Spin Dynamics Simulation Studies of Nanoscale Classical Heisenberg Antiferromagnets

by

Zhuofei Hou

(Under the direction of David P. Landau)

Abstract

Monte Carlo and spin dynamics techniques have been used to perform large-scale simulations of the dynamic behavior of a nanoscale, classical, Heisenberg antiferromagnet on a simple cubic lattice. Systems are of size $L \times L \times D$ with linear sizes $L \leqslant 40$ and $D \leqslant 40$ at a temperature below the Néel temperature. Nanoparticles are modeled with completely free boundary conditions, i.e., six free surfaces, and nanofilms are modeled with two free-surfaces in the spatial z-direction and periodic boundaries parallel to the surfaces in the x-,y-directions. Results are compared to those for the "infinite"system with fully periodic boundary conditions. The temporal evolutions of spin configurations were determined numerically from coupled equations of motion for individual spins using a fast spin dynamics algorithm based on the fourth-order Suzuki-Trotter decomposition of exponential operators, with initial spin configurations generated by Monte Carlo simulations. The local dynamic structure factor $S(\mathbf{r}_0, \mathbf{q}, \omega)$ was calculated from the local space- and time-displaced spin-spin correlation function, where $\mathbf{r}_0$ denotes the starting point from which the correlation function is calculated. Multiple excitation peaks for wave vectors within the first Brillouin zone appear in the spin-wave spectra for the transverse component of the dynamic structure factor $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ in the classical Heisenberg antiferromagnetic nanofilms and nanoparticles, which

are lacking if periodic boundary conditions are used. With the assumption of q-space spin-wave reflection with broken momentum conservation due to free-surface confinements, we successfully explained the locations of those excitations quantitatively in the linear dispersion region. Meanwhile, we also observed two novel quantized spin-wave excitation modes in the spatial z-direction in nanofilms for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$. Results of this study indicate the presence of new forms of spin-wave excitation behavior which have yet to be observed experimentally but could be directly tested through neutron scattering experiments on nanoscale $RbMnF_3$ particles or films.

INDEX WORDS: spin waves, correlation function, dynamic structure factor, neutron scattering experiment, dispersion relationship, Hybrid Monte Carlo, spin dynamics, Suzuki-Trotter decomposition, antiferromagnetic, nanostructure, classical Heisenberg model, completely free boundary conditions, partially free boundary conditions

Spin Dynamics Simulation Studies of Nanoscale Classical Heisenberg Antiferromagnets

by

Zhuofei Hou

B.S., Nankai University, China, 1998

M.S., University of Georgia, 2010

A Dissertation Submitted to the Graduate Faculty

of The University of Georgia in Partial Fulfillment

of the

Requirements for the Degree

Doctor of Philosophy

Athens, Georgia

2011

Spin Dynamics Simulation Studies of Nanoscale Classical Heisenberg Antiferromagnets

by

Zhuofei Hou

Approved:

Major Professor: David P. Landau

Committee: Heinz-Bernd Schüttler
William Dennis
Shan-Ho Tsai

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
August 2011

Dedication

This paper is dedicated to my wife Lijun, my daughter Sophia, and my parents.

Acknowledgments

I would first like to give my deepest gratitude to Prof. David P. Landau, my advisor and mentor for my six years as his student in the Center for Simulational Physics at the University of Georgia. I am grateful for his enthusiastic support and caring, his insightful advice and guidance with constant encouragement, and great patience throughout my entire research and academic years. I have immensely benefited from his knowledge and research experience on how to analyze problems with a strong attention to small details, meanwhile, without losing understanding physics as a big picture.

I would also like to thank all the other members of my advisory committee, Dr. Heinz-Bernd Schüttler, Dr. William Dennis and Dr. Shan-Ho Tsai, for their insights and perspective as physicists and for spending time serving as my committee members.

I appreciate greatly for the opportunity to collaborate with Dr. G. Malcolm Stocks and Dr. Greg Brown for providing research funding support for this work, as well as their hospitality during my visits to the Oak Ridge National Laboratory (ORNL).

I want to express my sincere appreciation to Dr. Shan-Ho Tsai. Besides serving as my committee member, Dr. Shan-Ho Tsai has generously spent time discussing with me about both physics and computing techniques applied in this work. I would like to express my appreciation to the guidance of Dr. Xiuping Tao, whose research project I took over, for her strong help and great patience at the beginning stage of my research.

All people in the Department of Physics and Astronomy offer me a friendly and supportive atmosphere. I want to thank Mr. Mike Caplinger and Mr. Jeff Deroshia, the network services specialists, for their diligent supportive work and help. I would also like to express my special appreciation to Ms. Linda Lee, the administrative specialist of the Center for Simulational Physics, for her extremely cordial and kind help.

I should also thank my church family who gave me and my family tremendous help and encouragement through years.

I thank my parents and my parents-in-law for being supportive all the time. Finally, in a very inadequate manner, I want to thank my dear wife and my dear little daughter for their constant love, support and standing with me as a family.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Magnetism is one of the oldest phenomena in solid state physics, but nevertheless still far from being fully understood, although there have been extensive experimental and theoretical studies of this phenomenon.

The equilibrium magnetic system has been the most investigated magnetic system, in which both static and dynamic magnetic properties of magnetism are studied. As one of the principal static magnetic properties, the *magnetic ordering* has been the most widely studied property since the turn of the century before last, when Curie [1] in 1895 and Weiss [2, 3] in 1904 and 1907 laid down the experimental and theoretical foundations for the quantitative study of static properties of magnetic system. They introduced the *Curie-Weiss law* which describes the dependence of magnetization on temperature in paramagnets, and which later was generalized to describe the phenomenon of ferromagnetism. As its name suggests, magnetic ordering introduces a regular ordered magnetic structure in many crystals; This means that in the absence of an external field the mean magnetic moment of at least one atom in each unit cell of the crystal is non-zero. *Ferromagnetism* is the most well known form of magnetic ordering. In addition to ferromagnetism, other types of spontaneous magnetic ordering of magnetic moments including *antiferromagnetism* and *ferrimagnetism* [4] were discovered in the middle of the century before last. In ferromagnets, the simplest type of magnetically ordered crystals, such as Fe, Ni, and Co, the mean magnetic moments of all the atoms have the same orientation provided that the temperature of the ferromagnet does not exceed a critical value, i.e., *Curie temperature* T_c. For this reason, ferromagnets have spontaneous magnetic moments, i.e., non-zero macroscopic magnetic moments, even in the

absence of an external field. In antiferromagnets such as carbonates, anhydrous sulphates, oxides and fluorides of the transition metals (Mn, Ni, Co, Rb and Fe), the mean atomic magnetic moments compensate each other within each unit cell in zero external magnetic field. In other words, an antiferromagnet consists of a set of antiparallel magnetic sublattices, each of which has a non-zero mean magnetic moment. This type of magnetic ordering occurs if the temperature of the antiferromagnet is lower than a critical temperature, known as the Néel temperature T_N. As for ferrimagnetism, another type of magnetic ordering, there exists a number of antiparallel magnetic sublattices whose magnetic moments are uncompensated in contrast to antiferromagnetism. Thus, ferrites exhibit spontaneous net magnetic moments even though there is no external field. Examples of this type magnetic crystal are compounds of transition metals, such as the salts $MnO{\cdot}Fe_2O_3$ and $3Y_2O_3{\cdot}5Fe_2O_3$.

As the temperature approaches the critical temperature, independent of the form of the magnetic ordering, usually a *second order phase transition* occurs between the ordered phase and the paramagnetic phase. The static critical behavior at the critical temperature is manifested by divergent correlation length and some thermodynamic quantities, e.g. specific heat C_v and magnetic susceptibility χ_m, according to power laws with a set of constant indices known as *static critical exponents*, e.g., α, β, γ, which describe the behavior of thermodynamic quantities near continuous phase transitions. It is a remarkable fact that phase transitions arising in different physical systems often possess the same set of static critical exponents. This phenomenon is known as *universality*. Based on the universality of static critical exponents, the principle of universality [5] was developed in 1967. This theory states that physical systems with different physics, Hamiltonian and geometries can be grouped into "Universality Classes" with common symmetry and common static critical exponents. Universality is a prediction of the *renormalization group theory* of phase transitions, which states that the thermodynamic properties of a system near a phase transition depend only on a small number of features, such as dimensionality and symmetry, and are insensitive to

the underlying microscopic properties of the system. The divergence of the correlation length is the essential point.

Other theoretical work have been done to study those static critical phenomena by introducing simple models, e.g., Ising model [6] and Heisenberg model [7]. A *mean field theory* was developed by Landau [8] in 1937 to describe those static critical phenomena quantitatively, but inaccurately, by ignoring magnetic fluctuations which become increasingly important as the system approaches the critical region. In 1944, the first analytical work was done by Onsager [9] by solving the 2-D Ising model exactly which provides an exact mathematical description of a second order transition. Further theoretical work on static critical phenomena, on models for which no exact solution is possible, have been carried out by using numerical series expansion techniques [5] and Monte Carlo simulations [10, 11]. With Monte Carlo simulations, the study of static critical phenomena has been widely expanded to even more complex physical systems [12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22].

The magnetic order in ferromagnets and antiferromagnets is the result of correlation between the directions of the electron spins on individual atoms. The correlation between the directions of atomic spins in magnetically ordered crystals leads to the existence of a particular type of collective mode in such crystals. To understand the origin of these modes consider to begin with a isotropic ferromagnet at $T = 0$. All the atomic magnetic moments then have the same direction which corresponds to minimum energy of the ferromagnet. Let us now deflect the magnetic moment of a particular atom and let go. This change of direction will not remain localized at the original atom, but owing to the presence of the correlation between spins, it will be propagated through the crystal in the form of a mode of wave motion. Such waves are called as *spin waves* which give rise to dynamic magnetic properties of magnetism in the crystal. Spin waves were theoretically discovered by Bloch [23] in 1930, and were first observed through the inelastic neutron scattering experiment developed by Brockhouse [24] in 1960. In this experiment, the energy loss of a beam of neutrons that excite a spin wave is measured, typically as a function of scattering vector (or equivalently

momentum transfer), temperature and external magnetic field. Inelastic neutron scattering measurements can determine the *dynamic structure factor* $S(\mathbf{q},\omega)$, which is the Fourier transform of the space- and time-displaced spin-spin correlation function, and the *dispersion curve* for spin waves, which defines a characteristic dependence of the frequency on the wave vector, i.e., dispersion relationship.

The theoretical study of the dynamic magnetic properties of magnetism has been carried out in three thermodynamic regimes: the hydrodynamic regime with $T < T_c$ and $q\xi \ll 1$, the critical regime with $T \sim T_c$ and $q\xi \gg 1$ and the high temperature regime with $T > T_c$ and $q\xi \ll 1$, where q is the wave vector of spin wave and ξ is the correlation length, which diverged as temperature approached the critical temperature T_c.

The *linear spin-wave theory*, which describes the collective mode of magnetic moments of atoms at the limit of zero temperature, is the simplest low-temperature approximation for the Heisenberg model. This theory describes the dynamic magnetic properties of low-temperature magnetism in terms of a dispersion curve expressing the spin-wave excitation energy frequency ω as a function of the wave vector q. A general theory of spin-wave interaction at the low temperature limit was developed by Dyson [25, 26] in 1956. In this theory, a temperature dependence of $T^{3/2}$ for the dispersion curve was given. In 1969, a general hydrodynamic spin-wave theory, which is applicable for the whole hydrodynamic regime, was developed by Halperin and Hohenberg [27] in 1969. Later, renormalization group theory was developed and applied to study the dynamic magnetic properties in both the hydrodynamic regime [28] and critical regime [29].

In the critical regime, the dynamic properties are defined by the phenomenon of *critical slowing down*, in which as the temperature approaches to T_c the relaxation time goes to infinity, and thus the system approaches thermal equilibrium extremely slowly. This phenomenon was proposed by Van Hove [30] and Landau [31] in 1954. They pointed out that the transport or kinetic coefficient in no case can diverge as fast as the magnetic susceptibility. Later in 1969, in their work on the theory of dynamic critical phenomena, Halperin and

Hohenberg [32] developed quantitative scaling laws to describe this phenomenon by introducing the classification of the different dynamic universality classes in terms of a *dynamic critical exponent* z, which depends on the conservation laws, lattice dimension and the static critical exponents. According to the classification of dynamic universality classes proposed by Hohenberg and Halperin [33] in 1977, the classical Heisenberg ferromagnet is of class J for which the order parameter (the uniform magnetization) is conserved in the critical dynamics, and the classical Heisenberg antiferromagnet is of class G for which the order parameter (the staggered magnetization) is not conserved. To study the critical dynamics, spin dynamics simulation techniques have been extensively used. The spin dynamics simulation technique is a simulation technique to simulate the real time evolution of magnetic spin system and further to determine the dynamic structure factor $S(\mathbf{q}, \omega)$ at a specific temperature.

The deterministic time-dependent dynamic properties of "infinite" bulk magnetic systems with periodic boundary conditions (PBC) has been extensively studied by experiments [34, 35, 36] and spin dynamics simulations [37, 38, 39, 40, 41, 42] with classical Heisenberg models. Early simulations for the transverse component of the dynamic structure factor, $S^T(\mathbf{q}, \omega)$, on isotropic, antiferromagnetic body-centered cubic systems at temperatures below the critical temperature T_c, show a single spin-wave excitation peak of finite intensity with finite width, becoming narrow and increasing in excitation energy frequency ω as T decreases. The behavior approaches the predictions of linear spin-wave theory, and a diffusive central peak appears increasing in strength with increasing T at $\omega = 0$, which is in qualitative agreement with experiments [35, 36]. Large-scale computer simulations carried out by Tsai, Bunker and Landau [39] on antiferromagnetic, isotropic, simple cubic systems below T_c found that by fitting the line shape of $S^T(\mathbf{q}, \omega)$ to a Lorentzian form [28], in the [100] direction the dispersion curves are approximately linear for wave vector within the first Brillouin zone. For increasing T towards T_c the dispersion curve turns into a power law, reflecting the crossover from hydrodynamics to critical behavior with the dynamic critical exponent estimated to be $z = 1.43(0.03)$, which is in agreement with the experimental esti-

mate of the dynamic critical exponent $z = 1.43(0.04)$ [36]. This estimate is slightly lower than the theoretical predicted value [29, 32, 33, 43] of $z = 1.5$ for an isotropic, three-dimensional Heisenberg antiferromagnet. With larger lattice sizes, and thus smaller values of q than previous simulations, Tsai and Landau [41] probed the asymptotic critical region in momentum to the limit of $q \sim 0$, and estimated $z = 1.49(0.03)$, which is in good agreement with the renormalization group theory and dynamic scaling predictions. The dynamic behavior of the longitudinal component of the dynamic structure factor, $S^L(\mathbf{q}, \omega)$, has been studied by Bunker and Landau [44]. For both the isotropic and anisotropic antiferromagnets, both annihilation and creation two-spin-wave peaks are observed. The splitting of longitudinal spin-wave peak into two spin-wave peaks with the energy separation of twice the energy gap at the Brillouin zone center are predicted for all anisotropic antiferromagnets.

Recent developments in the field of magnetic material applications brought much attention to the static and dynamic properties of confined magnetic elements of small dimensions [45]. Recent experiments [46, 47, 48] on micron-scale array elements showed quantized and localized spin-wave excitation modes as eigen-excitations by the selection rules introduced by laterally confined boundary conditions of the elements. In addition, the intrinsic broken translational invariance caused by confinement effects in one or more directions in those small laterally confined magnetic elements leads to a broken conservation law of corresponding momentum for a spin wave [49]. The broken conservation law of momentum brings uncertainty into the wave vector for a specific spin-wave excitation energy. This uncertainty is reported to be inversely proportional to the confinement length [50]. Therefore, instead of a continuous spin-wave spectrum with spin-wave excitation energy uniquely determined by each wave vector, quantized spin-wave excitation modes, each of them observed within a given wave-vector interval, are obtained from those experiments.

As for nanoscale magnetic systems, extensive Monte Carlo simulations have been performed recently by Brown *et al* [51, 52] to study *thermoinduced magnetization* (TiM) in antiferromagnetic nanoparticles. TiM is a ferromagnetic response predicted to occur in nanopar-

ticles of normally antiferromagnetic materials, with the magnetization vanishing at $T = 0$ and increasing linearly with increasing temperature in the vicinity of zero point. TiM is predicted to be an intrinsic property of the antiferromagnetic Heisenberg model below the Néel temperature T_N, and have a volume dependence as a function of both temperature and anisotropy.

In order to gain further understanding of the dynamic properties of nanoscale magnetic systems, in this study we carried out large-scale spin dynamics simulations of the dynamic behavior of the nanoscale classical Heisenberg antiferromagnet on a simple cubic lattice. We modeled nanoparticles and nanofilms with completely and partially free boundary conditions, respectively. We focus mainly on the spin-wave excitation spectra for the transverse component of the local dynamic structure factor $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ obtained in the nanoscale, classical Heisenberg isotropic antiferromagnet at a temperature below the Néel temperature but also have results for nanoscale, anisotropic Heisenberg antiferromagnet.

The remainder of the contents is as follows: Chapter 2 contains the theoretical background and the introduction of inelastic neutron scattering experiment. First, related fundamental concepts of thermodynamics, statistical physics and magnetism are introduced. Second, the theory of classical spin dynamics and the general linear spin-wave theory are discussed. Third, inelastic neutron scattering techniques and scattering function are described. Chapter 3 presents details of the simulation techniques applied in this work, including model and boundary condition setup, hybrid Monte Carlo methods, derivation of equations of motion for classical spin systems, Suzuki-Trotter decomposition algorithm for the integration for solving the equations of motion, and post analysis methods. In Chapter 4, we present and discuss our simulation results and show how conclusions are drawn. First, several static results for nanoscale spin systems are presented. Next, dynamic results of local dynamic structure factor for those nanoscale spin systems are presented. Chapter 5 presents conclusions of our work. Appendix A and B list our data generating and analyzing programs developed in C++ for the simulations. The $\Psi - Mag$ Toolset (or Toolkit) [53], a secondary

C++ template library developed by Oak Ridge National Laboratory (ORNL) for computational magnetism and serving as a prototype for a more general library for computational material science , was utilized in this work.

Chapter 2

Experimental and Theoretical Background

2.1 Fundamental Concepts of Thermodynamics and Statistical Physics

In statistical mechanics, the partition function Z encodes the statistical properties of a system in thermodynamic equilibrium. It is a function of temperature and other parameters, such as the Hamiltonian of the system. Most of the aggregate thermodynamic variables of the system, such as the internal energy, free energy, specific heat, and susceptibility, can be expressed in terms of the partition function or its derivatives. The partition function for a canonical ensemble is defined as [54]

$$Z = \sum_{all\ states} e^{-\beta \mathcal{H}}, \tag{2.1}$$

where the "inverse temperature", β, is defined as $\beta = 1/k_B T$, with k_B denoting the Boltzmann constant, and $\mathcal{H}$ denoting the Hamiltonian of the system. The term $e^{-\beta \mathcal{H}}$ is known as the Boltzmann factor. The summation defined in Eqn.(2.1) is over all possible microstates of the system and should be rewritten as appropriate integrals for a system with continuous degrees of freedom, e.g., Heisenberg spin system. The partition function can be related to thermodynamic properties because it has a very important statistical meaning. The probability P_s that the system occupies microstate s is given by

$$P_s = e^{-\beta \mathcal{H}(s)}/Z. \tag{2.2}$$

And the free energy of a system can be determined by

$$F = -\ln Z/\beta. \tag{2.3}$$

Other thermodynamic quantities can be calculated from the derivatives of the free energy defined in Eqn.(2.3). The internal energy is given by

$$U = k_B T^2 \frac{\partial \ln Z}{\partial T}. \tag{2.4}$$

The equilibrium statistical expected value, or ensemble average for a thermodynamic quantity A at a specific temperature, such as thermodynamic energy, magnetization and specific heat, can be determined by

$$\langle A(T) \rangle = \frac{1}{Z} \sum_s A(s) e^{-\beta \mathcal{H}(s)}, \tag{2.5}$$

where $\langle \ldots \rangle$ denotes the ensemble average and $A(s)$ is the microscopic value of A in the s-th state. Thus, the first moment of $\mathcal{H}$, i.e. internal energy $U \equiv \langle \mathcal{H} \rangle$, and its second moment $\langle \mathcal{H}^2 \rangle$ are given by

$$\begin{aligned} \langle \mathcal{H} \rangle &= \sum_s \mathcal{H}(s) e^{-\beta \mathcal{H}(s)} / \sum_s e^{-\beta \mathcal{H}(s)}, \\ \langle \mathcal{H}^2 \rangle &= \sum_s \mathcal{H}^2(s) e^{-\beta \mathcal{H}(s)} / \sum_s e^{-\beta \mathcal{H}(s)}. \end{aligned} \tag{2.6}$$

Therefore, we obtain the specific heat defined in terms of the fluctuations of the internal energy as [54]

$$k_B T^2 C_\nu = \langle \mathcal{H}^2 \rangle - \langle \mathcal{H} \rangle^2 = \langle (\mathcal{H} - \langle \mathcal{H} \rangle)^2 \rangle_{NVT} = \langle (\triangle U)^2 \rangle_{NVT}. \tag{2.7}$$

Similar fluctuation relations also hold for many other quantities, such as the isothermal susceptibility $\chi = (\partial \langle M \rangle / \partial H)_T$ is defined in terms of the fluctuation of the magnetization M as

$$k_B T \chi = \langle M^2 \rangle - \langle M \rangle^2. \tag{2.8}$$

This important equation is known as the *fluctuation-response theorem* [55], which shows that the linear response of a magnet to a small magnetic field reduces to the probing of fluctuations that are present even in the absence of the magnetic field. The larger the fluctuations, the higher the susceptibility. In turn, the high susceptibilities indicate large fluctuations, as in the vicinity of critical Curie point [56].

2.2 Theory of Magnetism

2.2.1 Local Magnetic moment and the Curie-Weiss Law

The modern theory of magnetism started at the turn of the twentieth century with the theoretical study made in understanding magnetic transitions with various theoretical models [57]. Those models have in common the feature that they assume the magnetic moments with fixed sizes to be localized on fixed lattice sites, and that they influence one another through pairwise interactions with an energy that achieves its maximum value of J when the moments are parallel or antiparallel in orientation.

Using the concept of local magnetic moment with a fixed size, Langevin [58] in 1905 firstly explained the Curie law of magnetic susceptibility. He started with a set of atomic magnetic moments each with a fixed magnitude m under an external magnetic field H applied in the z direction. Thus, the statistical mean value of the magnetization is parallel to the external field and its value per atom at a temperature T is given by

$$\begin{aligned} \langle m_z \rangle &= \int d\Omega m \cos\theta \exp\left(mH\cos\theta/k_BT\right) / \int d\Omega \exp\left(mH\cos\theta/k_BT\right) \\ &= mL(x), \\ L(x) &= \langle \cos\theta \rangle = \coth x - 1 = x/3 - x^3/45 + \ldots, \\ x &= mH/k_BT, \end{aligned} \tag{2.9}$$

where $m\cos\theta$ is the z component of a magnetic moment, k_B the Boltzmann constant and the integrals are over the solid angle $d\Omega$. $L(x)$ is known as the *Langevin function*, which gives the component parallel to H of a unit vector in the direction of the magnetization, by definition of the angle θ, namely,

$$\frac{\langle m_z \rangle}{m} = \langle \cos\theta \rangle = L(mH/k_BT). \tag{2.10}$$

Using an expansion form of $L(x)$ gives the following expression for the magnetic susceptibility which is inversely proportional to the temperature T, which is also known as *Curie's*

Law:

$$\chi = N_0 \lim_{H\to 0} \langle m_z \rangle / H = N_0 m^2 / 3k_B T \equiv \mathcal{C}/T, \tag{2.11}$$

where N_0 is the number of atoms in the crystal, and $\mathcal{C}$ is the *Curie Constant*.

In 1907, two years after Langevin's theory of paramagnetism, Pierre Weiss [59] proposed a phenomenological theory of ferromagnetism in which he assumed that the atomic magnetic moments in solids interact with one another through a molecular field, i.e., *Weiss molecular field*, proportional to the average magnetization. To approximate its effect, a molecular field term $\Gamma\langle m_z \rangle$ was added to the external field in the Langevin equation Eqn.(2.9) and obtained

$$\begin{aligned} \langle m_z \rangle &= mL(y), \\ y &= m(H + \Gamma\langle m_z \rangle)/k_B T, \end{aligned} \tag{2.12}$$

Ferromagnetism is described by Eqn.(2.12) with $\langle m_z \rangle > 0$ for $H = 0$. By using the expansion form Eqn.(2.9) of $L(y)$, the condition for ferromagnetism is given by

$$T < T_c = m^2 \Gamma / 3k_B, \tag{2.13}$$

where T_c is the *Curie temperature* below which ferromagnetism occurs. The susceptibility at $T > T_c$ is obtained from Eqn.(2.12) as

$$\chi = \mathcal{C}/(T - T_c), \tag{2.14}$$

where $\mathcal{C}$ is the *Curie Constant* given by Eqn.(2.11). Equation (2.14) is known as the *Curie-Weiss law*, which applies both for ferromagnets and for antiferromagnets. The temperature dependence of magnetization and that of the inverse susceptibility as obtained by the Weiss theory are shown in Figure 2.1. These forms of M vs. T and χ^{-1} vs. T relations are commonly observed in almost all ferromagnets.

The approximation approach of the *Curie-Weiss law* to ferromagnetism is widely known as the *mean-field theory* or *molecular-field theory*. In this theory the exchange interaction is replaced by an average molecular field; deviations from the average, i.e. fluctuations of

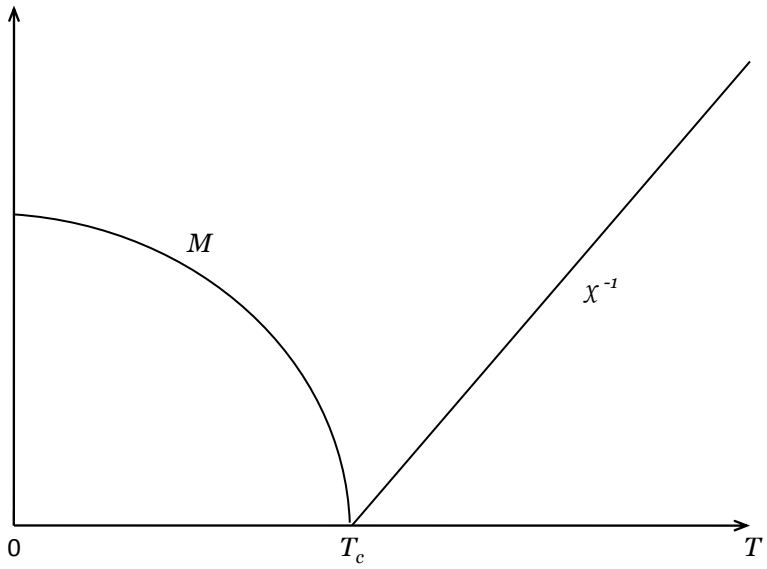

Figure 2.1: The temperature dependence of magnetization and that of the inverse magnetic susceptibility according to the Weiss theory.

the molecular field, are ignored. Therefore, this approximation is not appropriate especially in the paramagnetic region close to the transition temperature T_c. The observed transition temperature to the ordered state is expected to be generally lower than predicted by the molecular-field approximation. There have been many attempts to extend the molecular-field theory to take account of the energy due to the spin correlations. Representative approaches include the methods proposed by Weiss [60] in 1948, which corresponds to the Bethe approximation for the order-disorder transition in binary alloys, and the constant-coupling approximation [61, 62].

2.2.2 Critical Exponents and Scaling Hypothesis

Magnetic materials generally undergo a *phase transition* at a critical temperature T_c, which is accompanied by singular behavior in physics quantities such as the susceptibility and the specific heat. The singularities are characterized by *critical exponents* which are universal quantities. More modern studies of the *critical region*, namely the near-vicinity of the Curie point, are based on two general assumptions, or axioms [65]. The first one is that the *asymptotic* behavior governing the approach to the critical point, i.e., the Curie temperature T_c, of all physical quantities is a power law in $|T - T_c|$.

In accordance with this basic assumption, for the limit $T \to T_c$, several critical exponents, or *critical indices*, are defined. In particular, for the specific heat,

$$C \sim |T - T_c|^{-\alpha}, \tag{2.15}$$

for the spontaneous magnetization below T_c,

$$M \sim |T - T_c|^{-\beta}, \tag{2.16}$$

and for the magnetic susceptibility,

$$\chi \sim |T - T_c|^{-\gamma}. \tag{2.17}$$

It can be proven by general thermodynamics that α and γ have the same value for both $T > T_c$ and $T < T_c$. The three exponents in the above equations, i.e. α, β, and γ, are called "critical exponents", and there exists a simple relation, the so-called *Rushbrooke equality*, relating the exponents [54]

$$\alpha + 2\beta + \gamma = 2. \tag{2.18}$$

In principle, the power laws in Eqns.(2.15 - 2.17) are based on experimental observations [63], and are not just arbitrary assumptions.

The second basic assumption of the theories of critical exponents is known as the *scaling hypothesis*. It assumes first of all that ξ, the spin *correlation length*, over which fluctuations of the magnetization are correlated, increases and diverges at $T = T_c$ with decreasing

temperature towards to T_c, i.e.,

$$\xi \sim |T - T_c|^{-\nu}, \tag{2.19}$$

where ν is the critical exponent defining the diverging behavior of ξ in the near-vicinity of the Curie point. It further assumes that in the critical region, the dominating temperature-dependence of all the physical quantities of the system is only through their dependence on ξ. If the length scale is increased by a factor, the correlation length shrinks by the same factor. The quantity of $|T - T_c|$ then increases according to Eqn.(2.19), and all the physical quantities will also change by fixed power laws. However, $\xi \to \infty$ as $T \to T_c$, and the scaled system can be *renormalized*, namely be mapped back on the original system. This procedure is the basis of an important tool for calculating the critical exponents, known as the renormalization group theory [64].

2.3 Theory of Classical Spin Dynamics

Spin-wave theory is a good approximation to study dynamic properties of spin systems described by the *Heisenberg Hamiltonian*. Consider an excited state in a $S = \frac{1}{2}$ spin system in which a single spin is flipped from the complete ferromagnetic state, i.e. the ground state in which all spins are aligned. This excited state is not an eigenstate of the Heisenberg Hamiltonian. Due to the exchange interaction, the flipped spin is not localized and moves around in the lattice. Thus, the eigenstate is a state in which a wave of spins is excited. At low temperatures, states excited above the ground state can be well approximated as a collection of independent spin waves. Such a spin-wave theory was proposed by Bloch [66] in 1930. The $T^{3/2}$ law for the decrease of the spontaneous magnetization of ferromagnets was derived from this theory.

2.3.1 The Heisenberg Model

A key aspect of magnetic modeling is the description of individual spins. The spins may be quantum mechanical or classical, isotropic or anisotropic. A very important model is the

Heisenberg model. For a $S = \frac{1}{2}$ spin system, the quantum-mechanical Heisenberg model in terms of the *quantum Heisenberg Hamiltonian* is given by

$$\mathcal{H} = -2\sum_{i>j} J_{ij}\hat{\mathbf{S}}_i \cdot \hat{\mathbf{S}}_j - g\mu_0\mu_B \sum_i \mathbf{H}_i \cdot \hat{\mathbf{S}}_i, \tag{2.20}$$

where $\hat{\mathbf{S}}$ is the spin operator closely related to Pauli matrices as $\hat{\mathbf{S}} = \frac{1}{2}\sigma$. μ_0 is the vacuum permeability and μ_B is the Bohr magneton. Ignoring the orbital magnetic moment contribution, $g = 2.0023$. $\mathbf{H}_i$ is the local magnetic field acting on the i-th spin. The spin in the $S = \frac{1}{2}$ Heisenberg model has two states ($\uparrow$ and $\downarrow$). Compare to the above quantum-mechanical Heisenberg model, the classical Heisenberg model with $S = \infty$ has a continuum of states, which is defined by the *classical Heisenberg Hamiltonian* as

$$\mathcal{H} = -2\sum_{i>j} J_{ij}\mathbf{S}_i \cdot \mathbf{S}_j - g\mu_0\mu_B \sum_i \mathbf{H}_i \cdot \mathbf{S}_i, \tag{2.21}$$

where $\mathbf{S}_i$ is the classical spin vector with $\mid \mathbf{S}_i \mid = 1$. J_{ij} in Eqn.(2.20) and Eqn.(2.21) denotes the coupling coefficient of exchange interaction between the i-th and j-th spins. As introduced in Chapter 1, the exchange interaction comes from a purely quantum-mechanical exchange effect which results from the fact that electrons obey Fermi-Dirac statistics. In both quantum-mechanical and classical models, the J_{ij} may be positive (ferromagnetism) or negative (antiferromagnetism). The Heisenberg exchange interaction is not restricted to the nearest neighbors (nn), although the interactions tend to rapidly decrease with an increasing distance to the second-nearest neighbors (sn) and even more distant neighbors.

The classical Heisenberg Hamiltonian with anisotropy in this work takes the form as

$$\mathcal{H} = -\sum_{<i,j>} J_{ij}(S_i^x S_j^x + S_i^y S_j^y + \lambda S_i^z S_j^z) + D\sum_i (S_i^z)^2, \tag{2.22}$$

where J_{ij} is the nearest-neighbor coupling coefficient of exchange interaction in the nearest-neighbor spin pair denoted as $< i, j >$. λ is the exchange anisotropy and D is the single-site anisotropy. For an *isotropic* system, $\lambda = 1$ and $D = 0$. For a *uniaxial* anisotropic system, in which the spins intrinsically tend to align to z-axis, $\lambda \geq 1$ and $D > 0$. For a *planar*

anisotropic system, in which the spins intrinsically tend to align to xy-plane, $\lambda \leq 1$ and $D < 0$. Figure 2.2 shows three energy landscapes for isotropic, uniaxial anisotropic, and planar anisotropic systems, respectively.

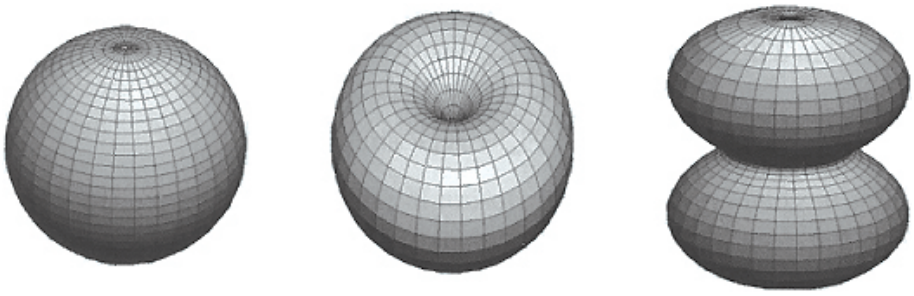

Figure 2.2: Anisotropy-energy landscapes: isotropic ($\lambda = 1$ and $D = 0$) landscape with an energy uniformity in all spatial orientations (left); uniaxial anisotropic ($\lambda \geq 1$ and $D > 0$) landscape with an easy axis along the z-axis (middle); planar anisotropic ($\lambda \leq 1$ and $D < 0$) landscape with an easy plane in the xy-plane.

2.3.2 Equations of Motion for the Spin Dynamics

The Heisenberg model has true dynamics with the real time evolution of spins governed by the *equations of motion* (EOM). The general recipe of spin dynamics is to generate initial spin states drawn from a canonical ensemble using Monte Carlo methods, and to use these as starting states for the integration of the coupled equations of motion. The SD method has been effective for the study of the dynamic properties of Heisenberg ferromagnets [37, 42] and antiferromagnets [38, 40, 41, 39].

The equations of motion can be derived from the quantum-mechanical commutator

$$\frac{d\hat{\mathbf{S}}_i}{dt} = -\frac{i}{\hbar}[\hat{\mathbf{S}}_i, \mathcal{H}]. \tag{2.23}$$

As an example, we can consider the Heisenberg Hamiltonian defined in Eqn.(2.22) for the quantum-mechanical isotropic case, i.e., $\mathcal{H} = -\sum_{<j,k>} J_{jk}\hat{\mathbf{S}}_j \cdot \hat{\mathbf{S}}_k$. The x-component of the

commutator given in Eqn.(2.23) is

$$\begin{aligned}
&\left[\hat{S}_i^x, -\sum_{<j,k>} J_{jk}(\hat{S}_j^x\hat{S}_k^x + \hat{S}_j^y\hat{S}_k^y + \hat{S}_j^z\hat{S}_k^z)\right] \\
= &\left[\hat{S}_i^x, -\sum_{<j,k>} J_{jk}\hat{S}_j^x\hat{S}_k^x\right] + \left[\hat{S}_i^x, -\sum_{<j,k>} J_{jk}\hat{S}_j^y\hat{S}_k^y\right] + \left[\hat{S}_i^x, -\sum_{<j,k>} J_{jk}\hat{S}_j^z\hat{S}_k^z\right] \\
= &\ 0 + \left[\hat{S}_i^x, -\sum_{<j,k>} J_{jk}\hat{S}_j^y\hat{S}_k^y\right] + \left[\hat{S}_i^x, -\sum_{<j,k>} J_{jk}\hat{S}_j^z\hat{S}_k^z\right] \\
= &\left[\hat{S}_i^x, -\sum_{<j,k>} J_{jk}\hat{S}_j^y\hat{S}_k^y\right] + \left[\hat{S}_i^x, -\sum_{<j,k>} J_{jk}\hat{S}_j^z\hat{S}_k^z\right] \\
= &-\sum_k J_{ik}\hat{S}_k^y\left[\hat{S}_i^x, \hat{S}_i^y\right] - \sum_k J_{ik}\hat{S}_k^z\left[\hat{S}_i^x, \hat{S}_i^z\right] \\
= &-i\sum_k J_{ik}\hat{S}_k^y\hat{S}_i^z + i\sum_k J_{ik}\hat{S}_k^z\hat{S}_i^y \\
= &-i\sum_k J_{ik}(\hat{S}_k^y\hat{S}_i^z - \hat{S}_k^z\hat{S}_i^y) \\
= &\ i(\hat{\mathbf{H}}_i^{eff}\times\hat{\mathbf{S}}_i)_x,
\end{aligned} \tag{2.24}$$

where $\hat{\mathbf{H}}_i^{eff}$ is an *effective field* acting on $\hat{\mathbf{S}}_i$ defined as

$$\hat{\mathbf{H}}_i^{eff} \equiv -2\sum_j J_{ij}\hat{\mathbf{S}}_j, \tag{2.25}$$

where the factor 2 comes from the fact that J_{ij} is just half of the actual bonding energy between $\hat{\mathbf{S}}_i$ and $\hat{\mathbf{S}}_j$ according to Eqn.(2.22). Combining Eqn.(2.23) and Eqn.(2.24), we obtain

$$\frac{d\hat{\mathbf{S}}_i}{dt} = \frac{1}{\hbar}\hat{\mathbf{H}}_i^{eff}\times\hat{\mathbf{S}}_i. \tag{2.26}$$

Let $S\to\infty$ and normalize the length of spin to unity, we obtain the *classical equation of motion* as

$$\frac{d\mathbf{S}_i}{dt} = \frac{1}{\hbar}\mathbf{H}_i^{eff}\times\mathbf{S}_i. \tag{2.27}$$

where $\mathbf{S}$ is a classical spin vector with unit length. In numerical operations, $\hbar$ is usually taken as 1 to reduce units.

Although the Eqn.(2.27) is derived for an isotropic Heisenberg Hamiltonian, it is also valid for the general Hamiltonian form defined in Eqn.(2.22).

2.3.3 Linear Spin-Wave Theory

At the temperature limit $T \to 0$, spins deviate slightly from the ground state of complete ordering and form an excitation of very low energy. The elementary excitations are in the form of linear spin waves with an infinite lifetime. As a result of infinite lifetime, the dynamic structure factor for a linear spin wave is a delta function at the value of $\omega(q)$ defined by the dispersion curve.

Kittel [4] gave a classical description of the linear spin-wave dispersion relationship for one- and three-dimensional ferromagnets on simple cubic lattices with nearest-neighbor interactions. Ashcroft and Mermin [68] took a quantum-mechanical approach considering arbitrary exchange interaction range to get the dispersion relation. Here we will not show the details of their work on the linear spin-wave calculations.

At the long wavelength limit $q \to 0$, independent of lattice structure and direction in q-space, the dispersion relation for isotropic ferromagnets is quadratic as

$$\omega = D(0)q^2, \tag{2.28}$$

and the dispersion relation for isotropic antiferromagnets is linear as

$$\omega = D(0)q, \tag{2.29}$$

where $D(0)$ is the spin-wave *stiffness coefficient* at zero temperature, defined as

$$D(0) = 2JSa^2, \tag{2.30}$$

where J is the nearest-neighbor exchange interaction coefficient, S is the spin length, and a is the lattice parameter. The dispersion curve in the [100], [110] and [111] directions in q-space are shown as a summary in Table 2.1 for simple-cubic (SC) isotropic ferromagnetic and antiferromagnetic systems.

Table 2.1: Linear spin-wave dispersion curve for the simple-cubic isotropic ferromagnet and antiferromagnet with the nearest-neighbor only interactions

Simple-cubic	Ferromagnet	Antiferromagnet
[100]	$4J\sin^2(\frac{qa}{2})$	$J\sqrt{2\big[9-8\cos(qa)-\cos(2qa)\big]}$
[110]	$8J\sin^2(\frac{qa}{2})$	$2J\sqrt{2\big[3-2\cos(qa)-\cos(2qa)\big]}$
[111]	$12J\sin^2(\frac{qa}{2})$	$6J\sin(qa)$

From the linear spin-wave theory, Bloch [66, 67] obtained the so called Bloch $T^{3/2}$ law for general interaction range. This law stats that at low temperature the spontaneous magnetization should deviate from its saturation value by an amount proportional to $T^{3/2}$, i.e.,

$$M(T) = M(0)\Big[1 - BT^{3/2} + \mathcal{O}(T^{5/2})\Big]. \tag{2.31}$$

Bloch $T^{3/2}$ law has been well confirmed by the experiment [69] in1964.

For a quantum-mechanical spin system, the temperature dependence of the spin-wave stiffness [26] in the low temperature limit is given by

$$D(T) = D(0)(1 - AT^{5/2}). \tag{2.32}$$

The zeroth order approximation, which is approached at the low temperature limit, for both a quantum-mechanical and a classical spin system is

$$D(T) = D(0)M = \begin{cases} D(0)(1 - AT^{3/2}), & \text{for} \quad spin - \frac{1}{2} \\ D(0)(1 - AT), & \text{for} \quad spin - \infty \end{cases} \tag{2.33}$$

where M is the order parameter, either uniform magnetization or staggered magnetization. Since the temperature dependence of magnetization varies from $\sim (1 - AT^{3/2})$ for spin-$\frac{1}{2}$ to

$\sim (1 - AT)$ for spin-∞, we can see there is a breakdown at the low temperature limit where the quantum fluctuation effect becomes increasingly important.

2.4 Dynamic Structure factor

The magnetism of materials originates from the magnetic moment of electrons, which are either itinerant or localized. The electron has a characteristic angular momentum $\hbar\mathbf{s}$ associated with its spin degree of freedom, and has *spin magnetic moment* as

$$\mathbf{u_s} = -g\mu_B\mathbf{s}, \tag{2.34}$$

where μ_B is the *Bohr magneton*; g is called the g-value and the magnitude for free electron is $g = 2.0023$.

In addition to the spin magnetic moment, the electron also has the *orbital magnetic moment* as

$$\mathbf{u_o} = -\frac{e}{2mc}(\mathbf{r} \times \mathbf{p}) = -\mu_B\mathbf{l}, \tag{2.35}$$

which arise from the orbital motion of the electron. $\mathbf{r}$ is the position vector of the electron; $\mathbf{p}$ is the momentum and $\mathbf{l}$ is the angular momentum divided by $\hbar$.

The atomic nuclei in materials also have magnetic moments. Similar to Eqn.(2.34), the magnetic moment of a nucleus with nuclear spin $\mathbf{I}$ is

$$\mathbf{u_N} = -g_N\mu_N\mathbf{I}, \tag{2.36}$$

where μ_N is called the *nuclear magneton* with the magnitude of $\mu_N = \mu_B/1836$; g_N is a quantity corresponding to the g-value and is of order of unity.

In most cases, the orbital contribution is negligibly small, and the magnetic moments of atomic nuclei can also be ignored compared with those of electrons. The "magnetic"electrons, either itinerant or localized, reside in partly occupied energy bands, which are often the highest occupied energy states in solid. Thus, those electrons also contribute to the binding energy and bulk thermodynamic properties of the solid. An understanding of the origin of

electron magnetic behavior is not only significant for the understanding of magnetism itself, but also important for the understanding of other fundamental solid state properties.

For most materials, the dynamic properties of the magnetism are defined by the time dependence of the site energies and the spin orientations. These can be uniquely determined in terms of pair correlation function.

To illustrate the quantitative relation between the dynamic structure factor and the correlation function, we define the space- and time-displaced spin-spin correlation function $C^k(\mathbf{r}-\mathbf{r}',t)$ for a periodic lattice as

$$C^k(\mathbf{r}-\mathbf{r}',t) = \langle S^k_{\mathbf{r}}(t) S^k_{\mathbf{r}'}(0)\rangle - \langle S^k_{\mathbf{r}}(t)\rangle\langle S^k_{\mathbf{r}'}(0)\rangle, \tag{2.37}$$

where $\langle \ldots \rangle$ denotes the ensemble average; $k = x$, y or z; $S^k_{\mathbf{r}}(t)$ stands for the k component of a spin at the lattice site $\mathbf{r}$ and time t. For a system with periodic boundary conditions the second term in Eqn.(2.37) gives a constant term of M^2 and can be dropped in a Fourier transformation to get the dynamic structure factor $S^k(\mathbf{q},\omega)$, which is defined by

$$\begin{aligned} S^k(\mathbf{q},\omega) &= \frac{1}{N}\sum_{\mathbf{r},\mathbf{r}'} e^{i\mathbf{q}\cdot(\mathbf{r}-\mathbf{r}')} \int_{-\infty}^{+\infty} e^{-i\omega t} C^k(\mathbf{r}-\mathbf{r}',t)\frac{dt}{\sqrt{2\pi}} \\ &= \frac{1}{N}\sum_{\mathbf{r},\mathbf{r}'} e^{i\mathbf{q}\cdot(\mathbf{r}-\mathbf{r}')} \int_{-\infty}^{+\infty} e^{-i\omega t} \langle S^k_{\mathbf{r}}(t) S^k_{\mathbf{r}'}(0)\rangle \frac{dt}{\sqrt{2\pi}}, \end{aligned} \tag{2.38}$$

where N is the total number of spins in a lattice. The factor $1/N$ comes from the discrete space Fourier transform and it ensures the inverse space Fourier transform of $S^k(\mathbf{q},t)$ recovers $C^k(\mathbf{r}-\mathbf{r}',t)$, i.e.,

$$\begin{aligned} & \sum_{\mathbf{q}} e^{i\mathbf{q}\cdot(\mathbf{r}-\mathbf{r}')} S^k(\mathbf{q},t) \\ = & \sum_{\mathbf{q}} e^{i\mathbf{q}\cdot(\mathbf{r}-\mathbf{r}')} \left(\frac{1}{N}\sum_{\mathbf{r_1},\mathbf{r}'_1} C^k(\mathbf{r_1}-\mathbf{r}'_1,t) e^{i\mathbf{q}\cdot(\mathbf{r_1}-\mathbf{r}'_1)} \right) \\ = & \frac{1}{N}\sum_{\mathbf{r_1},\mathbf{r}'_1} C^k(\mathbf{r_1}-\mathbf{r}'_1,t) \sum_{\mathbf{q}} e^{i\mathbf{q}\cdot((\mathbf{r_1}-\mathbf{r}'_1)-(\mathbf{r}-\mathbf{r}'))} \\ = & \frac{1}{N}\sum_{\mathbf{r_1},\mathbf{r}'_1} C^k(\mathbf{r_1}-\mathbf{r}'_1,t) N\delta_{\mathbf{r_1}-\mathbf{r}'_1,\mathbf{r}-\mathbf{r}'} \\ = & C^k(\mathbf{r}-\mathbf{r}',t). \end{aligned} \tag{2.39}$$

2.5 Inelastic Neutron Magnetic Scattering Experiment

Inelastic neutron scattering is an experimental technique commonly used in condensed matter research to study atomic and molecular motion as well as magnetic and crystal field excitations. It distinguishes itself from other neutron scattering techniques by resolving the change in kinetic energy that occurs when the collision between neutrons and the sample is an inelastic one. Results are generally obtained as the dynamic structure factor $S(\mathbf{q},\omega)$, sometimes also as the dynamic magnetic susceptibility $\chi(\mathbf{q},\omega)$. The dynamic structure factor is related to the dynamic magnetic susceptibility $\chi(\mathbf{q},\omega)$ by [70]

$$S(\mathbf{q},\omega) = \mathcal{I}\big[\chi(\mathbf{q},\omega)\big], \tag{2.40}$$

where $\mathcal{I}\big[\dots\big]$ denotes taking the imaginary part, the scattering vector $\mathbf{q}$ is the difference between incoming and outgoing wave vector, and $\hbar\omega$ is the energy change experienced by the sample. The scattering results of the dynamic structure factor $S(\mathbf{q},\omega)$ and the dynamic magnetic susceptibility $\chi(\mathbf{q},\omega)$ can be plotted as functions of ω, which can often be interpreted in the same way as the spectra obtained by conventional spectroscopic techniques.

2.5.1 Properties of Neutrons

Thermal neutrons, which are free neutrons that are Boltzmann distributed with $k_BT = 0.0253eV \sim 300K$ at room temperature [71], have been used in the inelastic neutron scattering experiment to explore many important features of condensed matter, especially the dynamic magnetic properties, due to their intrinsic properties. In many substances, thermal neutrons have a much larger effective cross-section than faster neutrons, and can therefore be absorbed more easily by any atomic nuclei that they collide with. The neutron is a subatomic particle with a mass of $\sim 1.675 \times 10^{-27}kg$, zero net electric charge, a spin angular momentum of $\hbar/2$ and a magnetic moment of $1.913\mu_N$ ($\mu_N = \mu_B/1836$) [72].

First, as a result of their mass, thermal neutrons have a de Broglie wavelength of $\sim 1.81\mathring{A}$, which is of the order of magnitude of the interatomic spacing in most solids. Thus, the

interference effects between the incoming neutrons and the scattering sample will yield inner structural information of the scattering materials.

Second, the neutron has zero net electronic charge. This enables a neutron to penetrate deeply into the target sample with the penetrating depth $10\,cm$, which is much larger than those of low energy electrons ($\sim 10 - 1000\,\mathring{A}$), low energy photons ($\sim 1000 - 5000\,\mathring{A}$) and X-rays ($\sim 1 - 5\,mm$). The lack of charge also means that neutrons are unaffected by the Coulomb barrier, which allows neutrons to come close to the nuclei, and then be directly scattered off of the nuclei. This property makes the neutron a perfect probe of lattice structure and dynamic phonon or magnon excitations.

Third, the energy of the thermal neutron is of the same order as that of most magnetic excitations in condensed matter. When the neutron is inelastically scattered by the creation or annihilation of an excitation, the energy change of the neutron is a large portion of its initial energy. Thus, measurement of the change in the energy of the neutron provides an accurate method to obtain information about the energy excitation spectra. In a real scattering experiment, the key experimental variables are the change in the neutron energy and the associated change in the wave vector. We denote the transfer of energy and momentum to the target sample by $\hbar\omega$ and $\hbar\mathbf{q}$, respectively. In the rest of this document, we drop $\hbar$ in our discussions for simplicity, i.e., ω represents both the excitation frequency and the energy transfer; $\mathbf{q}$ represents both the wave vector and the momentum transfer.

Last, neutrons are fermions with spin $\frac{\hbar}{2}$. Their spin magnetic moments interact with those of the unpaired electrons in a material. It is through this interaction that the magnetic properties of a material can be probed. In the inelastic scattering experiment, not only the energies of magnetic excitations, but also the space- and time-dependent spin-spin correlations can be investigated.

2.5.2 Inelastic Neutron Scattering

In an inelastic neutron scattering experiment both the momentum and the energy transfer between the neutron and the sample are determined. What is actually measured in a magnetic scattering experiment is the *partial differential cross-section* [73] defined as

$$\left(\frac{d^2\sigma}{d\Omega dE}\right)_{\alpha\beta} = \frac{N_{\alpha\beta}(\theta,\phi)}{\Phi_\alpha d\Omega dE}, \tag{2.41}$$

where α and $\beta = x, y,$ or z. $N_{\alpha\beta}$ is the number of neutrons with incident polarization α and scattered polarization β scattered into differential solid angle $d\Omega$ in the direction of (θ,ϕ) with final energy between E and $E+dE$. Φ_α is the flux of incident neutrons with polarization α. By using *Fermi's Golden rule* and the first *Born approximation* [73, 74, 75], we obtain the cross-section as

$$\frac{d^2\sigma}{d\Omega dE} = \frac{k'}{k}\left(\frac{m_n}{2\pi\hbar^2}\right)^2 \sum_{\lambda\sigma} p_{\lambda\sigma} \sum_{\lambda'\sigma'} \left|\left\langle \mathbf{k}'\sigma'\lambda' \middle| \hat{V} \middle| \mathbf{k}\sigma\lambda \right\rangle\right|^2 \delta(E_{\lambda'} - E_\lambda - \hbar\omega)\delta(\mathbf{k}' - \mathbf{k} - \mathbf{q}), \tag{2.42}$$

where k is the magnitude of momentum, m_n is the mass of the neutron, $p_{\lambda\sigma}$ is the probability of the state $\lambda\sigma$ to be occupied, σ and λ are the spin indexes of the neutrons and the scattering system respectively, and $\hat{V}$ is the interaction potential term of the Hamiltonian of the scattering system. The non-primed quantities are for the initial states before scattering and primed quantities are for final states after scattering. When only magnetic interactions are considered the following relationship can be derived from Eqn.(2.42) [75] as

$$\left(\frac{d^2\sigma}{d\Omega dE}\right)_{\alpha\beta} = \frac{k'}{k}\left|f(\mathbf{q})\right|^2 \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} dt e^{-i\omega t} \sum_{\mathbf{r}} \left\langle S_0^\alpha(0) S_{\mathbf{r}}^\beta(t)\right\rangle e^{i\mathbf{q}\cdot\mathbf{r}}, \tag{2.43}$$

where $f(\mathbf{q})$ is the neutron scattering form factor and contains all of the microscopic physics relevant to the interacting neutrons and electrons. If the scattering is measured with the same incident and scattering polarization, i.e., $\alpha = \beta$, then from Eqns.(2.38) and (2.43) we get

$$\left(\frac{d^2\sigma}{d\Omega dE}\right)_{\alpha\alpha} = \frac{k'}{k}\left|f(\mathbf{q})\right|^2 S^\alpha(\mathbf{q},\omega). \tag{2.44}$$

The above Eqn.(2.44) gives an expression for the dynamic structure factor as a function of experimentally observable quantities measured by neutron scattering. On the other hand, the dynamic structure factor can also be calculated with spin dynamics simulation techniques.

Inelastic neutron scattering triple-axis (or three-axis) spectrometer (TAS) is the most versatile and useful instrument for the use in inelastic scattering because it allows one to probe nearly any coordinate in energy and momentum space in a precisely controlled manner. The concept of the triple-axis spectrometry method was first developed by Brockhouse [76] in 1961. The first results from the prototype triple-axis spectrometer were published in 1955 and the first true triple-axis spectrometer was built in 1956. The three axes correspond to the axes of rotation of the monochromator, the sample, and the analyzer. The orientation of the three axes can be varied independently. The monochromator defines the direction and magnitude of the momentum of the incident neutron beam and the analyzer performs a similar function for the scattered or final beam, which is Bragg scattered by the sample. Figure 2.3 shows a schematic diagram of a TAS at Oak Ridge National Laboratory [77].

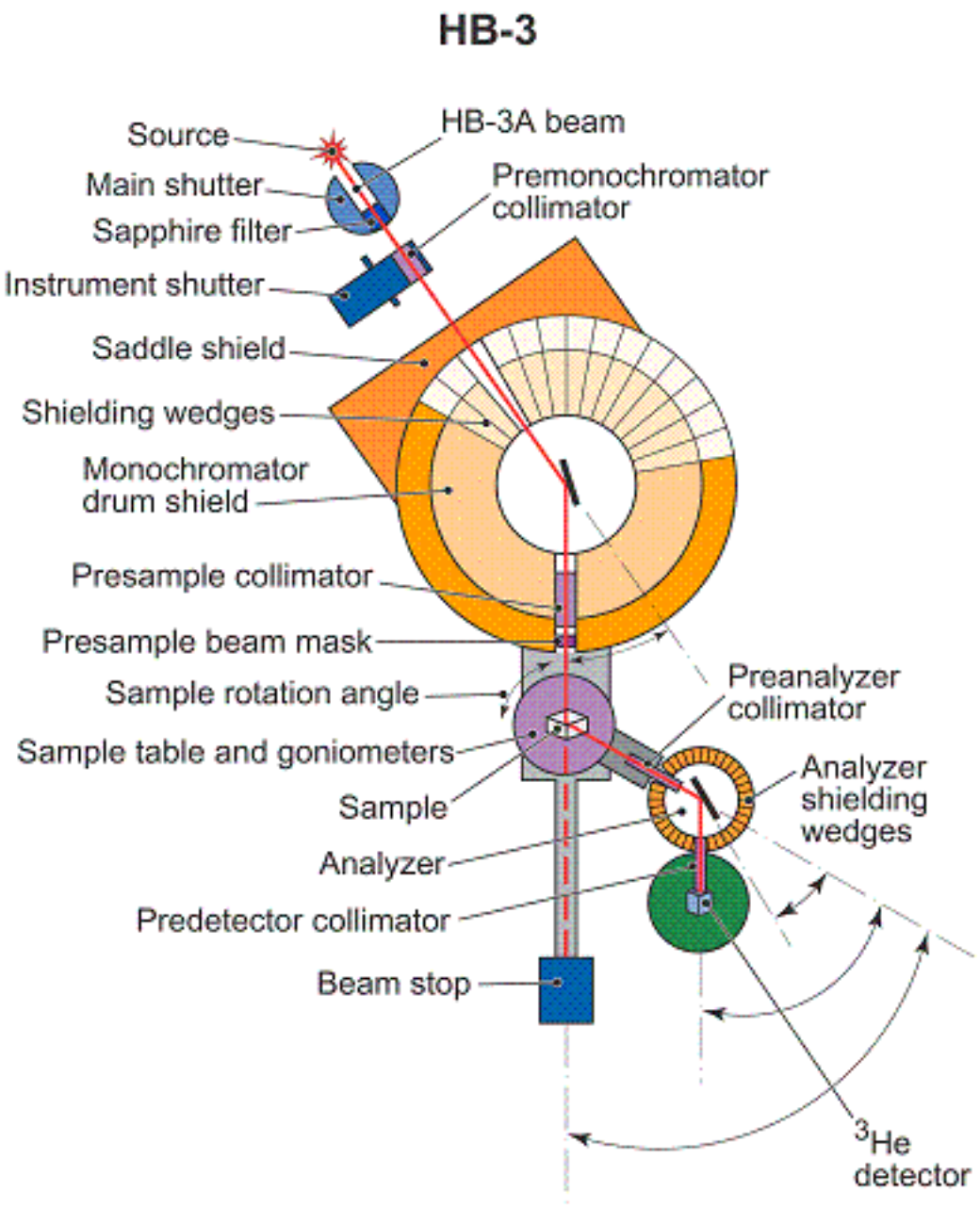

Figure 2.3: A schematic diagram of an inelastic neutron scattering triple-axis spectrometer at Oak Ridge National Laboratory (after Ref.[77]).

Chapter 3

Simulation Methods

3.1 Overview of Spin Dynamics

As stated in Section 2.3.2, the Heisenberg model has true dynamics with the real time evolution of spins governed by the equations of motion. The general recipe of spin dynamics is to generate N equilibrium spin configurations drawn from a canonical ensemble at a specific temperature T using a hybrid Monte Carlo (MC) method, and to use these N equilibrium spin configurations as starting states for the integration of the coupled equations of motion using spin dynamics (SD) method, with the real SD time evolving from $t = 0$ to $t = ndt$, where n is the total number of SD time steps and dt is the SD time step ($dt = 0.2/|J|$). From those data the local space- and time-displaced spin-spin correlation function $C(\mathbf{r}_0, \mathbf{r}, t)$ is calculated. We take a set of N initial conditions of a finite lattice size and average their results for the local space-time correlation function. If this set of N configurations is an equilibrium distribution at the temperature T, then the average over all the $C(\mathbf{r}_0, \mathbf{r}, t)$ will be a result for the local space-time correlation function at T for a finite lattice size. The hybrid MC method is an efficient simulation method by which we use to generate such an equilibrium distribution of states at a given temperature for a given lattice size and a given model.

The local space-time spin-spin correlation function is Fourier transformed in the post calculation, from which we get the local dynamic structure factor $S(\mathbf{r}_0, \mathbf{q}, \omega)$. Detailed information on the quantities $C(\mathbf{r}_0, \mathbf{r}, t)$ and $S(\mathbf{r}_0, \mathbf{q}, \omega)$ can be found in Section 3.4.

To illustrate the whole process of SD simulation, Figure 3.1 gives an overview of a spin dynamics simulation decomposed into three procedural sub-processes, i.e., the static MC, the dynamic SD, and the post calculation.

In this chapter, we will first discuss the hybrid MC method we used to obtain the equilibrium states including the three MC methods which constitute the algorithm, i.e., the Metropolis algorithm [78], the Wolff algorithm [80], and the Overrelaxation algorithm [81]. Then, the numerical integration method we used for SD, i.e., the fourth-order Suzuki-Trotter decomposition algorithm will be discussed. Next, a description of the computational methods for calculating the local space-time correlation function and the local dynamic structure factor will be given.

3.2 The Monte Carlo Method

For an isothermal system $\mathcal{G}$ governed by the Hamiltonian $\mathcal{H}(\mathbf{x})$, where $\mathbf{x}$ is the phase vector in phase space, the thermal average of any observable $A(\mathbf{x})$ is defined in the canonical ensemble as

$$\begin{aligned} \langle A(\mathbf{x}) \rangle_\beta &= \frac{1}{Z} \int d\mathbf{x} e^{-\beta \mathcal{H}(\mathbf{x})} A(\mathbf{x}), \\ Z &= \int d\mathbf{x} e^{-\beta \mathcal{H}(\mathbf{x})}, \end{aligned} \tag{3.1}$$

where $\beta = 1/k_B T$. The normalized Boltzmann probability density describing the statistical weight with which the configuration $\mathbf{x}$ occurs in thermal equilibrium is defined as

$$p(\mathbf{x}) = \frac{1}{Z} e^{-\beta \mathcal{H}(\mathbf{x})}. \tag{3.2}$$

Although Eqn.(3.2) gives a formally exact description of the probability distribution $p(\mathbf{x})$, it is impossible to carry out the integration in the general case with a high-dimensional phase space.

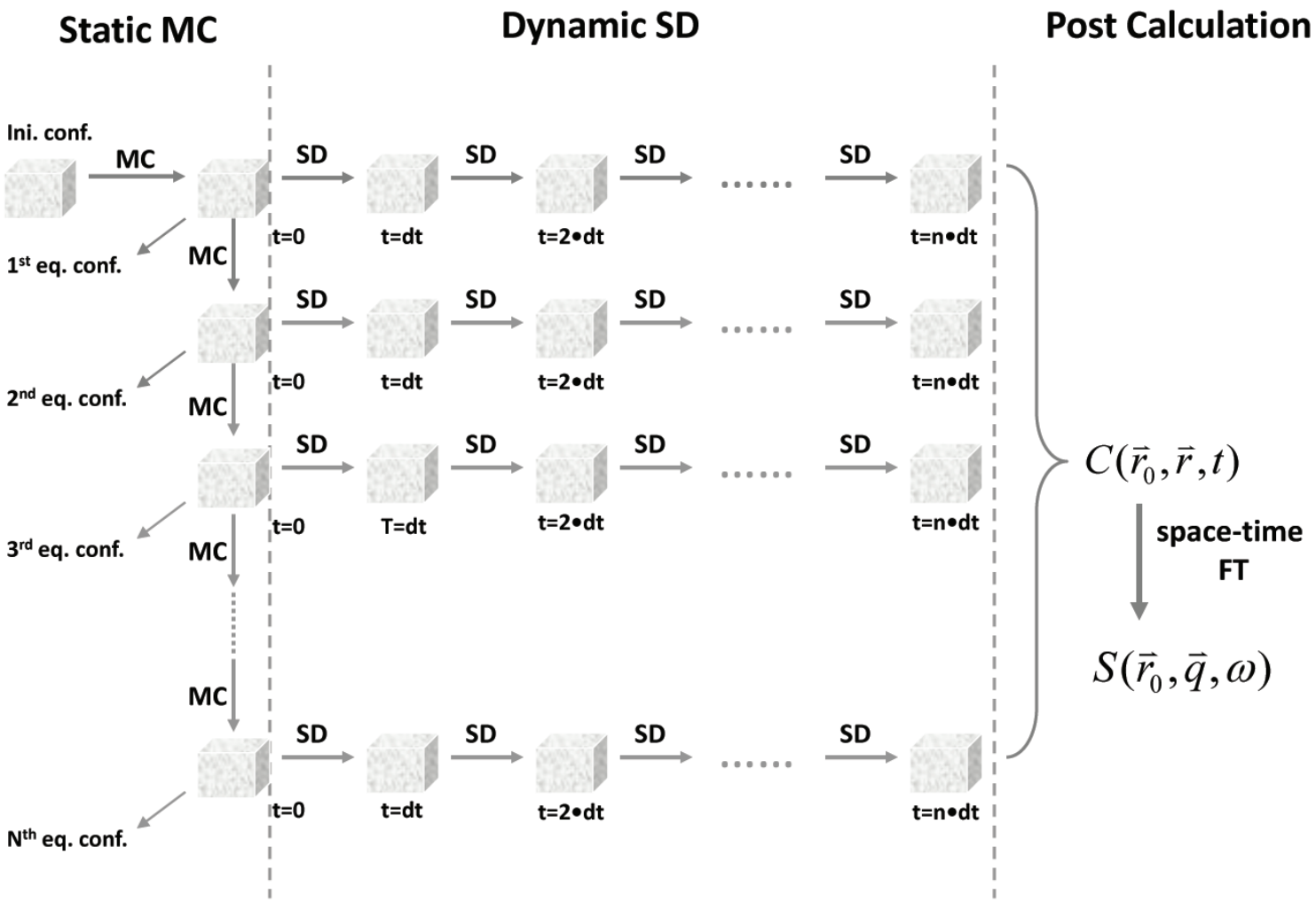

Figure 3.1: An overview of spin dynamics simulation: the static MC, the dynamic SD, and the post calculation phases.

3.2.1 The Simple sampling Monte Carlo Method

The Monte Carlo method in equilibrium statistical mechanics starts from an approximation to the exact equation Eqn.(3.1), where one integrates overall states $\{\mathbf{x}\}$ with their $p(\mathbf{x})$, by a summation using only a characteristic subset of phase space points $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_M\}$, which forms a statistical sample. As $M \to \infty$, we obtain an approximation to Eqn.(3.1) as [11]

$$\left\langle A(\mathbf{x})\right\rangle_\beta \approx \overline{A(\mathbf{x})_\beta} = \frac{\sum_{i=1}^{M} A(\mathbf{x}_i) e^{-\beta\mathcal{H}(\mathbf{x}_i)}}{\sum_{i=1}^{M} e^{-\beta\mathcal{H}(\mathbf{x}_i)}}. \tag{3.3}$$

where M is the number of states in the subset. The problem of selecting a representative distribution of states in a high-dimensional phase space can be solved by using a random number generator to select the M sampled states as random points in phase space. This method is known as *simple sampling* [11] Monte Carlo, which is the simplest of the Monte Carlo sampling techniques and is usually inefficient and even unreliable for a thermodynamic quantity as a function with any sharp peak. This is because many of chosen states in $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_M\}$ are physically improbable and contribute little to the average. It would be more statistically sensible to sample preferentially around physically favorable states. To realize this purpose, *Metropolis importance sampling* Monte Carlo [78] has been developed.

3.2.2 The Metropolis importance sampling Monte Carlo Method

Metropolis *et al.* realized the importance sampling Monte Carlo method by constructing a Markov process where each state $\mathbf{x}_l$ is constructed from a previous state $\mathbf{x}_k$ via a transition probability $W(\mathbf{x}_k \to \mathbf{x}_l)$ based on the energy difference between the initial and final states.

The equilibrium probability distribution of a classical system being at the phase point $\mathbf{x}_l$ at the Monte Carlo time t is

$$P_{\mathbf{x}_l}(t) = \frac{1}{Z} e^{-\beta\mathcal{H}(\mathbf{x}_l)}. \tag{3.4}$$

And the equilibrium probability distribution obeys a master equation [54]

$$\frac{\partial P_{\mathbf{x}_l}(t)}{\partial t} = -\sum_{l\neq k} \left[P_{\mathbf{x}_l}(t) W(\mathbf{x}_l \to \mathbf{x}_k) - P_{\mathbf{x}_k}(t) W(\mathbf{x}_k \to \mathbf{x}_l)\right], \tag{3.5}$$

where $W(\mathbf{x}_k \rightarrow \mathbf{x}_l)$ is the transition rate from point $\mathbf{x}_l$ to point $\mathbf{x}_k$ in phase space. For a system in equilibrium, we have $\partial P_{\mathbf{x}_l}(t)/\partial t = 0$. This condition imposes the principle of detailed balance

$$P_{\mathbf{x}_l}(t)W(\mathbf{x}_l \rightarrow \mathbf{x}_k) = P_{\mathbf{x}_k}(t)W(\mathbf{x}_k \rightarrow \mathbf{x}_l). \tag{3.6}$$

Then from Eqn.(3.4) and Eqn.(3.6)

$$\frac{P_{\mathbf{x}_k}(t)}{P_{\mathbf{x}_l}(t)} = \frac{W(\mathbf{x}_l \rightarrow \mathbf{x}_k)}{W(\mathbf{x}_k \rightarrow \mathbf{x}_l)} = e^{-\delta\mathcal{H}\beta}, \tag{3.7}$$

where $\delta\mathcal{H}$ is the energy change between the initial and final states, i.e., $(\mathcal{H}(\mathbf{x}_k) - \mathcal{H}(\mathbf{x}_l))$. This relation does not uniquely specify $W(\mathbf{x}_l \rightarrow \mathbf{x}_k)$. A convenient choice [79] that fits Eqn.(3.7) is

$$W(\mathbf{x}_l \rightarrow \mathbf{x}_k) = \begin{cases} e^{-\delta\mathcal{H}\beta}, & if\ \delta\mathcal{H} > 0 \\ 1, & otherwise \end{cases} \tag{3.8}$$

The recipe of the Metropolis importance sampling is given as follows

(1) Choose an initial configuration

(2) Randomly pick a site i

(3) Propose a change of the spin at site i, and calculate the energy change ΔE

(4) Generate a random number r such that $0 < r < 1$(uniformly distributed on the interval $[0, 1]$)

(5) Make the proposed change of the spin if $r < \exp(-\beta\Delta E)$
$\Rightarrow$ Complete a single Monte Carlo step

(6) Pick a new site and go to step **(3)**

This process can be repeated as many times as needed to realize a random walk across phase space that will trace out a set of M states which can be used in the sampling. In

practice, to improve the accuracy of the method, one needs to discard the first certain number of steps to approach the equilibrium and then use the subsequent steps to collect data and determine the properties of a system.

One limitation of the Monte Carlo method is the correlation, which diverges as $T \to T_c$, between successive generated states. There are two kinds of relaxation time [11, 54, 79]. The first one is called the *non-linear relaxation time* τ_{nl} which can be used to examine the approach to equilibrium. τ_{nl} can be derived by defining a non-linear relaxation function

$$\phi_M(t) = \frac{\langle M(t) - M(\infty) \rangle}{\langle M(0) - M(\infty) \rangle}, \tag{3.9}$$

where t is the Monte Carlo time in terms of MCS. $\phi_M(t)$ describes relaxation of magnetization M approaching its equilibrium value and has an exponential decay at long time. The non-linear relaxation time τ_{nl} is defined as [54]

$$\tau_{nl} = \int_0^\infty \phi_M(t) dt. \tag{3.10}$$

τ_{nl} diverges as

$$\tau_{nl} \sim \xi^{z_{nl}^M}, \tag{3.11}$$

where ξ is the correlation length and z_{nl}^M has a relationship with static critical exponents β and ν and the dynamic critical exponent z as [54]

$$z = z_{nl}^M + \beta/\nu. \tag{3.12}$$

Similarly, if the non-linear relaxation is calculated for the internal energy then

$$z = z_{nl}^M + (1 - \alpha)/\nu. \tag{3.13}$$

The other kind of relaxation time is called *linear relaxation time* τ_l. τ_l is associated with the normalized linear relaxation function which describes time correlations *within* equilibrium and is defined as

$$\phi_A(t) = \frac{\langle A(0)A(t) \rangle - \langle A \rangle^2}{\langle A^2 \rangle - \langle A \rangle^2}, \tag{3.14}$$

where t is the Monte Carlo time in terms of Monte Carlo steps (MCS) and A is a thermodynamic quantity, e.g., M or E. $\phi_A(0) = 1$ and $\phi_A(t)$ decays asymptotically with t as [54]

$$\phi_A(t) \to e^{-t/\tau_l}, \qquad t \to \infty. \tag{3.15}$$

As $T \to T_c$, the equilibrium linear relaxation time τ_l diverges as

$$\tau_l \sim \xi^z, \tag{3.16}$$

And according to Eqn.(2.19), $\xi \sim |T - T_c|^{-\nu}$, then

$$\tau_l \sim |T - T_c|^{-z\nu}. \tag{3.17}$$

As $T \to T_c$, τ_l diverges dramatically and thus causes long temporal correlation between successive generated states, which is known as *critical slowing down*. To lower this correlation and improve the efficiency of the Monte Carlo simulation, multiple simulation techniques have been developed such as the Wolff cluster flipping and overrelaxation algorithms. Both algorithms are collective updating algorithms, in contrast to the local updating scheme of Metropolis.

3.2.3 The Wolff Algorithm

The Wolff cluster flipping algorithm is appropriate for spins with continuous degrees of freedom.It updates clusters of spins simultaneously to reduce the critical slowing down at the critical region. This algorithm turns the original classical Heisenberg spin model with uniform interaction into an Ising model with inhomogeneous couplings. An elementary cluster update step, which is adapted for interactions beyond nearest neighbors, consists of the following operations [80]

(1) Randomly choose a unit direction vector $\hat{r}$ and a spin $\mathbf{S}_i$ at site i as the first spin of a cluster to be constructed

(2) Visit all neighboring spins $\mathbf{S}_j$ that have exchange interaction with $\mathbf{S}_i$ and assign bond between $\mathbf{S}_i$ and $\mathbf{S}_j$ according to the probability

$$p = 1 - e^{min\left[0,\, 2\beta J_{ij}(\hat{r}\cdot\mathbf{S}_i)(\hat{r}\cdot\mathbf{S}_j)\right]}$$

(3) Visit all neighboring spins $\mathbf{S}_j$ and repeat step **(2)** iteratively until the process stops

(4) Flip the cluster of spins connected with bonds with respect to the hyperplane orthogonal to $\hat{r}$

The Wolff cluster algorithm is ergodic and detailed balance is fulfilled. This algorithm is efficient for Heisenberg exchange interaction without exchange anisotropy. Single site anisotropy can be taken into account by creating the cluster using only the exchange interaction term of the Hamiltonian then flipping the cluster according to a probability determined by the single site anisotropy term of the Hamiltonian.

3.2.4 The Overrelaxation Algorithm

The overrelaxation algorithm, which was borrowed from the concept that used to speed up convergence of matrix inversion [81, 82], is not a stand-alone Monte Carlo method because it does not guarantee ergodicity, and can be added into hybrid Monte Carlo to reduce the correlation between successive states and thus increase the efficiency of the simulation. This algorithm is only applicable to the Heisenberg model without single site anisotropy. Since no random number creation is required in an overrelaxation step, it is deterministic and much more efficient than either a Metropolis or Wolff cluster step.

For an isotropic Heisenberg model, each spin is in an effective field given by Eqn.(2.25), and the energy of the spin is determined by

$$E_i = \mathbf{H}_i^{eff} \cdot \mathbf{S}_i. \tag{3.18}$$

Any precession of the spin around the effective field $\mathbf{H}_i^{eff}$ will not change the total energy of the system.

In practice, a lattice is decomposed into two sublattices so that there is exchange interaction between spins belonging to different sublattices. So all spins in one sublattice can be precessed simultaneously around their individual effective fields with energy conserved. An effective way is to precess each spin 180° around its effective field. If this updating is performed on each sublattice, one sublattice at a time, then all spins will be reoriented at constant energy. The spin updating via overrelaxation is essentially microcanonical. To obtain canonical ensemble average of thermodynamic quantities, a sensible combination of overrelaxation with Metropolis Monte Carlo or with Metropolis-Wolff hybrid Monte Carlo is imperative. Since we are not performing any high precision Monte Carlo simulations, the optimized hybrid Monte Carlo, i.e., two Metropolis with eight overrelaxation steps for an isotropic system and two Metropolis with eight Wolff steps for an anisotropic system, which has been determined to be optimal in a previous study [37], is applied in this work.

3.3 The Spin Dynamics Method

3.3.1 Methods for Integration of the Equation of Motion

Given an equilibrium distribution of starting states at a temperature T, we need a numerical integration method to solve the coupled differential equations of motion. Traditional integration methods include Runge-Kutta and predictor-corrector methods [83]. Runge-Kutta methods are an important family of iterative methods for the approximation of solutions of differential equations. Runge-Kutta methods propagate a solution over an interval by combining the information from several Euler-style steps, and then using the information

obtained to match a Taylor series expansion up to some higher order. The fourth-order Runge-Kutta method has been so commonly used that it is often referred to as "RK4", "classical Runge-Kutta method"or simply as the "RungeKutta method". Predictor-corrector methods are numerical methods that proceed in two steps. First, the prediction step extrapolates a rough approximation of the desired quantity one step advance. Second, the corrector step refines the extrapolation using derivative information at the new point. The predictor-corrector methods have been used in some early spin dynamics simulations [37, 84].

A severe restriction on the size of the SD time step is applied to any kind of numerical integration methods. This restriction is imposed by the accuracy within which the numerical method observes the conservation law of the dynamics. It is evident from Eqn.(2.27) that the length of spin $\mathbf{S}_i$ at lattice site i and the total energy of a system are conserved. Additionally, for a isotropic Heisenberg model, the uniform magnetization $M = \Sigma_i \mathbf{S}_i$ should also be a conserved quantity. The predictor-corrector methods observe the conservation laws for the uniform magnetization to within the machine accuracy and other conservation laws within the accuracy set by the truncation error [86]. Typically, it limits the SD time step to about $\delta t = 0.01/|J|$ and the total integration time about $600/|J|$ [87].

As mentioned in Section 3.2.4 on the overrelaxation method, for the Heisenberg Hamiltonian without single site anisotropy, each spin subjected to an effective field undergoes Larmor precession around this field, which is itself changing in time. The lattice can be decomposed into two sublattices so that a spin on one sublattice performs a Larmor precession in a local field $\mathbf{\Omega}$ of neighboring spins which are all located on the other sublattice. For the Heisenberg Hamiltonian with only nearest-neighbor interactions there are only two sublattices if the lattice is simple cubic or body-center cubic. This procedure has been implemented as the Suzuki-Trotter decomposition [88, 89] algorithm. A similar method was developed independently by Frank, Huang and Leimkuhler [90] and named as the *Staggered Red-Black Method*.

3.3.2 Suzuki-Trotter Decomposition Algorithm

The Suzuki-Trotter decomposition treats the dynamics directly through the rotation of a spin about its local field $\mathbf{\Omega}$ by an angle $\alpha = |\mathbf{\Omega}|\Delta t$. This procedure guarantees the conservation of the spin length $\mathbf{S}$ and energy to within machine accuracy. Unlike the predictor-corrector algorithm, conservation of magnetization is only observed to within truncation error. Denoting the two sublattices by $\mathcal{A}$ and $\mathcal{B}$, respectively. We can write Eqn.(2.27) into

$$\begin{aligned} \frac{d}{dt}\mathbf{S}_{k\in\mathcal{A}} &= \mathbf{\Omega}_{\mathcal{B}}[\{\mathbf{S}\}] \times \mathbf{S}_{k\in\mathcal{A}}, \\ \frac{d}{dt}\mathbf{S}_{k\in\mathcal{B}} &= \mathbf{\Omega}_{\mathcal{A}}[\{\mathbf{S}\}] \times \mathbf{S}_{k\in\mathcal{B}}, \end{aligned} \tag{3.19}$$

where $\mathbf{\Omega}_{\mathcal{A}}[\{\mathbf{S}\}]$ and $\mathbf{\Omega}_{\mathcal{B}}[\{\mathbf{S}\}]$ denote the local field produced by the spins on sublattice $\mathcal{A}$ and $\mathcal{B}$, respectively. The equations in Eqn.(3.19) can be reduced into a linear system of differential equations if the spins on the other sublattice are kept fixed. This suggests an alternating update scheme with the spins $\mathbf{S}_{k\in\mathcal{A}}$ rotated for the fixed $\mathbf{S}_{k\in\mathcal{B}}$ and vice versa. In this alternating update scheme, the scalar products $\mathbf{\Omega}_{\mathcal{B}}[\{\mathbf{S}\}] \cdot \mathbf{S}_{k\in\mathcal{A}}$ remains constant during the update of $\mathbf{S}_{k\in\mathcal{A}}$ and the scalar products $\mathbf{\Omega}_{\mathcal{A}}[\{\mathbf{S}\}] \cdot \mathbf{S}_{k\in\mathcal{B}}$ remains constant during the update of $\mathbf{S}_{k\in\mathcal{B}}$. Thus, the energy is exactly conserved during this alternating update scheme. The conservation of magnetization according to the symmetry of the Hamiltonian, however, will not be observed during the above updating scheme.

To describe the updating operations on the spin configuration during the integration of the equation of motion, we use a vector y to represent a full spin configuration, and decompose it into two sublattice vector components $y_{\mathcal{A}}$ and $y_{\mathcal{B}}$ according to $y = (y_{\mathcal{A}}, y_{\mathcal{B}})$. The cross products in Eqn.(3.19) can be expressed by matrices A and B which are the infinitesimal generators of rotation of the spin configuration $y_{\mathcal{A}}$ on sublattice $\mathcal{A}$ at fixed $y_{\mathcal{B}}$ and of the spin configuration $y_{\mathcal{B}}$ on sublattice $\mathcal{B}$ at fixed $y_{\mathcal{A}}$, respectively. The update of the configuration y from time t to $t + \Delta t$ can then be expressed by an exponential matrix operator as

$$y(t + \Delta t) = e^{(A+B)\Delta t} y(t). \tag{3.20}$$

The exponential operators $e^{(A+B)}\Delta t$ in Eqn.(3.20), which rotates each spin of the configuration, has no simple explicit form, because the rotation axis for each spin depends on the configuration itself. However, the operators $e^{A\Delta t}$ and $e^{B\Delta t}$, which rotates $y_{\mathcal{A}}$ at fixed $y_{\mathcal{B}}$ and rotates $y_{\mathcal{B}}$ at fixed $y_{\mathcal{A}}$, respectively, have a simple explicit form.

For an isotropic Heisenberg model with $D = 0$ in Eqn.(2.22), for each $k \in \mathcal{A}$, we have

$$\mathbf{\Omega}_{\mathcal{A}}[\{\mathbf{S}\}] = -J \sum_{l=NN(k)} \mathbf{S}_l \equiv \mathbf{\Omega}_k, \tag{3.21}$$

where $NN(k)$ denotes the nearest neighbor on $y_{\mathcal{B}}$ of spin $\mathbf{S}_k$. From Eqn.(3.19) and Eqn.(3.21) we obtain [85, 86]

$$\mathbf{S}_k(t+\Delta t) = \frac{\mathbf{\Omega}_k(\mathbf{\Omega}_k \cdot \mathbf{S}_k(t))}{\mathbf{\Omega}_k^2} + \left[\mathbf{S}_k(t) - \frac{\mathbf{\Omega}_k(\mathbf{\Omega}_k \cdot \mathbf{S}_k(t))}{\mathbf{\Omega}_k^2}\right] \cos(|\mathbf{\Omega}_k|\Delta t) + \frac{\mathbf{\Omega}_k \times \mathbf{S}_k(t)}{|\mathbf{\Omega}_k|} \sin(|\mathbf{\Omega}_k|\Delta t). \tag{3.22}$$

According to Eqn.(3.22), we have

$$\mathbf{\Omega}_k \cdot \mathbf{S}_k(t + \Delta t) = \mathbf{\Omega}_k \cdot \mathbf{S}_k(t). \tag{3.23}$$

The above equation explicitly confirms energy conservation. For $k \in \mathcal{B}$, Eqn(3.22) also holds in the same form.

The lowest-order Suzuki-Trotter decomposition of Eqn.(3.20) is

$$e^{(A+B)\Delta t} = e^{A\Delta t} e^{B\Delta t} + \mathcal{O}(\Delta t^2), \tag{3.24}$$

which is only correct up to terms of the order Δt^2. The magnetization will only be conserved up to terms of the order Δt, which is insufficient for practical purpose. To increase the order of the truncation error of the algorithm and thus improve the conservation of magnetization, the second-order Suzuki-Trotter decomposition of the exponential operator in Eqn.(3.20) is used and given by [89]

$$e^{(A+B)\Delta t} = e^{A\frac{\Delta t}{2}} e^{B\Delta t} e^{A\frac{\Delta t}{2}} + \mathcal{O}(\Delta t^3), \tag{3.25}$$

and the fourth-order Suzuki-Trotter decomposition is

$$e^{(A+B)\Delta t} = \prod_{i=1}^{5} e^{p_i A\frac{\Delta t}{2}} e^{p_i B\Delta t} e^{p_i A\frac{\Delta t}{2}} + \mathcal{O}(\Delta t^5), \tag{3.26}$$

where $p_1 = p_2 = p_4 = p_5 \equiv p = 1/(4 - 4^{1/3}) \simeq 0.41449$ and $p_3 = 1 - 4p \simeq -0.65780$, which are determined by a recursive scheme developed by Suzuki *et al.* [89].

The higher the order of the algorithm the more computer time each SD time step will take. All Suzuki-Trotter algorithms use more computer time per SD time step than the predictor-corrector method. This time can be compensated for by using larger SD time steps since the higher order algorithm is significantly more accurate. In this work, we applied the fourth-order Suzuki-Trotter decomposition of the exponential operator with an SD time step of $\Delta t = 0.2/|J|$. A comparison on the performance of Suzuki-Trotter decomposition algorithms with the second and fourth order with different Δt is detailed in Section 4.2.1.

The Suzuki-Trotter decomposition algorithm discussed above can be generalized to the case of anisotropic ferromagnetic and antiferromagnetic systems where the anisotropy is single site in nature with $D \neq 0$ [86]. For a spin on sublattice $\mathcal{A}$, the equation of motion is

$$\frac{d}{dt}\mathbf{S}_{k\in\mathcal{A}} = \mathbf{\Omega}_{\mathcal{B}}[\{\mathbf{S}\}] \times \mathbf{S}_{k\in\mathcal{A}} - 2DS^z_{k\in\mathcal{A}}\mathbf{e}_z \times \mathbf{S}_{k\in\mathcal{A}}, \tag{3.27}$$

where $\mathbf{e}_z$ is the unit vector pointing along the z-axis; the equation of motion for a spin on sublattice $\mathcal{B}$ obeys the same form. In contrast to the isotropic case with Eqn.(3.19), the equation of motion for each individual spin on each sublattice is nonlinear. In practice, to include the effects of the nonlinearity in Eqn.(3.27), we add a single site anisotropy into the condition for energy conservation [Eqn.(3.23)], and obtain

$$\mathbf{\Omega}_k \cdot \mathbf{S}_k(t + \Delta t) - D[S^z_k(t + \Delta t)]^2 = \mathbf{\Omega}_k \cdot \mathbf{S}_k(t) - D[S^z_k(t)]^2 \tag{3.28}$$

for $k \in \mathcal{A}$ and $k \in \mathcal{B}$, where $\mathbf{\Omega}_k$ is given by Eqn.(3.21). If we make the transformation $\mathbf{\Omega} \to \tilde{\mathbf{\Omega}}$ such that

$$\tilde{\mathbf{\Omega}} \cdot \mathbf{S}_k(t + \Delta t) = \tilde{\mathbf{\Omega}} \cdot \mathbf{S}_k(t), \tag{3.29}$$

then we will be able to perform the dynamics using Eqn.(3.22) with $\mathbf{\Omega}$ replaced by $\tilde{\mathbf{\Omega}}$. From Eqn.(3.28) and Eqn.(3.29), we obtain $\tilde{\mathbf{\Omega}}$ as

$$\tilde{\mathbf{\Omega}} = \mathbf{\Omega} - D(0, 0, S^z_k(t) + S^z_k(t + \Delta t)). \tag{3.30}$$

The solution of Eqn.(3.30) requires that $S_k^z(t+\Delta t)$ must be known in advance. This can be solved through an iterative scheme. In this work, we started from the initial value of $S_k^z(t+\Delta t)$ with the second order *Taylor* expansion as

$$\begin{aligned} S_k^z(t+\Delta t) &= S_k^z(t) + \frac{dS_k^z(t)}{dt}dt \\ &= S_k^z(t) + \left\{\Omega_k^x(t)S_k^y(t) - \Omega_k^y(t)S_k^x(t)\right\}dt, \end{aligned} \tag{3.31}$$

where $\Omega_k^x(t)$ and $\Omega_k^y(t)$ are the x and y components of the effective field [Eqn.(3.21)] on site k at spin dynamics time t. We perform an update according to the decomposition given by Eqn.(3.26). We then perform a second update using results for $S_k^z(t+\Delta t)$ from the previous iteration as the input. To guarantee a good accuracy, we continue this process repeatedly. As an iterative procedure, it leads to s substantial slowdown of the integration algorithm, where the energy is no longer exactly conserved. In this work, to balance the trade-off between accuracy and efficiency, we used three iterations at each spin dynamics time.

3.4 The Post Calculation on Local Correlation Functions

3.4.1 Model Nanostructures and Boundary Conditions

In this work, we modeled nanofilms and nanoparticles on a simple cubic lattice of different linear dimensions with two kinds of boundary conditions which are required both geometrically and physically by the modeling, i.e., *partially* free boundary conditions and *completely* free boundary conditions, respectively.

Model nanofilms consist of $L_{xy}^2 L_z$ spins on a $L_{xy} \times L_{xy} \times L_z$ simple cubic lattice with partially free boundary conditions with two free-surfaces in the spatial z-direction, which is labeled as the *"FBCZ"* direction, and periodic boundaries parallel to the surfaces in the x- and y-direction, which is labeled as the *"PBCXY"* direction in this study. L_{xy} denotes linear dimensions in the x-, y-directions and L_z denotes linear dimension in the z-direction, respectively.

Model nanoparticles consist of L^3 spins on a $L \times L \times L$ simple cubic lattice with completely free boundary conditions with six free surfaces. The model nanofilm and model nanoparticle are illustrated graphically in Figure 3.2 and Figure 3.3, respectively.

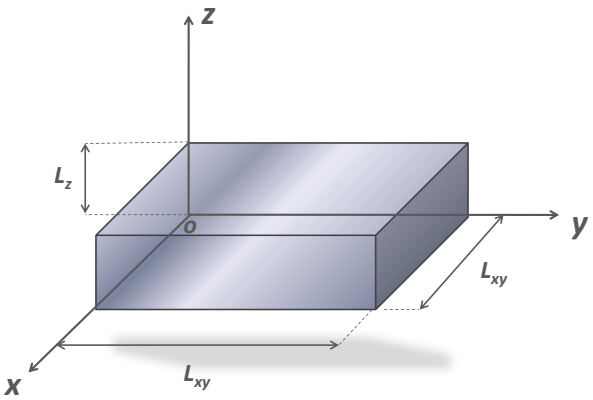

Figure 3.2: Model nanofilm with two free-surfaces in the spatial z-direction and periodic boundaries parallel to the surfaces in the x-, y-directions on $L_{xy} \times L_{xy} \times L_z$ simple cubic lattices. L_{xy} denotes linear dimensions in the x-, y-directions and L_z denotes linear dimension in the z-direction, respectively.

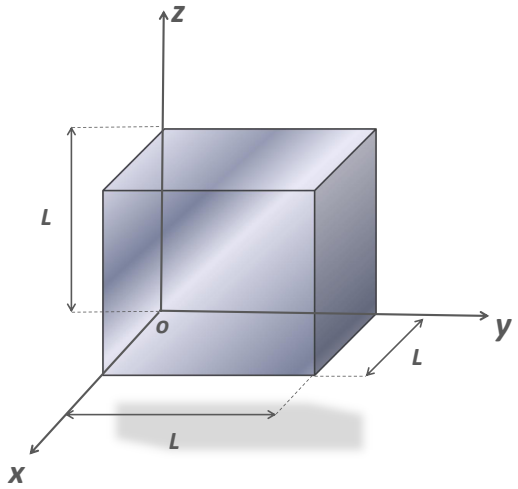

Figure 3.3: Model nanoparticle with completely free boundary conditions with six free surfaces on $L \times L \times L$ simple cubic lattices.

3.4.2 Local Correlation Function

As is known, fully periodic boundary conditions have been implemented to preserve the translational invariance and emulate "infinite"systems. In the modeling of nanofilms and nanoparticles in this work, we introduced free boundary conditions either partially in one spatial direction or completely in all spatial directions. As one of consequences of introducing free boundary conditions, the translational invariance of system is broken in the directions we introduced free boundary conditions. Accordingly, to express the broken translational symmetry, the formalism of the space- and time-displaced spin-spin correlation function has been modified from a translational invariant one with a form of $C(\mathbf{r}, t)$ to a localized one with a form of $C(\mathbf{r}_0, \mathbf{r}, t)$, where the parameter $\mathbf{r}_0$ denotes a fixed lattice site as the starting point for the calculation of the local correlation. According to the specific localization performed, $\mathbf{r}_0$ can be chosen to be fixed at the bulk center, the surface center, or even the lattice corner, i.e., $\mathbf{r}_0 \Rightarrow Bulk\ Center$, $Surface\ Center$, or $Lattice\ Corner$.

The definition of the local space- and time-displaced spin-spin correlation function is defined as

$$C^k(\mathbf{r}_0, \mathbf{r}, t) = \langle S^k(\mathbf{r}_0, t_0) S^k(\mathbf{r}_0 + \mathbf{r}, t_0 + t) \rangle - \langle S^k(\mathbf{r}_0, t_0) \rangle \langle S^k(\mathbf{r}_0 + \mathbf{r}, t_0 + t) \rangle, \tag{3.32}$$

where $\mathbf{r}_0$ and t_0 denote the spatial and temporal starting points for the local correlation function, respectively; $\mathbf{r}$ and t denote the spatial and time intervals, respectively; $\langle \ldots \rangle$ gives the ensemble average; $k = x$, y or z; $S^k(\mathbf{r}_0 + \mathbf{r}, t_0 + t)$ stands for the k component of a spin at the lattice site $\mathbf{r} + \mathbf{r}_0$ and the time $t_0 + t$. To give the details, the formalisms of the local correlation function $C(\mathbf{r}_0, \mathbf{r}, t)$ defined for model nanoparticles and model nanofilms are discussed separately in Section 3.4.3 and 3.4.4.

3.4.3 Local Correlation Function for Model Nanoparticles

For model nanoparticles with completely free boundary conditions on simple cubic lattices, we define the first and second terms in Eqn.(3.32) as

$$\begin{aligned}
\langle S^k(\mathbf{r}_0,t_0)S^k(\mathbf{r}_0+\mathbf{r},t_0+t)\rangle &= \frac{1}{S(n_t-t+1)}\sum_{s=1}^{S}\sum_{t_0=0}^{n_t-t} S^k(\mathbf{r}_0,t_0)S^k(\mathbf{r}_0+\mathbf{r},t_0+t)\big|_s, \\
\langle S^k(\mathbf{r}_0,t_0)\rangle &= \frac{1}{S(n_t+1)}\sum_{s=1}^{S}\sum_{t_0=0}^{n_t} S^k(\mathbf{r}_0,t_0), \\
\langle S^k(\mathbf{r}_0+\mathbf{r},t_0+t)\rangle &= \frac{1}{S(n_t-t+1)}\sum_{s=1}^{S}\sum_{t_0=0}^{n_t-t} S^k(\mathbf{r}_0+\mathbf{r},t_0+t)\big|_s, \qquad (3.33)
\end{aligned}$$

where S is the parameter giving the total number of geometrically symmetric [100] directions, along which we perform the correlation calculations; $s \in [1,\, S]$ is an integer index on those S directions; n_t is the total number of SD time steps.

With an odd lattice size L, where $L = 2i+1,\ i = 1, 2, 3, \ldots$, the bulk center and six free surface centers can be determined uniquely. As $\mathbf{r}_0 \Rightarrow Bulk\ Center$, we have six symmetric [100] directions off from the bulk center to six surface centers, as shown in Figure 3.4, i.e., $S_{\mathbf{r}_0 \Rightarrow Bulk\ Center} = 6$; As $\mathbf{r}_0 \Rightarrow Surface\ Center$, we define two sets of symmetric [100] directions, which are called *within-surface* directions, along which we calculate the spin-spin correlation off from each surface center within the free surface, and *off-surface* directions along which we calculate the correlation off from each surface center to the bulk center, as shown in Figure 3.5, respectively. In within-surface directions, we have four symmetric [100] directions for each of six surface centers, and thus we have $S^{within-surface}_{\mathbf{r}_0 \Rightarrow Surface\ Center} = 24$; In off-surface directions, we only have one [100] direction for each of six surface centers, and thus we have $S^{off-surface}_{\mathbf{r}_0 \Rightarrow Surface\ Center} = 6$; As $\mathbf{r}_0 \Rightarrow Lattice\ Corner$, we have three symmetric [100] directions off from each of eight lattice corners, as shown in Figure 3.6, i.e., $S_{\mathbf{r}_0 \Rightarrow Lattice\ Corner} = 24$.

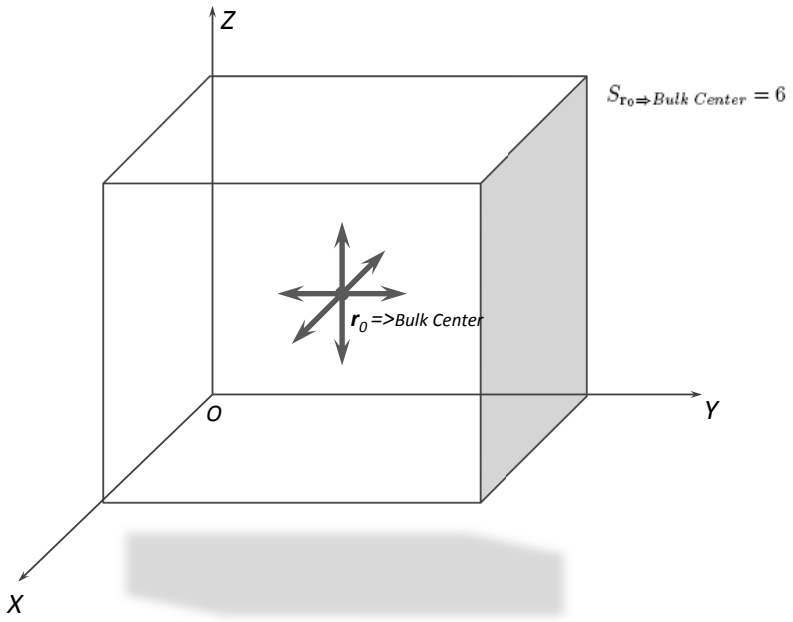

Figure 3.4: Geometrically symmetric [100] directions along which we calculate the spin-spin correlation as $\mathbf{r}_0 \Rightarrow Bulk\ Center$ with $S_{\mathbf{r}_0 \Rightarrow Bulk\ Center} = 6$.

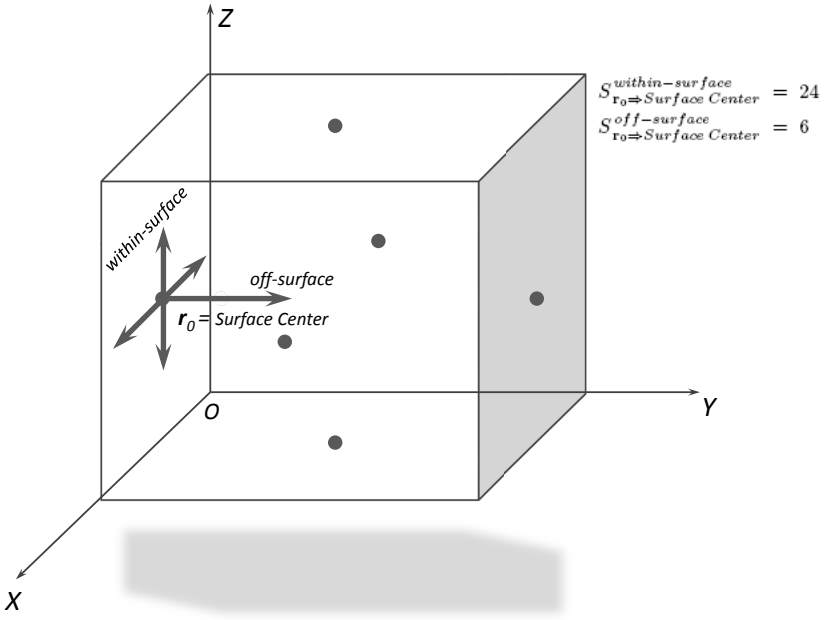

Figure 3.5: Geometrically symmetric [100] directions along which we calculate the spin-spin correlation as $\mathbf{r}_0 \Rightarrow Surface\ Center$ with $S^{within-surface}_{\mathbf{r}_0 \Rightarrow Surface\ Center} = 24$ and $S^{off-surface}_{\mathbf{r}_0 \Rightarrow Surface\ Center} = 6$.

With an even lattice size L, where $L = 2i$, $i = 2, 3, \ldots$, the bulk-center spin and surface-center spins can not be determined uniquely. To overcome this problem, we take the geometric

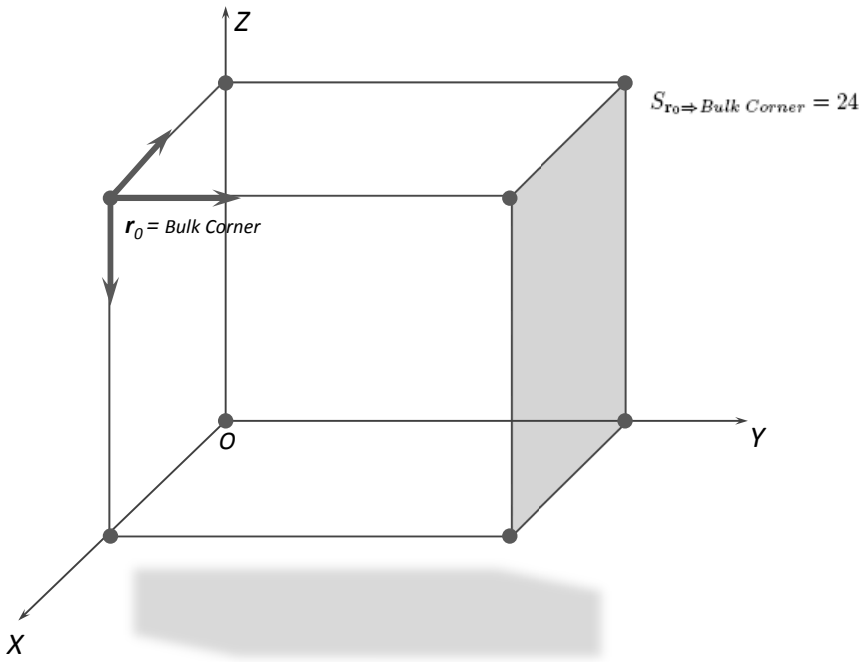

Figure 3.6: Geometrically symmetric [100] directions along which we calculate the spin-spin correlation as $\mathbf{r}_0 \Rightarrow Lattice\ Corner$ with $S_{\mathbf{r}_0 \Rightarrow Lattice\ Corner} = 24$.

mean bulk- and surface-center spins into our calculations. Figure 3.7 and Figure 3.8 illustrate the method we used to define twelve mean bulk-center spins. We take the unit cell at the center of the bulk and magnify it, as shown in the top right portion of the figures. Four spins, which are on the same sublattice, are labeled with solid dots at four symmetric corners of the unit cell. Those four spins are taken as both geometrically and physically equivalent. Six dashed lines on the unit cell surfaces connect any two spins as a pair. We define the average of two spins in a pair as a mean bulk-center spin, which is labeled with an open circle. As is shown in the figures, we have twelve such mean bulk-center spins which are located separately on six unit cell surfaces. From each mean bulk-center spin, a [100] direction is defined and labeled with a thick arrow, along which we perform the calculation of the local correlation. Thus, we have twelve symmetric [100] directions as $\mathbf{r}_0 \Rightarrow Bulk\ Center$, i.e., $S_{\mathbf{r}_0 \Rightarrow Bulk\ Center} = 12$, for an even lattice size L. Accordingly, the bulk-center spin component

$S^k(\mathbf{r}_0)$ in Eqn.(3.33) is replaced by the component of a mean bulk-center spin, according to which [100] direction is used in the calculation.

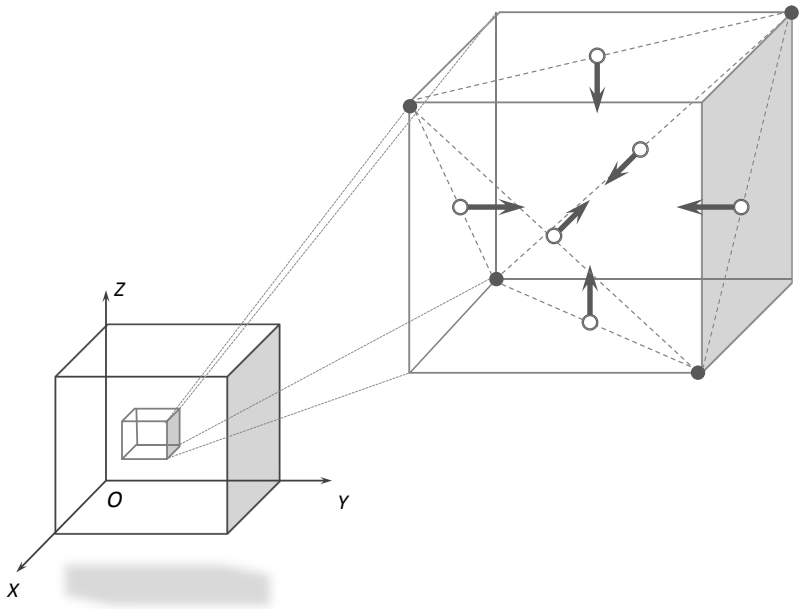

Figure 3.7: Illustration of the method defining six mean bulk-center spins and six symmetric [100] directions on one sublattice of the center unit cell.

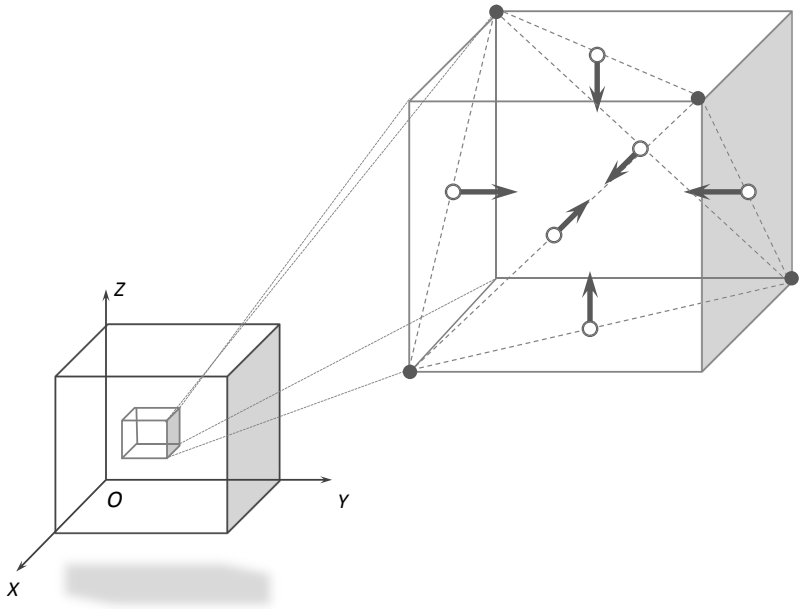

Figure 3.8: Illustration of the method defining six mean bulk-center spins and six symmetric [100] directions on another sublattice of the center unit cell.

Figure 3.9 and Figure 3.10 illustrate the method we used to define twelve mean surface-center spins. Similar to the method we used to define twelve mean bulk-center spins, we take the unit square at the center of each free surface and define the average of two diagonal spins which are on the same sublattice as a mean surface-center spin, which is labeled with an open circle. In within-surface directions, we have four symmetric [100] directions for each of twelve mean surface centers, and thus we have $S^{within-surface}_{\mathbf{r}_0 \Rightarrow Surface\ Center} = 48$; In off-surface directions, we only have one [100] direction for each of twelve mean surface centers, and thus we have $S^{off-surface}_{\mathbf{r}_0 \Rightarrow Surface\ Center} = 12$. Accordingly, the surface-center spin component $S^k(\mathbf{r}_0)$ in Eqn.(3.33) is replaced by the component of a mean surface-center spin, according to which [100] direction is used in the calculation.

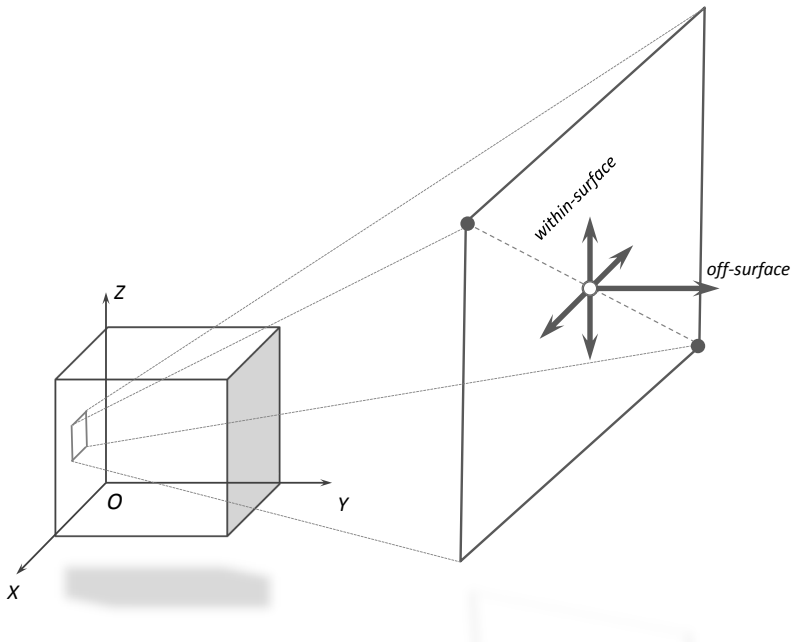

Figure 3.9: Illustration of the method defining six mean surface-center spins and six symmetric [100] directions on one sublattice of the center unit cell.

3.4.4 Local Correlation Function for Model Nanofilms

For model nanofilms with partially free boundary conditions on simple cubic lattices, as discussed before, we defined the *FBCZ* direction in the spatial z-direction with two free surfaces, and the *PBCXY* direction in the x- and y-direction with periodic boundaries parallel

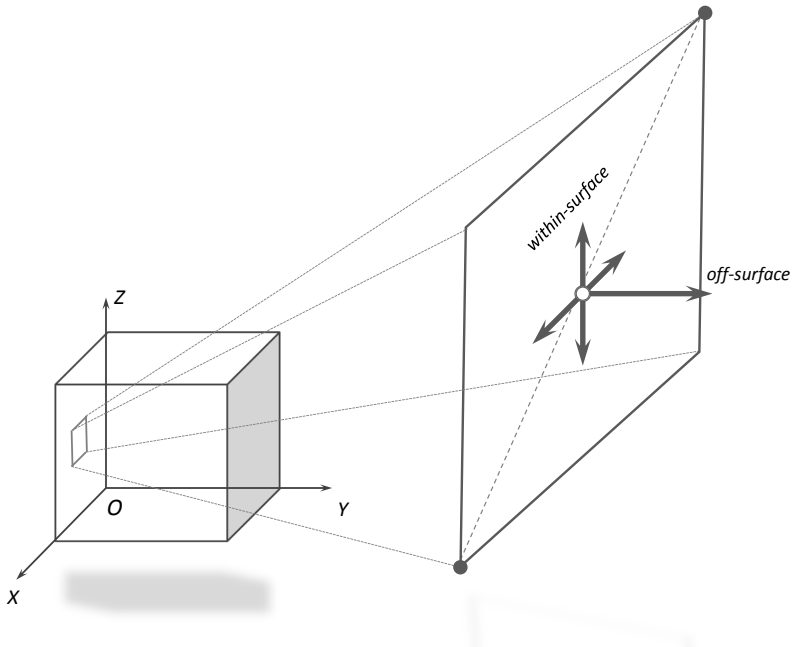

Figure 3.10: Illustration of the method defining six mean surface-center spins and six symmetric [100] directions on another sublattice of the center unit cell.

to the free surfaces. Since the boundary conditions are periodic, the translational invariance is preserved in the $PBCXY$ directions. In this part of work, we calculated the local correlations both in $PBCXY$ and $FBCZ$ directions with even linear dimensions L_{xy} and L_z, i.e., $L_{xy} = 2i$ and $L_z = 2i$, $i = 2, 3, \ldots$. The starting point $\mathbf{r}_0$ for the calculation of the local correlation is chosen to be at the bulk center, i.e., $\mathbf{r}_0 \Rightarrow Bulk\ Center$.

Figure 3.11 illustrates how we define sixteen mean bulk-center spins when we calculate the local correlations in the $PBCXY$ directions. Similar to the method we used to define mean bulk-center spins for nanoparticles with an even lattice size L, for nanofilms with the translational invariance in the $PBCXY$ directions, we take the center unit cell and extend it horizontally in the x- and y-direction into two rectangular tubes crossing at the center, which are magnified and illustrated in Figure 3.12 and Figure 3.13, respectively. In Figure 3.12, each open circle represents a mean bulk-center spin, which is geometrically averaged from spins

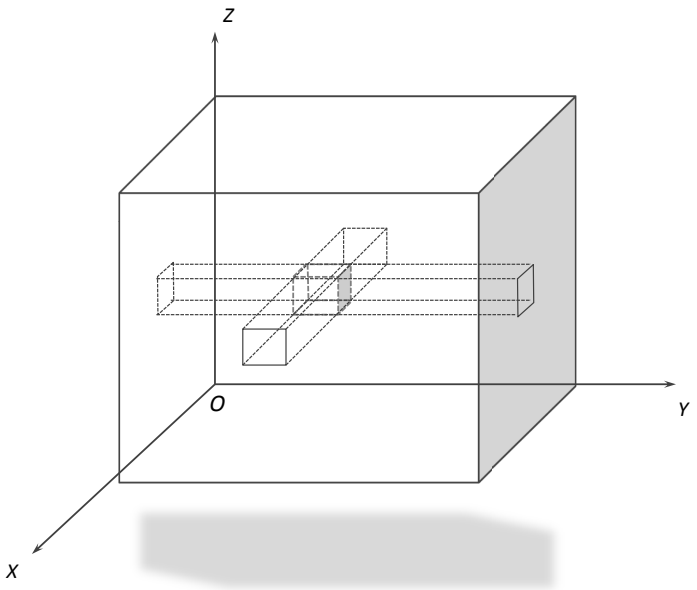

Figure 3.11: Illustration of the method defining sixteen mean bulk-center spins for the calculation of the local correlation in the $PBCXY$ directions.

of the same sublattice on each side wall of the rectangular tube. Thus, we have eight such mean bulk-center spins averaged from two sublattices on four side walls. From each mean bulk-center spin, a [100] direction is defined and labeled with a thick arrow, along which we perform the calculation of the local correlation. Figure 3.13 shows another rectangular tube which is perpendicular to the one in Figure 3.12. We have another eight mean bulk-center spins and eight symmetric [100] directions. Thus, totally we have sixteen symmetric [100] directions as $\mathbf{r}_0 \Rightarrow Bulk\ Center$, i.e., $S_{\mathbf{r}_0 \Rightarrow Bulk\ Center} = 16$, for nanofilms with an even lattice size L in the $PBCXY$ directions. Accordingly, to calculate the local correlation function, the bulk-center spin component $S^k(\mathbf{r}_0)$ in Eqn.(3.33) is replaced by the component of a mean bulk-center spin, according to which [100] direction is used in the calculation.

Figure 3.14 and Figure 3.15 illustrate how we define four mean bulk-center spins when we calculate the local correlations in the $FBCZ$ directions. Similar to the geometric extension

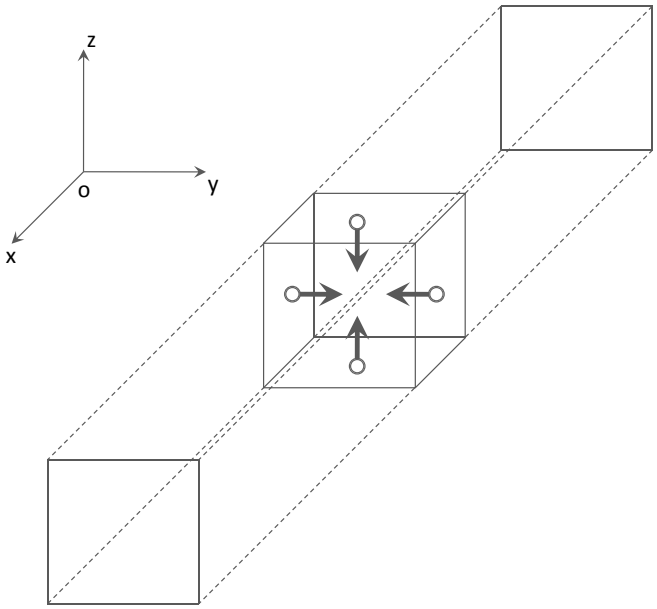

Figure 3.12: Illustration of the method defining eight mean bulk-center spins and eight symmetric [100] directions from two sublattices on one extended rectangular tube for the calculation of the local correlation in the $PBCXY$ directions.

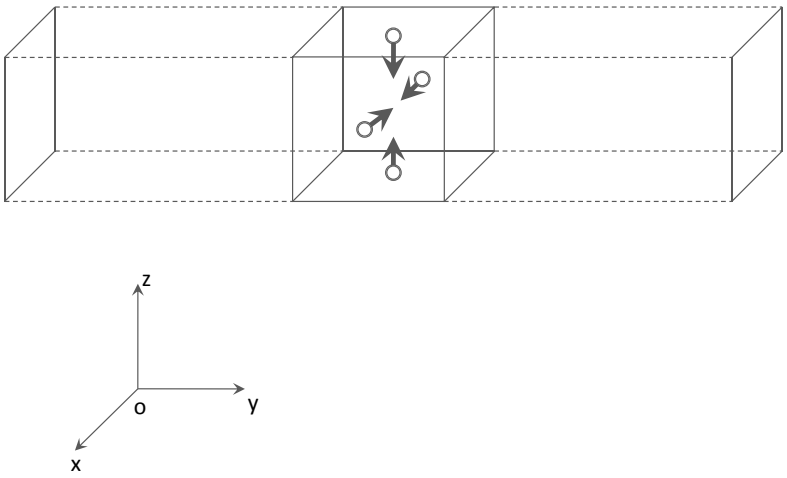

Figure 3.13: Illustration of the method defining eight mean bulk-center spins and eight symmetric [100] directions from two sublattices on another extended rectangular tube for the calculation of the local correlation in the $PBCXY$ directions.

we conducted to define mean bulk-center spins for the calculation of the local correlation in the *PBCXY* directions, with the translational invariance in the *PBCXY* directions, we extend the top and bottom surfaces of the center unit cell horizontally into two center planes, which are labeled in the figures as *"Plane-0"* and *"Plane-1"*. On the Plane-0, an open circle represents a mean bulk-center spin, which is geometrically averaged from spins of the same sublattice. Thus, we have two such mean bulk-center spins averaged from two sublattices on that plane. From each mean bulk-center spin, a [100] direction is defined and labeled with a thick arrow, along which we perform the calculation of the local correlation. On the Plane-1, we have another two mean bulk-center spins and two symmetric [100] directions. Thus, totally we have four symmetric [100] directions as $\mathbf{r}_0 \Rightarrow Bulk\ Center$, i.e., $S_{\mathbf{r}_0 \Rightarrow Bulk\ Center} = 4$, for nanofilms with an even lattice size L in the *FBCZ* directions. Accordingly, to calculate the local correlation function, the bulk-center spin component $S^k(\mathbf{r}_0)$ in Eqn.(3.33) is replaced by the component of a mean bulk-center spin, according to which [100] direction is used in the calculation.

3.4.5 Local Dynamic Structure Factor

Similar to the definition of the dynamic structure factor $S^k(\mathbf{q}, \omega)$, defined in Eqn.(2.38) for the "infinite"system with fully periodic boundary conditions, the definition of the local dynamic structure factor $S(\mathbf{r}_0, \mathbf{q}, \omega)$, which is the Fourier transform of the local space- and time-displaced spin-spin correlation function $C(\mathbf{r}_0, \mathbf{r}, t)$, is given by

$$\begin{aligned} S^k(\mathbf{r}_0, \mathbf{q}, \omega) &= \frac{1}{N} \sum_{\mathbf{r}} e^{i\mathbf{q}\cdot\mathbf{r}} \int_{-\infty}^{+\infty} e^{-i\omega t} C^k(\mathbf{r}_0, \mathbf{r}, t) \frac{dt}{\sqrt{2\pi}} \\ &= \frac{1}{N} \int_{-\infty}^{+\infty} e^{-i\omega t} \frac{dt}{\sqrt{2\pi}} \sum_{\mathbf{r}} e^{i\mathbf{q}\cdot\mathbf{r}} C^k(\mathbf{r}_0, \mathbf{r}, t), \end{aligned} \tag{3.34}$$

where $k = x$, y or z; N is the total number of spins in a lattice (discussed in Eqn.(2.39)).

Due to limited computer resources, there are two major practical limitations on the computer simulation of dynamic behavior, i.e., finite evolution or integration time and finite system size. In our simulations, the simple cubic lattice sizes used range from 10 to 40; the

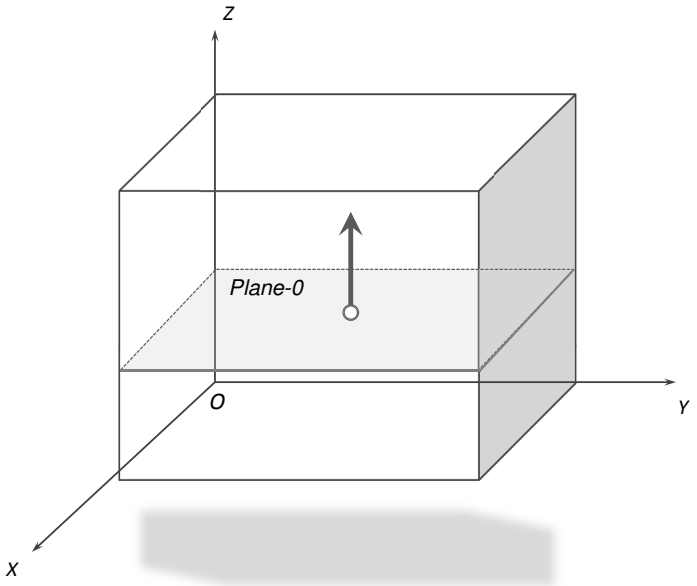

Figure 3.14: Illustration of the method defining two mean bulk-center spins and two symmetric [100] directions from two sublattices on one center plan for the calculation of the local correlation in the *FBCZ* directions.

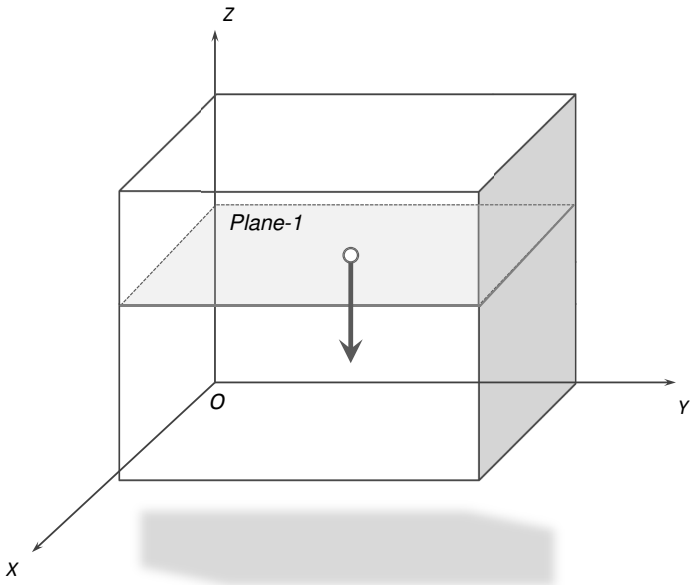

Figure 3.15: Illustration of the method defining two mean bulk-center spins and two symmetric [100] directions from two sublattices on another center plan for the calculation of the local correlation in the *FBCZ* directions.

SD time cutoff, which is denoted as t_c in simulations, is usually the order of 10^3 to 10^4, i.e., $t_c \sim 10^3$ to 10^4. t_c should be distinguished from n_t, the total number of SD time steps, defined previously in Eqn.(3.33). With the finite integration time t_c, the above Eqn.(3.34) can be rewritten as

$$S^k(\mathbf{r}_0, \mathbf{q}, \omega) = \frac{1}{N} \int_{-t_c}^{+t_c} e^{-i\omega t} \frac{dt}{\sqrt{2\pi}} \sum_{\mathbf{r}} e^{i\mathbf{q}\cdot\mathbf{r}} C^k(\mathbf{r}_0, \mathbf{r}, t). \tag{3.35}$$

The calculation of the local correlation is performed in [100] direction, i.e., in momentum space $\mathbf{q} = (q, 0, 0)$. q is determined differently according to $\mathbf{r}_0$ for the calculation of the local correlation, as given below,

- For $\mathbf{r}_0 \Rightarrow$*Bulk Center* or *Surface Center (within-surface)*:

$$q = \begin{cases} \frac{2\pi n_q}{L}, & n_q = 0, 1, 2, \ldots, n_{q_{max}} \equiv \frac{L}{2} \quad \text{for } \textit{even } L = 2i,\ i = 2, 3, \ldots \\ \frac{2\pi n_q}{L}, & n_q = 0, 1, 2, \ldots, n_{q_{max}} \equiv \frac{L-1}{2} \quad \text{for } \textit{odd } L = 2i+1,\ i = 1, 2, 3, \ldots \end{cases} \tag{3.36}$$

- For $\mathbf{r}_0 \Rightarrow$*Lattice Corner* or *Surface Center (off-surface)*:

$$q = \frac{2\pi n_q}{L}, n_q = 0, 1, 2, \ldots, n_{q_{max}} \equiv L \tag{3.37}$$

Combined with Eqn.(3.32), the discrete spatial Fourier transform in Eqn.(3.34) can be written as

$$\begin{aligned} \sum_{\mathbf{r}} e^{i\mathbf{q}\cdot\mathbf{r}} C^k(\mathbf{r}_0, \mathbf{r}, t) &= \sum_{\mathbf{r}} e^{i\mathbf{q}\cdot\mathbf{r}} [\langle S^k(\mathbf{r}_0, t_0) S^k(\mathbf{r}_0 + \mathbf{r}, t_0 + t)\rangle - \langle S^k(\mathbf{r}_0, t_0)\rangle \langle S^k(\mathbf{r}_0 + \mathbf{r}, t_0 + t)\rangle] \\ &= term\,1 - term\,2, \end{aligned} \tag{3.38}$$

where

$$\begin{aligned} term\,1 &= \sum_{\mathbf{r}} e^{i\mathbf{q}\cdot\mathbf{r}}\langle S^k(\mathbf{r}_0,t_0)S^k(\mathbf{r}_0+\mathbf{r},t_0+t)\rangle, \\ term\,2 &= \sum_{\mathbf{r}} e^{i\mathbf{q}\cdot\mathbf{r}}\langle S^k(\mathbf{r}_0,t_0)\rangle\langle S^k(\mathbf{r}_0+\mathbf{r},t_0+t)\rangle \\ &= \langle S^k(\mathbf{r}_0,t_0)\rangle \sum_{\mathbf{r}} e^{i\mathbf{q}\cdot\mathbf{r}}\langle S^k(\mathbf{r}_0+\mathbf{r},t_0+t)\rangle \\ &= \mathcal{S}\sum_{\mathbf{r}} e^{i\mathbf{q}\cdot\mathbf{r}}\langle S^k(\mathbf{r}_0+\mathbf{r},t_0+t)\rangle, \end{aligned} \tag{3.39}$$

where $\mathcal{S} \equiv \langle S^k(\mathbf{r}_0,t_0)\rangle$ is a constant independent of $\mathbf{r}$ and t. Without considering the $\langle\cdots\rangle$ ensemble average (discussed in Eqns.(3.33)), for $\mathbf{q} = (q,0,0)$, $term\,1$ and $term\,2$ can be written as

$$\begin{aligned} term\,1 &= \sum_{\mathbf{r}} e^{i\mathbf{q}\cdot\mathbf{r}} S^k(\mathbf{r}_0,t_0)S^k(\mathbf{r}_0+\mathbf{r},t_0+t) \\ &= \sum_{r_x} e^{iqr_x} \sum_{r_y,r_z} S^k(\mathbf{r}_0,t_0)S^k(\mathbf{r}_0+\mathbf{r},t_0+t) \\ &= \sum_{r_x} e^{iqr_x} S^k(\mathbf{r}_0,t_0) \sum_{r_y,r_z} S^k(\mathbf{r}_0+\mathbf{r},t_0+t) \\ &= \sum_{r_x} e^{iqr_x} S^k(\mathbf{r}_0,t_0) \sum_{plane-r_x} S^k(\mathbf{r}_0+\mathbf{r},t_0+t), \end{aligned} \tag{3.40}$$

$$\begin{aligned} term\,2 &= \mathcal{S}\sum_{\mathbf{r}} e^{i\mathbf{q}\cdot\mathbf{r}} S^k(\mathbf{r}_0+\mathbf{r},t_0+t) \\ &= \mathcal{S}\sum_{r_x} e^{iqr_x} \sum_{r_y,r_z} S^k(\mathbf{r}_0+\mathbf{r},t_0+t) \\ &= \mathcal{S}\sum_{r_x} e^{iqr_x} \sum_{plane-r_x} S^k(\mathbf{r}_0+\mathbf{r},t_0+t). \end{aligned} \tag{3.41}$$

Thus, combining Eqn.(3.40) and Eqn.(3.41), we have

$$\begin{aligned} \sum_{\mathbf{r}} e^{i\mathbf{q}\cdot\mathbf{r}} C^k(\mathbf{r}_0,\mathbf{r},t) &= term\,1 - term\,2 \\ &= \sum_{r_x} e^{iqr_x}\left[S^k(\mathbf{r}_0,t_0)\sum_{plane-r_x} S^k(\mathbf{r}_0+\mathbf{r},t_0+t) - \mathcal{S}\sum_{plane-r_x} S^k(\mathbf{r}_0+\mathbf{r},t_0+t)\right] \\ &= \sum_{r_x} e^{iqr_x} C_L^k(\mathbf{r}_0,r_x,t), \end{aligned} \tag{3.42}$$

where $C_L^k(\mathbf{r}_0, r_x, t)$ is defined as the *local intermediate sum product*. At this point, we consider the $\langle \cdots \rangle$ ensemble average discussed in Eqns.(3.33), and rewrite $C_L^k(\mathbf{r}_0, r_x, t)$ as

$$\begin{aligned} C_L^k(\mathbf{r}_0, r_x, t) &= \frac{1}{S(n_t - t + 1)} \sum_{s=1}^{S} \sum_{t_0=0}^{n_t - t} \left[S^k(\mathbf{r}_0, t_0) \sum_{plane - r_x} S^k(\mathbf{r}_0 + \mathbf{r}, t_0 + t) \right]_s - \\ & \frac{1}{S(n_t - t + 1)} \sum_{s=1}^{S} \sum_{t_0=0}^{n_t - t} \left[\mathcal{S} \sum_{plane - r_x} S^k(\mathbf{r}_0 + \mathbf{r}, t_0 + t) \right]_s , \end{aligned} \tag{3.43}$$

where S is the total number of geometrically symmetric [100] directions discussed in Section 3.4.3 and 3.4.4; $s \in [1,\, S]$ is an integer index on those S directions; $\mathcal{S} \equiv \langle S^k(\mathbf{r}_0, t_0) \rangle = \frac{1}{S(n_t+1)} \sum_{s=1}^{S} \sum_{t_0=0}^{n_t} S^k(\mathbf{r}_0, t_0)$; $\mathbf{r}_0$ and t_0 denote the spatial and temporal starting points for the local correlation function, respectively, and n_t is the total number of SD time steps.

In the case of antiferromagnets, the wave vectors are measured with respect to the $\Lambda \equiv (\pi, \pi, \pi)$ point, i.e., $\mathbf{q} \to \mathbf{q} + \Lambda$, which corresponds to the Brillouin zone center. Then the discrete spatial Fourier transform in Eqn.(3.42) can be rewritten as

$$\begin{aligned} & \sum_{\mathbf{r}} e^{i(\mathbf{q}+\Lambda)\cdot\mathbf{r}} C^k(\mathbf{r}_0, \mathbf{r}, t) \\ = & \sum_{\mathbf{r}} e^{i\Lambda\cdot\mathbf{r}} e^{i\mathbf{q}\cdot\mathbf{r}} C^k(\mathbf{r}_0, \mathbf{r}, t) \\ = & \sum_{r_x} e^{iqr_x} \left[S^k(\mathbf{r}_0, t_0) \sum_{plane - r_x} e^{i\Lambda\cdot\mathbf{r}} S^k(\mathbf{r}_0 + \mathbf{r}, t_0 + t) - \mathcal{S} \sum_{plane - r_x} e^{i\Lambda\cdot\mathbf{r}} S^k(\mathbf{r}_0 + \mathbf{r}, t_0 + t) \right] \\ = & \sum_{r_x} e^{iqr_x} \left[S^k(\mathbf{r}_0, t_0) \sum_{plane - r_x} e^{i\Lambda\cdot\mathbf{r}} S^k(\mathbf{r}_0 + \mathbf{r}, t_0 + t) - e^{-i\Lambda\cdot\mathbf{r}_0} \mathcal{S} \sum_{plane - r_x} e^{i\Lambda\cdot(\mathbf{r}_0+\mathbf{r})} S^k(\mathbf{r}_0 + \mathbf{r}, t_0 + t) \right] \\ = & \sum_{r_x} e^{iqr_x} \left[S^k(\mathbf{r}_0, t_0) \sum_{plane - r_x} f_1 S^k(\mathbf{r}_0 + \mathbf{r}, t_0 + t) - f_2 \mathcal{S} \sum_{plane - r_x} f_3 S^k(\mathbf{r}_0 + \mathbf{r}, t_0 + t) \right] \\ = & \sum_{r_x} e^{iqr_x} C_L^k(\mathbf{r}_0, r_x, t), \end{aligned} \tag{3.44}$$

where $f_1 \equiv e^{i\Lambda\cdot\mathbf{r}}$, $f_2 \equiv e^{-i\Lambda\cdot\mathbf{r}_0}$, and $f_3 \equiv e^{i\Lambda\cdot(\mathbf{r}_0+\mathbf{r})}$ are newly introduced parameters with values that can be easily calculated as

$$f_1 = \begin{cases} +1 & \textit{if two spins } S(\mathbf{r}_0) \textit{ and } S(\mathbf{r}_0+\mathbf{r}) \textit{ are on the same sublattice,} \\ -1 & \textit{otherwise.} \end{cases} \tag{3.45}$$

$$f_2, f_3 = \begin{cases} +1 & \textit{if the summation over spatial coordinates } x,\ y, \textit{ and } z \textit{ of the spin} \\ & S(\mathbf{r}_0) \textit{ or } S(\mathbf{r}_0+\mathbf{r}) \textit{ is even,} \\ -1 & \textit{otherwise.} \end{cases} \tag{3.46}$$

Thus, in the case of antiferromagnets, the local intermediate sum product $C_L^k(\mathbf{r}_0, r_x, t)$ with the $\langle \cdots \rangle$ ensemble average taken into account has a similar form as the one defined in Eqn.(3.43) as

$$\begin{aligned} C_L^k(\mathbf{r}_0, r_x, t) &= \frac{1}{S(n_t - t + 1)} \sum_{s=1}^{S} \sum_{t_0=0}^{n_t - t} \left[S^k(\mathbf{r}_0, t_0) \sum_{plane - r_x} f_1 S^k(\mathbf{r}_0 + \mathbf{r}, t_0 + t) \right]_s - \\ & \frac{1}{S(n_t - t + 1)} \sum_{s=1}^{S} \sum_{t_0=0}^{n_t - t} \left[f_2 \mathcal{S} \sum_{plane - r_x} f_3 S^k(\mathbf{r}_0 + \mathbf{r}, t_0 + t) \right]_s , \end{aligned} \tag{3.47}$$

In addition to the ensemble average discussed in Eqns.(3.33), we must also average over the multiple integration runs. Each run starts from an initial equilibrium configuration drawn from the canonical ensemble (discussed in Section 3.1). It is obtained by starting from a random distribution, running Monte Carlo and discarding the first $10\tau_{nl}$ MCS [54] (Monte Carlo time) configurations, or discarding the first $10\tau_l$ MCS configurations if the starting equilibrium configuration is used for integration.

The above defined *local sum product* $C_L^k(\mathbf{r}_0, r_x, t)$ should be distinguished from the local space- and time-displaced spin-spin correlation function $C(\mathbf{r}_0, \mathbf{r}, t)$ because the former is partially Fourier transformed from the latter. To calculate the local dynamic structure factor in [100] direction with $\mathbf{q} = (q, 0, 0)$, we combine Eqn.(3.35), Eqn.(3.44) and Eqn.(3.42) and

obtain

$$\begin{aligned} S^k(\mathbf{r}_0,\mathbf{q},\omega) &= \frac{1}{N}\int_{-t_c}^{+t_c} e^{-i\omega t}\frac{dt}{\sqrt{2\pi}}\sum_{r_x} e^{iqr_x}C_L^k(\mathbf{r}_0,r_x,t) \\ &= \frac{1}{N}\int_{-t_c}^{+t_c} e^{-i\omega t}\frac{dt}{\sqrt{2\pi}}\sum_{r_x=-n_{q_{max}}}^{n_{q_{max}}} e^{iqr_x}C_L^k(\mathbf{r}_0,r_x,t) \\ &= \frac{2}{N}\int_{-t_c}^{+t_c} e^{-i\omega t}\frac{dt}{\sqrt{2\pi}}\sum_{r_x=0}^{n_{q_{max}}} \cos(qr_x)C_L^k(\mathbf{r}_0,r_x,t) \\ &= \frac{4}{N}\int_{0}^{+t_c} \cos(\omega t)\frac{dt}{\sqrt{2\pi}}\sum_{r_x=0}^{n_{q_{max}}} \cos(qr_x)C_L^k(\mathbf{r}_0,r_x,t), \end{aligned} \tag{3.48}$$

where $n_{q_{max}}$ is defined in Eqn.(3.36) and Eqn.(3.37). In last two lines, we replace e^{iqr_x} by $\cos(qr_x)$ and $e^{-i\omega t}$ by $\cos(\omega t)$ because there is space and time reversal symmetry due to constant energy in the simulation, and thus the imaginary components of both the time and space Fourier transforms vanish.

We have finite time cutoff t_c in the time Fourier transform. The finite time cutoff will result in oscillations in $S(\mathbf{r}_0,\mathbf{q},\omega)$ of frequency $\frac{2\pi}{t_c}$. To alleviated this problem, we did the convolution of the dynamic structure factor with a Gaussian resolution function in frequency ω, i.e.,

$$S^k(\mathbf{r}_0,\mathbf{q},\omega) = \frac{4}{N}\int_{0}^{+t_c} \cos(\omega t)e^{-\frac{1}{2}(t\delta_\omega)^2}\frac{dt}{\sqrt{2\pi}}\sum_{r_x=0}^{n_{q_{max}}} \cos(qr_x)C_L^k(\mathbf{r}_0,r_x,t), \tag{3.49}$$

where δ_ω is a parameter determining the width of the resolution function and needs to be chosen properly such that the effects of the finite time cutoff can be neglected.

In the simulations for ferromagnetic systems with periodic boundary conditions, after we generate an equilibrium initial spin configuration, we choose a cartesian coordinate frame of reference in the spin space such that its z-axis is parallel to the uniform magnetization M, which is a constant of motion and the order parameter of the system. We then evolve the spin configuration with t_c, i.e., SD time cutoff, to a maximum time t_{max}. The dynamic structure factor can then be regrouped in the spin space into a longitudinal component

$$S^L(\mathbf{q},\omega) = S^z(\mathbf{q},\omega), \tag{3.50}$$

and a transverse component

$$S^T(\mathbf{q},\omega) = \frac{1}{2}\big(S^x(\mathbf{q},\omega) + S^y(\mathbf{q},\omega)\big), \tag{3.51}$$

since x and y components are equivalent. The longitudinal component $S^L(\mathbf{q},\omega)$ of propagating excitations for all classical Heisenberg models was found to be made up of two-spin wave peaks [38, 44].

In the simulations for antiferromagnetic nanostructures, we applied the same approach to regroup the local dynamic structure factor in the spin space into a longitudinal component

$$S^L(\mathbf{r}_0,\mathbf{q},\omega) = S^z(\mathbf{r}_0,\mathbf{q},\omega), \tag{3.52}$$

and a transverse component

$$S^T(\mathbf{r}_0,\mathbf{q},\omega) = \frac{1}{2}\big(S^x(\mathbf{r}_0,\mathbf{q},\omega) + S^y(\mathbf{r}_0,\mathbf{q},\omega)\big). \tag{3.53}$$

However, for the antiferromagnet the order parameter, i.e., the staggered magnetization, is not a constant of motion; therefore, regrouping components of the spin parallel (longitudinal component) and perpendicular (transverse component) to the order parameter is challenging. As we integrate the equations of motion, the direction of the staggered magnetization changes slightly because it is not a conserved quantity. Our approach to overcome this problem is to rotate the coordinate frame of reference continually after each integration step so that the z-axis is to be realigned to the staggered magnetization and restored the z-axis as the longitudinal direction. This procedure is called as the *z-polarization* in our simulations.

3.4.6 The Ψ-*Mag* Toolset

Our simulations are carried out by utilizing elements of the $\Psi - Mag$ Toolset (or Toolkit) [53], which is a secondary C++ template library developed by ORNL for computational magnetism and serves as a prototype for a more general library for computational material science. Its design is modeled after and inspired by the generic programming techniques of the C++ standard template library (C++ STL). The major Ψ-*Mag* elements applied in our simulation codes include

1. EHModel.h: Generic extendable Heisenberg Hamiltonian
2. EHTerms.h: Generic term in Heisenberg Hamiltonian
3. EHTerms_ExchangeScaler.h: Scalar exchange interaction term in the Heisenberg Hamiltonian
4. EHTerms_AnisotropyUniaxial.h: Single-site anisotropy term in the Heisenberg Hamiltonian
5. HeisenbergSpinStepper.h: Metropolis Monte Carlo
6. OverRelaxation.h: Overrelaxation algorithms
7. WolffCluster.h: Wolff cluster flipping algorithms
8. RotationOperator.h: Single spin rotation
9. SuzukiTrotter_ani.h: Originally SuzukiTrotter.h, which arranges the operators in the sequence defined in Eqn.(3.26) for isotropic systems. It was modified to SuzukiTrotter_ani.h to add the iterative procedure for anisotropic systems
10. LatticeIndexMapper.h: Mapping between lattice coordinates and the lattice index
11. StatisticalMoments.h: Statistical moments for error bars

Using Toolset requires the specification of lattice structure, neighbor list, exchange parameter list for each lattice site, Monte Carlo (Monte Carlo, Hybrid Monte Carlo composition, number of Monte Carlo steps to discard for equilibrium, and number of Monte Carlo steps to run), and spin dynamics (SD time step dt, total number of SD time steps n_t, the SD time cutoff t_c, and sublattice decomposition $\mathcal{A}$ and $\mathcal{B}$).

Chapter 4

Results

4.1 Static Monte Carlo Results

4.1.1 Free Boundary Effects on the Antiferromagnetic Order Parameter of Nanoparticles

In the beginning of our simulations, we investigated the static effects of free boundaries on the antiferromagnetic order parameter ϕ; namely, the staggered magnetization, of nanoparticles at a specific temperature $T = 0.4T_N$. T_N is the Néel temperature which has been determined to a high degree of accuracy of $T_N = 1.442929(77)|J|/k_B$, by Chen *et al.* [18] for the isotropic Heisenberg system with simple cubic lattice geometry.

The antiferromagnetic order parameter, i.e., the staggered magnetization, is defined as [51]

$$\phi = \left| \frac{1}{L^3} \sum_{x=1}^{L} \sum_{y=1}^{L} \sum_{z=1}^{L} (-1)^{x+y+z} S(x, y, z) \right|, \tag{4.1}$$

where L is the lattice size of nanoparticles; $S(x, y, z)$ is the spin located at $\mathbf{r} = (x, y, z)$.

To study the free boundary effects on the order parameter, we did the so-called "cellization"procedure, in which we split the simple cubic lattice into non-overlapped $2 \times 2 \times 2$ unit cells, each with eight spins. We then calculate the staggered magnetization for each unit cell. As the location shifts from the center unit cell to the boundary one in a specific direction, the unit-cell staggered magnetization varies as a function of the displacement from the center unit cell in that direction. Figures 4.1, 4.2, and 4.3 show the results in [100], [110], and [111] directions, respectively, on a lattice with a lattice size of $L = 18$ at $T = 0.4T_N$. In addition to the isotropic system with $D/|J| = 0$, we also performed the calculation on

anisotropic systems with $D/|J| \neq 0$, as shown in the figures. For the isotropic system, in a previous study [37] that used a hybrid of the Metropolis and overrelaxation algorithms a hybrid step of 2 Metropolis sweeps and 8 overrelaxation steps was determined to be optimal. For anisotropic systems, 1 hybrid MC step consists of 2 Metropolis sweeps and 8 Wolff cluster spin flips [40]. For both isotropic and anisotropic systems, 2×10^4 hybrid MC steps were discarded to approach equilibrium and 2×10^4 hybrid MC steps were used for the canonical ensemble average at $T = 0.4T_N$. Error bars in the figures are smaller than the symbols.

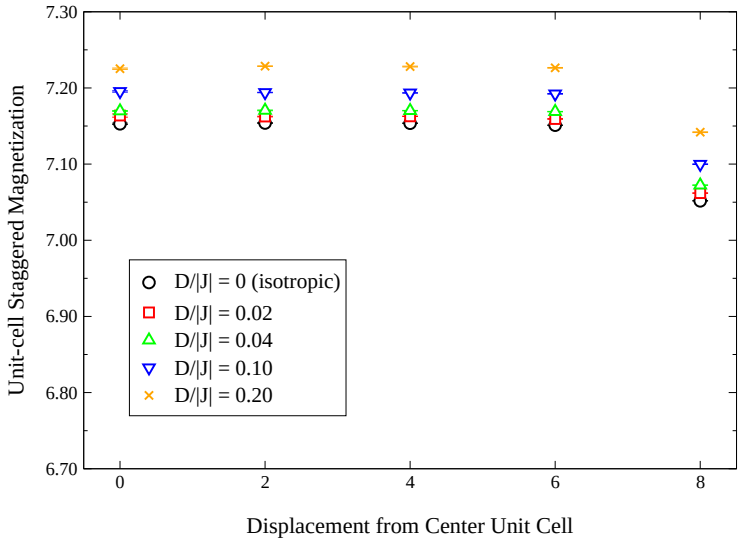

Figure 4.1: The unit-cell staggered magnetization as a function of the displacement with the unit of lattice constant a from the center unit cell in the [100] direction for nanoparticles with $L = 18$ at $T = 0.4T_N$. Boundary unit cell has the displacement of 8. Error bars are smaller than the symbols.

As shown in the figures, for both isotropic and anisotropic systems, the unit-cell staggered magnetization drops as the displacement from the center unit cell approaches the maximum values, i.e., the free boundaries of the system, in [100], [110], and [111] directions, respectively. Approximately, the drop in the [100] direction is ~ 0.10, in the [110] direction ~ 0.20, and in the [111] direction ~ 0.35. Thus, if we take the drop of the order parameter as the

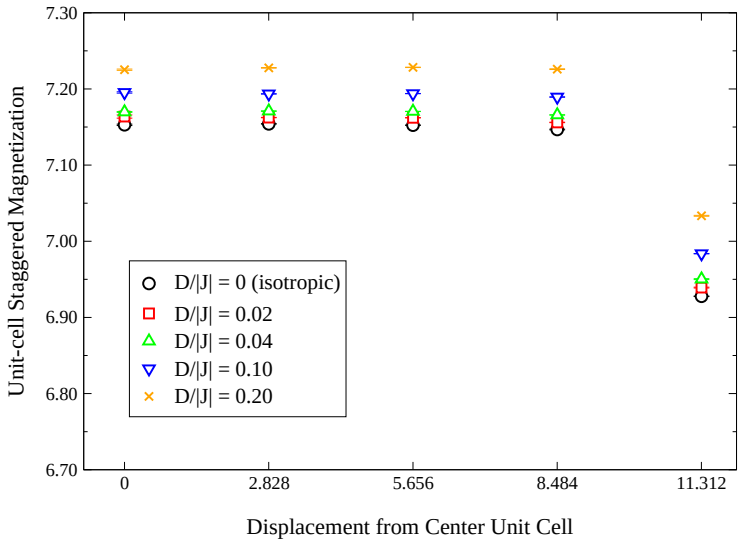

Figure 4.2: The unit-cell staggered magnetization as a function of the displacement with the unit of lattice constant a from the center unit cell in the [110] direction for nanoparticles with $L = 18$ at $T = 0.4T_N$. Boundary unit cell has the displacement of 11.312. Error bars are smaller than the symbols.

criteria of the magnitude of free boundary effects, the maximum boundary effect on the antiferromagnetic order parameter is in the [111] direction.

4.1.2 Thermodynamic Properties of Nanoparticles

In this section, we show the static Monte Carlo simulation results for thermodynamic quantities of nanoparticles, as functions of temperature T. Figures 4.4 and 4.5 show the data for the antiferromagnetic order parameter ϕ and the mean energy per spin E/N, respectively, where N is the total number of spins on the modeled nanoparticles with three lattice sizes, i.e., $L = 5$, 11, and 21; Figures 4.6 and 4.7 show the results for the specific heat C_ν and the magnetic susceptibility χ, respectively. Our simulations were conducted both on isotropic systems with $D/|J| = 0$ and anisotropic systems with $D/|J| = 1.0$ with the classical Heisenberg Hamiltonian defined in Eqn.(2.22). As discussed before, for isotropic systems, 1 hybrid MC step consists of 2 Metropolis sweeps plus 8 overrelaxation steps; for anisotropic

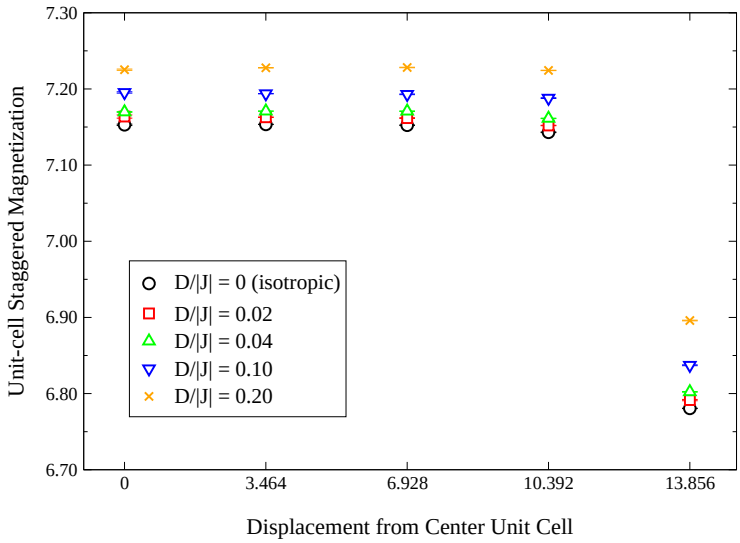

Figure 4.3: The unit-cell staggered magnetization as a function of the displacement with the unit of lattice constant a from the center unit cell in the [111] direction for nanoparticles with $L = 18$ at $T = 0.4T_N$. Boundary unit cell has the displacement of 13.856. Error bars are smaller than the symbols.

systems, 1 hybrid MC step consists of 2 Metropolis sweeps plus 8 Wolff cluster spin flips. We performed annealing simulations in three temperature ranges, i.e., $k_BT/|J| = 0.4 \sim 0.7$, $0.7 \sim 1.7$, and $1.7 \sim 3.0$. For temperature ranges $k_BT/|J| = 0.4 \sim 0.7$ and $0.7 \sim 1.7$, 5×10^4 hybrid MC steps were discarded to approach equilibrium and 5×10^4 hybrid MC steps were used for the canonical ensemble average for three lattice sizes. For the temperature range $k_BT/|J| = 1.7 \sim 3.0$, 1×10^4 hybrid MC steps were discarded to approach equilibrium and 1×10^4 hybrid MC steps were used for the canonical ensemble average for three lattice sizes. Error bars in the figures are smaller than the symbols.

As shown in Figure 4.6, for nanoparticles with free boundary conditions, the Néel temperature T_N varies as the lattice size varies; a larger systems has a higher T_N. For isotropic systems with $D/|J| = 0$, $k_BT_N/|J| \sim 1.04$ with $L = 5$, $k_BT_N/|J| \sim 1.24$ with $L = 11$; $k_BT_N/|J| \sim 1.36$ with $L = 21$; For anisotropic systems with $D/|J| = 1.0$, $k_BT_N/|J| \sim 1.32$

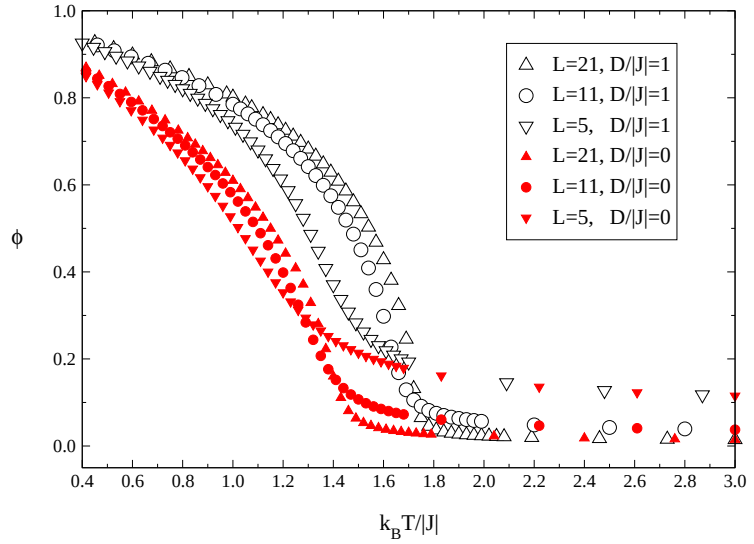

Figure 4.4: The antiferromagnetic order parameter ϕ as a function of temperature T for isotropic (red solid symbols) and anisotropic (black open symbols) nanoparticles with completely free boundary conditions with $L = 5$, 11, and 21. Error bars are smaller than the symbols.

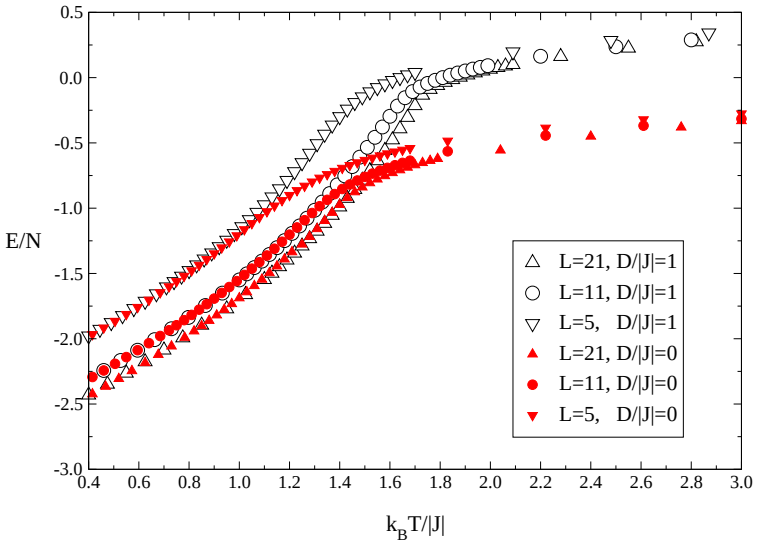

Figure 4.5: The mean energy per spin E/N as a function of temperature T for isotropic (red solid symbols) and anisotropic (black open symbols) nanoparticles with completely free boundary conditions with $L = 5$, 11, and 21. Error bars are smaller than the symbols.

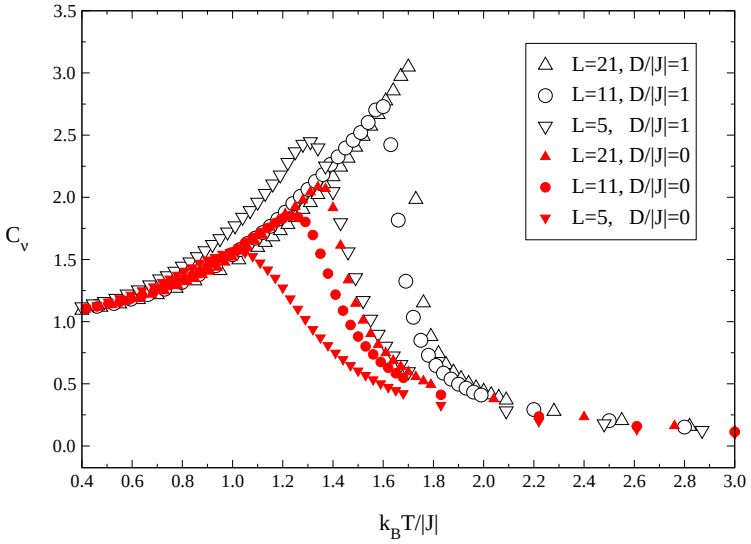

Figure 4.6: The specific heat C_ν as a function of temperature T for isotropic (red solid symbols) and anisotropic (black open symbols) nanoparticles with completely free boundary conditions with $L = 5$, 11, and 21. Error bars are smaller than the symbols.

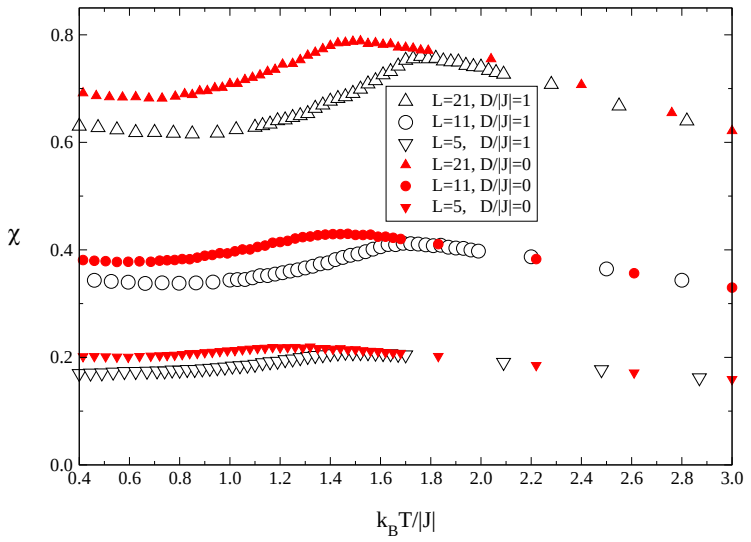

Figure 4.7: The magnetic susceptibility χ as a function of temperature T for isotropic (red solid symbols) and anisotropic (black open symbols) nanoparticles with completely free boundary conditions with $L = 5$, 11, and 21. Error bars are smaller than the symbols.

with $L = 5$, $k_B T_N/|J| \sim 1.59$ with $L = 11$; $k_B T_N/|J| \sim 1.70$ with $L = 21$, i.e., for the same lattice size the T_N of an anisotropic system is higher than that of the isotropic one.

4.2 Spin Dynamics Simulation Results

4.2.1 SD Time Series of Magnetization: A Test on Effects of SD Time Step of Algorithm with Nanoparticles

As discussed in Section 3.3.2, the greatest advantage of the Suzuki-Trotter decomposition algorithm is its capability for handling large SD time steps and the exact conservation of spin length $|\mathbf{S}|$ [86]. For isotropic systems it also conserves the energy exactly. For anisotropic system, energy conservation can be obtained to a high accuracy using iterative schemes; exact magnetization conservation, however, is lost by the algorithm. In this work, to obtain a higher degree of accuracy, we applied the fourth-order Suzuki-Trotter decomposition algorithm. To determine the SD time step dt to be used in the time integration, we conducted tests on dt based on its performance on magnetization conservation.

We first consider an isotropic, antiferromagnetic nanoparticle on a simple cubic lattice with a lattice size of $L = 10$ at a temperature $k_B T/|J| = 1.0$. We performed the Suzuki-Trotter algorithm with $n_t = 10400$, where n_t is the total number of SD time steps, and four different SD time steps, i.e., $dt = 0.1/|J|$, $0.2/|J|$, $0.3/|J|$, and $0.4/|J|$, and obtained the SD time series of the z component of the total magnetization denoted as $M_z(t)$, as shown in Figure 4.8.

Next, we consider anisotropic, antiferromagnetic nanoparticles with $D/|J| = 0.01$ on a simple cubic lattice with lattice sizes of $L = 10$, 20, and 40 at the same temperature $k_B T/|J| = 1.0$. We performed the SD simulations with the same $n_t = 10400$ and four SD time steps, i.e., $dt = 0.1/|J|$, $0.2/|J|$, $0.3/|J|$, and $0.4/|J|$. For anisotropic systems we used an iterative procedure in the spin dynamics, and the total number of iterations we used is 3. Figures 4.9, 4.10 and 4.11 show $M_z(t)$ for three lattice sizes, respectively.

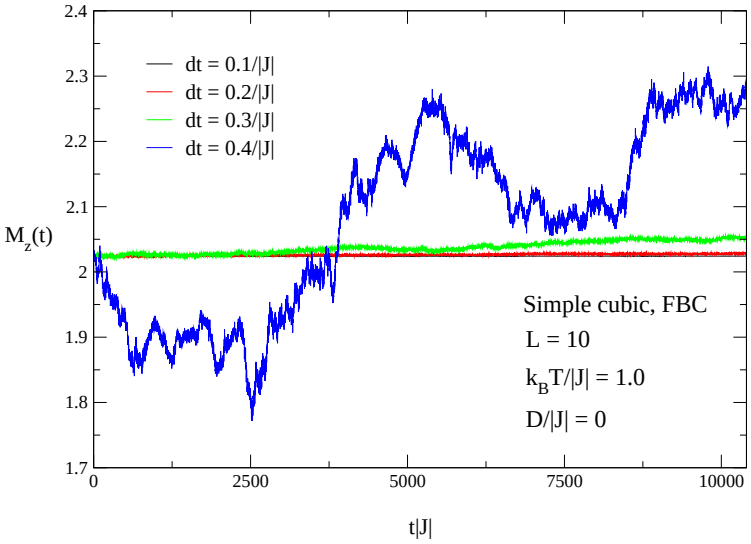

Figure 4.8: The SD time series of the z component of the total magnetization $M_z(t)$ for an isotropic, antiferromagnetic nanoparticle on a simple cubic lattice with a lattice size of $L = 10$ at a temperature $k_BT/|J| = 1.0$, with $n_t = 10400$ and $dt = 0.1/|J|$, $0.2/|J|$, $0.3/|J|$, and $0.4/|J|$.

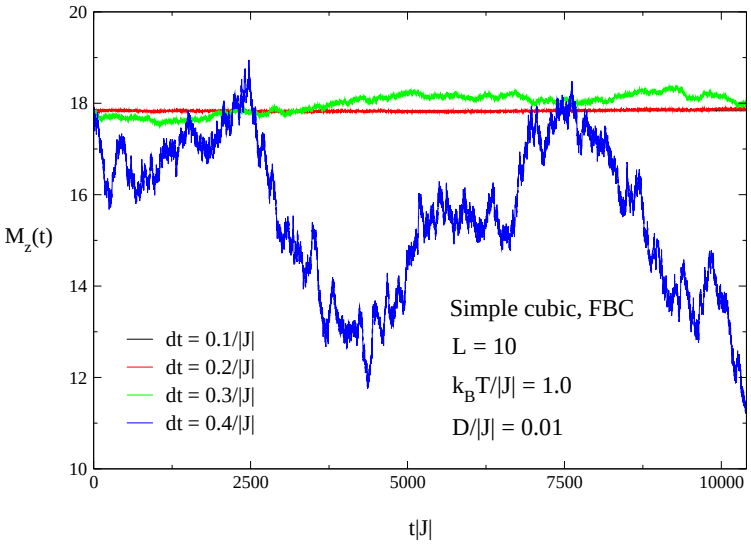

Figure 4.9: The SD time series of the z component of the total magnetization $M_z(t)$ for an anisotropic, antiferromagnetic nanoparticle with $D/|J| = 0.01$ on a simple cubic lattice with a lattice size of $L = 10$ at a temperature $k_BT/|J| = 1.0$, with $n_t = 10400$ and $dt = 0.1/|J|$, $0.2/|J|$, $0.3/|J|$, and $0.4/|J|$.

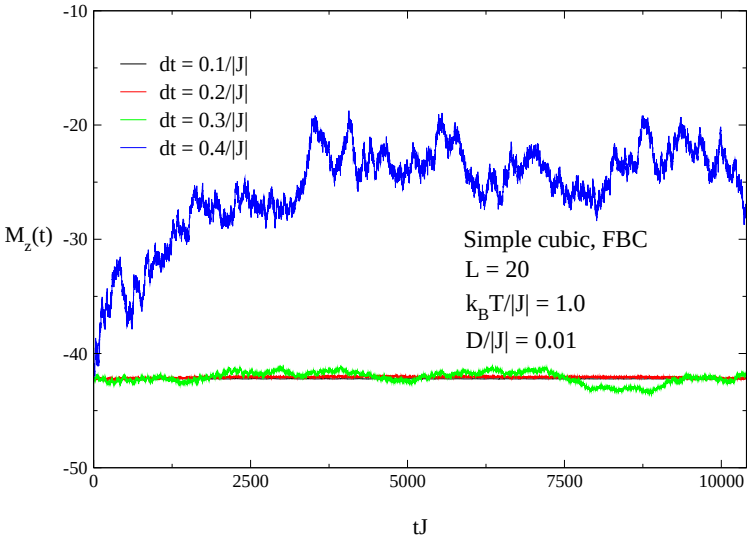

Figure 4.10: The SD time series of the z component of the total magnetization $M_z(t)$ for an anisotropic, antiferromagnetic nanoparticle with $D/|J| = 0.01$ on a simple cubic lattice with a lattice size of $L = 20$ at a temperature $k_BT/|J| = 1.0$, with $n_t = 10400$ and $dt = 0.1/|J|$, $0.2/|J|$, $0.3/|J|$, and $0.4/|J|$.

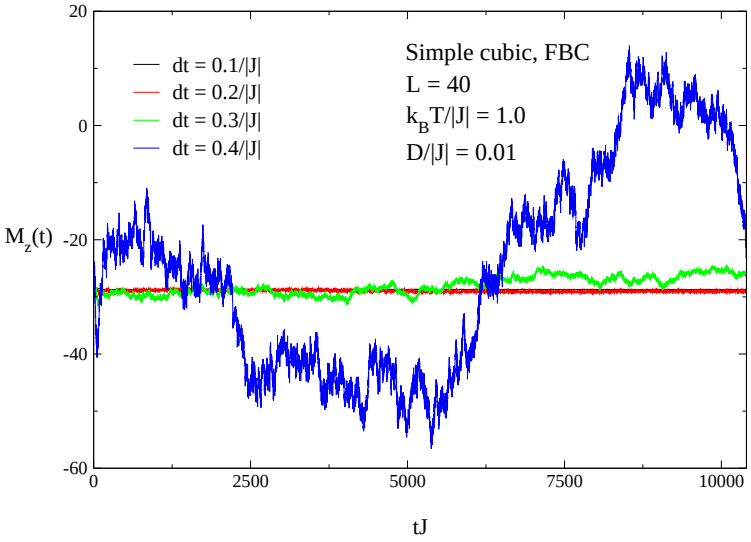

Figure 4.11: The SD time series of the z component of the total magnetization $M_z(t)$ for an anisotropic, antiferromagnetic nanoparticle with $D/|J| = 0.01$ on a simple cubic lattice with a lattice size of $L = 40$ at a temperature $k_BT/|J| = 1.0$, with $n_t = 10400$ and $dt = 0.1/|J|$, $0.2/|J|$, $0.3/|J|$, and $0.4/|J|$.

As shown in the figures, the Suzuki-Trotter decomposition algorithm yields substantially smaller magnetization fluctuations with smaller SD time steps of $dt = 0.1/|J|$ and $0.2/|J|$ than those with $dt = 0.3/|J|$ and $0.4/|J|$. To balance the trade-off between accuracy and efficiency, we used $dt = 0.2/|J|$ as the SD time step in this work.

4.2.2 Comparison with Low Temperature Spin-Wave Theory

In order to demonstrate the proper functioning of the SD algorithm, we performed spin dynamics runs at $T = 0.1T_N$ and $T = 0.2T_N$. At those low temperatures the spin-wave frequency should match the linear spin-wave results given in Section 2.3.3. Our simulations were performed for the isotropic ferro- and antiferromagnet on simple cubic lattices of $L = 24$ with periodic boundary conditions. We found agreement between the positions of spin-wave peaks and the predicted dispersion curve from linear spin-wave theory as shown in the figures 4.12 and 4.13. It was also observed that our simulation results are slight lower than the predicted values from linear spin-wave theory. This slight lowering of simulation results are due to the finite temperature and finite lattice size used in our simulations.

4.2.3 Isotropic Antiferromagnetic Nanofilms in *PBCXY* [100] Directions I: spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$

In this section, we give the results for the transverse component of the local dynamic structure factor $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ with $\mathbf{r}_0 \Rightarrow$ *Bulk Center* for isotropic, antiferromagnetic nanofilms on $L_{xy} \times L_{xy} \times L_z$ simple cubic lattice. The results were obtained in the *PBCXY* [100] directions (discussed in Section 3.4.4), i.e., the directions parallel to the free surfaces, as shown in Figure 4.14.

We performed spin dynamics runs at a specific temperature $T = 0.4T_N$. We discarded 5×10^3 hybrid MC steps to approach the first equilibrium spin configuration and used 5×10^3 hybrid MC steps to generate each of the rest of $N - 1$ equilibrium spin configurations

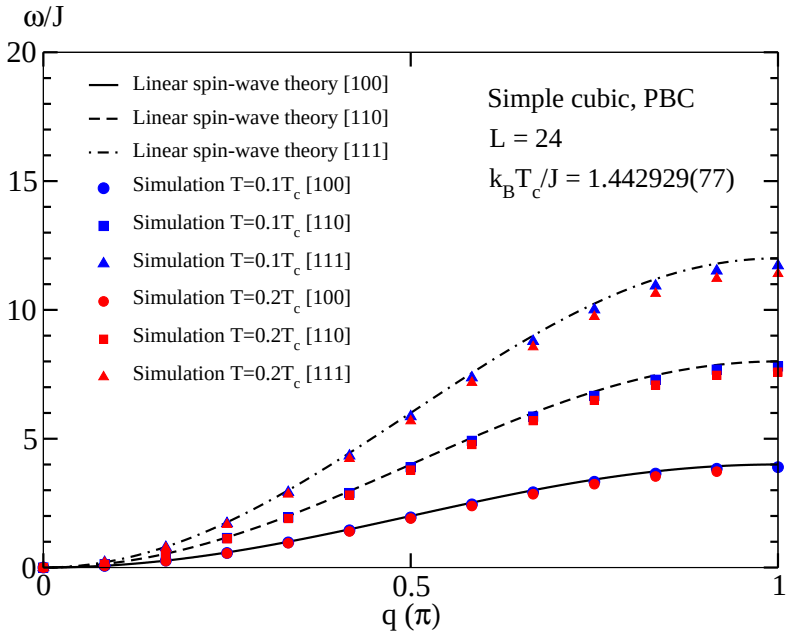

Figure 4.12: Comparison with linear spin-wave theory results for the isotropic ferromagnet. Three black lines give the predicted dispersion curves in the [100], [110], and [111] momentum space directions, respectively. Solid symbols give simulation results at $T = 0.1T_c$ (blue symbols) and $T = 0.2T_c$ (red symbols) in three momentum space directions, respectively. Error bars are smaller than the symbols.

drawn from a canonical ensemble at $T = 0.4T_N$. The total number of SD time steps in our simulations was $n_t = 5000$; the SD time cutoff was $t_c = 4000$; the SD time step was $dt = 0.2/|J|$.

Figure 4.15 shows the spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$, obtained from isotropic, antiferromagnetic nanofilms with the same $L_{xy} = 20$ and three different thicknesses, i.e., $L_z = 10$, 20, and 30. For convenience, we labeled the y-axis of our results with $S^T(\mathbf{r}_0, n_q, \omega)$ with $n_q = 0, 1, 2, \ldots$ (defined in Eqn.(3.36) and Eqn.(3.37)). In the figure, we give the spectra for $n_q = 0, 1, 2, \ldots, 5$.

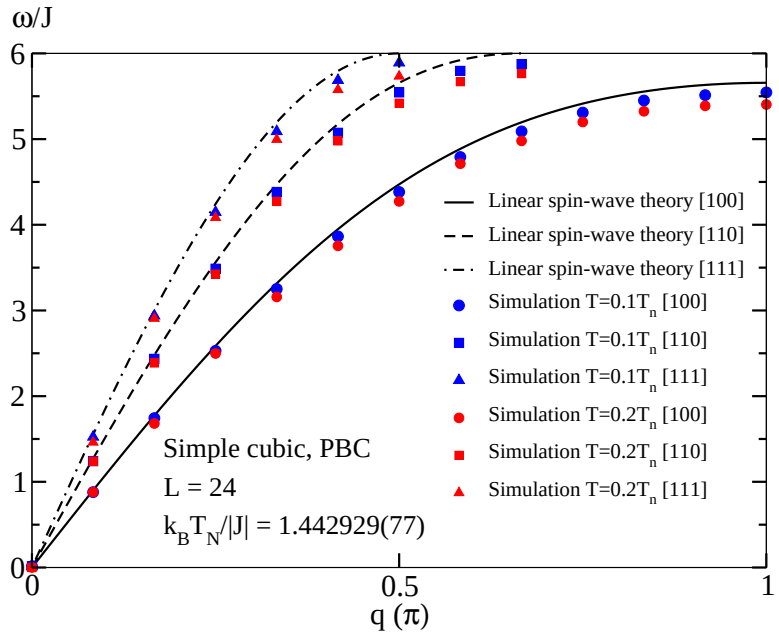

Figure 4.13: Comparison with linear spin-wave theory results for the isotropic antiferromagnet. Three black lines give the predicted dispersion curves in the [100], [110], and [111] momentum space directions, respectively. Solid symbols give simulation results at $T = 0.1T_N$ (blue symbols) and $T = 0.2T_N$ (red symbols) in three momentum space directions, respectively. Error bars are smaller than the symbols.

The vertical dashed lines in the figure labeled with ω_{PBC} show the single spin-wave excitation locations for each wave vector of the "infinite"system with periodic boundary conditions. Three major observations have been made pertinent to the above spectra:

1. Multiple excitation peaks for wave vectors within the first Brillouin zone appear in the spin-wave spectra for the transverse component of the local dynamic structure factor $S^T(\mathbf{r}_0, n_q, \omega)$ in the classical Heisenberg isotropic antiferromagnetic nanofilms, which are lacking if periodic boundary conditions are used (discussed in Section 4.2.5);

2. The width of the energy gap between the main peak and the secondary peak with a higher energy frequency, which is denoted as $\Delta\omega_{12}$, in spectra for some wave vectors, e.g., $n_q = 1, 2, 3$, is closely related to the linear dimension of the system (discussed in Section 4.2.4);

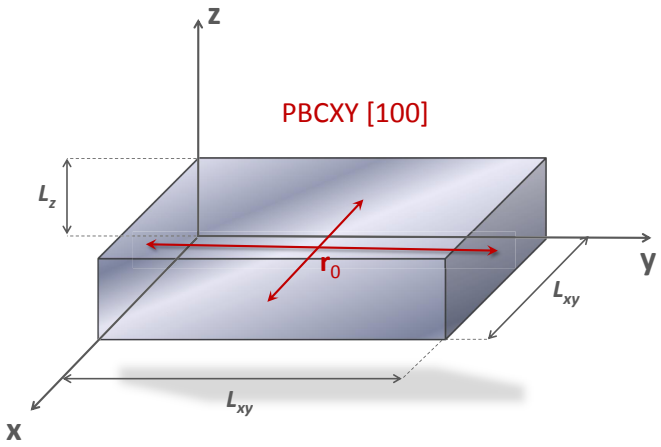

Figure 4.14: *PBCXY* [100] directions with $\mathbf{r}_0 \Rightarrow$ *Bulk Center* for isotropic, antiferromagnetic nanofilms.

3. Negative spin-wave excitation peaks originated from the time phase flip of local correlation function were observed in spectra for each wave vector.

In addition to the above three major observations, we also observed that, as the thickness L_z of nanofilms, i.e., the distance between the free surfaces, becomes larger, the main excitation peak for some wave vectors, e.g., $n_q = 1, 2, 3$, shifts closer to the ω_{PBC}. This observation is reasonable considering the free-surface effects become weaker as free surfaces increasingly get separated from each other, and thus the dynamics behaves more like that of the "infinite"system with periodic boundary conditions.

To complete our results, we give another set of spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$, obtained from isotropic, antiferromagnetic nanofilms with the same thickness $L_z = 10$ and three different horizontal dimensions, i.e., $L_{xy} = 10$, 20, and 30, as shown in Figure 4.16.

There have been similar observations made on the spectra in Figure 4.16 and Figure 4.15. It should be noted that,

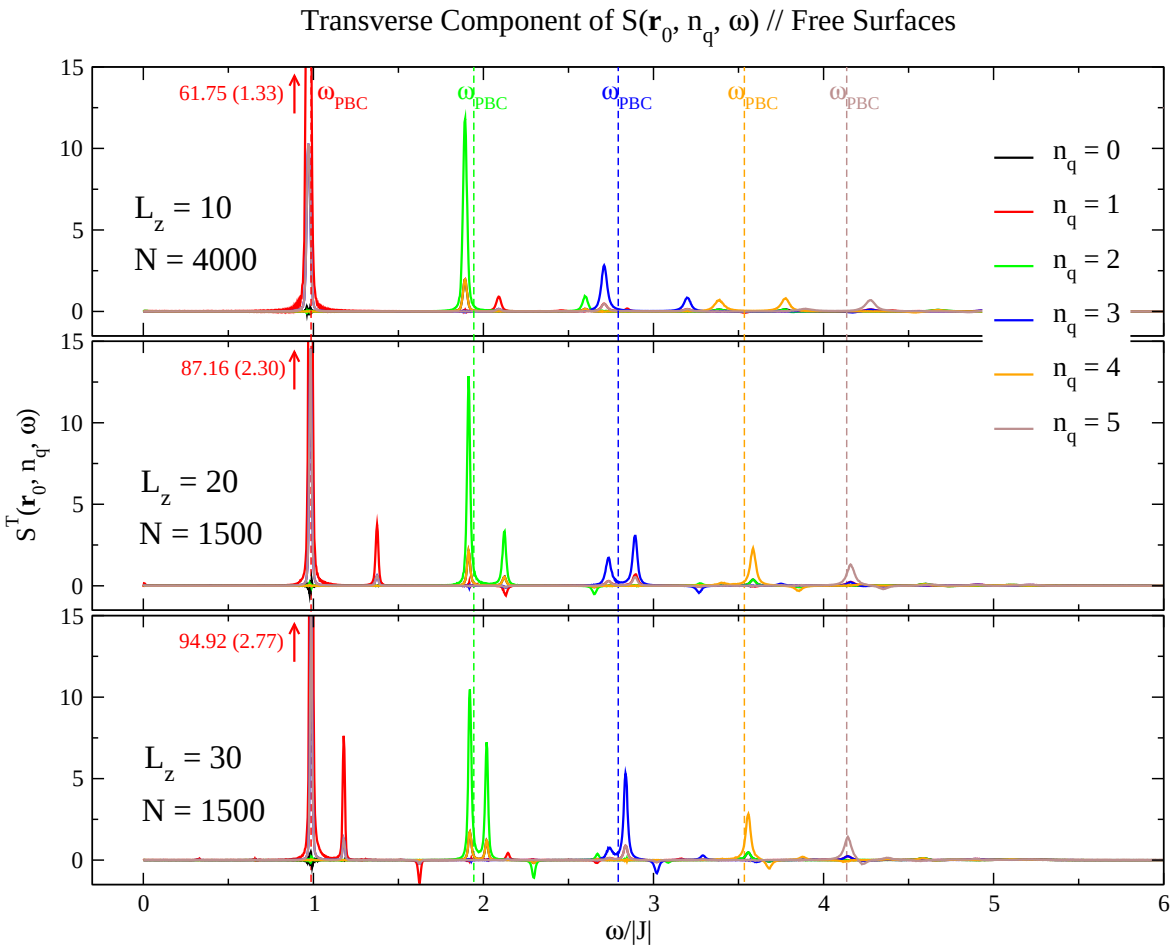

Figure 4.15: The spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ obtained from isotropic, antiferromagnetic nanofilms with the same $L_{xy} = 20$ and three different thicknesses, i.e., $L_z = 10$, 20, and 30. The results were obtained in the $PBCXY$ [100] directions, i.e., the directions parallel to the free surfaces, with $\mathbf{r}_0 \Rightarrow Bulk\ Center$ at $T = 0.4T_N$ with SD parameters of $n_t = 5000$, $t_c = 4000$, and $dt = 0.2/|J|$. We give the spectra for $n_q = 0, 1, 2, \ldots, 5$. N is the total number of initial configurations.

- The bigger oscillations of the spectra for $n_q = 1$ in Figure 4.16 for $L_{xy} = 20$ and 30 are due to the finite time cutoff t_c, which introduces oscillations into the results of the Fourier transformation. These oscillations, however, can be smoothed out by convoluting the local correlation function with a Gaussian resolution function $e^{-\frac{1}{2}t\delta_\omega}$ in the time Fourier transformation, where δ_ω is a parameter determining the resolution

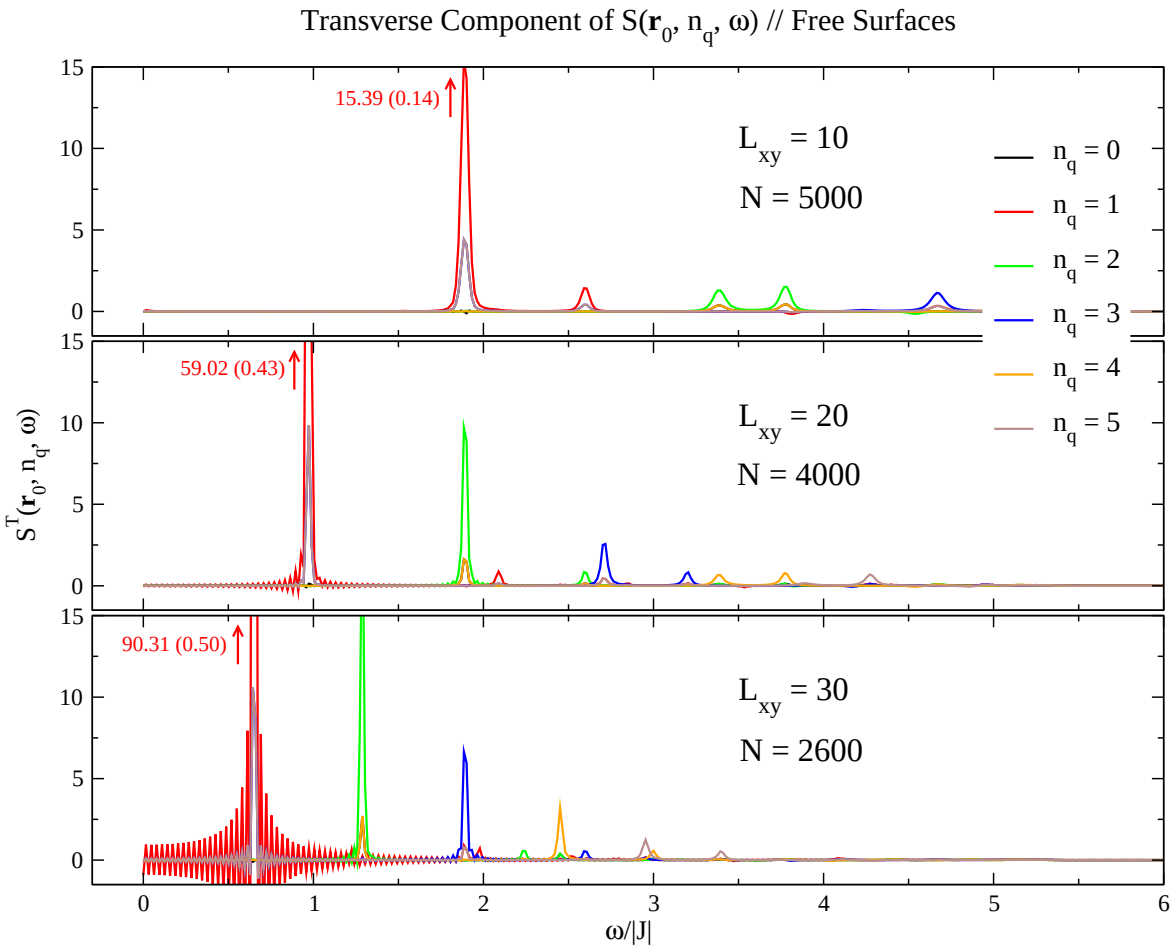

Figure 4.16: The spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ obtained from isotropic, antiferromagnetic nanofilms with the same $L_z = 10$ and three different horizontal dimensions, i.e., $L_{xy} = 10$, 20, and 30. The results were obtained in the *PBCXY* [100] directions, i.e., the directions parallel to the free surfaces, with $\mathbf{r}_0 \Rightarrow$ *Bulk Center* at $T = 0.4T_N$ with SD parameters of $n_t = 5000$, $t_c = 4000$, and $dt = 0.2/|J|$. We give the spectra for $n_q = 0, 1, 2, \ldots, 5$. N is the total number of initial configurations.

in frequency and needs to be chosen properly such that effects of the cutoff in the evolution time can be neglected [37];

- The shifting to lower energy frequency of the main excitation peak in Figure 4.16 is due to the finite-size effects in the xy-directions with periodic boundary conditions.

4.2.4 Isotropic Antiferromagnetic Nanofilms in *PBCXY* [100] Directions II: Systematic Examination of Size Effects on $\Delta\omega_{12}$

We investigated the relationship between $\Delta\omega_{12}$ and the linear dimension of the system, i.e., L_z and L_{xy}. Figure 4.17 shows $\Delta\omega_{12}$ as an exponentially decaying function of $L_z^{1/3}$ for $n_q = 1, 2, 3$, respectively; the inset shows $\Delta\omega_{12}$ for $n_q = 1, 2, 3$ with one lattice size of $L_{xy} = L_z = 20$. Figure 4.18 shows $\Delta\omega_{12}$ as a logarithmically increasing function of L_{xy} for $n_q = 1, 2, \ldots 5$; the inset shows $\Delta\omega_{12}$ for $n_q = 1, 2, 3$ with one lattice size of $L_z = 10$ and $L_{xy} = 20$.

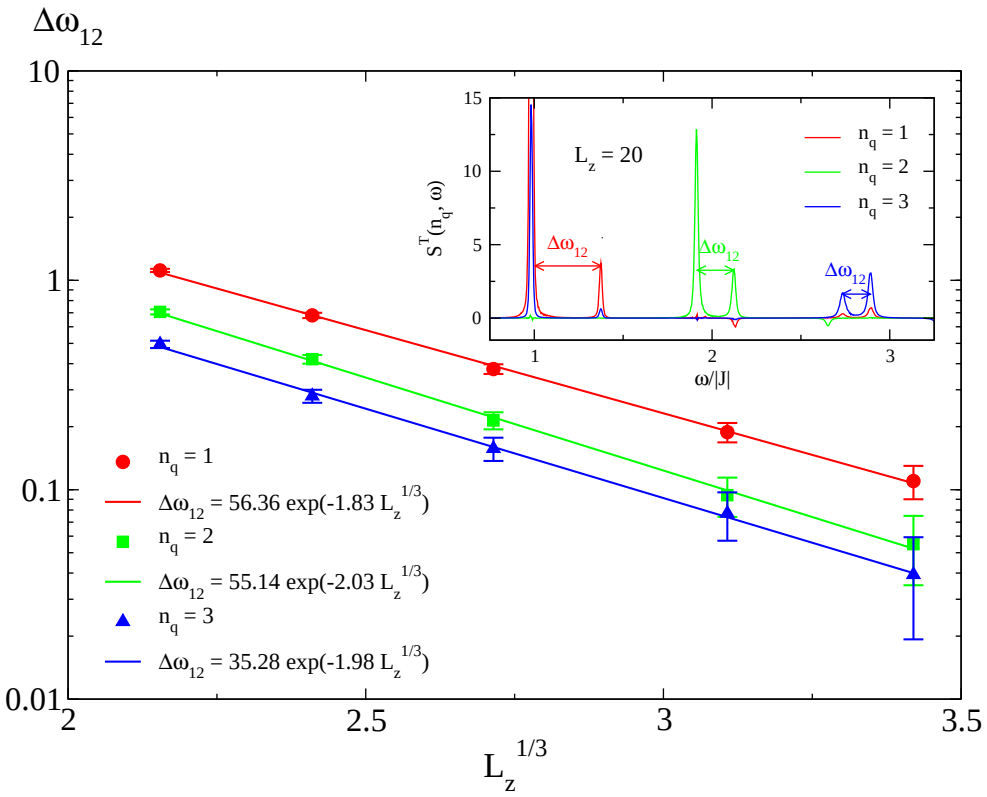

Figure 4.17: $\Delta\omega_{12}$ as an exponentially decaying function of $L_z^{1/3}$ with the same L_{xy}=20 for $n_q = 1, 2, 3$, respectively; the inset shows $\Delta\omega_{12}$ for $n_q = 1, 2, 3$ with one lattice size of $L_z = 20$.

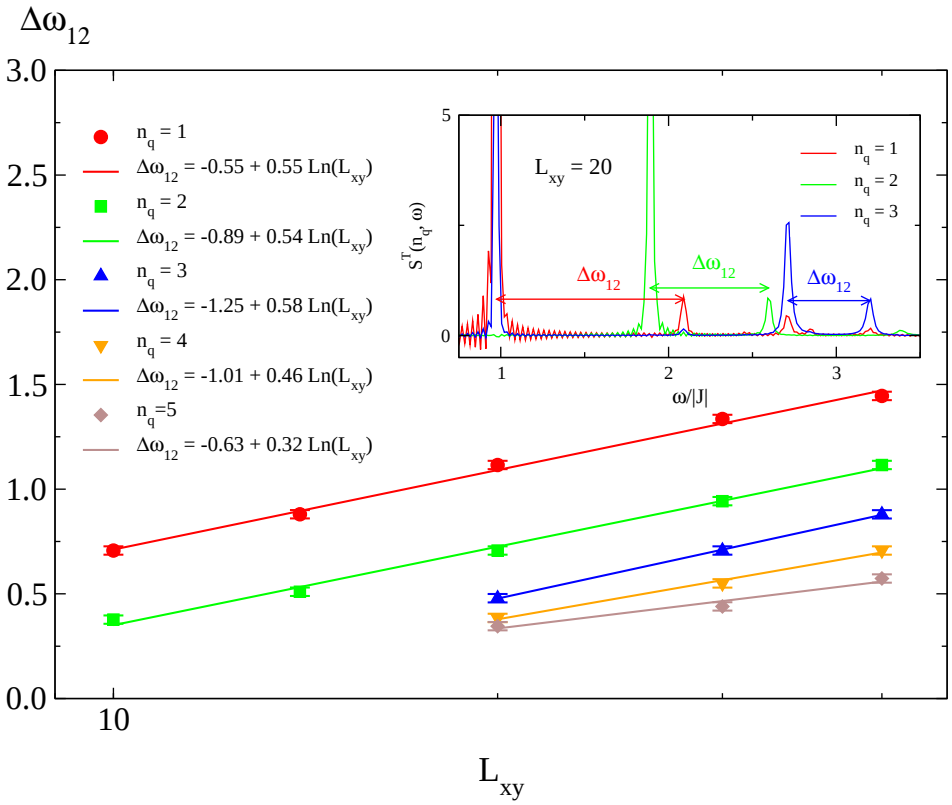

Figure 4.18: $\Delta\omega_{12}$ as a logarithmically increasing function of L_{xy} with the same $L_z = 10$ for $n_q = 1, 2, \ldots 5$; the inset shows $\Delta\omega_{12}$ for $n_q = 1, 2, 3$ with one lattice size of $L_{xy} = 20$.

4.2.5 Isotropic Antiferromagnetic Nanofilms in *PBCXY* [100] Directions III: Quantitative Explanation of Multiple Spin-Wave Excitations with the Assumption of Q-Space Spin-Wave Reflection

As discussed in Section 4.2.3, the most significant observation of the spectra for $S^T(\mathbf{r}_0, n_q, \omega)$ in isotropic, antiferromagnetic nanofilms is those multiple spin-wave excitation peaks with high resolution. To illustrate the high resolution of those multiple spin-wave peaks, Figure 4.19 shows the comparison between the magnitude of those multiple spin-wave peaks and the magnitude of the intrinsic noise in our simulations for $n_q = 1$ of the nanofilm with $L_{xy} = L_z = 20$. As shown in the figure, there is a significant difference in the magnitude, which makes it very difficult to draw a conclusion that those multiple spin-wave excitations

simply originate from noise fluctuations. There should be real physics superimposed on the much less intense noise background. Note that the noise is $\sim 10^{-4}$ as big as the single spin-wave peak.

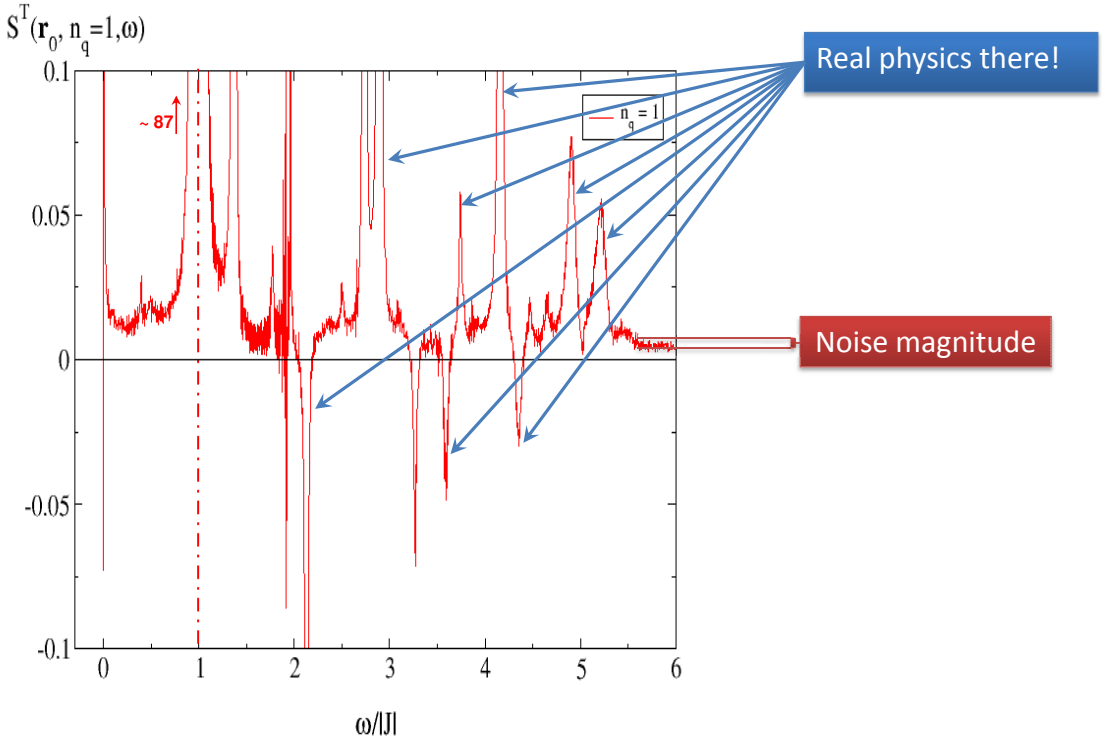

Figure 4.19: The high resolution of multiple spin-wave excitation peaks; the comparison between the magnitude of those multiple spin-wave peaks and the magnitude of the intrinsic noise in our simulations for $n_q = 1$ of the nanofilm with $L_{xy} = L_z = 20$. Note that the noise is $\sim 10^{-4}$ as big as the single spin-wave peak.

As mentioned in Chapter 1, for those small laterally confined magnetic systems like nanofilms or nanoparticles, there is intrinsic broken translational invariance caused by free-surface confinement effects in one or more directions, which leads to a broken conservation law of corresponding momentum for a spin wave [49]. The broken conservation law of momentum brings uncertainty into the wave vector for a specific spin-wave excitation energy. To explain those multiple spin-wave excitation peaks, we proposed the assumption of q-space spin-wave reflection with broken momentum conservation as follows,

In the linear dispersion region with small momentum q, the reflected spin-wave energy and momentum should satisfy a geometric relationship defined by

$$\frac{\omega_{refl}}{\omega_{bulk}} = \frac{q_{refl}}{q_{bulk}}, \tag{4.2}$$

where q_{bulk} and q_{refl} are the bulk momentum and reflected momentum, respectively; ω_{bulk} and ω_{refl} are bulk energy frequency and reflected energy frequency, respectively.

Figure 4.20 gives an illustration of this assumption.

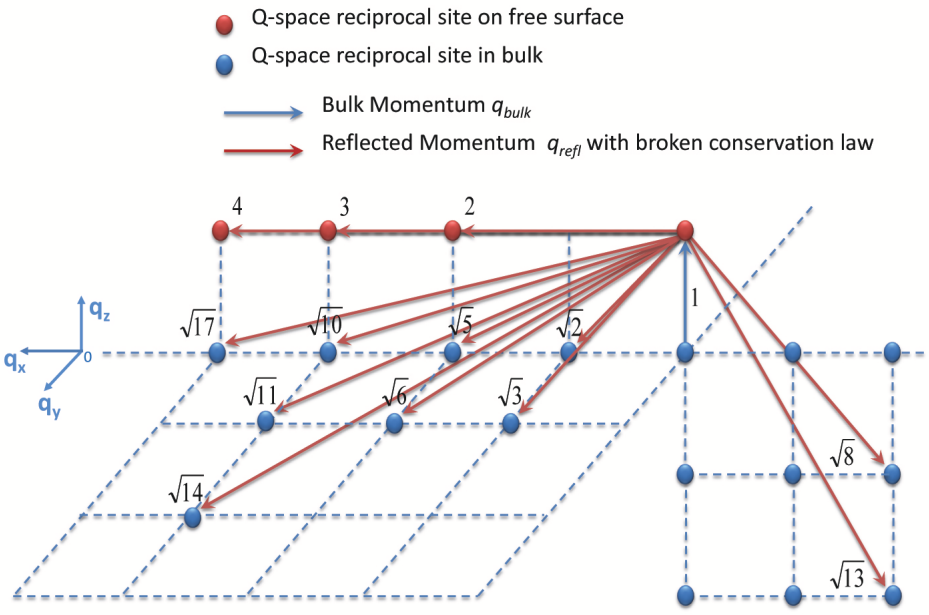

Figure 4.20: An illustration of the assumption of q-space spin-wave reflection with broken momentum conservation.

With the assumption of q-space spin-wave reflection with broken momentum conservation, we successfully explained those multiple spin-wave excitation spectra quantitatively in the linear dispersion region with small momentum q. Figure 4.21 gives the same spectra shown in Figure 4.19. The thick red dashed line gives the single spin-wave excitation location for the wave vector of $n_q = 1$ of the system with periodic boundary conditions; the thick

back dashed line labeled with $\omega = \omega_{bulk}$ gives the bulk excitation location for the wave vector of $n_q = 1$ of the nanofilm. To locate multiple spin-wave excitation locations quantitatively, we took the bulk energy frequency $\omega = \omega_{bulk}$ and then multiplied it with all possible ratios of $\frac{q_{refl}}{q_{bulk}}$, which are illustrated in Figure 4.20. The results of those multiplications are shown by thin black dashed lines with a ratio multiplying ω labeled on each.

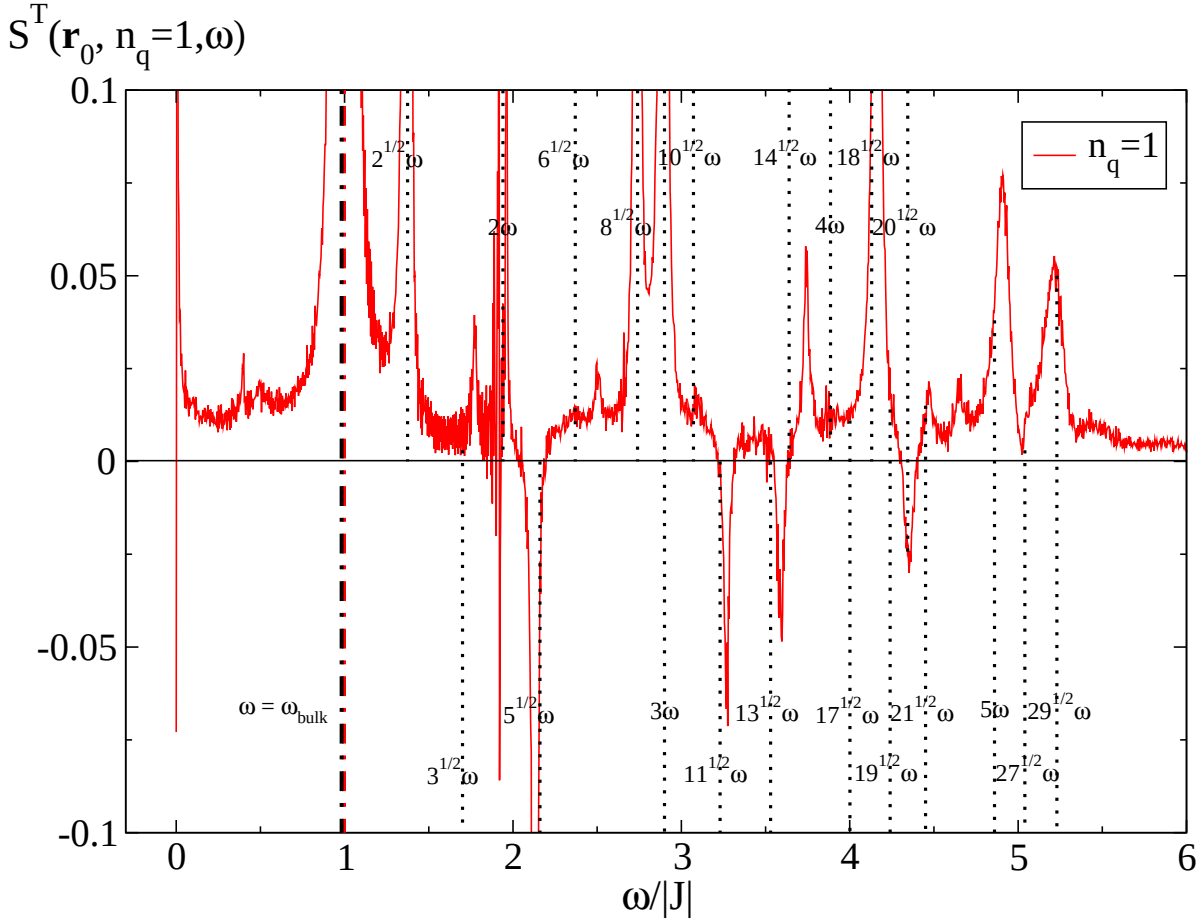

Figure 4.21: Determination of multiple spin-wave excitation locations with the assumption of q-space spin-wave reflection with broken momentum conservation in the linear dispersion region with small momentum of $n_q = 1$ for an isotropic, antiferromagnetic nanofilm with a lattice size of $L_{xy} = L_z = 20$. The results were obtained in the $PBCXY$ [100] directions, i.e., the directions parallel to the free surfaces, with $\mathbf{r}_0 \Rightarrow Bulk\ Center$ at $T = 0.4T_N$ with SD parameters of $n_t = 5000$, $t_c = 4000$, and $dt = 0.2/|J|$. The thick red dashed line gives the single spin-wave excitation location for the wave vector of $n_q = 1$ of the system with periodic boundary conditions; the thick back dashed line labeled with $\omega = \omega_{bulk}$ gives the bulk excitation location for the wave vector of $n_q = 1$ of the nanofilm.

Similarly, Figure 4.22 shows the determination of multiple spin-wave excitation locations quantitatively with $n_q = 2$.

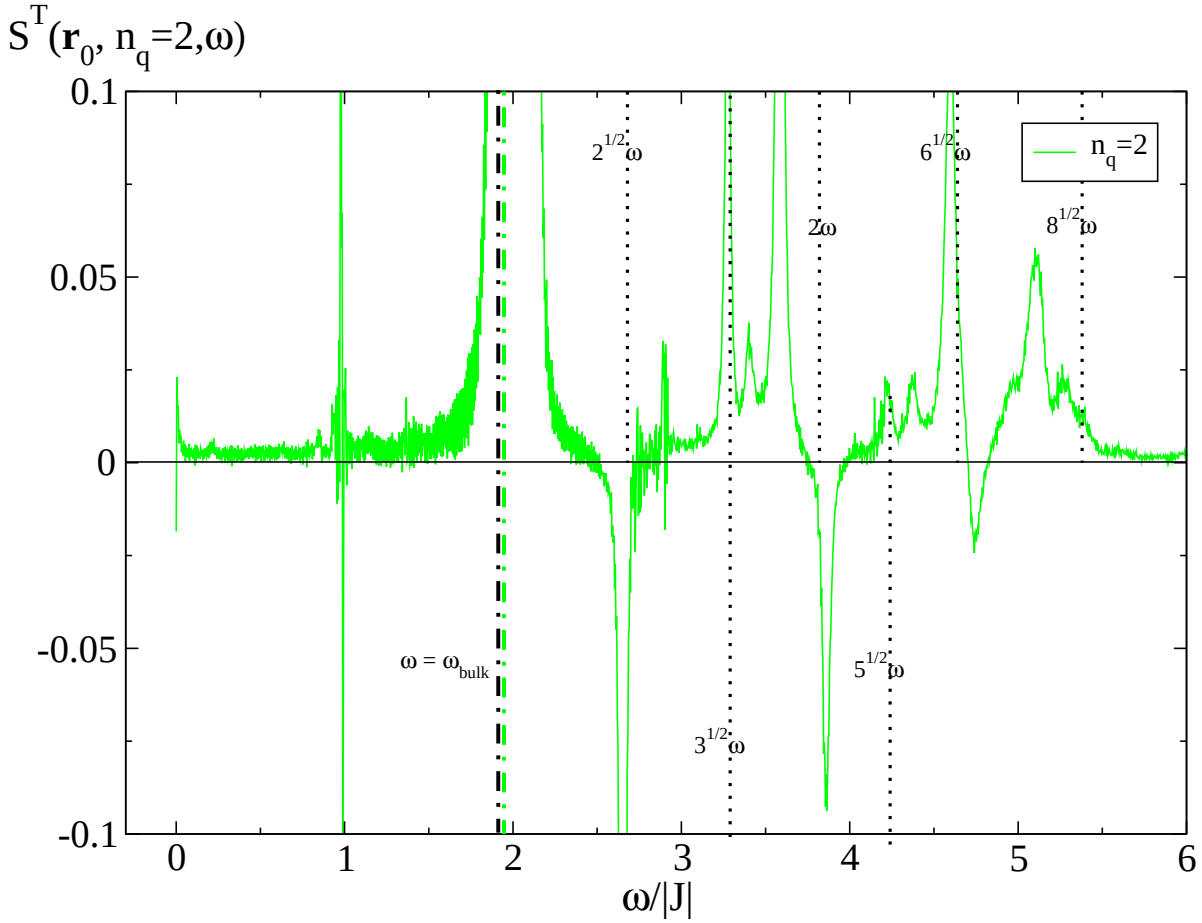

Figure 4.22: Determination of multiple spin-wave excitation locations with the assumption of q-space spin-wave reflection with broken momentum conservation in the linear dispersion region with small momentum of $n_q = 2$ for an isotropic, antiferromagnetic nanofilm with a lattice size of $L_{xy} = L_z = 20$. The results were obtained in the $PBCXY$ [100] directions, i.e., the directions parallel to the free surfaces, with $\mathbf{r}_0 \Rightarrow Bulk\ Center$ at $T = 0.4T_N$ with SD parameters of $n_t = 5000$, $t_c = 4000$, and $dt = 0.2/|J|$. The thick green dashed line gives the single spin-wave excitation location for the wave vector of $n_q = 2$ of the system with periodic boundary conditions; the thick back dashed line labeled with $\omega = \omega_{bulk}$ gives the bulk excitation location for the wave vector of $n_q = 2$ of the nanofilm.

Comparing the results in Figure 4.21 and Figure 4.22, we observed that, with our assumption of q-space spin-wave reflection, the proportion of successfully explained multiple excitations with $n_q = 1$ is larger than that with $n_q = 2$, which means our assumption works better

with $n_q = 1$ than $n_q = 2$, i.e., works better with a smaller momentum. This is observation is reasonable considering that the linear dispersion region with small momentum q is the region for the assumption to be correctly applied.

4.2.6 Isotropic Antiferromagnetic Nanofilms in *FBCZ* [100] Directions: spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$

In this section, we give the results for the transverse component of the local dynamic structure factor $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ with $\mathbf{r}_0 \Rightarrow$ *Bulk Center* for isotropic, antiferromagnetic nanofilms in the *FBCZ* [100] directions (discussed in Section 3.4.4), i.e., the directions perpendicular to the free surfaces, as shown in Figure 4.23.

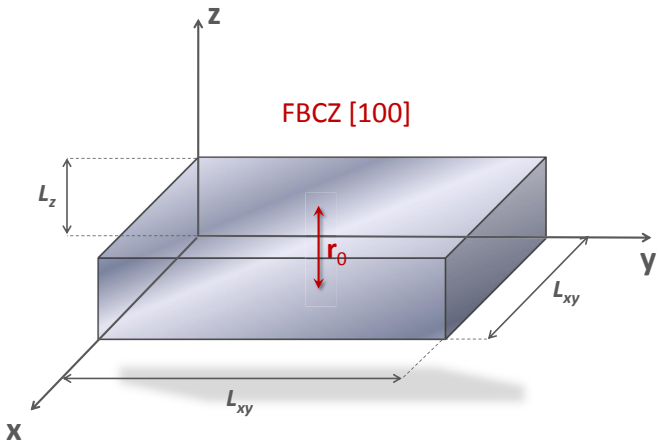

Figure 4.23: *FBCZ* [100] directions with $\mathbf{r}_0 \Rightarrow$ *Bulk Center* for isotropic, antiferromagnetic nanofilms.

We performed spin dynamics runs using the same simulation conditions and parameters as introduced in the beginning paragraphs of Section 4.2.3. Figure 4.24 shows the spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$, obtained from an isotropic, antiferromagnetic nanofilm with a lattice size of $L_{xy} = L_z = 20$. In the figure, we give the spectra for $S^T(\mathbf{r}_0, n_q, \omega)$ with $n_q = 0, 1, \ldots, 4$.

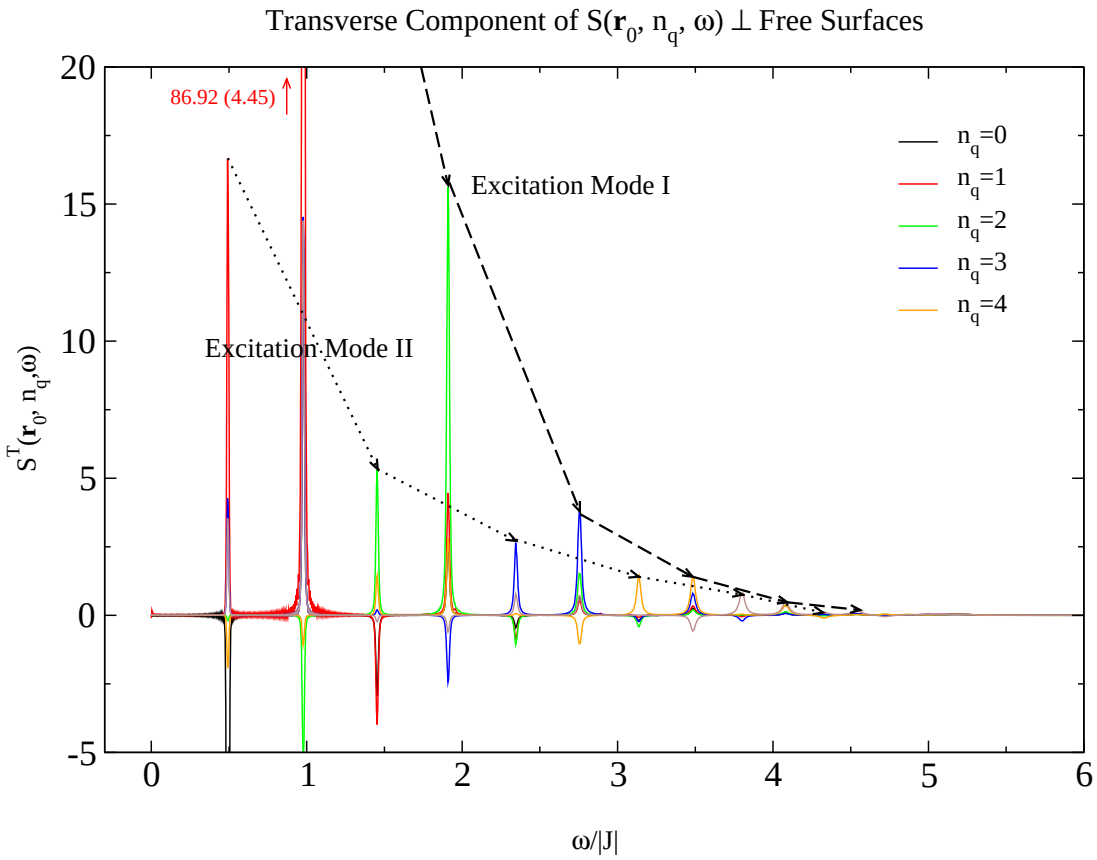

Figure 4.24: The spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ obtained from an isotropic, antiferromagnetic nanofilm with a lattice size of $L_{xy} = L_z = 20$. The results were obtained in the *FBCZ* [100] directions, i.e., the directions perpendicular to the free surfaces, with $\mathbf{r}_0 \Rightarrow$ *Bulk Center* at $T = 0.4T_N$ with SD parameters of $n_t = 5000$, $t_c = 4000$, and $dt = 0.2/|J|$. We give the spectra for $n_q = 0, 1, \ldots, 4$.

As shown in the figure, we observed two novel quantized spin-wave excitation modes for $S^T(\mathbf{r}_0, n_q, \omega)$, i.e., "Excitation Mode I"and "Excitation Mode II", in the spatial z-direction in isotropic, antiferromagnetic nanofilms. This is a new form of spin-wave excitation behavior which needs further study, but at least our results indicate that those novel quantized excitation modes could be potentially caused by, but not limited to the free-surface confinement effects.

4.2.7 Isotropic Antiferromagnetic Nanoparticles I: spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$

In this section, following the results we obtained from isotropic, antiferromagnetic nanofilms, we give the results for the transverse component of the local dynamic structure factor $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ with $\mathbf{r}_0 \Rightarrow$ *Bulk Center* for isotropic, antiferromagnetic nanoparticles on $L \times L \times L$ simple cubic lattice. The results, discussed in Section 3.4.3, were obtained in six symmetric [100] directions, as shown in Figure 4.25.

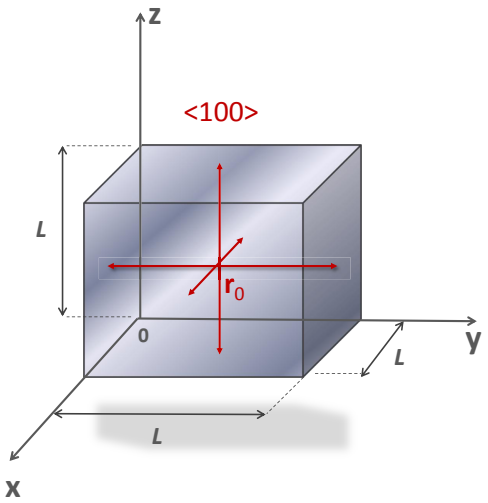

Figure 4.25: [100] directions with $\mathbf{r}_0 \Rightarrow$ *Bulk Center* for isotropic, antiferromagnetic nanoparticles.

Our simulations were conducted at the same temperature $T = 0.4T_N$ we used for the simulations on nanofilms. We discarded 10^4 hybrid MC steps to approach the first equilibrium spin configuration and used 10^4 hybrid MC steps to generate each of the rest of $N - 1$ equilibrium spin configurations drawn from a canonical ensemble at $T = 0.4T_N$. The total number of SD time steps in our simulations was $n_t = 5000$; the SD time cutoff was $t_c = 4000$; the SD time step was $dt = 0.2/|J|$.

Figure 4.26 shows the spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$, obtained from isotropic, antiferromagnetic nanoparticles with $L = 10$, 14, and 20. For the convenience of labeling, we labeled the y-axis

of our results with $S^T(\mathbf{r}_0, n_q, \omega)$ with $n_q = 0, 1, 2, \ldots$ (defined in Eqn.(3.36) and Eqn.(3.37)). In the figure, we give the spectra for $S^T(\mathbf{r}_0, n_q, \omega)$ with $n_q = 1, 2, \ldots, 5$.

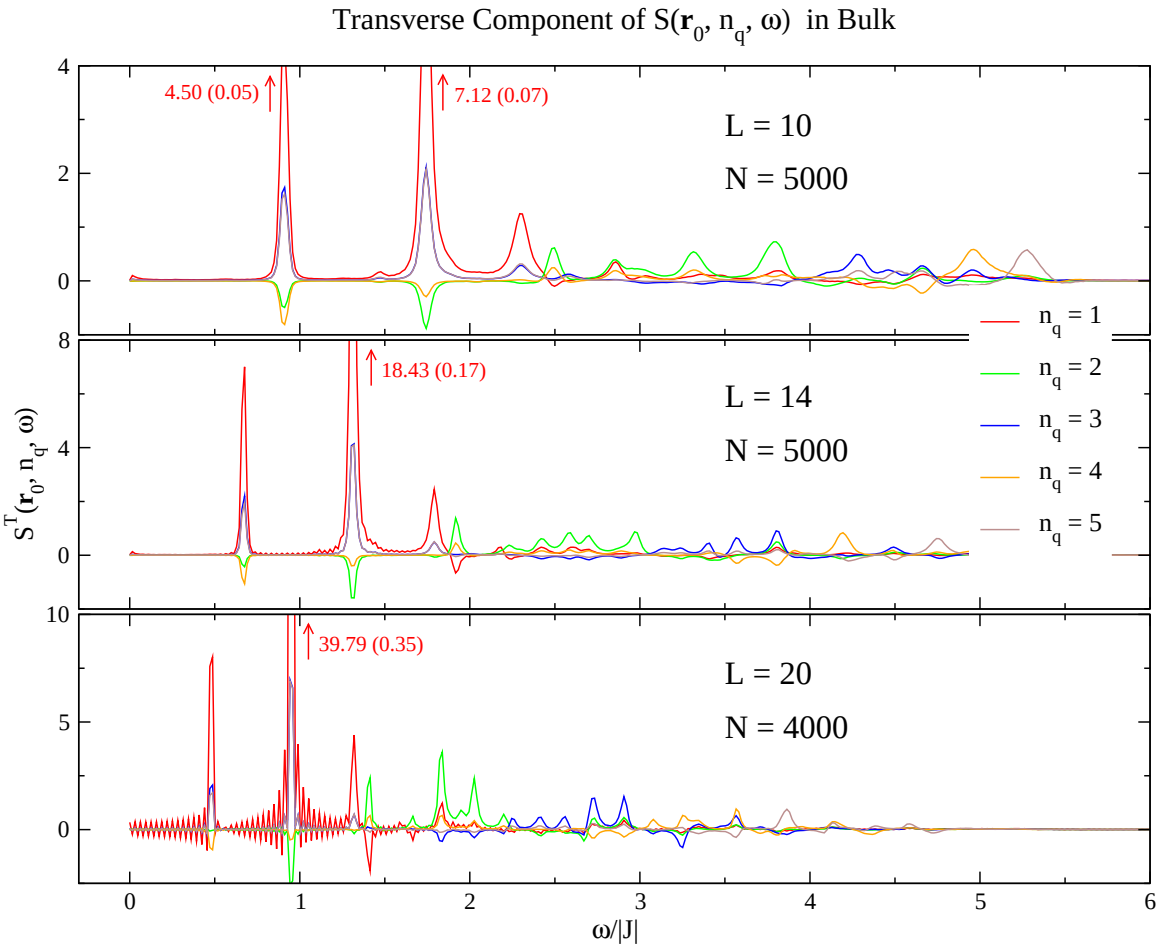

Figure 4.26: The spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ obtained from isotropic, antiferromagnetic nanoparticles with $L = 10$, 14, and 20. The results were obtained in the [100] directions with $\mathbf{r}_0 \Rightarrow$ *Bulk Center* at $T = 0.4T_N$ with SD parameters of $n_t = 5000$, $t_c = 4000$, and $dt = 0.2/|J|$. We give the spectra for $n_q = 1, 2, \ldots, 5$.

Three major observations have been made pertinent to the above spectra for isotropic, antiferromagnetic nanoparticles:

1. Multiple excitation peaks for wave vectors within the first Brillouin zone appear in the spin-wave spectra for the transverse component of the local dynamic structure factor $S^T(\mathbf{r}_0, n_q, \omega)$ in the classical Heisenberg isotropic antiferromagnetic nanoparticles, which are lacking if periodic boundary conditions are used;

2. The width of the energy gap between the main peak and the secondary peak with a higher energy frequency for wave vectors, e.g., $n_q = 1$, is closely related to the linear dimension of the system;

3. A secondary excitation peak with approximately half the energy of the main peak appears for all wave vectors, which is lacking in the spectra for nanofilms.

4. Negative spin-wave excitation peaks were observed in spectra for each wave vector.

As noted previously in Section 4.2.3, the bigger oscillations of the spectra for $n_q = 1$ in Figure 4.26 for $L = 20$ are due to the finite time cutoff t_c, which introduces oscillations can be smoothed out by convoluting the local correlation function with a Gaussian resolution function in the time Fourier transformation; and the shifting to lower energy frequency of the main excitation peak in the figure is due to the finite-size effects.

4.2.8 Isotropic Antiferromagnetic Nanoparticles II: Quantitative Explanation of Multiple Spin-Wave Excitations with the Assumption of Q-Space Spin-Wave Reflection

As shown in Figure 4.26, the spectra for isotropic, antiferromagnetic nanoparticles are even more complicated than the spectra for isotropic, antiferromagnetic nanofilms given in Section 4.2.3. Not only are there many more multiple spin-wave excitations for each wave vector, but also the excitation patterns themselves become more intricate. Those observations are due to the fact that the completely laterally confined nanoparticles have much stronger free-surface effects on dynamics than those of nanofilms. However, in the linear dispersion region with the assumption of q-space spin-wave reflection, we can still determine the locations of those excitations quantitatively.

Figure 4.27 gives the spectra for $n_q = 1$ of the nanoparticle with a lattice size of $L = 10$. The thick red dashed line gives the single spin-wave excitation location for the wave vector of $n_q = 1$ of the system with periodic boundary conditions; the thick back dashed line

labeled with $\omega = \omega_{bulk}$ gives the bulk excitation location for the wave vector of $n_q = 1$ of the nanoparticle. The determination of multiple spin-wave excitation locations are shown by thin black dashed lines with a ratio multiplying ω labeled on each.

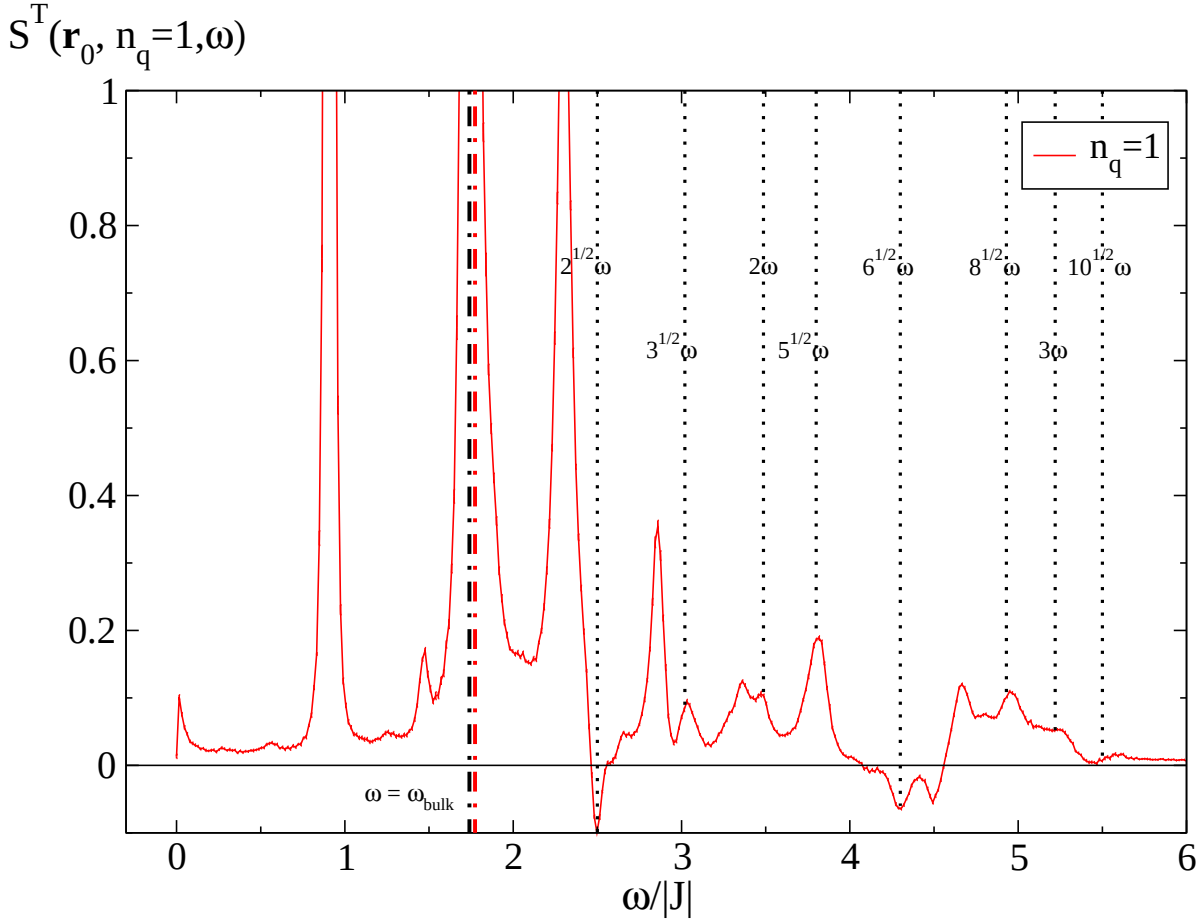

Figure 4.27: Determination of multiple spin-wave excitation locations with the assumption of q-space spin-wave reflection with broken momentum conservation in the linear dispersion region with small momentum of $n_q = 1$ for an isotropic, antiferromagnetic nanoparticle with a lattice size of $L = 10$. The results were obtained in the [100] directions with $\mathbf{r}_0 \Rightarrow Bulk\ Center$ at $T = 0.4T_N$ with SD parameters of $n_t = 5000$, $t_c = 4000$, and $dt = 0.2/|J|$. The thick red dashed line gives the single spin-wave excitation location for the wave vector of $n_q = 1$ of the system with periodic boundary conditions; the thick back dashed line labeled with $\omega = \omega_{bulk}$ gives the bulk excitation location for the wave vector of $n_q = 1$ of the nanoparticle.

4.2.9 Anisotropic Antiferromagnetic Nanoparticles: spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$

To complete our results, in Figure 4.28 we show the data of the transverse component of the local dynamic structure factor $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ with $\mathbf{r}_0 \Rightarrow$ *Bulk Center* for anisotropic, antiferromagnetic nanoparticles with $L = 15$, 21, and 29; Figure 4.29 and Figure 4.30 show the data of $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ with $\mathbf{r}_0 \Rightarrow Surface\ Center$ with $L = 15$ in *within-surface* and *off-surface* directions, respectively; Figure 4.31 shows the data of $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ with $\mathbf{r}_0 \Rightarrow$ *Lattice Corner* with $L = 15$.

We performed spin dynamics simulations at the temperature $T = 0.4T_N$. We discarded 5×10^3 hybrid MC steps to approach the first equilibrium spin configuration and used 5×10^3 hybrid MC steps to generate each of the rest of $N-1$ equilibrium spin configurations drawn from a canonical ensemble at $T = 0.4T_N$. The total number of SD time steps in our simulations was $n_t = 2000$; the SD time cutoff was $t_c = 1000$; the SD time step was $dt = 0.2/|J|$.

Compared to the spectra shown in Figure 4.26, the spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ for anisotropic, antiferromagnetic nanoparticles have higher spin-wave excitation energies than those for isotropic ones. As $\mathbf{r}_0 \Rightarrow$ *Bulk Center*, the main excitation peak can be easily located and compared to the single spin-wave excitation locations for each wave vector of the system with periodic boundary conditions. As $\mathbf{r}_0 \Rightarrow Surface\ Center$, the excitation patterns in the *within-surface* directions behave more like those of single spin-wave excitations than in the *off-surface* directions. For the spectra in the *off-surface* directions with $\mathbf{r}_0 \Rightarrow Surface\ Center$ and the spectra with $\mathbf{r}_0 \Rightarrow$ *Lattice Corner*, however, further study is needed to determine all excitation locations.

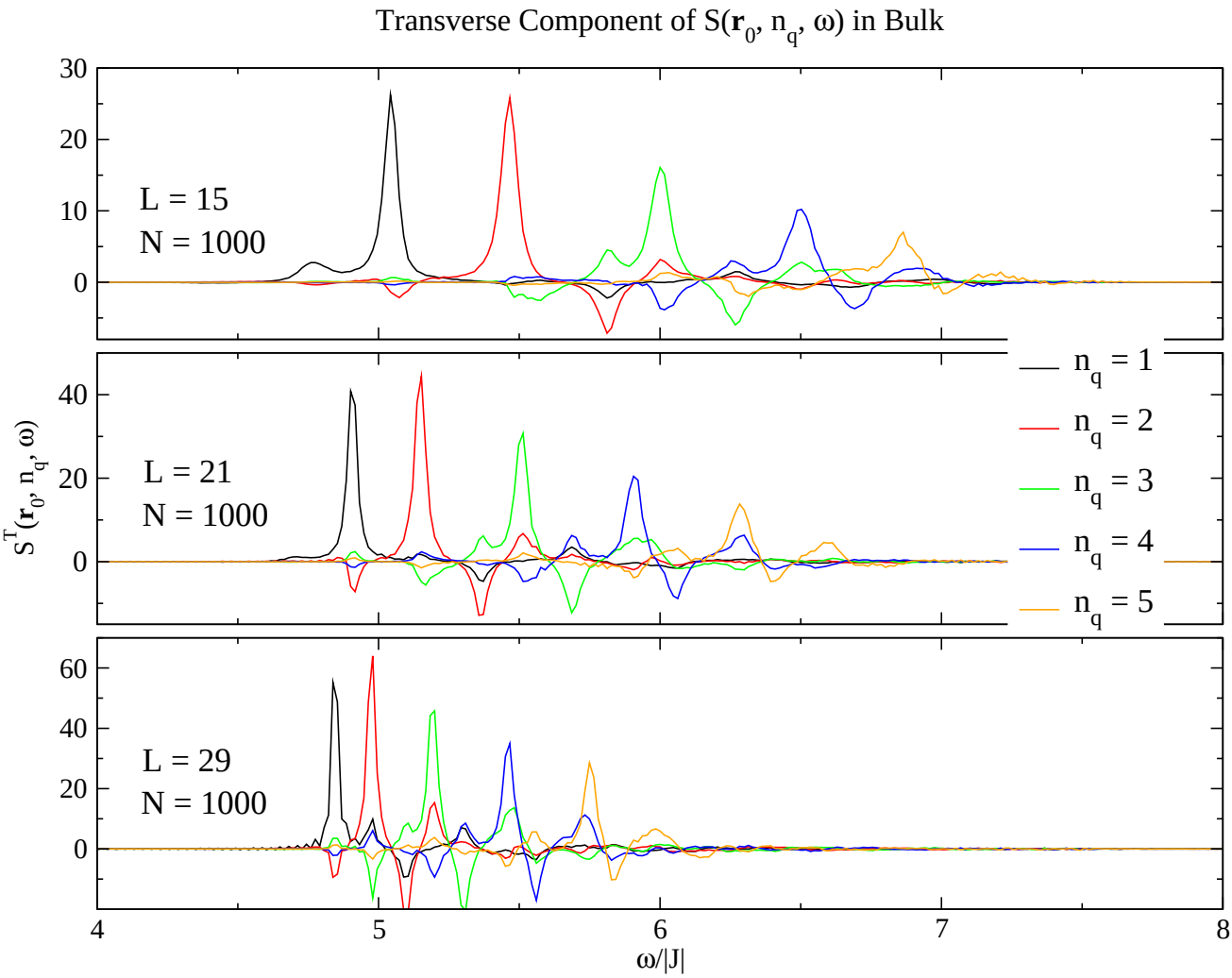

Figure 4.28: The spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ obtained from anisotropic, antiferromagnetic nanoparticles with $L = 15$, 21, and 29. The results were obtained in the [100] directions with $\mathbf{r}_0 \Rightarrow$ *Bulk Center* at $T = 0.4T_N$ with SD parameters of $n_t = 2000$, $t_c = 1000$, and $dt = 0.2/|J|$. We give the spectra for $n_q = 1, 2, \ldots, 5$. N is the total number of initial configurations.

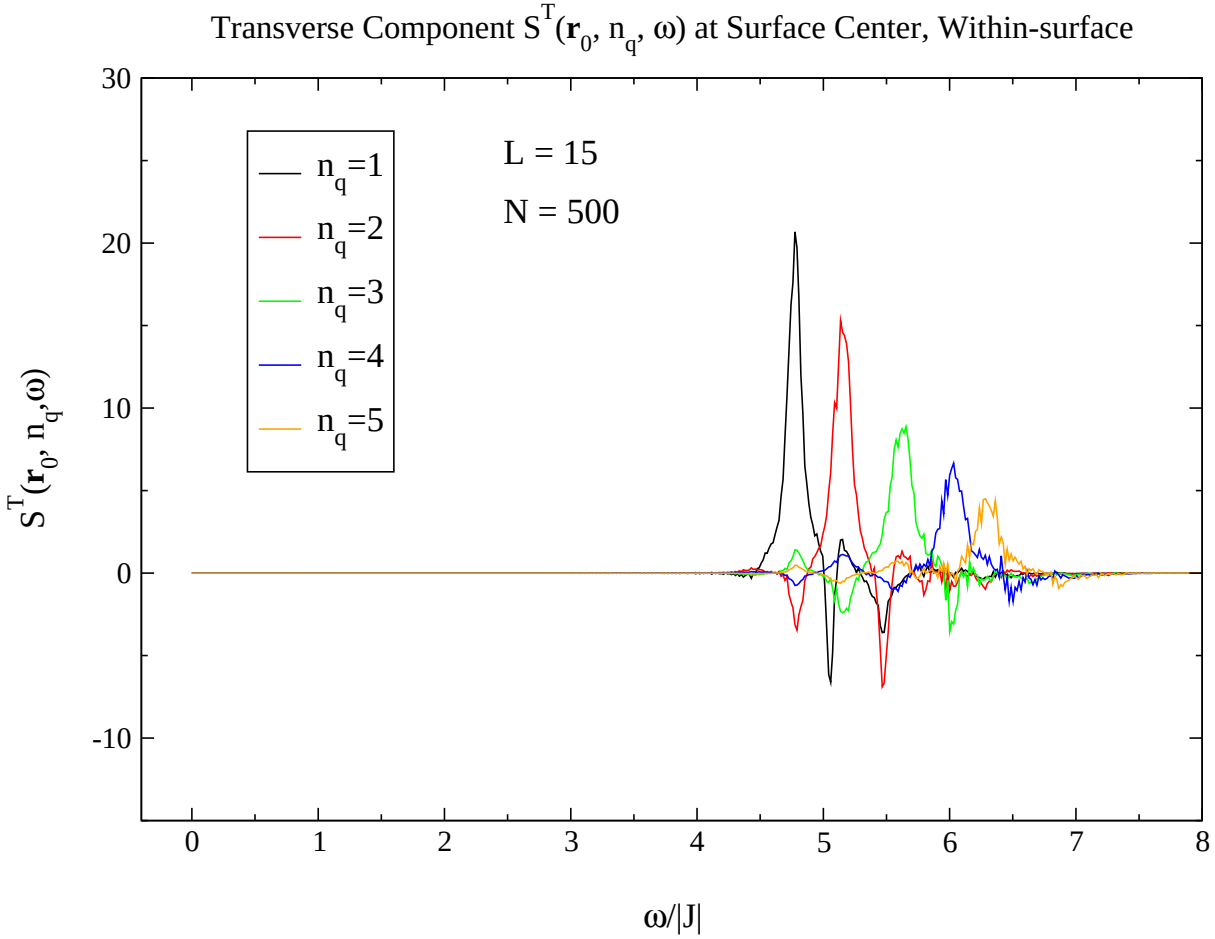

Figure 4.29: The spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ obtained from an anisotropic, antiferromagnetic nanoparticle with $L = 15$. The results were obtained in the *within-surface* directions with $\mathbf{r}_0 \Rightarrow Surface\ Center$ at $T = 0.4T_N$ with SD parameters of $n_t = 2000$, $t_c = 1000$, and $dt = 0.2/|J|$. We give the spectra for $n_q = 1, 2, \ldots, 5$. N is the total number of initial configurations.

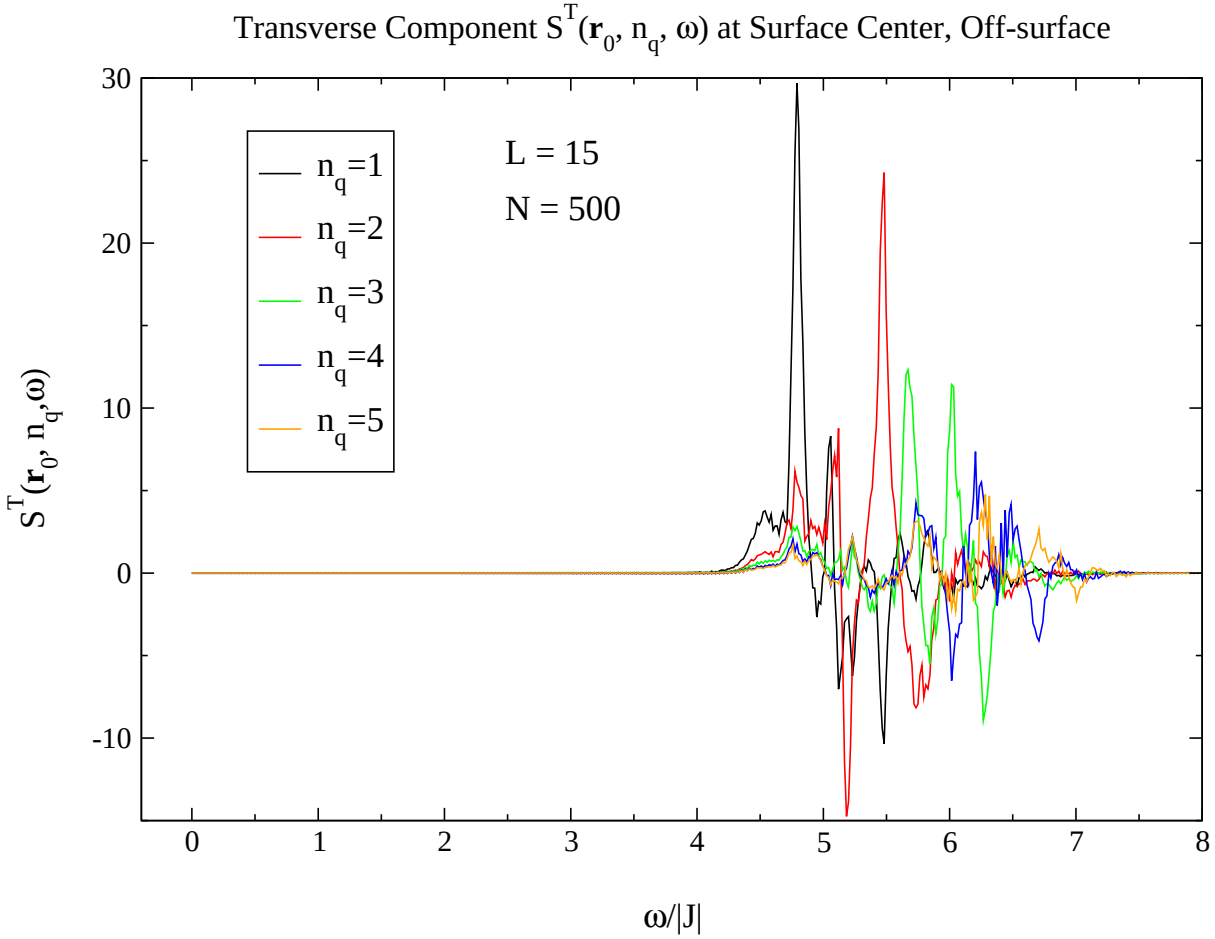

Figure 4.30: The spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ obtained from an anisotropic, antiferromagnetic nanoparticle with $L = 15$. The results were obtained in the *off-surface* directions with $\mathbf{r}_0 \Rightarrow Surface\ Center$ at $T = 0.4T_N$ with SD parameters of $n_t = 2000$, $t_c = 1000$, and $dt = 0.2/|J|$. We give the spectra for $n_q = 1, 2, \ldots, 5$. N is the total number of initial configurations.

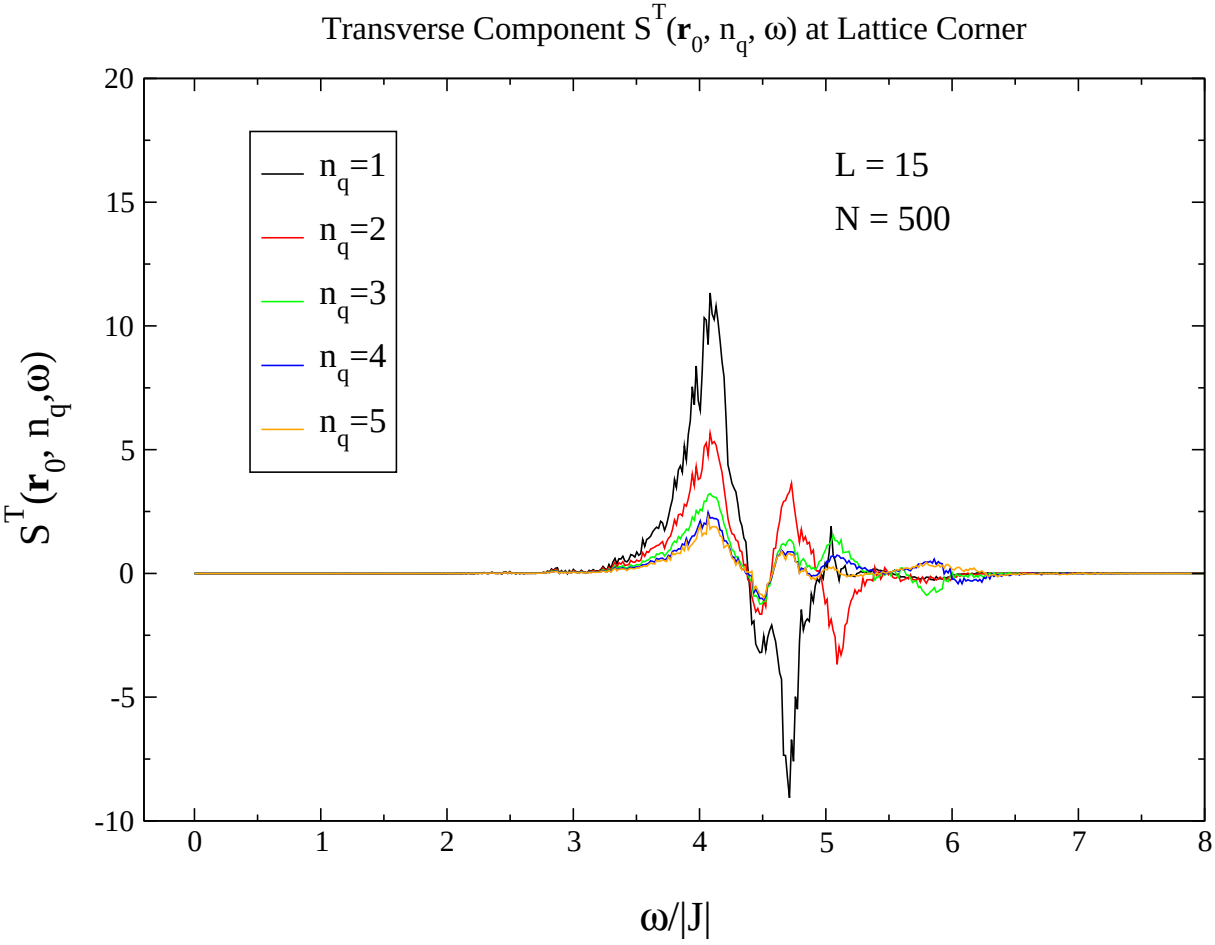

Figure 4.31: The spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ obtained from an anisotropic, antiferromagnetic nanoparticle with $L = 15$. The results were obtained in the [100] directions with $\mathbf{r}_0 \Rightarrow$ *Lattice Corner* at $T = 0.4T_N$ with SD parameters of $n_t = 2000$, $t_c = 1000$, and $dt = 0.2/|J|$. We give the spectra for $n_q = 1, 2, \ldots, 5$. N is the total number of initial configurations.

Chapter 5

Conclusion

With large scale Monte Carlo and spin dynamics simulations, we have investigated the dynamic behavior of antiferromagnetic nanostructures on a simple cubic lattice geometry, using both an isotropic and an anisotropic classical Heisenberg model of classical spins with unit length and with the nearest-neighbor exchange interactions. Nanoparticles are modeled with completely free boundary conditions, and nanofilms are modeled with partially free boundary conditions i.e., two free-surfaces in the spatial z-direction and periodic boundaries parallel to the surfaces in the x-,y-directions. Hybrid Monte Carlo methods are used to obtain the static properties of modeled nanostructures. The Monte Carlo methods are also used to generate equilibrium spin configurations as initial states of the coupled differential equations of motion. A fast spin dynamics algorithm based on the fourth-order Suzuki-Trotter decomposition of exponential operators has been applied to integrate the equations of motion. Our spin dynamics simulations are performed at a low temperature $T = 0.4T_N$. The integrations are carried to $t_c = 4000$ with an SD time step $dt = 0.2/|J|$.

With the time evolution of the spin configurations, the local space- and time-displaced spin-spin correlation function $C(\mathbf{r}_0, \mathbf{r}, t)$ is calculated, where $\mathbf{r}_0$ denotes the starting point from which the correlation function is calculated and can be chosen to be fixed at the bulk center or the surface center of nanoparticles and nanofilms, or the lattice corner of nanoparticles in the simulations. The local dynamic structure factor $S(\mathbf{r}_0, \mathbf{q}, \omega)$ is the Fourier transformation of $C(\mathbf{r}_0, \mathbf{r}, t)$, which can be observed in inelastic magnetic neutron scattering. For the temperature $T = 0.4T_N$, compared to the single spin-wave excitation spectra for the

"infinite"system with fully periodic boundary conditions, much more complicated excitation spectra for the transverse component of the local dynamic structure factor $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ appear in the nanoscale classical Heisenberg antiferromagnets. The spectra for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ have multiple excitation peaks for wave vectors within the first Brillouin zone, which are lacking if periodic boundary conditions are used. We were able to simulate these systems with sufficiently high accuracy such that multiple excitation peaks distinguish themselves from the intrinsic simulation noise by showing a significant difference in magnitude between the two signals. A systematic examination of size effects on $\Delta\omega_{12}$ with isotropic, antiferromagnetic nanofilms has shown that, for small wave vectors the width of the energy gap between the main peak and the secondary peak with a higher excitation energy behaves as an exponentially decaying function of $L_z^{1/3}$ and a logarithmically increasing function of L_{xy}. With the assumption of q-space spin-wave reflection with broken momentum conservation due to lateral free-surface confinements, we successfully explained the locations of those excitations quantitatively for isotropic, antiferromagnetic nanostructures in the linear dispersion region with small wave vectors.

Moreover, we have also observed two novel quantized spin-wave excitation modes for $S^T(\mathbf{r}_0, \mathbf{q}, \omega)$ in the spatial z-direction of isotropic, antiferromagnetic nanofilms , which is a novel spin-wave excitation behavior needs further study.

Results of this study indicate the presence of new forms of spin-wave excitation behavior which have yet to be observed experimentally but could be directly tested through neutron scattering experiments on nanoscale $RbMnF_3$ films or particles.

Bibliography

[1] P. Curie, J. Phys. **4**, 197, 263 (1895).

[2] P. Weiss, J. Phys. Radium **3**, 194 (1904).

[3] P. Weiss, J. Phys. Radium **6**, 661 (1907).

[4] C. Kittel, Introduction to Solid State Physics, 7th ed., John Wiley & Sons Inc. (1996).

[5] M. E. Fisher, Rep. Prog. Phys. **30**, 615 (1967).

[6] E. Ising, Z. Phys. **31**, 253 (1925).

[7] W. Heisenberg, Z. Phys. **49**, 619 (1928).

[8] L. D. Landau, Phys. Z. Sowjun **11**, 26, 545 (1937).

[9] L. Onsager, Phys. Rev. **65**, 117 (1944).

[10] D. W. Heermann, Computer Simulation Methods in Theoretical Physics, 2nd ed., Springer-Verlag, Berlin (1990).

[11] K. Binder and D. W. Heermann, Monte Carlo Simulation in Statistical Physics, 2nd corrected ed., Springer-Verlag, Berlin (1988).

[12] D. P. Landau, Phys. Rev. B **16**, 4164 (1977).

[13] D. P. Landau and K. Binder, Phys. Rev. B **17**, 2328 (1978).

[14] D. P. Landau, Phys. Rev. B **27**, 5604 (1983).

[15] S. Wansleben and D. P. Landau, Phys. Rev. B **43**, 6006 (1991).

[16] P. Peczak, A. M. Ferrenberg and D. P. Landau, Phys. Rev. B **43**, 6087 (1991).

[17] A. M. Ferrenberg and D. P. Landau, Phys. Rev. B **44**, 5081 (1991).

[18] K. Chen, A. M. Ferrenberg and D. P. Landau, Phys. Rev. B **48**, 3249 (1993).

[19] A. Bunker, B. D. Gaulin and C. Kallin, Phys. Rev. B **48**, 15861 (1993).

[20] C. Holm and W. Janke, Phys. Lett. A **173**, 8 (1993).

[21] L. Hua and J. W. Tucker, J. Mag. and Mag. Mater. **140-144**, 1509 (1995).

[22] R. G. Brown and M. Ciftan, Phys. Rev. Lett. **76**, 1352 (1996).

[23] F. Bloch, Z. Phys. **61**, 201 (1930).

[24] B. N. Brockhouse, Bull. Amer. Phys. Soc. **5**, 462 (no. 47) (1960).

[25] F. J. Dyson, Phys. Rev. **102**, 1217 (1956).

[26] F. J. Dyson, Phys. Rev. **102**, 1230 (1956).

[27] B. I. Halperin and P. C. Hohenberg, Phys. Rev. **188**, 898 (1969).

[28] G. F. Mazenko, M. J. Nolan and R. Freedman, Phys. Rev. B **18**, 2281 (1978).

[29] R. Freedman and G. F. Mazenko, Phys. Rev. Lett. **34**, 1575 (1975); Phys. Rev. B **13**, 4967 (1976).

[30] G. Placzek and L. Van Hove, Phys. Rev. **93**, 1207 (1954).

[31] L. D. Landau and E. M. Lifshitz, Statistical Physics, Part I, 3rd ed. (Course of Theoretical Physics, Vol 5), Butterworth-Heinemann, Oxford (1980).

[32] B. I. Halperin and P. C. Hohenberg, Phys. Rev. **177**, 952 (1969).

[33] P. C. Hohenberg and B. I. Halperin, Rev. Mod. Phys. **49**, 435 (1977).

[34] A. Tucciarone, H.Y. Lau, L.M. Corliss, A. Delapalme, and J.M. Hastings, Phys. Rev. B **4**, 3206 (1971).

[35] U.J. Cox, R.A. Cowley, S. Bates, and L.D. Cussen, J. Phys. Condens. Matter **1**, 3031 (1989).

[36] R. Coldea, R.A. Cowley, T.G. Perring, D.F. McMorrow, and B.Roessli, Phys. Rev. B **57**, 5281 (1998).

[37] K. Chen and D.P. Landau, Phys. Rev. B **49**, 3266 (1994).

[38] A. Bunker, K. Chen, and D.P. Landau, Phys. Rev. B **54**, 9259 (1996).

[39] S.-H. Tsai, A. Bunker, and D.P. Landau, Phys. Rev. B **61**, 333 (2000).

[40] S.-H. Tsai, A. Bunker, and D.P. Landau, Comput. Phys. Commum. **147**, Issues 1-2, Pages 97-100 (2002).

[41] S.-H. Tsai and D.P. Landau, Phys. Rev. B **67**, 104411 (2003).

[42] X. Tao, D.P. Landau, T.C. Schulthess, G.M. Stocks, Phys. Rev. Lett. **95**, 087207 (2005).

[43] A. Cuccoli, S. W. Lovesey and V. Tognetti, J. Phys.: Condens. Matter **6**, 7553 (1994).

[44] A. Bunker and D.P. Landau, Phys. Rev. Lett. **85**, 2601 (2000).

[45] B. Hillebrands and K. Ounadjela, in Spin Wave Confinement, S. O. Demokritov, Editor, Pan Stanford Publishing, Singapore (2009).

[46] C. Mathieu, J. Jorzick, A. Frank, S. O. Demokritov, B. Hillebrands, A.N. Slavin, B. Bartenlian, C. Chappert, D. Decanini, F. Rousseaux, and E. Cambril, Phys. Rev. Lett. **81**, 3968 (1998).

[47] J. Jorzick, C. Krämer, S. O. Demokritov, B. Hillebrands, B. Bartenlian, C. Chappert, D. Decanini, F. Rousseaux, E. Cambril, E. Søndergard, M. Bailleul, C. Fermon, and A.N. Slavin, J. Appl. Phys. **89**, 7091 (2001).

[48] M.P. Kostyley, V.E. Demidov, U.-H. Hansen, and S. O. Demokritov, Phys. Rev. B **76**, 224414 (2007).

[49] S. O. Demokritov, Editor, Spin Wave Confinement, Pan Stanford Publishing, Singapore (2009).

[50] J. Jorzick, S. O. Demokritov, C. Mathieu, B. Hillebrands, B. Bartenlian, C. Chappert, F. Rousseaux, A.N. Slavin, Phys. Rev. B **60**, 15194 (1999).

[51] G. Brown and A. Janotti, M. Eisenbach and G.M. Stocks, Phys. Rev. B **72**, 140405 (2005).

[52] G. Brown, J. Appl. Phys. **103**, 07D504 (2008).

[53] www.ccs.ornl.gov/mri/repository/Psi-Mag/.

[54] D. P. Landau and K. Binder, A Guide to Monte Carlo Simulations in Statistical Physics, Cambridge University Press, Cambridge (2000).

[55] K. Yosida, Theory of Magnetism, Springer-Verlag, Berlin (1996).

[56] R. Skomski, Simple Models of Magnetism, Oxford University Press, New York (2008).

[57] P. Langevin, J. Phys. (Paris) **4**, 678 (1905).

[58] T. Moriya, Spin Fluctuations in Itinerant Electron Magnetism, Springer-Verlag, Berlin (1985).

[59] P. R. Weiss, J. Phys. Radium **6**, 661 (1907).

[60] P. R. Weiss, Phys. Rev. **74**, 1493 (1948).

[61] T. Oguchi, Prog. Theor. Phys. **13**, 148 (1955).

[62] P. W. Kasteleijn and J. van Kranendonk, Physica **22**, 317 (1956).

[63] P. Heller, Repts. Prog. Phys. **30**, 731 (1967).

[64] M. E. Fisher, Rev. Modern Phys. **46**, 597 1974.

[65] A. Aharoni, Introduction to the Theory of Ferromagnetism, 2nd ed., Oxford University Press, New York (2000).

[66] F. Bloch, Z. Physik **61**, 206 (1930).

[67] F. Bloch, Z. Physik **74**, 295 (1932).

[68] N. W. Ashcroft and N. D. Mermin, Solid State Physics, Harcourt College Publishers (1976).

[69] F. Holtzberg, T. R. McMuire, S. Methfessel and J. C. Suits, J. Appl. Phys. **35**, 1033 (1964).

[70] L. P. Kadanoff and P. C. Martin, Ann. Phys. **24**, 419 (1963).

[71] S. W. Lovesay, Theory of Neutron Scattering from Condensed Matter, Oxford, Clarendon (1984).

[72] G. L. Squires, Thermal Nuetron Scattering, Cambridge University Press, Cambridge (1978).

[73] L. Van. Hove, Phys. Rev. **95**, 249 (1954).

[74] L. Van. Hove, Phys. Rev. **95**, 1374 (1954).

[75] W. Marshall and S. W. Lovesay, Theory of Thermal Neutron Scattering, Oxford, Clarendon (1971).

[76] B. N. Brockhouse, Rev. Mod. Phys. **67**, 735 (1995).

[77] http://neutrons.ornl.gov/instruments/HFIR/HB3/.

[78] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller, J. Chem. Phys. **21**, 1087 (1953).

[79] K. Binder, Monte Carlo Methods in Statistical Physics, Springer-Verlag, Berlin (1987).

[80] U. Wolff, Phys. Rev. Lett. **62**, 361 (1989).

[81] M. Creutz, Phys. Rev. D **36**, 515 (1987).

[82] F. R. Brown and T. J. Woch, Phys. Rev. Lett. **58**, 2394 (1987).

[83] W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterling, Numerical Recipes in C++: The Art of Scientific Computing, Cambridge University Press (2002).

[84] H. G. Evertz and D. P. Landau, Phys. Rev. B **54**, 12302 (1996).

[85] R. E. Watson, M. Blume and G. H. Vineyard, Phys. Rev. **181**, 811 (1969).

[86] M. Krech, A. Bunker and D. P. Landau, Comput. Phys. Commun. **111**, 1 (1998).

[87] D. P. Landau, S.-H. Tsai, M. Krech and A. Bunker, Int. J. of Mod. Phys. C **10**, 1541 (1999).

[88] M. Suzuki, J. Math. Phys. **26**, 601 (1985).

[89] M. Suzuki and K. Umeno, Computer Simulation Studies in Condensed Matter Physics VI, D. P. Landau, K. K. Mon and H. B. Schüttler, Editors, Springer-Verlag (1993).

[90] J. Frank, W. Huang and B. Leimkuhler, J. Comp. Phys. **133**, 160 (1997).

Appendix A

Data Generation Programs

A.1 Compilation and Execution

A.1.1 Compilation with *makefile*

♯ This is a makefile

♯ -DLOCAL_BC_PBCXY define the function to calculate local correlation function for nanofilms with $\mathbf{r}_0 \Rightarrow$ *Bulk Center*

CFLAGS = -ansi -pedantic

OPT = -O3

all:

g++ $(CFLAGS) $(OPT) -DLOCAL_BC_PBCXY -I ./include -I ./ main.cpp -o main.exe

A.1.2 Execution

Prepare *in.dat* and use the following command to execute.

main.exe in.dat

A.2 Input Data File *in.dat*

```
#============================PARAMETERS================================
                    outFileTitle      h004J-1A0h2w0i0o8DkNkC1nkI1r0_OIC_
                       modelType      Heisenberg
                      initConfig      ORDERED
                  masterSeedType      RANDOM
                     latDimension     3
```

```
                  latSize(zyx)      4 10 10
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~BC~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
           BC(pbc/pbcXY/fbc/sbc)      pbcXY
#~~~~~~~~~~~~~~~~~~~~~~~~~~DoWhat(0:no/1:yes)~~~~~~~~~~~~~~~~~~~~~~~~~~~~
              iDoWhat(0:SD/3:MC)      0
#-----------------------------------------------------------------------
                     print_EMhs      0
                     print_EMtm      0
#-----------------------------------------------------------------------
               doCorre_Real(pbc)      0
                 doCorre_Real_In      (100)
                print_Corre_Real      0
#-----------------------------------------------------------------------
                doCorre_Fly(pbc)      0
                 print_Corre_Fly      0
#-----------------------------------------------------------------------
          doCorre_LocalSL(f/sbc)      0
              doCorre_LocalSL_In      (100)
            print_Corre_LocalSL      0
                      startIndex      52
              localSL_Direction      udlrf
                whichDirToPrint      f
               ave_On_Direction      0
           localSL_Ave_Direction      udlr
#-----------------------------------------------------------------------
         doCorre_Local_BC(f/sbc)      0
            print_Corre_Local_BC      0
#-----------------------------------------------------------------------
         doCorre_Local_SC(f/sbc)      0
            print_Corre_Local_SC      0
#-----------------------------------------------------------------------
         doCorre_Local_CR(f/sbc)      0
            print_Corre_Local_CR      0
#-----------------------------------------------------------------------
   doCorre_Local_BC_PBCXY(pbcXY)      0
      print_Corre_Local_BC_PBCXY      0
#-----------------------------------------------------------------------
       doCorre_FSUR_PBCXY(pbcXY)      1
          print_Corre_FSUR_PBCXY      1
#-----------------------------------------------------------------------
                   doCellization      0
                 print_(OP/COM)      OP
                    print_2Cell      0
                     cellIndex1      50
                     cellIndex2      0
                  print_Quant-R      0
#-----------------------------------------------------------------------
```

```
                withSingleConfig      0
                 noAveOnCombined      0
                 doAveOnCombined      1
                         NConfig      1
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Anisotropy~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                   isAnisotropic      0
                     aniStrength      0
              aniDirection(Dxyz)      0  0  1
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~SD~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                    SuzukiOrder      4
                             nt      2000
                             tc      1000
                             dt      0.2
                          tIntv      20
                 totalIterations      1
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~MC~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                              T      0.4
                         NHeisen      2
                          NWolff      0
                          NIsing      0
                       NOverRelax      8
                        NDiscard      2000
                         NHybrid      2000
                      everyNSteps      20
                             EBin      0.000025
                             MBin      0.000025
#~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~J~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
                         NShells      1
                         NSubLat      2
                  N_per_Site_pbc      6
                      sh1Neigh_J      -1.0
#================================END=====================================
```

A.3 Source Code Samples

A.3.1 *main.cpp*

```
#include <iostream>
#include <cstdlib>

#include "timer.h"

#include "build_Program.h"


// Main program
```

```
int main(int argc, char* argv[])
{
 if(argc!=2) {
   std::cerr<<"Usage: <program> <input file>"<<std::endl;
   return EXIT_FAILURE;
 }
 timer   timing;
 std::cout<<"\nLatest Compile Time: "<<timing.compile_time()<<std::endl;
 std::cout<<"Program Start Time: "<<timing.currentTime()<<std::endl;

 Build_Program     program(argv[1]);
 program.run();

 std::cout<<"Program Run Time(D/H:M:S): "<<timing()<<std::endl<<std::endl;
 return EXIT_SUCCESS;
} // End of main
```

A.3.2 *build_Program.h*

```
#ifndef BUILD_PROGRAM_H
#define BUILD_PROGRAM_H

#include <vector>
#include <iostream>
#include <cstdlib>
#include <ctime>

#include "Real.h"              // double (or single if -DSINGLE_PRECISION)
#include "Vec.h"               // define 3-D vectors, e.g. spins and fields
#include "timer.h"             // get time
#include "Random.h"            // use various kinds of random number generators
#include "EHModel.h"           // Extended Heisenberg Model Hamiltonian
#include "Metropolis.h"        // importance sampling MC method

#include "input.h"             // read data in in.dat
#include "sclattice.h"         // define SC lattice and its sublattices
#include "build_EHModel.h"     // build up Hamiltonian with JList
#include "database.h"          // given a spin configuration, collect data
#include "spindynamics_Bunker.h"
                               // following Bunker's style to run SuzukiTrotter
#include "simulator.h"         // run the SD process
#include "controller.h"        // pass T to simulation


/*
#if   defined(USE_MPAX)
```

```
      typedef psimag::MPAX RndGen;
#elif defined(USE_RAN2)
      typedef psimag::Ran2 RndGen;
#elif defined(USE_R1279)
      typedef psimag::R1279 RndGen;
#endif
*/


class Build_Program
{
 typedef std::vector<unsigned int>                                    DataContainer;
 typedef psimag::Vec<psimag::Real,3>                                  SpinType;
 typedef psimag::Vec<psimag::Real,3>                                  FieldType;
 typedef std::vector<SpinType>                                        SpinContainer;
 typedef std::vector<FieldType>                                       FieldContainer;
 typedef psimag::Ran2                                                 RndGen;
 typedef Input<RndGen,DataContainer>                                  In;
 typedef ScLattice<DataContainer>                                     SC;
 typedef psimag::EHModel<SpinContainer,FieldContainer>                EHModel_D0;
 typedef psimag::EHModel<SpinContainer,FieldContainer>                EHModel_D;
 typedef Build_EHModel<SpinContainer,FieldContainer,0>                Build_EHModel_D0;
 typedef Build_EHModel<SpinContainer,FieldContainer,1>                Build_EHModel_D;
 typedef Database<In,SC,EHModel_D,DataContainer,SpinContainer>        DataBase;
 typedef Spindynamics<In,SC,EHModel_D0,SpinContainer,FieldType>       SpinDynamics;
 typedef Simulator<EHModel_D0,EHModel_D,psimag::Metropolis,
                   RndGen,In,DataBase,SpinDynamics,
                   SpinContainer,FieldType>                           Simulate;
 typedef Controller<In,Simulate>                                      Control;

 public:
         explicit Build_Program(const char*);
         ~Build_Program() {}

         void run();

 private:
         Build_Program(Build_Program&);
         Build_Program& operator=(const Build_Program&);

         SpinContainer         spins;
         In                    in;
         RndGen                ran;
         SC                    sc;
         EHModel_D0            model_D0;
         EHModel_D             model_D;
         Build_EHModel_D0      buildModel_D0;
```

```
        Build_EHModel_D       buildModel_D;
        DataBase              database;
        SpinDynamics          spindynamics;
        Simulate              simulate;
        Control               control;
};

/*****************************************************************************/

Build_Program::Build_Program(const char* infile)
: spins(),
  in(infile,spins),
  ran(in.para().master_random_seed),
  sc(in),
  model_D0(),
  model_D(),
  buildModel_D0(model_D0,sc),
  buildModel_D(model_D,sc,in),
  database(in,sc,model_D),
  spindynamics(in,sc,model_D0),
  simulate(sc,model_D0,model_D,ran,in,database,spindynamics),
  control(in,simulate)
{}

inline void Build_Program::run() { control.run(spins); }

#endif
```

A.3.3 *spindynamics_Bunker.h*

```
#ifndef SPINDYNAMICS_BUNKER_H
#define SPINDYNAMICS_BUNKER_H

#include <vector>

#include "Real.h"


// Spindynamics is designed to run SD in one SDdt (not decompTime!)
template<class Input, class SC, class EHModel_D0, class SpinContainer,
         class FieldType>
class Spindynamics
{
 typedef std::vector<unsigned int>                    DataContainer;

 public:
```

```
        Spindynamics(const Input&, const SC&, EHModel_DO&);
        ~Spindynamics();

        void run(SpinContainer&);
        void run(SpinContainer&, const std::vector<bool>&);

    const std::vector<DataContainer>& getSublat_index() const
        { return sublat_index; }

private:
        Spindynamics(const Spindynamics&);
        Spindynamics& operator=(const Spindynamics&);

    void ROTSPIN_A_Tylor(const psimag::Real, SpinContainer&, psimag::Real*);
    void ROTSPIN_B_Tylor(const psimag::Real, SpinContainer&, psimag::Real*);
    void ROTSPIN_A(const psimag::Real, SpinContainer&, psimag::Real*);
    void ROTSPIN_B(const psimag::Real, SpinContainer&, psimag::Real*);

        void ROTSPIN_A_Tylor(const psimag::Real, SpinContainer&, psimag::Real*,
                             const std::vector<bool>&);
        void ROTSPIN_B_Tylor(const psimag::Real, SpinContainer&, psimag::Real*,
                             const std::vector<bool>&);
        void ROTSPIN_A(const psimag::Real, SpinContainer&, psimag::Real*,
                       const std::vector<bool>&);
        void ROTSPIN_B(const psimag::Real, SpinContainer&, psimag::Real*,
                       const std::vector<bool>&);

        const Input&                                    in;
    const SC&                                       sc;
        const unsigned int                              totalIterations;
    const psimag::Real                              DT,
                                                    RP1, RP2, RP3, RP4, RP5,
                                                    RP1h, RP2h, RP3h, RP4h,
                                                        RP5h, RP12h, RP23h,
                                                        RP34h, RP45h,
                             CJT,
                             D;
        DataContainer                                   sublat_index_A,
                                                    sublat_index_B;
    std::vector<DataContainer>                      sublat_index,
                                                    nl;
    psimag::Real*                                   decompTime_futureSz_A1;
    psimag::Real*                                   decompTime_futureSz_A2;
    psimag::Real*                                   decompTime_futureSz_A3;
    psimag::Real*                                   decompTime_futureSz_A4;
    psimag::Real*                                   decompTime_futureSz_A5;
    psimag::Real*                                   decompTime_futureSz_A6;
```

```
    psimag::Real*                                   decompTime_futureSz_B1;
    psimag::Real*                                   decompTime_futureSz_B2;
    psimag::Real*                                   decompTime_futureSz_B3;
    psimag::Real*                                   decompTime_futureSz_B4;
    psimag::Real*                                   decompTime_futureSz_B5;
};

/*************************************************************************/

template<class Input, class SC, class EHModel_D0, class SpinContainer,
         class FieldType>
Spindynamics<Input,SC,EHModel_D0,SpinContainer,FieldType>::
Spindynamics(const Input& input, const SC& scLatt, EHModel_D0& model_D0)
: in(input),
  sc(scLatt),
  totalIterations(in.para().totalIterations),
  DT(in.para().SDdt),
  RP1(0.414490771794376*DT),
  RP2(0.414490771794376*DT),
  RP3(-0.657963087177503*DT),
  RP4(0.414490771794376*DT),
  RP5(0.414490771794376*DT),
  RP1h(RP1*0.5),
  RP2h(RP2*0.5),
  RP3h(RP3*0.5),
  RP4h(RP4*0.5),
  RP5h(RP5*0.5),
  RP12h(RP1h+RP2h),
  RP23h(RP2h+RP3h),
  RP34h(RP3h+RP4h),
  RP45h(RP4h+RP5h),
  CJT(in.para().exchange_factor[0]),
  D(in.para().D),
  sublat_index_A(sc.sublattice_length()[0]),
  sublat_index_B(sc.sublattice_length()[1]),
  sublat_index(2),
  nl(sc.neighbor_list_forBunker()),
  decompTime_futureSz_A1(NULL),
  decompTime_futureSz_A2(NULL),
  decompTime_futureSz_A3(NULL),
  decompTime_futureSz_A4(NULL),
  decompTime_futureSz_A5(NULL),
  decompTime_futureSz_A6(NULL),
  decompTime_futureSz_B1(NULL),
  decompTime_futureSz_B2(NULL),
  decompTime_futureSz_B3(NULL),
  decompTime_futureSz_B4(NULL),
```

```
  decompTime_futureSz_B5(NULL)
{
 for(unsigned int i=0; i<sublat_index_A.size(); i++)
   sublat_index_A[i] = sc.sublattice_index()[i];
 for(unsigned int i=0; i<sublat_index_B.size(); i++)
   sublat_index_B[i] = sc.sublattice_index()[sc.sublattice_length()[0]+i];

 sublat_index[0].assign(sublat_index_A.begin(),sublat_index_A.end());
 sublat_index[1].assign(sublat_index_B.begin(),sublat_index_B.end());

 decompTime_futureSz_A1 = new psimag::Real[sublat_index_A.size()];
 decompTime_futureSz_A2 = new psimag::Real[sublat_index_A.size()];
 decompTime_futureSz_A3 = new psimag::Real[sublat_index_A.size()];
 decompTime_futureSz_A4 = new psimag::Real[sublat_index_A.size()];
 decompTime_futureSz_A5 = new psimag::Real[sublat_index_A.size()];
 decompTime_futureSz_A6 = new psimag::Real[sublat_index_A.size()];
 decompTime_futureSz_B1 = new psimag::Real[sublat_index_B.size()];
 decompTime_futureSz_B2 = new psimag::Real[sublat_index_B.size()];
 decompTime_futureSz_B3 = new psimag::Real[sublat_index_B.size()];
 decompTime_futureSz_B4 = new psimag::Real[sublat_index_B.size()];
 decompTime_futureSz_B5 = new psimag::Real[sublat_index_B.size()];
}

template<class Input, class SC, class EHModel_D0, class SpinContainer,
         class FieldType>
Spindynamics<Input,SC,EHModel_D0,SpinContainer,FieldType>::~Spindynamics()
{
 delete [] decompTime_futureSz_A1;
 delete [] decompTime_futureSz_A2;
 delete [] decompTime_futureSz_A3;
 delete [] decompTime_futureSz_A4;
 delete [] decompTime_futureSz_A5;
 delete [] decompTime_futureSz_A6;
 delete [] decompTime_futureSz_B1;
 delete [] decompTime_futureSz_B2;
 delete [] decompTime_futureSz_B3;
 delete [] decompTime_futureSz_B4;
 delete [] decompTime_futureSz_B5;
}

// run SD in one SDdt (not decompTime!)
template<class Input, class SC, class EHModel_D0, class SpinContainer,
         class FieldType>
void Spindynamics<Input,SC,EHModel_D0,SpinContainer,FieldType>::
run(SpinContainer& spins)
{
 SpinContainer   spins0;
```

```
 spins0.assign(spins.begin(),spins.end());

 // Taylor iteration_4th order Suzuki_Trotter
 ROTSPIN_A_Tylor(RP1h,spins,decompTime_futureSz_A1);
 ROTSPIN_B_Tylor(RP1,spins,decompTime_futureSz_B1);
 ROTSPIN_A_Tylor(RP12h,spins,decompTime_futureSz_A2);
 ROTSPIN_B_Tylor(RP2,spins,decompTime_futureSz_B2);
 ROTSPIN_A_Tylor(RP23h,spins,decompTime_futureSz_A3);
 ROTSPIN_B_Tylor(RP3,spins,decompTime_futureSz_B3);
 ROTSPIN_A_Tylor(RP34h,spins,decompTime_futureSz_A4);
 ROTSPIN_B_Tylor(RP4,spins,decompTime_futureSz_B4);
 ROTSPIN_A_Tylor(RP45h,spins,decompTime_futureSz_A5);
 ROTSPIN_B_Tylor(RP5,spins,decompTime_futureSz_B5);
 ROTSPIN_A_Tylor(RP5h,spins,decompTime_futureSz_A6);

 // Iterative iterations_4th order Suzuki_Trotter
 if(in.para().isAnisotropic==true && totalIterations>1) {
   for(unsigned int iIteration=1; iIteration<totalIterations; iIteration++) {
     spins.assign(spins0.begin(),spins0.end());
     ROTSPIN_A(RP1h,spins,decompTime_futureSz_A1);
     ROTSPIN_B(RP1,spins,decompTime_futureSz_B1);
     ROTSPIN_A(RP12h,spins,decompTime_futureSz_A2);
     ROTSPIN_B(RP2,spins,decompTime_futureSz_B2);
     ROTSPIN_A(RP23h,spins,decompTime_futureSz_A3);
     ROTSPIN_B(RP3,spins,decompTime_futureSz_B3);
     ROTSPIN_A(RP34h,spins,decompTime_futureSz_A4);
     ROTSPIN_B(RP4,spins,decompTime_futureSz_B4);
     ROTSPIN_A(RP45h,spins,decompTime_futureSz_A5);
     ROTSPIN_B(RP5,spins,decompTime_futureSz_B5);
     ROTSPIN_A(RP5h,spins,decompTime_futureSz_A6);
   }
 }
}

template<class Input, class SC, class EHModel_D0, class SpinContainer,
         class FieldType>
void Spindynamics<Input,SC,EHModel_D0,SpinContainer,FieldType>::
run(SpinContainer& spins, const std::vector<bool>& surfaceLabel)
{
 SpinContainer   spins0;
 spins0.assign(spins.begin(),spins.end());

 // Taylor iteration_4th order Suzuki_Trotter
 ROTSPIN_A_Tylor(RP1h,spins,decompTime_futureSz_A1,surfaceLabel);
 ROTSPIN_B_Tylor(RP1,spins,decompTime_futureSz_B1,surfaceLabel);
 ROTSPIN_A_Tylor(RP12h,spins,decompTime_futureSz_A2,surfaceLabel);
 ROTSPIN_B_Tylor(RP2,spins,decompTime_futureSz_B2,surfaceLabel);
```

```
 ROTSPIN_A_Tylor(RP23h,spins,decompTime_futureSz_A3,surfaceLabel);
 ROTSPIN_B_Tylor(RP3,spins,decompTime_futureSz_B3,surfaceLabel);
 ROTSPIN_A_Tylor(RP34h,spins,decompTime_futureSz_A4,surfaceLabel);
 ROTSPIN_B_Tylor(RP4,spins,decompTime_futureSz_B4,surfaceLabel);
 ROTSPIN_A_Tylor(RP45h,spins,decompTime_futureSz_A5,surfaceLabel);
 ROTSPIN_B_Tylor(RP5,spins,decompTime_futureSz_B5,surfaceLabel);
 ROTSPIN_A_Tylor(RP5h,spins,decompTime_futureSz_A6,surfaceLabel);

 // Iterative iterations_4th order Suzuki_Trotter
 if(in.para().isAnisotropic==true && totalIterations>1) {
   for(unsigned int iIteration=1; iIteration<totalIterations; iIteration++) {
     spins.assign(spins0.begin(),spins0.end());
     ROTSPIN_A(RP1h,spins,decompTime_futureSz_A1,surfaceLabel);
     ROTSPIN_B(RP1,spins,decompTime_futureSz_B1,surfaceLabel);
     ROTSPIN_A(RP12h,spins,decompTime_futureSz_A2,surfaceLabel);
     ROTSPIN_B(RP2,spins,decompTime_futureSz_B2,surfaceLabel);
     ROTSPIN_A(RP23h,spins,decompTime_futureSz_A3,surfaceLabel);
     ROTSPIN_B(RP3,spins,decompTime_futureSz_B3,surfaceLabel);
     ROTSPIN_A(RP34h,spins,decompTime_futureSz_A4,surfaceLabel);
     ROTSPIN_B(RP4,spins,decompTime_futureSz_B4,surfaceLabel);
     ROTSPIN_A(RP45h,spins,decompTime_futureSz_A5,surfaceLabel);
     ROTSPIN_B(RP5,spins,decompTime_futureSz_B5,surfaceLabel);
     ROTSPIN_A(RP5h,spins,decompTime_futureSz_A6,surfaceLabel);
   }
 }
}

template<class Input, class SC, class EHModel_D0, class SpinContainer,
         class FieldType>
void Spindynamics<Input,SC,EHModel_D0,SpinContainer,FieldType>::
ROTSPIN_A_Tylor(const psimag::Real DF, SpinContainer& spins,
                psimag::Real* decompTime_futureSz_A)
{
 psimag::Real WX, WY, WZ, W2,
              WXSx, WXSy, WXSz, WdotS,
      SINF, COSF;
 unsigned int index;
 for(unsigned int i=0; i<sublat_index_A.size(); i++) {
   index = sublat_index_A[i];
   WX = 0.;
   WY = 0.;
   WZ = 0.;
   for(unsigned int ni=0; ni<nl[index].size(); ni++) {
     WX += spins[nl[index][ni]][0]*-CJT;
     WY += spins[nl[index][ni]][1]*-CJT;
     WZ += spins[nl[index][ni]][2]*-CJT;
   }
```

```
   WZ -= D*(2.*spins[index][2]+(WX*spins[index][1]-WY*spins[index][0])*DF);

   W2   = WX*WX+WY*WY+WZ*WZ;
   WXSx = WY*spins[index][2]-WZ*spins[index][1];
   WXSy = WZ*spins[index][0]-WX*spins[index][2];
   WXSz = WX*spins[index][1]-WY*spins[index][0];
   WdotS=(WX*spins[index][0]+WY*spins[index][1]+WZ*spins[index][2])/W2;

   SINF = (sin(sqrt(W2)*DF))/sqrt(W2);
   COSF = cos(sqrt(W2)*DF)-1.0;

   spins[index][0] = spins[index][0]+(spins[index][0]-WX*WdotS)*COSF+WXSx*SINF;
   spins[index][1] = spins[index][1]+(spins[index][1]-WY*WdotS)*COSF+WXSy*SINF;
   spins[index][2] = spins[index][2]+(spins[index][2]-WZ*WdotS)*COSF+WXSz*SINF;

   decompTime_futureSz_A[i] = spins[index][2];
 }
 return;
}

template<class Input, class SC, class EHModel_D0, class SpinContainer,
         class FieldType>
void Spindynamics<Input,SC,EHModel_D0,SpinContainer,FieldType>::
ROTSPIN_B_Tylor(const psimag::Real DF, SpinContainer& spins,
        psimag::Real* decompTime_futureSz_B)
{
 psimag::Real WX, WY, WZ, W2,
              WXSx, WXSy, WXSz, WdotS,
              SINF, COSF;
 unsigned int index;
 for(unsigned int i=0; i<sublat_index_B.size(); i++) {
   index = sublat_index_B[i];
   WX = 0.;
   WY = 0.;
   WZ = 0.;
   for(unsigned int ni=0; ni<nl[index].size(); ni++) {
     WX += spins[nl[index][ni]][0]*-CJT;
     WY += spins[nl[index][ni]][1]*-CJT;
     WZ += spins[nl[index][ni]][2]*-CJT;
   }
   WZ -= D*(2.*spins[index][2]+(WX*spins[index][1]-WY*spins[index][0])*DF);

   W2   = WX*WX+WY*WY+WZ*WZ;
   WXSx = WY*spins[index][2]-WZ*spins[index][1];
   WXSy = WZ*spins[index][0]-WX*spins[index][2];
   WXSz = WX*spins[index][1]-WY*spins[index][0];
   WdotS=(WX*spins[index][0]+WY*spins[index][1]+WZ*spins[index][2])/W2;
```

```
   SINF = (sin(sqrt(W2)*DF))/sqrt(W2);
   COSF = cos(sqrt(W2)*DF)-1.0;

   spins[index][0] = spins[index][0]+(spins[index][0]-WX*WdotS)*COSF+WXSx*SINF;
   spins[index][1] = spins[index][1]+(spins[index][1]-WY*WdotS)*COSF+WXSy*SINF;
   spins[index][2] = spins[index][2]+(spins[index][2]-WZ*WdotS)*COSF+WXSz*SINF;

   decompTime_futureSz_B[i] = spins[index][2];
  }
  return;
}

template<class Input, class SC, class EHModel_D0, class SpinContainer,
         class FieldType>
void Spindynamics<Input,SC,EHModel_D0,SpinContainer,FieldType>::
ROTSPIN_A(const psimag::Real DF, SpinContainer& spins,
          psimag::Real* decompTime_futureSz_A)
{
 psimag::Real WX, WY, WZ, W2,
              WXSx, WXSy, WXSz, WdotS,
              SINF, COSF;
 unsigned int index;
 for(unsigned int i=0; i<sublat_index_A.size(); i++) {
   index = sublat_index_A[i];
   WX = 0.;
   WY = 0.;
   WZ = 0.;
   for(unsigned int ni=0; ni<nl[index].size(); ni++) {
     WX += spins[nl[index][ni]][0]*-CJT;
     WY += spins[nl[index][ni]][1]*-CJT;
     WZ += spins[nl[index][ni]][2]*-CJT;
   }
   WZ -= D*(spins[index][2]+decompTime_futureSz_A[i]);

   W2   = WX*WX+WY*WY+WZ*WZ;
   WXSx = WY*spins[index][2]-WZ*spins[index][1];
   WXSy = WZ*spins[index][0]-WX*spins[index][2];
   WXSz = WX*spins[index][1]-WY*spins[index][0];
   WdotS=(WX*spins[index][0]+WY*spins[index][1]+WZ*spins[index][2])/W2;

   SINF = (sin(sqrt(W2)*DF))/sqrt(W2);
   COSF = cos(sqrt(W2)*DF)-1.0;

   spins[index][0] = spins[index][0]+(spins[index][0]-WX*WdotS)*COSF+WXSx*SINF;
   spins[index][1] = spins[index][1]+(spins[index][1]-WY*WdotS)*COSF+WXSy*SINF;
   spins[index][2] = spins[index][2]+(spins[index][2]-WZ*WdotS)*COSF+WXSz*SINF;
```

```
   decompTime_futureSz_A[i] = spins[index][2];
 }
 return;
}

template<class Input, class SC, class EHModel_D0, class SpinContainer,
         class FieldType>
void Spindynamics<Input,SC,EHModel_D0,SpinContainer,FieldType>::
ROTSPIN_B(const psimag::Real DF, SpinContainer& spins,
          psimag::Real* decompTime_futureSz_B)
{
 psimag::Real WX, WY, WZ, W2,
              WXSx, WXSy, WXSz, WdotS,
              SINF, COSF;
 unsigned int index;
 for(unsigned int i=0; i<sublat_index_B.size(); i++) {
   index = sublat_index_B[i];
   WX = 0.;
   WY = 0.;
   WZ = 0.;
   for(unsigned int ni=0; ni<nl[index].size(); ni++) {
     WX += spins[nl[index][ni]][0]*-CJT;
     WY += spins[nl[index][ni]][1]*-CJT;
     WZ += spins[nl[index][ni]][2]*-CJT;
   }
   WZ -= D*(spins[index][2]+decompTime_futureSz_B[i]);

   W2   = WX*WX+WY*WY+WZ*WZ;
   WXSx = WY*spins[index][2]-WZ*spins[index][1];
   WXSy = WZ*spins[index][0]-WX*spins[index][2];
   WXSz = WX*spins[index][1]-WY*spins[index][0];
   WdotS=(WX*spins[index][0]+WY*spins[index][1]+WZ*spins[index][2])/W2;

   SINF = (sin(sqrt(W2)*DF))/sqrt(W2);
   COSF = cos(sqrt(W2)*DF)-1.0;

   spins[index][0] = spins[index][0]+(spins[index][0]-WX*WdotS)*COSF+WXSx*SINF;
   spins[index][1] = spins[index][1]+(spins[index][1]-WY*WdotS)*COSF+WXSy*SINF;
   spins[index][2] = spins[index][2]+(spins[index][2]-WZ*WdotS)*COSF+WXSz*SINF;

   decompTime_futureSz_B[i] = spins[index][2];
 }
 return;
}

template<class Input, class SC, class EHModel_D0, class SpinContainer,
```

```
         class FieldType>
void Spindynamics<Input,SC,EHModel_D0,SpinContainer,FieldType>::
ROTSPIN_A_Tylor(const psimag::Real DF, SpinContainer& spins,
                psimag::Real* decompTime_futureSz_A,
                const std::vector<bool>& surfaceLabel)
{
 psimag::Real WX, WY, WZ, W2,
              WXSx, WXSy, WXSz, WdotS,
              SINF, COSF;
 unsigned int index;
 for(unsigned int i=0; i<sublat_index_A.size(); i++) {
   index = sublat_index_A[i];
   if(!surfaceLabel[index]) {
     WX = 0.;
     WY = 0.;
     WZ = 0.;
     for(unsigned int ni=0; ni<nl[index].size(); ni++) {
       WX += spins[nl[index][ni]][0]*-CJT;
       WY += spins[nl[index][ni]][1]*-CJT;
       WZ += spins[nl[index][ni]][2]*-CJT;
     }
     WZ -= D*(2.*spins[index][2]+(WX*spins[index][1]-WY*spins[index][0])*DF);

     W2   = WX*WX+WY*WY+WZ*WZ;
     WXSx = WY*spins[index][2]-WZ*spins[index][1];
     WXSy = WZ*spins[index][0]-WX*spins[index][2];
     WXSz = WX*spins[index][1]-WY*spins[index][0];
     WdotS=(WX*spins[index][0]+WY*spins[index][1]+WZ*spins[index][2])/W2;

     SINF = (sin(sqrt(W2)*DF))/sqrt(W2);
     COSF = cos(sqrt(W2)*DF)-1.0;

     spins[index][0] = spins[index][0]+(spins[index][0]-WX*WdotS)*COSF+WXSx*SINF;
     spins[index][1] = spins[index][1]+(spins[index][1]-WY*WdotS)*COSF+WXSy*SINF;
     spins[index][2] = spins[index][2]+(spins[index][2]-WZ*WdotS)*COSF+WXSz*SINF;
   }
   decompTime_futureSz_A[i] = spins[index][2];
 }
 return;
}

template<class Input, class SC, class EHModel_D0, class SpinContainer,
         class FieldType>
void Spindynamics<Input,SC,EHModel_D0,SpinContainer,FieldType>::
                  ROTSPIN_B_Tylor(const psimag::Real DF, SpinContainer& spins,
                  psimag::Real* decompTime_futureSz_B,
                  const std::vector<bool>& surfaceLabel)
```

```
{
 psimag::Real WX, WY, WZ, W2,
              WXSx, WXSy, WXSz, WdotS,
              SINF, COSF;
 unsigned int index;
 for(unsigned int i=0; i<sublat_index_B.size(); i++) {
   index = sublat_index_B[i];
   if(!surfaceLabel[index]) {
     WX = 0.;
     WY = 0.;
     WZ = 0.;
     for(unsigned int ni=0; ni<nl[index].size(); ni++) {
       WX += spins[nl[index][ni]][0]*-CJT;
       WY += spins[nl[index][ni]][1]*-CJT;
       WZ += spins[nl[index][ni]][2]*-CJT;
     }
     WZ -= D*(2.*spins[index][2]+(WX*spins[index][1]-WY*spins[index][0])*DF);

     W2   = WX*WX+WY*WY+WZ*WZ;
     WXSx = WY*spins[index][2]-WZ*spins[index][1];
     WXSy = WZ*spins[index][0]-WX*spins[index][2];
     WXSz = WX*spins[index][1]-WY*spins[index][0];
     WdotS=(WX*spins[index][0]+WY*spins[index][1]+WZ*spins[index][2])/W2;

     SINF = (sin(sqrt(W2)*DF))/sqrt(W2);
     COSF = cos(sqrt(W2)*DF)-1.0;

     spins[index][0] = spins[index][0]+(spins[index][0]-WX*WdotS)*COSF+WXSx*SINF;
     spins[index][1] = spins[index][1]+(spins[index][1]-WY*WdotS)*COSF+WXSy*SINF;
     spins[index][2] = spins[index][2]+(spins[index][2]-WZ*WdotS)*COSF+WXSz*SINF;
   }
   decompTime_futureSz_B[i] = spins[index][2];
  }
  return;
}

template<class Input, class SC, class EHModel_D0, class SpinContainer,
         class FieldType>
void Spindynamics<Input,SC,EHModel_D0,SpinContainer,FieldType>::
ROTSPIN_A(const psimag::Real DF, SpinContainer& spins,
          psimag::Real* decompTime_futureSz_A,
          const std::vector<bool>& surfaceLabel)
{
 psimag::Real WX, WY, WZ, W2,
              WXSx, WXSy, WXSz, WdotS,
              SINF, COSF;
 unsigned int index;
```

```
 for(unsigned int i=0; i<sublat_index_A.size(); i++) {
   index = sublat_index_A[i];
   if(!surfaceLabel[index]) {
     WX = 0.;
     WY = 0.;
     WZ = 0.;
     for(unsigned int ni=0; ni<nl[index].size(); ni++) {
       WX += spins[nl[index][ni]][0]*-CJT;
       WY += spins[nl[index][ni]][1]*-CJT;
       WZ += spins[nl[index][ni]][2]*-CJT;
     }
     WZ -= D*(spins[index][2]+decompTime_futureSz_A[i]);

     W2   = WX*WX+WY*WY+WZ*WZ;
     WXSx = WY*spins[index][2]-WZ*spins[index][1];
     WXSy = WZ*spins[index][0]-WX*spins[index][2];
     WXSz = WX*spins[index][1]-WY*spins[index][0];
     WdotS=(WX*spins[index][0]+WY*spins[index][1]+WZ*spins[index][2])/W2;

     SINF = (sin(sqrt(W2)*DF))/sqrt(W2);
     COSF = cos(sqrt(W2)*DF)-1.0;

     spins[index][0] = spins[index][0]+(spins[index][0]-WX*WdotS)*COSF+WXSx*SINF;
     spins[index][1] = spins[index][1]+(spins[index][1]-WY*WdotS)*COSF+WXSy*SINF;
     spins[index][2] = spins[index][2]+(spins[index][2]-WZ*WdotS)*COSF+WXSz*SINF;
   }
   decompTime_futureSz_A[i] = spins[index][2];
 }
 return;
}

template<class Input, class SC, class EHModel_D0, class SpinContainer,
         class FieldType>
void Spindynamics<Input,SC,EHModel_D0,SpinContainer,FieldType>::
ROTSPIN_B(const psimag::Real DF, SpinContainer& spins,
          psimag::Real* decompTime_futureSz_B,
          const std::vector<bool>& surfaceLabel)
{
 psimag::Real WX, WY, WZ, W2,
              WXSx, WXSy, WXSz, WdotS,
              SINF, COSF;
 unsigned int index;
 for(unsigned int i=0; i<sublat_index_B.size(); i++) {
   index = sublat_index_B[i];
   if(!surfaceLabel[index]) {
     WX = 0.;
     WY = 0.;
```

```
      WZ = 0.;
      for(unsigned int ni=0; ni<nl[index].size(); ni++) {
        WX += spins[nl[index][ni]][0]*-CJT;
        WY += spins[nl[index][ni]][1]*-CJT;
        WZ += spins[nl[index][ni]][2]*-CJT;
      }
      WZ -= D*(spins[index][2]+decompTime_futureSz_B[i]);

      W2   = WX*WX+WY*WY+WZ*WZ;
      WXSx = WY*spins[index][2]-WZ*spins[index][1];
      WXSy = WZ*spins[index][0]-WX*spins[index][2];
      WXSz = WX*spins[index][1]-WY*spins[index][0];
      WdotS=(WX*spins[index][0]+WY*spins[index][1]+WZ*spins[index][2])/W2;

      SINF = (sin(sqrt(W2)*DF))/sqrt(W2);
      COSF = cos(sqrt(W2)*DF)-1.0;

      spins[index][0] = spins[index][0]+(spins[index][0]-WX*WdotS)*COSF+WXSx*SINF;
      spins[index][1] = spins[index][1]+(spins[index][1]-WY*WdotS)*COSF+WXSy*SINF;
      spins[index][2] = spins[index][2]+(spins[index][2]-WZ*WdotS)*COSF+WXSz*SINF;
    }
    decompTime_futureSz_B[i] = spins[index][2];
  }
  return;
}

#endif
```

A.3.4 *simulator.h*

```
#ifndef SIMULATOR_H
#define SIMULATOR_H

#include <fstream>
#include <vector>
#include <cmath>
#include <cstdlib>
#include <ctime>

#include "Real.h"
#include "HeisenbergSpinStepper.h"
#include "IsingFlip.h"
#include "WolffCluster.h"
#include "OverRelaxation.h"
#include "RotationOperator.h"
#include "LatticeIndexMapper.h"
```

```
template<class EHModel_D0, class EHModel_D, class AcceptProb, class RndGen,
         class Input, class DataBase, class SpinDynamics, class SpinContainer,
         class FieldType>
class Simulator
{
 typedef psimag::RotationOperator<typename SpinContainer::value_type,FieldType>
         RotationOp;

 public:
         template<class SC>
         Simulator(const SC&, EHModel_D0&, EHModel_D&, RndGen&, const Input&,
                   DataBase&, SpinDynamics&);
         ~Simulator() {}

         void run(SpinContainer&, psimag::Real);

 private:
         Simulator(const Simulator&);
         Simulator& operator=(const Simulator&);

         void MC_moves(SpinContainer&, psimag::Real);
         void MC_moves(SpinContainer&, psimag::Real, const std::vector<bool>&);

         void zPolarize_Iso(SpinContainer&);
         void zPolarize_Iso(SpinContainer&, const std::vector<bool>&);

         // monte carlo moves
         psimag::HeisenbergSpinStepper<EHModel_D,AcceptProb>   heisenberg;
         psimag::WolffCluster<AcceptProb>                      wolff;
         psimag::IsingFlip<EHModel_D0, AcceptProb>             ising;
         psimag::OverRelaxation<RotationOp, EHModel_D0>        overRelax;

         RndGen&                                               ran;
         const Input&                                          input;
         DataBase&                                             database;
         SpinDynamics&                                         spindynamics;

         const std::vector<unsigned int>&                      sublatticeLabel;
         const std::vector<bool>&                              surfaceLabel;
};

/*********************************************************************************/

template<class EHModel_D0, class EHModel_D, class AcceptProb, class RndGen,
         class Input, class DataBase, class SpinDynamics, class SpinContainer,
```

```
         class FieldType>
template<class SC>
Simulator<EHModel_D0,EHModel_D,AcceptProb,RndGen,Input,DataBase,SpinDynamics,
SpinContainer,FieldType>::
Simulator(const SC& sc, EHModel_D0& md_D0, EHModel_D& md_D, RndGen& rn,
          const Input& in, DataBase& dbase, SpinDynamics& spind)
: heisenberg(md_D),
  wolff(sc.neighbor_list(), sc.neighbor_size(), sc.J_list()),
  ising(md_D0),
  overRelax(md_D0),
  ran(rn),
  input(in),
  database(dbase),
  spindynamics(spind),
  sublatticeLabel(sc.sublattice_label()),
  surfaceLabel(sc.surface_label())
{}

template<class EHModel_D0, class EHModel_D, class AcceptProb, class RndGen,
         class Input, class DataBase, class SpinDynamics, class SpinContainer,
         class FieldType>
void Simulator<EHModel_D0,EHModel_D,AcceptProb,RndGen,Input,DataBase,
               SpinDynamics,SpinContainer,FieldType>
::run(SpinContainer& spins, psimag::Real T)
{
 if(input.para().boundary!="sbc")
   { for(unsigned int i=0; i<input.para().NDiscard; i++) MC_moves(spins,T); }
 else { for(unsigned int i=0; i<input.para().NDiscard; i++)
   MC_moves(spins,T,surfaceLabel); }

 if(input.para().idowhat==3) {
   unsigned int NHybrid       = input.para().NHybrid,
                everyNSteps   = input.para().everyNSteps;
   bool         print_EMhs    = input.para().print_EMhs,
                print_EMtm    = input.para().print_EMtm,
                doCellization = input.para().doCellization;

   for(unsigned int i=0; i<NHybrid; i++) {

     #ifdef EM
     if(print_EMtm==true || print_EMhs==true) database.collect_EM(spins,i);
     #endif
     #ifdef CELL
     if(doCellization==true) database.collect_Cell(spins,i);
     #endif

     if(input.para().boundary!="sbc")
```

```
      { for(unsigned int j=0; j<everyNSteps; j++) MC_moves(spins,T); }
    else { for(unsigned int j=0; j<everyNSteps; j++)
      MC_moves(spins,T,surfaceLabel); }
  }

  #ifdef EM
  if(print_EMtm==true || print_EMhs==true) database.collect_EM(spins,NHybrid);
  #endif
  #ifdef CELL
  if(doCellization==true) database.collect_Cell(spins,NHybrid);
  #endif

  long int Id = time(NULL);
  #ifdef EM
  if(print_EMtm==true || print_EMhs==true) database.output_EM(Id);
  #endif
  #ifdef CELL
  if(doCellization==true) database.output_Cell(Id);
  #endif

  database.clearup();
}
else {
  unsigned int  NHybrid               = input.para().NHybrid,
                everyNSteps           = input.para().everyNSteps,
                NConfig               = input.para().NConfig,
                nt                    = input.para().nt;
  psimag::Real  SDdt                  = input.para().SDdt;
  bool          singleConfig          = input.para().singleConfig,
                print_EMhs            = input.para().print_EMhs,
                print_EMtm            = input.para().print_EMtm,
                doCorre_Fly           = input.para().doCorre_Fly,
                doCorre_Real          = input.para().doCorre_Real,
                doCorre_LocalSL       = input.para().doCorre_LocalSL,
                doCorre_Local_BC      = input.para().doCorre_Local_BC,
                doCorre_Local_SC      = input.para().doCorre_Local_SC,
                doCorre_Local_CR      = input.para().doCorre_Local_CR,
                doCorre_Local_BC_PBCXY = input.para().doCorre_Local_BC_PBCXY,
                doCorre_FSUR_PBCXY    = input.para().doCorre_FSUR_PBCXY,
                doCellization         = input.para().doCellization;

  SpinContainer spins0;
  spins0.assign(spins.begin(),spins.end());

  for(unsigned int iConfig=0; iConfig<NConfig; iConfig++) {
    std::cout<<"iConfig = "<<iConfig<<" of "<<NConfig<<" with dt = "
             <<std::setprecision(2)<<std::fixed<<SDdt<<std::endl;
```

```
if(input.para().isAnisotropic==false) {
  if(input.para().boundary!="sbc") zPolarize_Iso(spins);
  else zPolarize_Iso(spins,surfaceLabel);
}

for(unsigned int it=0; it<nt; it++) {

  #ifdef EM
  if(print_EMtm==true || print_EMhs==true) database.collect_EM(spins,it);
  #endif
  #ifdef FLY
  if(doCorre_Fly==true)          database.collect_Corr_Intermd(spins,it);
  #endif
  #ifdef REAL
  if(doCorre_Real==true)         database.collect_Corr_Real(spins,it);
  #endif
  #ifdef LOCAL_SINGLE_LINE
  if(doCorre_LocalSL==true)      database.collect_Corr_LocalSL(spins,it);
  #endif
  #ifdef LOCAL_BC
  if(doCorre_Local_BC==true)     database.collect_Corr_Local_BC(spins,it);
  #endif
  #ifdef LOCAL_SC
  if(doCorre_Local_SC==true)     database.collect_Corr_Local_SC(spins,it);
  #endif
  #ifdef LOCAL_CR
  if(doCorre_Local_CR==true)     database.collect_Corr_Local_CR(spins,it);
  #endif
  #ifdef LOCAL_BC_PBCXY
  if(doCorre_Local_BC_PBCXY==true)
    database.collect_Corr_Local_BC_PBCXY(spins,it);
  #endif
  #ifdef FSUR_PBCXY
  if(doCorre_FSUR_PBCXY==true)  database.collect_Corr_FSUR_PBCXY(spins,it);
  #endif
  #ifdef CELL
  if(doCellization==true)       database.collect_Cell(spins,it);
  #endif

  if(input.para().boundary!="sbc") spindynamics.run(spins);
  else spindynamics.run(spins,surfaceLabel);

  if(input.para().isAnisotropic==false&&input.para().exchange_factor[0]<0.)
  {
    if(input.para().boundary!="sbc") zPolarize_Iso(spins);
    else zPolarize_Iso(spins,surfaceLabel);
```

```
  }
}

#ifdef EM
if(print_EMtm==true || print_EMhs==true) database.collect_EM(spins,nt);
#endif
#ifdef FLY
if(doCorre_Fly==true)             database.collect_Corr_Intermd(spins,nt);
#endif
#ifdef REAL
if(doCorre_Real==true)            database.collect_Corr_Real(spins,nt);
#endif
#ifdef LOCAL_SINGLE_LINE
if(doCorre_LocalSL==true)         database.collect_Corr_LocalSL(spins,nt);
#endif
#ifdef LOCAL_BC
if(doCorre_Local_BC==true)        database.collect_Corr_Local_BC(spins,nt);
#endif
#ifdef LOCAL_SC
if(doCorre_Local_SC==true)        database.collect_Corr_Local_SC(spins,nt);
#endif
#ifdef LOCAL_CR
if(doCorre_Local_CR==true)        database.collect_Corr_Local_CR(spins,nt);
#endif
#ifdef LOCAL_BC_PBCXY
if(doCorre_Local_BC_PBCXY==true)
  database.collect_Corr_Local_BC_PBCXY(spins,nt);
#endif
#ifdef FSUR_PBCXY
if(doCorre_FSUR_PBCXY==true)
  database.collect_Corr_FSUR_PBCXY(spins,nt);
#endif
#ifdef CELL
if(doCellization==true)            database.collect_Cell(spins,nt);
#endif

#ifdef FLY
if(doCorre_Fly==true)              database.doCorre_Intermd();
#endif
#ifdef REAL
if(doCorre_Real==true)             database.doCorre_Real();
#endif
#ifdef LOCAL_SINGLE_LINE
if(doCorre_LocalSL==true)          database.doCorre_LocalSL();
#endif
#ifdef LOCAL_BC
if(doCorre_Local_BC==true)         database.doCorre_Local_BC();
```

```
#endif
#ifdef LOCAL_SC
if(doCorre_Local_SC==true)         database.doCorre_Local_SC();
#endif
#ifdef LOCAL_CR
if(doCorre_Local_CR==true)         database.doCorre_Local_CR();
#endif
#ifdef LOCAL_BC_PBCXY
if(doCorre_Local_BC_PBCXY==true)   database.doCorre_Local_BC_PBCXY();
#endif
#ifdef FSUR_PBCXY
if(doCorre_FSUR_PBCXY==true)       database.doCorre_FSUR_PBCXY();
#endif

long int Id = time(NULL);
#ifdef EM
if(print_EMtm==true || print_EMhs==true)
  database.output_EM(Id,SDdt,iConfig);
#endif
#ifdef FLY
if(doCorre_Fly==true && singleConfig==true)
  database.output_Corr_Intermd(Id,SDdt,iConfig);
#endif
#ifdef REAL
if(doCorre_Real==true && singleConfig==true)
  database.output_Corr_Real(Id,SDdt,iConfig);
#endif
#ifdef LOCAL_SINGLE_LINE
if(doCorre_LocalSL==true && singleConfig==true)
  database.output_Corr_LocalSL(Id,SDdt,iConfig);
#endif
#ifdef LOCAL_BC
if(doCorre_Local_BC==true && singleConfig==true)
  database.output_Corr_Local_BC(Id,SDdt,iConfig);
#endif
#ifdef LOCAL_SC
if(doCorre_Local_SC==true && singleConfig==true)
  database.output_Corr_Local_SC(Id,SDdt,iConfig);
#endif
#ifdef LOCAL_CR
if(doCorre_Local_CR==true && singleConfig==true)
  database.output_Corr_Local_CR(Id,SDdt,iConfig);
#endif
#ifdef LOCAL_BC_PBCXY
if(doCorre_Local_BC_PBCXY==true && singleConfig==true)
  database.output_Corr_Local_BC_PBCXY(Id,SDdt,iConfig);
#endif
```

```
#ifdef FSUR_PBCXY
if(doCorre_FSUR_PBCXY==true && singleConfig==true)
  database.output_Corr_FSUR_PBCXY(Id,SDdt,iConfig);
#endif
#ifdef CELL
if(doCellization==true)
  database.output_Cell(Id,iConfig,SDdt);
#endif

#ifdef EM
if(print_EMtm==true || print_EMhs==true)
  database.clearup();
#endif
#ifdef FLY
if(doCorre_Fly==true && singleConfig==true)            database.clearup();
#endif
#ifdef REAL
if(doCorre_Real==true && singleConfig==true)           database.clearup();
#endif
#ifdef LOCAL_SINGLE_LINE
if(doCorre_LocalSL==true && singleConfig==true)        database.clearup();
#endif
#ifdef LOCAL_BC
if(doCorre_Local_BC==true && singleConfig==true)       database.clearup();
#endif
#ifdef LOCAL_SC
if(doCorre_Local_SC==true && singleConfig==true)       database.clearup();
#endif
#ifdef LOCAL_CR
if(doCorre_Local_CR==true && singleConfig==true)       database.clearup();
#endif
#ifdef LOCAL_BC_PBCXY
if(doCorre_Local_BC_PBCXY==true && singleConfig==true) database.clearup();
#endif
#ifdef FSUR_PBCXY
if(doCorre_FSUR_PBCXY==true && singleConfig==true)     database.clearup();
#endif
#ifdef CELL
if(doCellization==true)                                database.clearup();
#endif

if(iConfig!=NConfig-1) {
  if(input.para().boundary!="sbc")
    { for(unsigned int i=0; i<NHybrid; i++) MC_moves(spins0,T); }
  else { for(unsigned int i=0; i<NHybrid; i++)
    MC_moves(spins0,T,surfaceLabel); }
  spins.assign(spins0.begin(),spins0.end());
```

```
    }
  } //iconfig

  if(singleConfig==false) {
    #ifdef FLY
    if(doCorre_Fly==true)       database.output_Corr_Intermd(time(NULL),SDdt);
    #endif
    #ifdef REAL
    if(doCorre_Real==true)      database.output_Corr_Real(time(NULL),SDdt);
    #endif
    #ifdef LOCAL_SINGLE_LINE
    if(doCorre_LocalSL==true)   database.output_Corr_LocalSL(time(NULL),SDdt);
    #endif
    #ifdef LOCAL_BC
    if(doCorre_Local_BC==true)  database.output_Corr_Local_BC(time(NULL),SDdt);
    #endif
    #ifdef LOCAL_SC
    if(doCorre_Local_SC==true)  database.output_Corr_Local_SC(time(NULL),SDdt);
    #endif
    #ifdef LOCAL_CR
    if(doCorre_Local_CR==true)  database.output_Corr_Local_CR(time(NULL),SDdt);
    #endif
    #ifdef LOCAL_BC_PBCXY
    if(doCorre_Local_BC_PBCXY==true)
      database.output_Corr_Local_BC_PBCXY(time(NULL),SDdt);
    #endif
    #ifdef FSUR_PBCXY
    if(doCorre_FSUR_PBCXY==true)
      database.output_Corr_FSUR_PBCXY(time(NULL),SDdt);
    #endif

    database.clearup();
  }
 }
}

template<class EHModel_D0, class EHModel_D, class AcceptProb, class RndGen,
         class Input, class DataBase, class SpinDynamics, class SpinContainer,
         class FieldType>
void Simulator<EHModel_D0,EHModel_D,AcceptProb,RndGen,Input,DataBase,SpinDynamics,
               SpinContainer,FieldType>
::MC_moves(SpinContainer& Spins, psimag::Real T)
{
 for(unsigned int j=0; j<input.para().NHeisenberg; j++)
   for(unsigned int k=0; k<Spins.size(); k++)       heisenberg(ran,0.,Spins,T);
 for(unsigned int j=0; j<input.para().NWolff;j++)   wolff(ran,0.,Spins,T);
 for(unsigned int j=0; j<input.para().NIsing;j++)        ising(ran,0.,Spins,T);
```

```
 for(unsigned int j=0; j<input.para().NOverRelax; j++) {
   for(unsigned int k=0; k<input.para().NSubLat; k++) {
     unsigned int iSub = static_cast<int>(input.para().NSubLat*ran()),
                  subsize = spindynamics.getSublat_index()[iSub].size();
     int p=overRelax(ran,0.,Spins,spindynamics.getSublat_index()[iSub],subsize);
   }
 }
}

template<class EHModel_D0, class EHModel_D, class AcceptProb, class RndGen,
         class Input, class DataBase, class SpinDynamics, class SpinContainer,
         class FieldType>
void Simulator<EHModel_D0,EHModel_D,AcceptProb,RndGen,Input,DataBase,
               SpinDynamics,SpinContainer,FieldType>
::MC_moves(SpinContainer& Spins, psimag::Real T,
           const std::vector<bool>& surfaceLabel)
{
 for(unsigned int j=0; j<input.para().NHeisenberg; j++)
   for(unsigned int k=0; k<Spins.size(); k++)
     heisenberg(ran,0.,Spins,T,surfaceLabel);
 for(unsigned int j=0; j<input.para().NOverRelax; j++) {
   for(unsigned int k=0; k<input.para().NSubLat; k++) {
     unsigned int iSub = static_cast<int>(input.para().NSubLat*ran()),
                  subsize = spindynamics.getSublat_index()[iSub].size();
     int p=overRelax(ran,0.,Spins,spindynamics.getSublat_index()[iSub],
                     subsize,surfaceLabel);
   }
 }
}

template<class EHModel_D0, class EHModel_D, class AcceptProb, class RndGen,
         class Input, class DataBase, class SpinDynamics, class SpinContainer,
         class FieldType>
void Simulator<EHModel_D0,EHModel_D,AcceptProb,RndGen,Input,DataBase,SpinDynamics,
               SpinContainer,FieldType>
::zPolarize_Iso(SpinContainer& Spins)
{
 psimag::Real m  = 0., mx = 0., my = 0., mz = 0.;
 psimag::Real zcax, zcay, zcaz, ycaz, ycax, ycay, xcax, xcay, xcaz;
 psimag::Real sx, sy, sz;;
 if(input.para().exchange_factor[0]>0.) {
   for(unsigned int i=0;i<Spins.size();i++) {
     mx += Spins[i][0];
     my += Spins[i][1];
     mz += Spins[i][2];
   }
 }
```

```
 else {
   for(unsigned int i=0;i<Spins.size();i++) {
     if(sublatticeLabel[i]==0) {
       mx += Spins[i][0];
       my += Spins[i][1];
       mz += Spins[i][2];
     }
     else {
       mx -= Spins[i][0];
       my -= Spins[i][1];
       mz -= Spins[i][2];
     }
   }
 }

 mx = mx/static_cast<double>(Spins.size());
 my = my/static_cast<double>(Spins.size());
 mz = mz/static_cast<double>(Spins.size());
 m  = sqrt(mx*mx+my*my+mz*mz);

 zcax = mx/m;
 zcay = my/m;
 zcaz = mz/m;

 ycaz = sqrt(1.-pow(zcaz,2.));
 ycax = -zcax*zcaz/ycaz;
 ycay = -zcay*zcaz/ycaz;

 xcax = -zcay/ycaz;
 xcay = zcax/ycaz;
 xcaz = 0.;

 for(unsigned int i=0;i<Spins.size();i++) {
   sx = xcax*Spins[i][0]+xcay*Spins[i][1]+xcaz*Spins[i][2];
   sy = ycax*Spins[i][0]+ycay*Spins[i][1]+ycaz*Spins[i][2];
   sz = zcax*Spins[i][0]+zcay*Spins[i][1]+zcaz*Spins[i][2];
   Spins[i][0] = sx;
   Spins[i][1] = sy;
   Spins[i][2] = sz;
 }
 return ;
}

template<class EHModel_D0, class EHModel_D, class AcceptProb, class RndGen,
         class Input, class DataBase, class SpinDynamics, class SpinContainer,
         class FieldType>
void Simulator<EHModel_D0,EHModel_D,AcceptProb,RndGen,Input,DataBase,
```

```
              SpinDynamics,SpinContainer,FieldType>
::zPolarize_Iso(SpinContainer& Spins, const std::vector<bool>& surfaceLabel)
{
 psimag::Real m  = 0., mx = 0., my = 0., mz = 0.;
 psimag::Real zcax, zcay, zcaz, ycaz, ycax, ycay, xcax, xcay, xcaz;
 psimag::Real sx, sy, sz;;
 if(input.para().exchange_factor[0]>0.) {
   for(unsigned int i=0;i<Spins.size();i++) {
     mx += Spins[i][0];
     my += Spins[i][1];
     mz += Spins[i][2];
   }
 }
 else {
   for(unsigned int i=0;i<Spins.size();i++) {
     if(sublatticeLabel[i]==0) {
       mx += Spins[i][0];
       my += Spins[i][1];
       mz += Spins[i][2];
     }
     else {
       mx -= Spins[i][0];
       my -= Spins[i][1];
       mz -= Spins[i][2];
     }
   }
 }

 mx = mx/static_cast<double>(Spins.size());
 my = my/static_cast<double>(Spins.size());
 mz = mz/static_cast<double>(Spins.size());
 m  = sqrt(mx*mx+my*my+mz*mz);

 zcax = mx/m;
 zcay = my/m;
 zcaz = mz/m;

 ycaz = sqrt(1.-pow(zcaz,2.));
 ycax = -zcax*zcaz/ycaz;
 ycay = -zcay*zcaz/ycaz;

 xcax = -zcay/ycaz;
 xcay = zcax/ycaz;
 xcaz = 0.;

 for(unsigned int i=0;i<Spins.size();i++) {
   if(!surfaceLabel[i]) {
```

```
     sx = xcax*Spins[i][0]+xcay*Spins[i][1]+xcaz*Spins[i][2];
     sy = ycax*Spins[i][0]+ycay*Spins[i][1]+ycaz*Spins[i][2];
     sz = zcax*Spins[i][0]+zcay*Spins[i][1]+zcaz*Spins[i][2];
     Spins[i][0] = sx;
     Spins[i][1] = sy;
     Spins[i][2] = sz;
   }
 }
 return ;
}

#endif
```

A.3.5 *correlation_Local_BC_PBCXY.h*

```
#ifndef CORRELATION_LOCAL_BC_PBCXY_H
#define CORRELATION_LOCAL_BC_PBCXY_H

#include <fstream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <iomanip>
#include <string>
#include <sstream>
#include <cstdlib>
#include <cmath>

#include "Real.h"
#include "ContainerUtil.h"
#include "LatticeIndexMapper.h"


template<class Input, class SC, class DataContainer, class SpinContainer>
class Correlation_Local_BC_PBCXY
{
 typedef psimag::Real                                real;

 public:
         virtual ~Correlation_Local_BC_PBCXY() = 0;
         virtual void insert(const SpinContainer&, const unsigned int) = 0;
         virtual void doCorrelation_Local_BC() = 0;
         virtual void print(const long int, const real, const int) = 0;
         virtual void clearup() = 0;
 protected:
         template<class OS> void printTitle(const Input&, OS&, real, int) const;
```

```
};

/*************************************************************************/

template<class Input, class SC, class DataContainer, class SpinContainer>
Correlation_Local_BC_PBCXY<Input,SC,DataContainer,SpinContainer>::
~Correlation_Local_BC_PBCXY() {}

template<class Input, class SC, class DataContainer, class SpinContainer>
template<class OS>
void Correlation_Local_BC_PBCXY<Input,SC,DataContainer,SpinContainer>::
printTitle(const Input& in, OS& os, real SDdt, int iConfig) const
{
 in.print_input(os);
 std::ostringstream   oss;
 oss<<iConfig;
 os<<"#\n# Running For iConfig = "<<(iConfig==-1?"N/A":oss.str())
   <<" of NConfig = "<<in.para().NConfig<<" with SDdt = ";
 if(SDdt==-1) os<<"N/A"<<std::endl;
 else os<<SDdt<<std::endl;
}

/////////////////////////////////////////////////////////////////////////

template<class Input, class SC, class DataContainer, class SpinContainer>
class Correlation_Local_BC_PBCXY_EvenL :
public Correlation_Local_BC_PBCXY<Input,SC,DataContainer,SpinContainer>
{
 typedef psimag::Real                                real;
 typedef std::vector<real>                           VContainer;
 typedef std::vector<VContainer>                     VVContainer;
 typedef std::vector<VVContainer>                    VVVContainer;
 typedef std::vector<std::vector<DataContainer> >    BCS_IndexContainer;

 public:
        explicit Correlation_Local_BC_PBCXY_EvenL(const Input&, const SC&);
        virtual ~Correlation_Local_BC_PBCXY_EvenL();

        virtual void insert(const SpinContainer&, const unsigned int);
        virtual void doCorrelation_Local_BC();
        virtual void print(const long int, const real, const int);
        virtual void clearup();

 private:
     Correlation_Local_BC_PBCXY_EvenL(const Correlation_Local_BC_PBCXY_EvenL&);
     Correlation_Local_BC_PBCXY_EvenL& operator=(
                                    const Correlation_Local_BC_PBCXY_EvenL&);
```

```
      void findBCSIndex(BCS_IndexContainer&, BCS_IndexContainer&);
      void spinSumOnPlane_100(const SpinContainer&, VVContainer&, VVContainer&,
                      VVContainer&, VVContainer&, VVContainer&, VVContainer&);

         const Input&             in;
         const DataContainer&     L, sublattice_label;
         const unsigned int       N, PL_P, PL_F, BCPL_P, BCPL_F, nt, tc, tintv;

         BCS_IndexContainer       BCS_Index_P, BCS_Index_F;

         VVContainer   aveBCS_X_P, aveBCS_Y_P, aveBCS_Z_P,
                       aveBCS_X_F, aveBCS_Y_F, aveBCS_Z_F;
         VVVContainer  BCS_Time_X_P, BCS_Time_Y_P, BCS_Time_Z_P,
                       BCS_Time_X_F, BCS_Time_Y_F, BCS_Time_Z_F;

         VVContainer   SSOP0_XY_100, SSOP0_YZ_100, SSOP0_ZX_100,
                       SSOP1_XY_100, SSOP1_YZ_100, SSOP1_ZX_100,

                       aveSSOP0_XY_100, aveSSOP0_YZ_100, aveSSOP0_ZX_100,
                       aveSSOP1_XY_100, aveSSOP1_YZ_100, aveSSOP1_ZX_100;

         VVVContainer  SSOP0_XY_Time_100, SSOP0_YZ_Time_100, SSOP0_ZX_Time_100,
                       SSOP1_XY_Time_100, SSOP1_YZ_Time_100, SSOP1_ZX_Time_100;

         VVContainer   Cx_100_P, Cy_100_P, Cz_100_P,
                       Cx_100_F, Cy_100_F, Cz_100_F;
};

/*****************************************************************************/

template<class Input, class SC, class DataContainer, class SpinContainer>
Correlation_Local_BC_PBCXY_EvenL<Input,SC,DataContainer,SpinContainer>::
Correlation_Local_BC_PBCXY_EvenL(const Input& input, const SC& sc)
:in(input),
 L(in.para().L),
 sublattice_label(sc.sublattice_label()),
 N(sc.number_of_sites()),
 PL_P(L[1]),
 PL_F(L[0]),
 BCPL_P(L[1]/2-1),
 BCPL_F(L[0]/2-1),
 nt(in.para().nt),
 tc(in.para().tc),
 tintv(in.para().tintv)
{
 findBCSIndex(BCS_Index_P,BCS_Index_F);
```

```
aveBCS_X_P.resize(4); aveBCS_Y_P.resize(4); aveBCS_Z_P.resize(4);
aveBCS_X_F.resize(2); aveBCS_Y_F.resize(2); aveBCS_Z_F.resize(2);
for(unsigned int p=0; p<aveBCS_X_P.size(); p++)
{ aveBCS_X_P[p].resize(2,0.); aveBCS_Y_P[p].resize(2,0.);
  aveBCS_Z_P[p].resize(2,0.); }
for(unsigned int p=0; p<aveBCS_X_F.size(); p++)
{ aveBCS_X_F[p].resize(2,0.); aveBCS_Y_F[p].resize(2,0.);
  aveBCS_Z_F[p].resize(2,0.); }

BCS_Time_X_P.resize(nt+1);BCS_Time_Y_P.resize(nt+1);BCS_Time_Z_P.resize(nt+1);
BCS_Time_X_F.resize(nt+1);BCS_Time_Y_F.resize(nt+1);BCS_Time_Z_F.resize(nt+1);
for(unsigned int t=0; t<nt+1; t++) {
  BCS_Time_X_P[t].resize(4);BCS_Time_Y_P[t].resize(4);BCS_Time_Z_P[t].resize(4);
  BCS_Time_X_F[t].resize(2);BCS_Time_Y_F[t].resize(2);BCS_Time_Z_F[t].resize(2);
}
for(unsigned int t=0; t<nt+1; t++) {
  for(unsigned int p=0; p<4; p++)
  { BCS_Time_X_P[t][p].resize(2,0.); BCS_Time_Y_P[t][p].resize(2,0.);
     BCS_Time_Z_P[t][p].resize(2,0.); }
  for(unsigned int p=0; p<2; p++)
  { BCS_Time_X_F[t][p].resize(2,0.); BCS_Time_Y_F[t][p].resize(2,0.);
    BCS_Time_Z_F[t][p].resize(2,0.); }
}

SSOP0_XY_100.resize(PL_F); SSOP0_YZ_100.resize(PL_P); SSOP0_ZX_100.resize(PL_P);
SSOP1_XY_100.resize(PL_F); SSOP1_YZ_100.resize(PL_P); SSOP1_ZX_100.resize(PL_P);
for(unsigned int r=0; r<PL_F; r++) {
  SSOP0_XY_100[r].resize(3,0.);
  SSOP1_XY_100[r].resize(3,0.);
}
for(unsigned int r=0; r<PL_P; r++) {
  SSOP0_YZ_100[r].resize(3,0.); SSOP0_ZX_100[r].resize(3,0.);
  SSOP1_YZ_100[r].resize(3,0.); SSOP1_ZX_100[r].resize(3,0.);
}

aveSSOP0_XY_100.resize(PL_F); aveSSOP0_YZ_100.resize(PL_P);
aveSSOP0_ZX_100.resize(PL_P);
aveSSOP1_XY_100.resize(PL_F); aveSSOP1_YZ_100.resize(PL_P);
aveSSOP1_ZX_100.resize(PL_P);
for(unsigned int r=0; r<PL_F; r++) {
  aveSSOP0_XY_100[r].resize(3,0.);
  aveSSOP1_XY_100[r].resize(3,0.);
}
for(unsigned int r=0; r<PL_P; r++) {
  aveSSOP0_YZ_100[r].resize(3,0.); aveSSOP0_ZX_100[r].resize(3,0.);
  aveSSOP1_YZ_100[r].resize(3,0.); aveSSOP1_ZX_100[r].resize(3,0.);
```

```
 }

 SSOP0_XY_Time_100.resize(nt+1); SSOP0_YZ_Time_100.resize(nt+1);
 SSOP0_ZX_Time_100.resize(nt+1);
 SSOP1_XY_Time_100.resize(nt+1); SSOP1_YZ_Time_100.resize(nt+1);
 SSOP1_ZX_Time_100.resize(nt+1);
 for(unsigned int t=0; t<nt+1; t++) {
   SSOP0_XY_Time_100[t].resize(PL_F); SSOP0_YZ_Time_100[t].resize(PL_P);
   SSOP0_ZX_Time_100[t].resize(PL_P);
   SSOP1_XY_Time_100[t].resize(PL_F); SSOP1_YZ_Time_100[t].resize(PL_P);
   SSOP1_ZX_Time_100[t].resize(PL_P);
 }
 for(unsigned int t=0; t<nt+1; t++) {
   for(unsigned int r=0; r<PL_F; r++) {
     SSOP0_XY_Time_100[t][r].resize(3,0.);
     SSOP1_XY_Time_100[t][r].resize(3,0.);
   }
 }
 for(unsigned int t=0; t<nt+1; t++) {
   for(unsigned int r=0; r<PL_P; r++) {
     SSOP0_YZ_Time_100[t][r].resize(3,0.); SSOP0_ZX_Time_100[t][r].resize(3,0.);
     SSOP1_YZ_Time_100[t][r].resize(3,0.); SSOP1_ZX_Time_100[t][r].resize(3,0.);
   }
 }

 Cx_100_P.resize(PL_P/2+1); Cy_100_P.resize(PL_P/2+1); Cz_100_P.resize(PL_P/2+1);
 Cx_100_F.resize(PL_F/2+1); Cy_100_F.resize(PL_F/2+1); Cz_100_F.resize(PL_F/2+1);
 for(unsigned int r=0; r<Cx_100_P.size(); r++) {
   Cx_100_P[r].resize(tc+1,0.); Cy_100_P[r].resize(tc+1,0.);
   Cz_100_P[r].resize(tc+1,0.);
 }
 for(unsigned int r=0; r<Cx_100_F.size(); r++) {
   Cx_100_F[r].resize(tc+1,0.); Cy_100_F[r].resize(tc+1,0.);
   Cz_100_F[r].resize(tc+1,0.);
 }
}

template<class Input, class SC, class DataContainer, class SpinContainer>
Correlation_Local_BC_PBCXY_EvenL<Input,SC,DataContainer,SpinContainer>::
~Correlation_Local_BC_PBCXY_EvenL() {}

template<class Input, class SC, class DataContainer, class SpinContainer>
void Correlation_Local_BC_PBCXY_EvenL<Input,SC,DataContainer,SpinContainer>::
insert(const SpinContainer& spins, const unsigned int SDtime)
{
 for(unsigned int p=0; p<4; p++) {
   for(unsigned s=0; s<2; s++) {
```

```
    real sumBCS_X_P = 0.,
         sumBCS_Y_P = 0.,
         sumBCS_Z_P = 0.;
    for(unsigned int i=0; i<BCS_Index_P[p][s].size(); i++) {
      sumBCS_X_P += spins[BCS_Index_P[p][s][i]][0];
      sumBCS_Y_P += spins[BCS_Index_P[p][s][i]][1];
      sumBCS_Z_P += spins[BCS_Index_P[p][s][i]][2];
    }
    BCS_Time_X_P[SDtime][p][s] = sumBCS_X_P /
                                 static_cast<real>(BCS_Index_P[p][s].size());
    BCS_Time_Y_P[SDtime][p][s] = sumBCS_Y_P /
                                 static_cast<real>(BCS_Index_P[p][s].size());
    BCS_Time_Z_P[SDtime][p][s] = sumBCS_Z_P /
                                 static_cast<real>(BCS_Index_P[p][s].size());
  }
}
for(unsigned int p=0; p<2; p++) {
  for(unsigned s=0; s<2; s++) {
    real sumBCS_X_F = 0.,
         sumBCS_Y_F = 0.,
         sumBCS_Z_F = 0.;
    for(unsigned int i=0; i<BCS_Index_F[p][s].size(); i++) {
      sumBCS_X_F += spins[BCS_Index_F[p][s][i]][0];
      sumBCS_Y_F += spins[BCS_Index_F[p][s][i]][1];
      sumBCS_Z_F += spins[BCS_Index_F[p][s][i]][2];
    }
    BCS_Time_X_F[SDtime][p][s] = sumBCS_X_F /
                                 static_cast<real>(BCS_Index_F[p][s].size());
    BCS_Time_Y_F[SDtime][p][s] = sumBCS_Y_F /
                                 static_cast<real>(BCS_Index_F[p][s].size());
    BCS_Time_Z_F[SDtime][p][s] = sumBCS_Z_F /
                                 static_cast<real>(BCS_Index_F[p][s].size());
  }
}

spinSumOnPlane_100(spins,SSOP0_XY_100,SSOP0_YZ_100,SSOP0_ZX_100,SSOP1_XY_100,
                   SSOP1_YZ_100,SSOP1_ZX_100);

std::copy(SSOP0_XY_100.begin(),SSOP0_XY_100.end(),
          SSOP0_XY_Time_100[SDtime].begin());
std::copy(SSOP0_YZ_100.begin(),SSOP0_YZ_100.end(),
          SSOP0_YZ_Time_100[SDtime].begin());
std::copy(SSOP0_ZX_100.begin(),SSOP0_ZX_100.end(),
          SSOP0_ZX_Time_100[SDtime].begin());
std::copy(SSOP1_XY_100.begin(),SSOP1_XY_100.end(),
          SSOP1_XY_Time_100[SDtime].begin());
std::copy(SSOP1_YZ_100.begin(),SSOP1_YZ_100.end(),
```

```
          SSOP1_YZ_Time_100[SDtime].begin());
std::copy(SSOP1_ZX_100.begin(),SSOP1_ZX_100.end(),
          SSOP1_ZX_Time_100[SDtime].begin());

if(SDtime==nt) {
  for(unsigned int p=0; p<4; p++) {
    std::fill(aveBCS_X_P[p].begin(),aveBCS_X_P[p].end(),0.);
    std::fill(aveBCS_Y_P[p].begin(),aveBCS_Y_P[p].end(),0.);
    std::fill(aveBCS_Z_P[p].begin(),aveBCS_Z_P[p].end(),0.);
  }
  for(unsigned int p=0; p<4; p++) {
    for(unsigned int s=0; s<2; s++) {
      for(unsigned int it=0; it<nt+1; it++) {
        aveBCS_X_P[p][s] += BCS_Time_X_P[it][p][s];
        aveBCS_Y_P[p][s] += BCS_Time_Y_P[it][p][s];
        aveBCS_Z_P[p][s] += BCS_Time_Z_P[it][p][s];
      }
    }
  }
  for(unsigned int p=0; p<4; p++) {
    for(unsigned int s=0; s<2; s++) {
      aveBCS_X_P[p][s] /= static_cast<real>(nt+1);
      aveBCS_Y_P[p][s] /= static_cast<real>(nt+1);
      aveBCS_Z_P[p][s] /= static_cast<real>(nt+1);
    }
  }

  for(unsigned int p=0; p<2; p++) {
    std::fill(aveBCS_X_F[p].begin(),aveBCS_X_F[p].end(),0.);
    std::fill(aveBCS_Y_F[p].begin(),aveBCS_Y_F[p].end(),0.);
    std::fill(aveBCS_Z_F[p].begin(),aveBCS_Z_F[p].end(),0.);
  }
  for(unsigned int p=0; p<2; p++) {
    for(unsigned int s=0; s<2; s++) {
      for(unsigned int it=0; it<nt+1; it++) {
        aveBCS_X_F[p][s] += BCS_Time_X_F[it][p][s];
        aveBCS_Y_F[p][s] += BCS_Time_Y_F[it][p][s];
        aveBCS_Z_F[p][s] += BCS_Time_Z_F[it][p][s];
      }
    }
  }
  for(unsigned int p=0; p<2; p++) {
    for(unsigned int s=0; s<2; s++) {
      aveBCS_X_F[p][s] /= static_cast<real>(nt+1);
      aveBCS_Y_F[p][s] /= static_cast<real>(nt+1);
      aveBCS_Z_F[p][s] /= static_cast<real>(nt+1);
    }
```

```
    }

    for(unsigned int r=0; r<PL_F; r++) {
      std::fill(aveSSOP0_XY_100[r].begin(),aveSSOP0_XY_100[r].end(),0.);
      std::fill(aveSSOP1_XY_100[r].begin(),aveSSOP1_XY_100[r].end(),0.);
    }
    for(unsigned int r=0; r<PL_F; r++) {
      for(unsigned int c=0; c<3; c++) {
        for(unsigned int it=0; it<nt+1; it++) {
          aveSSOP0_XY_100[r][c] += SSOP0_XY_Time_100[it][r][c];
          aveSSOP1_XY_100[r][c] += SSOP1_XY_Time_100[it][r][c];
        }
      }
    }
    for(unsigned int r=0; r<PL_F; r++) {
      for(unsigned int c=0; c<3; c++) {
        aveSSOP0_XY_100[r][c] /= static_cast<real>(nt+1);
        aveSSOP1_XY_100[r][c] /= static_cast<real>(nt+1);
      }
    }
    for(unsigned int r=0; r<PL_P; r++) {
      std::fill(aveSSOP0_YZ_100[r].begin(),aveSSOP0_YZ_100[r].end(),0.);
      std::fill(aveSSOP0_ZX_100[r].begin(),aveSSOP0_ZX_100[r].end(),0.);
      std::fill(aveSSOP1_YZ_100[r].begin(),aveSSOP1_YZ_100[r].end(),0.);
      std::fill(aveSSOP1_ZX_100[r].begin(),aveSSOP1_ZX_100[r].end(),0.);
    }
    for(unsigned int r=0; r<PL_P; r++) {
      for(unsigned int c=0; c<3; c++) {
        for(unsigned int it=0; it<nt+1; it++) {
          aveSSOP0_YZ_100[r][c] += SSOP0_YZ_Time_100[it][r][c];
          aveSSOP0_ZX_100[r][c] += SSOP0_ZX_Time_100[it][r][c];
          aveSSOP1_YZ_100[r][c] += SSOP1_YZ_Time_100[it][r][c];
          aveSSOP1_ZX_100[r][c] += SSOP1_ZX_Time_100[it][r][c];
        }
      }
    }
    for(unsigned int r=0; r<PL_P; r++) {
      for(unsigned int c=0; c<3; c++) {
        aveSSOP0_YZ_100[r][c] /= static_cast<real>(nt+1);
        aveSSOP0_ZX_100[r][c] /= static_cast<real>(nt+1);
        aveSSOP1_YZ_100[r][c] /= static_cast<real>(nt+1);
        aveSSOP1_ZX_100[r][c] /= static_cast<real>(nt+1);
      }
    }
  }
}
```

```
template<class Input, class SC, class DataContainer, class SpinContainer>
void Correlation_Local_BC_PBCXY_EvenL<Input,SC,DataContainer,SpinContainer>::
doCorrelation_Local_BC()
{
 for(unsigned int t=0; t<tc+1; t++) {
   real ave_P = 1./static_cast<real>(nt-t+1)/8.,
        ave_F = 1./static_cast<real>(nt-t+1)/4.;
   for(unsigned int it=0; it<nt-t+1; it++) {
     // correlation in PBCXY direction
     // starting @ sub0
     Cx_100_P[0][t] +=
     (BCS_Time_X_P[it][0][0]*SSOP0_YZ_Time_100[it+t][BCPL_P][0] +
      BCS_Time_X_P[it][1][0]*SSOP0_YZ_Time_100[it+t][BCPL_P+1][0] +
      BCS_Time_X_P[it][2][0]*SSOP0_ZX_Time_100[it+t][BCPL_P][0] +
      BCS_Time_X_P[it][3][0]*SSOP0_ZX_Time_100[it+t][BCPL_P+1][0])*ave_P -
     (aveBCS_X_P[0][0]*aveSSOP0_YZ_100[BCPL_P][0] +
      aveBCS_X_P[1][0]*aveSSOP0_YZ_100[BCPL_P+1][0] +
      aveBCS_X_P[2][0]*aveSSOP0_ZX_100[BCPL_P][0] +
      aveBCS_X_P[3][0]*aveSSOP0_ZX_100[BCPL_P+1][0])*ave_P;
     Cy_100_P[0][t] +=
     (BCS_Time_Y_P[it][0][0]*SSOP0_YZ_Time_100[it+t][BCPL_P][1] +
      BCS_Time_Y_P[it][1][0]*SSOP0_YZ_Time_100[it+t][BCPL_P+1][1] +
      BCS_Time_Y_P[it][2][0]*SSOP0_ZX_Time_100[it+t][BCPL_P][1] +
      BCS_Time_Y_P[it][3][0]*SSOP0_ZX_Time_100[it+t][BCPL_P+1][1])*ave_P -
     (aveBCS_Y_P[0][0]*aveSSOP0_YZ_100[BCPL_P][1] +
      aveBCS_Y_P[1][0]*aveSSOP0_YZ_100[BCPL_P+1][1] +
      aveBCS_Y_P[2][0]*aveSSOP0_ZX_100[BCPL_P][1] +
      aveBCS_Y_P[3][0]*aveSSOP0_ZX_100[BCPL_P+1][1])*ave_P;
     Cz_100_P[0][t] +=
    (BCS_Time_Z_P[it][0][0]*SSOP0_YZ_Time_100[it+t][BCPL_P][2] +
     BCS_Time_Z_P[it][1][0]*SSOP0_YZ_Time_100[it+t][BCPL_P+1][2] +
     BCS_Time_Z_P[it][2][0]*SSOP0_ZX_Time_100[it+t][BCPL_P][2] +
     BCS_Time_Z_P[it][3][0]*SSOP0_ZX_Time_100[it+t][BCPL_P+1][2])*ave_P -
    (aveBCS_Z_P[0][0]*aveSSOP0_YZ_100[BCPL_P][2] +
     aveBCS_Z_P[1][0]*aveSSOP0_YZ_100[BCPL_P+1][2] +
     aveBCS_Z_P[2][0]*aveSSOP0_ZX_100[BCPL_P][2] +
     aveBCS_Z_P[3][0]*aveSSOP0_ZX_100[BCPL_P+1][2])*ave_P;
     // starting @ sub1
     Cx_100_P[0][t] +=
    (BCS_Time_X_P[it][0][1]*SSOP1_YZ_Time_100[it+t][BCPL_P][0] +
     BCS_Time_X_P[it][1][1]*SSOP1_YZ_Time_100[it+t][BCPL_P+1][0] +
     BCS_Time_X_P[it][2][1]*SSOP1_ZX_Time_100[it+t][BCPL_P][0] +
     BCS_Time_X_P[it][3][1]*SSOP1_ZX_Time_100[it+t][BCPL_P+1][0])*ave_P -
    (aveBCS_X_P[0][1]*aveSSOP1_YZ_100[BCPL_P][0] +
     aveBCS_X_P[1][1]*aveSSOP1_YZ_100[BCPL_P+1][0] +
     aveBCS_X_P[2][1]*aveSSOP1_ZX_100[BCPL_P][0] +
     aveBCS_X_P[3][1]*aveSSOP1_ZX_100[BCPL_P+1][0])*ave_P;
```

```
 Cy_100_P[0][t] +=
(BCS_Time_Y_P[it][0][1]*SSOP1_YZ_Time_100[it+t][BCPL_P][1] +
 BCS_Time_Y_P[it][1][1]*SSOP1_YZ_Time_100[it+t][BCPL_P+1][1] +
 BCS_Time_Y_P[it][2][1]*SSOP1_ZX_Time_100[it+t][BCPL_P][1] +
 BCS_Time_Y_P[it][3][1]*SSOP1_ZX_Time_100[it+t][BCPL_P+1][1])*ave_P -
(aveBCS_Y_P[0][1]*aveSSOP1_YZ_100[BCPL_P][1] +
 aveBCS_Y_P[1][1]*aveSSOP1_YZ_100[BCPL_P+1][1] +
 aveBCS_Y_P[2][1]*aveSSOP1_ZX_100[BCPL_P][1] +
 aveBCS_Y_P[3][1]*aveSSOP1_ZX_100[BCPL_P+1][1])*ave_P;
 Cz_100_P[0][t] +=
(BCS_Time_Z_P[it][0][1]*SSOP1_YZ_Time_100[it+t][BCPL_P][2] +
 BCS_Time_Z_P[it][1][1]*SSOP1_YZ_Time_100[it+t][BCPL_P+1][2] +
 BCS_Time_Z_P[it][2][1]*SSOP1_ZX_Time_100[it+t][BCPL_P][2] +
 BCS_Time_Z_P[it][3][1]*SSOP1_ZX_Time_100[it+t][BCPL_P+1][2])*ave_P -
(aveBCS_Z_P[0][1]*aveSSOP1_YZ_100[BCPL_P][2] +
 aveBCS_Z_P[1][1]*aveSSOP1_YZ_100[BCPL_P+1][2] +
 aveBCS_Z_P[2][1]*aveSSOP1_ZX_100[BCPL_P][2] +
 aveBCS_Z_P[3][1]*aveSSOP1_ZX_100[BCPL_P+1][2])*ave_P;

 // correlation in FBCZ direction
 // starting @ sub0
 Cx_100_F[0][t] +=
(BCS_Time_X_F[it][0][0]*SSOP0_XY_Time_100[it+t][BCPL_F][0] +
 BCS_Time_X_F[it][1][0]*SSOP0_XY_Time_100[it+t][BCPL_F+1][0])*ave_F -
(aveBCS_X_F[0][0]*aveSSOP0_XY_100[BCPL_F][0] +
 aveBCS_X_F[1][0]*aveSSOP0_XY_100[BCPL_F+1][0])*ave_F;
 Cy_100_F[0][t] +=
(BCS_Time_Y_F[it][0][0]*SSOP0_XY_Time_100[it+t][BCPL_F][1] +
 BCS_Time_Y_F[it][1][0]*SSOP0_XY_Time_100[it+t][BCPL_F+1][1])*ave_F -
(aveBCS_Y_F[0][0]*aveSSOP0_XY_100[BCPL_F][1] +
 aveBCS_Y_F[1][0]*aveSSOP0_XY_100[BCPL_F+1][1])*ave_F;
 Cz_100_F[0][t] +=
(BCS_Time_Z_F[it][0][0]*SSOP0_XY_Time_100[it+t][BCPL_F][2] +
 BCS_Time_Z_F[it][1][0]*SSOP0_XY_Time_100[it+t][BCPL_F+1][2])*ave_F -
(aveBCS_Z_F[0][0]*aveSSOP0_XY_100[BCPL_F][2] +
 aveBCS_Z_F[1][0]*aveSSOP0_XY_100[BCPL_F+1][2])*ave_F;
 // starting @ sub1
 Cx_100_F[0][t] +=
(BCS_Time_X_F[it][0][1]*SSOP1_XY_Time_100[it+t][BCPL_F][0] +
 BCS_Time_X_F[it][1][1]*SSOP1_XY_Time_100[it+t][BCPL_F+1][0])*ave_F -
(aveBCS_X_F[0][1]*aveSSOP1_XY_100[BCPL_F][0] +
 aveBCS_X_F[1][1]*aveSSOP1_XY_100[BCPL_F+1][0])*ave_F;
 Cy_100_F[0][t] +=
(BCS_Time_Y_F[it][0][1]*SSOP1_XY_Time_100[it+t][BCPL_F][1] +
 BCS_Time_Y_F[it][1][1]*SSOP1_XY_Time_100[it+t][BCPL_F+1][1])*ave_F -
(aveBCS_Y_F[0][1]*aveSSOP1_XY_100[BCPL_F][1] +
 aveBCS_Y_F[1][1]*aveSSOP1_XY_100[BCPL_F+1][1])*ave_F;
```

```
    Cz_100_F[0][t] +=
   (BCS_Time_Z_F[it][0][1]*SSOP1_XY_Time_100[it+t][BCPL_F][2] +
    BCS_Time_Z_F[it][1][1]*SSOP1_XY_Time_100[it+t][BCPL_F+1][2])*ave_F -
   (aveBCS_Z_F[0][1]*aveSSOP1_XY_100[BCPL_F][2] +
    aveBCS_Z_F[1][1]*aveSSOP1_XY_100[BCPL_F+1][2])*ave_F;
  }
}

// correlation in PBCXY direction
for(unsigned int r=1; r<Cx_100_P.size(); r++) {
  unsigned int r1 = BCPL_P+r,
               r2 = BCPL_P+1-r;
  for(unsigned int t=0; t<tc+1; t++) {
    real ave_P = 1./static_cast<real>(nt-t+1)/8.;
    for(unsigned int it=0; it<nt-t+1; it++) {

      // starting @ sub0
      Cx_100_P[r][t] +=
     (BCS_Time_X_P[it][0][0]*SSOP0_YZ_Time_100[it+t][r1][0] +
      BCS_Time_X_P[it][1][0]*SSOP0_YZ_Time_100[it+t][r2][0] +
      BCS_Time_X_P[it][2][0]*SSOP0_ZX_Time_100[it+t][r1][0] +
      BCS_Time_X_P[it][3][0]*SSOP0_ZX_Time_100[it+t][r2][0])*ave_P -
     (aveBCS_X_P[0][0]*aveSSOP0_YZ_100[r1][0] +
      aveBCS_X_P[1][0]*aveSSOP0_YZ_100[r2][0] +
      aveBCS_X_P[2][0]*aveSSOP0_ZX_100[r1][0] +
      aveBCS_X_P[3][0]*aveSSOP0_ZX_100[r2][0])*ave_P;
      Cy_100_P[r][t] +=
     (BCS_Time_Y_P[it][0][0]*SSOP0_YZ_Time_100[it+t][r1][1] +
      BCS_Time_Y_P[it][1][0]*SSOP0_YZ_Time_100[it+t][r2][1] +
      BCS_Time_Y_P[it][2][0]*SSOP0_ZX_Time_100[it+t][r1][1] +
      BCS_Time_Y_P[it][3][0]*SSOP0_ZX_Time_100[it+t][r2][1])*ave_P -
     (aveBCS_Y_P[0][0]*aveSSOP0_YZ_100[r1][1] +
      aveBCS_Y_P[1][0]*aveSSOP0_YZ_100[r2][1] +
      aveBCS_Y_P[2][0]*aveSSOP0_ZX_100[r1][1] +
      aveBCS_Y_P[3][0]*aveSSOP0_ZX_100[r2][1])*ave_P;
      Cz_100_P[r][t] +=
     (BCS_Time_Z_P[it][0][0]*SSOP0_YZ_Time_100[it+t][r1][2] +
      BCS_Time_Z_P[it][1][0]*SSOP0_YZ_Time_100[it+t][r2][2] +
      BCS_Time_Z_P[it][2][0]*SSOP0_ZX_Time_100[it+t][r1][2] +
      BCS_Time_Z_P[it][3][0]*SSOP0_ZX_Time_100[it+t][r2][2])*ave_P -
     (aveBCS_Z_P[0][0]*aveSSOP0_YZ_100[r1][2] +
      aveBCS_Z_P[1][0]*aveSSOP0_YZ_100[r2][2] +
      aveBCS_Z_P[2][0]*aveSSOP0_ZX_100[r1][2] +
      aveBCS_Z_P[3][0]*aveSSOP0_ZX_100[r2][2])*ave_P;
      // starting @ sub1
      Cx_100_P[r][t] +=
     (BCS_Time_X_P[it][0][1]*SSOP1_YZ_Time_100[it+t][r1][0] +
```

```
      BCS_Time_X_P[it][1][1]*SSOP1_YZ_Time_100[it+t][r2][0] +
      BCS_Time_X_P[it][2][1]*SSOP1_ZX_Time_100[it+t][r1][0] +
      BCS_Time_X_P[it][3][1]*SSOP1_ZX_Time_100[it+t][r2][0])*ave_P -
     (aveBCS_X_P[0][1]*aveSSOP1_YZ_100[r1][0] +
      aveBCS_X_P[1][1]*aveSSOP1_YZ_100[r2][0] +
      aveBCS_X_P[2][1]*aveSSOP1_ZX_100[r1][0] +
      aveBCS_X_P[3][1]*aveSSOP1_ZX_100[r2][0])*ave_P;
      Cy_100_P[r][t] +=
     (BCS_Time_Y_P[it][0][1]*SSOP1_YZ_Time_100[it+t][r1][1] +
      BCS_Time_Y_P[it][1][1]*SSOP1_YZ_Time_100[it+t][r2][1] +
      BCS_Time_Y_P[it][2][1]*SSOP1_ZX_Time_100[it+t][r1][1] +
      BCS_Time_Y_P[it][3][1]*SSOP1_ZX_Time_100[it+t][r2][1])*ave_P -
     (aveBCS_Y_P[0][1]*aveSSOP1_YZ_100[r1][1] +
      aveBCS_Y_P[1][1]*aveSSOP1_YZ_100[r2][1] +
      aveBCS_Y_P[2][1]*aveSSOP1_ZX_100[r1][1] +
      aveBCS_Y_P[3][1]*aveSSOP1_ZX_100[r2][1])*ave_P;
      Cz_100_P[r][t] +=
     (BCS_Time_Z_P[it][0][1]*SSOP1_YZ_Time_100[it+t][r1][2] +
      BCS_Time_Z_P[it][1][1]*SSOP1_YZ_Time_100[it+t][r2][2] +
      BCS_Time_Z_P[it][2][1]*SSOP1_ZX_Time_100[it+t][r1][2] +
      BCS_Time_Z_P[it][3][1]*SSOP1_ZX_Time_100[it+t][r2][2])*ave_P -
     (aveBCS_Z_P[0][1]*aveSSOP1_YZ_100[r1][2] +
      aveBCS_Z_P[1][1]*aveSSOP1_YZ_100[r2][2] +
      aveBCS_Z_P[2][1]*aveSSOP1_ZX_100[r1][2] +
      aveBCS_Z_P[3][1]*aveSSOP1_ZX_100[r2][2])*ave_P;
    }
  }
}

// correlation in FBCZ direction
for(unsigned int r=1; r<Cx_100_F.size(); r++) {
  unsigned int r1 = BCPL_F+r,
               r2 = BCPL_F+1-r;
  for(unsigned int t=0; t<tc+1; t++) {
    real ave_F = 1./static_cast<real>(nt-t+1)/4.;
    for(unsigned int it=0; it<nt-t+1; it++) {

      // starting @ sub0
      Cx_100_F[r][t] +=
     (BCS_Time_X_F[it][0][0]*SSOP0_XY_Time_100[it+t][r1][0] +
      BCS_Time_X_F[it][1][0]*SSOP0_XY_Time_100[it+t][r2][0])*ave_F -
     (aveBCS_X_F[0][0]*aveSSOP0_XY_100[r1][0] +
      aveBCS_X_F[1][0]*aveSSOP0_XY_100[r2][0])*ave_F;
      Cy_100_F[r][t] +=
     (BCS_Time_Y_F[it][0][0]*SSOP0_XY_Time_100[it+t][r1][1] +
      BCS_Time_Y_F[it][1][0]*SSOP0_XY_Time_100[it+t][r2][1])*ave_F -
     (aveBCS_Y_F[0][0]*aveSSOP0_XY_100[r1][1] +
```

```
       aveBCS_Y_F[1][0]*aveSSOP0_XY_100[r2][1])*ave_F;
       Cz_100_F[r][t] +=
      (BCS_Time_Z_F[it][0][0]*SSOP0_XY_Time_100[it+t][r1][2] +
       BCS_Time_Z_F[it][1][0]*SSOP0_XY_Time_100[it+t][r2][2])*ave_F -
      (aveBCS_Z_F[0][0]*aveSSOP0_XY_100[r1][2] +
       aveBCS_Z_F[1][0]*aveSSOP0_XY_100[r2][2])*ave_F;
       // starting @ sub1
       Cx_100_F[r][t] +=
      (BCS_Time_X_F[it][0][1]*SSOP1_XY_Time_100[it+t][r1][0] +
       BCS_Time_X_F[it][1][1]*SSOP1_XY_Time_100[it+t][r2][0])*ave_F -
      (aveBCS_X_F[0][1]*aveSSOP1_XY_100[r1][0] +
       aveBCS_X_F[1][1]*aveSSOP1_XY_100[r2][0])*ave_F;
       Cy_100_F[r][t] +=
      (BCS_Time_Y_F[it][0][1]*SSOP1_XY_Time_100[it+t][r1][1] +
       BCS_Time_Y_F[it][1][1]*SSOP1_XY_Time_100[it+t][r2][1])*ave_F -
      (aveBCS_Y_F[0][1]*aveSSOP1_XY_100[r1][1] +
       aveBCS_Y_F[1][1]*aveSSOP1_XY_100[r2][1])*ave_F;
       Cz_100_F[r][t] +=
      (BCS_Time_Z_F[it][0][1]*SSOP1_XY_Time_100[it+t][r1][2] +
       BCS_Time_Z_F[it][1][1]*SSOP1_XY_Time_100[it+t][r2][2])*ave_F -
      (aveBCS_Z_F[0][1]*aveSSOP1_XY_100[r1][2] +
       aveBCS_Z_F[1][1]*aveSSOP1_XY_100[r2][2])*ave_F;
     }
   }
 }
}

template<class Input, class SC, class DataContainer, class SpinContainer>
void Correlation_Local_BC_PBCXY_EvenL<Input,SC,DataContainer,SpinContainer>::
clearup()
{
 // (mandatory if singleConfig==1)
 Cx_100_P.clear(); Cy_100_P.clear(); Cz_100_P.clear();
 Cx_100_F.clear(); Cy_100_F.clear(); Cz_100_F.clear();
 Cx_100_P.resize(PL_P/2+1); Cy_100_P.resize(PL_P/2+1);
 Cz_100_P.resize(PL_P/2+1);
 Cx_100_F.resize(PL_F/2+1); Cy_100_F.resize(PL_F/2+1);
 Cz_100_F.resize(PL_F/2+1);
 for(unsigned int r=0; r<Cx_100_P.size(); r++) {
   Cx_100_P[r].resize(tc+1,0.); Cy_100_P[r].resize(tc+1,0.);
   Cz_100_P[r].resize(tc+1,0.);
 }
 for(unsigned int r=0; r<Cx_100_F.size(); r++) {
   Cx_100_F[r].resize(tc+1,0.); Cy_100_F[r].resize(tc+1,0.);
   Cz_100_F[r].resize(tc+1,0.);
 }
}
```

```
template<class Input, class SC, class DataContainer, class SpinContainer>
void Correlation_Local_BC_PBCXY_EvenL<Input,SC,DataContainer,SpinContainer>::
print(const long int Id, const real SDdt, const int iConfig)
{
 (Please refer to electronic version)
}

template<class Input, class SC, class DataContainer, class SpinContainer>
void Correlation_Local_BC_PBCXY_EvenL<Input,SC,DataContainer,SpinContainer>::
findBCSIndex(BCS_IndexContainer& BCS_Index_P, BCS_IndexContainer& BCS_Index_F)
{
 BCS_Index_P.resize(4);
 BCS_Index_F.resize(2);
 for(unsigned int p=0; p<4; p++) BCS_Index_P[p].resize(2);
 for(unsigned int p=0; p<2; p++) BCS_Index_F[p].resize(2);

 DataContainer cor(3);
 for(unsigned int i=0; i<N; i++) {
   psimag::index_to_coordinate(i,cor,L);

   // find BCS_Index in PBCXY direction
   if(cor[2]==BCPL_P   && (cor[0]==BCPL_F || cor[0]==BCPL_F+1)) {
     if(sublattice_label[i]==0) BCS_Index_P[0][0].push_back(i);
     else BCS_Index_P[0][1].push_back(i);
   }
   if(cor[2]==BCPL_P+1 && (cor[0]==BCPL_F || cor[0]==BCPL_F+1)) {
     if(sublattice_label[i]==0) BCS_Index_P[1][0].push_back(i);
     else BCS_Index_P[1][1].push_back(i);
   }
   if(cor[1]==BCPL_P   && (cor[0]==BCPL_F || cor[0]==BCPL_F+1)) {
     if(sublattice_label[i]==0) BCS_Index_P[2][0].push_back(i);
     else BCS_Index_P[2][1].push_back(i);
   }
   if(cor[1]==BCPL_P+1 && (cor[0]==BCPL_F || cor[0]==BCPL_F+1)) {
     if(sublattice_label[i]==0) BCS_Index_P[3][0].push_back(i);
     else BCS_Index_P[3][1].push_back(i);
   }

   // find BCS_Index for FBCZ direction
   if(cor[0]==BCPL_F) {
     if(sublattice_label[i]==0) BCS_Index_F[0][0].push_back(i);
     else BCS_Index_F[0][1].push_back(i);
   }
   if(cor[0]==BCPL_F+1) {
     if(sublattice_label[i]==0) BCS_Index_F[1][0].push_back(i);
     else BCS_Index_F[1][1].push_back(i);
```

```
  }
 }
 /*
 for(unsigned int i=0; i<BCS_Index_P.size(); i++) {
   std::cout<<"BC_P_"<<i<<":\n";
   for(unsigned int j=0; j<BCS_Index_P[i].size(); j++) {
     std::cout<<"sub"<<j<<": ";
     for(unsigned int k=0; k<BCS_Index_P[i][j].size();k++) {
       std::cout<<BCS_Index_P[i][j][k]<<"  ";
     }
     std::cout<<"Total #: "<<BCS_Index_P[i][j].size()<<std::endl;
   }
 }
 for(unsigned int i=0; i<BCS_Index_F.size(); i++) {
   std::cout<<"BC_F_"<<i<<":\n";
   for(unsigned int j=0; j<BCS_Index_F[i].size(); j++) {
     std::cout<<"sub"<<j<<": ";
     for(unsigned int k=0; k<BCS_Index_F[i][j].size();k++) {
       std::cout<<BCS_Index_F[i][j][k]<<"  ";
     }
     std::cout<<"Total #: "<<BCS_Index_F[i][j].size()<<std::endl;
   }
 }
 */
}

template<class Input, class SC, class DataContainer, class SpinContainer>
void Correlation_Local_BC_PBCXY_EvenL<Input,SC,DataContainer,SpinContainer>::
spinSumOnPlane_100(const SpinContainer& spins, VVContainer& SSOP0_XY_100,
                   VVContainer& SSOP0_YZ_100, VVContainer& SSOP0_ZX_100,
                   VVContainer& SSOP1_XY_100, VVContainer& SSOP1_YZ_100,
                   VVContainer& SSOP1_ZX_100)
{
 unsigned int index;
 DataContainer cor(3);

 if(in.para().exchange_factor[0]<0.) {
   real sumx0, sumy0, sumz0,
        sumx1, sumy1, sumz1;
   for(unsigned int iz=0; iz<L[0]; iz++) {
     sumx0 = 0.; sumy0 = 0.; sumz0 = 0.;
     sumx1 = 0.; sumy1 = 0.; sumz1 = 0.;
     for(unsigned int ix=0; ix<L[2]; ix++) {
       for(unsigned int iy=0; iy<L[1]; iy++) {
         cor[0] = iz;
         cor[1] = iy;
         cor[2] = ix;
```

```
      index = psimag::coordinate_to_index(cor,L);
      if(sublattice_label[index]==0) {
        sumx0 += spins[index][0];
        sumy0 += spins[index][1];
        sumz0 += spins[index][2];
        sumx1 -= spins[index][0];
        sumy1 -= spins[index][1];
        sumz1 -= spins[index][2];
      }
      else {
        sumx0 -= spins[index][0];
        sumy0 -= spins[index][1];
        sumz0 -= spins[index][2];
        sumx1 += spins[index][0];
        sumy1 += spins[index][1];
        sumz1 += spins[index][2];
      }
    }
  }
  SSOP0_XY_100[iz][0] = sumx0;
  SSOP0_XY_100[iz][1] = sumy0;
  SSOP0_XY_100[iz][2] = sumz0;
  SSOP1_XY_100[iz][0] = sumx1;
  SSOP1_XY_100[iz][1] = sumy1;
  SSOP1_XY_100[iz][2] = sumz1;
}
for(unsigned int ix=0; ix<L[2]; ix++) {
  sumx0 = 0.; sumy0 = 0.; sumz0 = 0.;
  sumx1 = 0.; sumy1 = 0.; sumz1 = 0.;
  for(unsigned int iy=0; iy<L[1]; iy++) {
    for(unsigned int iz=0; iz<L[0]; iz++) {
      cor[0] = iz;
      cor[1] = iy;
      cor[2] = ix;
      index = psimag::coordinate_to_index(cor,L);
      if(sublattice_label[index]==0) {
        sumx0 += spins[index][0];
        sumy0 += spins[index][1];
        sumz0 += spins[index][2];
        sumx1 -= spins[index][0];
        sumy1 -= spins[index][1];
        sumz1 -= spins[index][2];
      }
      else {
        sumx0 -= spins[index][0];
        sumy0 -= spins[index][1];
        sumz0 -= spins[index][2];
```

```
        sumx1 += spins[index][0];
        sumy1 += spins[index][1];
        sumz1 += spins[index][2];
      }
    }
  }
  SSOP0_YZ_100[ix][0] = sumx0;
  SSOP0_YZ_100[ix][1] = sumy0;
  SSOP0_YZ_100[ix][2] = sumz0;
  SSOP1_YZ_100[ix][0] = sumx1;
  SSOP1_YZ_100[ix][1] = sumy1;
  SSOP1_YZ_100[ix][2] = sumz1;
}
for(unsigned int iy=0; iy<L[1]; iy++) {
  sumx0 = 0.; sumy0 = 0.; sumz0 = 0.;
  sumx1 = 0.; sumy1 = 0.; sumz1 = 0.;
  for(unsigned int iz=0; iz<L[0]; iz++) {
    for(unsigned int ix=0; ix<L[2]; ix++) {
      cor[0] = iz;
      cor[1] = iy;
      cor[2] = ix;
      index = psimag::coordinate_to_index(cor,L);
      if(sublattice_label[index]==0) {
        sumx0 += spins[index][0];
        sumy0 += spins[index][1];
        sumz0 += spins[index][2];
        sumx1 -= spins[index][0];
        sumy1 -= spins[index][1];
        sumz1 -= spins[index][2];
      }
      else {
        sumx0 -= spins[index][0];
        sumy0 -= spins[index][1];
        sumz0 -= spins[index][2];
        sumx1 += spins[index][0];
        sumy1 += spins[index][1];
        sumz1 += spins[index][2];
      }
    }
  }
  SSOP0_ZX_100[iy][0] = sumx0;
  SSOP0_ZX_100[iy][1] = sumy0;
  SSOP0_ZX_100[iy][2] = sumz0;
  SSOP1_ZX_100[iy][0] = sumx1;
  SSOP1_ZX_100[iy][1] = sumy1;
  SSOP1_ZX_100[iy][2] = sumz1;
}
```

```
}
else {
  real sumx, sumy, sumz;
  for(unsigned int iz=0; iz<L[0]; iz++) {
    sumx = 0.; sumy = 0.; sumz = 0.;
    for(unsigned int ix=0; ix<L[2]; ix++) {
      for(unsigned int iy=0; iy<L[1]; iy++) {
        cor[0] = iz;
        cor[1] = iy;
        cor[2] = ix;
        index = psimag::coordinate_to_index(cor,L);
        sumx += spins[index][0];
        sumy += spins[index][1];
        sumz += spins[index][2];
      }
    }
    SSOP0_XY_100[iz][0] = sumx;
    SSOP0_XY_100[iz][1] = sumy;
    SSOP0_XY_100[iz][2] = sumz;
    SSOP1_XY_100[iz][0] = sumx;
    SSOP1_XY_100[iz][1] = sumy;
    SSOP1_XY_100[iz][2] = sumz;
  }
  for(unsigned int ix=0; ix<L[2]; ix++) {
    sumx = 0.; sumy = 0.; sumz = 0.;
    for(unsigned int iy=0; iy<L[1]; iy++) {
      for(unsigned int iz=0; iz<L[0]; iz++) {
        cor[0] = iz;
        cor[1] = iy;
        cor[2] = ix;
        index = psimag::coordinate_to_index(cor,L);
        sumx += spins[index][0];
        sumy += spins[index][1];
        sumz += spins[index][2];
      }
    }
    SSOP0_YZ_100[ix][0] = sumx;
    SSOP0_YZ_100[ix][1] = sumy;
    SSOP0_YZ_100[ix][2] = sumz;
    SSOP1_YZ_100[ix][0] = sumx;
    SSOP1_YZ_100[ix][1] = sumy;
    SSOP1_YZ_100[ix][2] = sumz;
  }
  for(unsigned int iy=0; iy<L[1]; iy++) {
    sumx = 0.; sumy = 0.; sumz = 0.;
    for(unsigned int iz=0; iz<L[0]; iz++) {
      for(unsigned int ix=0; ix<L[2]; ix++) {
```

```
          cor[0] = iz;
          cor[1] = iy;
          cor[2] = ix;
          index = psimag::coordinate_to_index(cor,L);
          sumx += spins[index][0];
          sumy += spins[index][1];
          sumz += spins[index][2];
        }
      }
      SSOP0_ZX_100[iy][0] = sumx;
      SSOP0_ZX_100[iy][1] = sumy;
      SSOP0_ZX_100[iy][2] = sumz;
      SSOP1_ZX_100[iy][0] = sumx;
      SSOP1_ZX_100[iy][1] = sumy;
      SSOP1_ZX_100[iy][2] = sumz;
    }
  }
  return ;
}

#endif
```

Appendix B

Data Analysis Programs

B.1 Fourier Transform with *FT_LC.cpp*

```
#include <iostream>
#include <fstream>
#include <iomanip>
#include <sstream>
#include <vector>
#include <string>
#include <cmath>
#include <cstdlib>

#include "Real.h"


void read_to_eol(std::ifstream& in) { while(in.get()!='\n'); }

int main(int argc, char* argv[])
{
 if(argc!=9) {
   std::cout<<"usage: <program> <1: L>\n"
            <<"                 <2: tc>\n"
            <<"                 <3: dt>\n"
            <<"                 <4: ratio - dw=PI/(tc*dt)"
            <<"                 <5: 0:no print SQT; 1:print SQT\n"
            <<"                 <6: 0:no print nqs-w; 1:print nqs-w\n"
            <<"                 <7: 0:BC; 1:SCW; 2:SCO; 3:CR>\n"
            <<"                 <8: LC-BC/SCW/SCO/CR input file\n";
   exit(-1);
 }
 if(atoi(argv[5])!=0 && atoi(argv[5])!=1) {
   std::cerr<<">>> Error: Wrong value for <5: 0:no print SQT; 1:print SQT>\n";
   exit(-1);
 }
 if(atoi(argv[6])!=0 && atoi(argv[6])!=1) {
   std::cerr<<">>> Error: Wrong value for <0:no print nqs; 1:print nqs>\n";
   exit(-1);
```

```
}
if(atoi(argv[7])!=0 && atoi(argv[7])!=1 && atoi(argv[7])!=2&&atoi(argv[7])!=3)
{
  std::cerr<<">>> Error: Wrong value for <6: 0:BC; 1:SCW; 2:SCO; 3:CR>\n";
  exit(-1);
}

const unsigned int L  = atoi(argv[1]),
                   tc = atoi(argv[2]),
                   ratio = atoi(argv[4]),
                   printSQT = atoi(argv[5]),
                   printnqs = atoi(argv[6]),
                   doWhat = atoi(argv[7]);
psimag::Real       dt = atof(argv[3]);

std::ifstream      ifs(argv[8],std::ios::in);

std::vector<std::vector<psimag::Real> > c_q00_x, c_q00_y, c_q00_z;
if(L%2==1) {
  if(doWhat==0 || doWhat==1) {
    c_q00_x.resize((L+1)/2);
    c_q00_y.resize((L+1)/2);
    c_q00_z.resize((L+1)/2);
  }
  else {
    c_q00_x.resize(L);
    c_q00_y.resize(L);
    c_q00_z.resize(L);
  }
}
else{
  if(doWhat==0 || doWhat==1) {
    c_q00_x.resize(L/2+1);
    c_q00_y.resize(L/2+1);
    c_q00_z.resize(L/2+1);
  }
  else {
    c_q00_x.resize(L);
    c_q00_y.resize(L);
    c_q00_z.resize(L);
  }
}
for(unsigned int r=0; r<c_q00_x.size(); r++) {
  c_q00_x[r].resize(tc+1,0.);
  c_q00_y[r].resize(tc+1,0.);
  c_q00_z[r].resize(tc+1,0.);
}
```

```
 while(ifs.peek()=='#') read_to_eol(ifs);

 unsigned int r, t;
 psimag::Real cx, cy, cz;
 while(ifs>>r>>t>>cx>>cy>>cz) {
   c_q00_x[r][t] = cx;
   c_q00_y[r][t] = cy;
   c_q00_z[r][t] = cz;
 }

 ifs.close();

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

 switch(doWhat) {
   case 0: { std::cout<<"\n>>> LC-BC input file has been read! Now find S(q,t)
                         ..."<<std::endl;  break; }
   case 1: { std::cout<<"\n>>> LC-SCW input file has been read! Now find S(q,t)
                         ..."<<std::endl; break; }
   case 2: { std::cout<<"\n>>> LC-SCO input file has been read! Now find S(q,t)
                         ..."<<std::endl; break; }
   case 3: { std::cout<<"\n>>> LC-CR input file has been read! Now find S(q,t)
                         ..."<<std::endl;  break; }
   default: ;
 }

 psimag::Real q,
              dq00 = 2.*PI/L,
              q00_dist = 1.0;
 std::vector<std::vector<psimag::Real> > Sqt_q00_x, Sqt_q00_y, Sqt_q00_z;
 Sqt_q00_x.resize(c_q00_x.size());
 Sqt_q00_y.resize(c_q00_x.size());
 Sqt_q00_z.resize(c_q00_x.size());
 for(unsigned int nq=0; nq<Sqt_q00_x.size(); nq++) {
   Sqt_q00_x[nq].resize(tc+1);
   Sqt_q00_y[nq].resize(tc+1);
   Sqt_q00_z[nq].resize(tc+1);
 }

 std::vector<std::vector<psimag::Real> > cosq00;
 cosq00.resize(c_q00_x.size());
 for(unsigned int nq=0; nq<cosq00.size(); nq++) cosq00[nq].resize(c_q00_x.size());
 for(unsigned int nq=0; nq<cosq00.size(); nq++) {
   psimag::Real q = nq*dq00;
   for(unsigned int r=0; r<cosq00[nq].size(); r++) {
     cosq00[nq][r] = cos(q*r);
```

```
  }
}

for(unsigned int nq=0; nq<cosq00.size(); nq++) {   // q_max = pi/q00_dist = pi
  for(unsigned int t=0; t<tc+1; t++) {   // q=(0, dq00, ..., pi)/q00_dist
    psimag::Real sum_q00_x = 0.,
                 sum_q00_y = 0.,
                 sum_q00_z = 0.;
    for(unsigned int r=0; r<cosq00[nq].size(); r++) {
      sum_q00_x += c_q00_x[r][t]*cosq00[nq][r];
      sum_q00_y += c_q00_y[r][t]*cosq00[nq][r];
      sum_q00_z += c_q00_z[r][t]*cosq00[nq][r];
    }
    Sqt_q00_x[nq][t] = sum_q00_x;
    Sqt_q00_y[nq][t] = sum_q00_y;
    Sqt_q00_z[nq][t] = sum_q00_z;
  }
}

if(printSQT==1) {
  std::ofstream ofs[Sqt_q00_x.size()];
  for(unsigned int nq=0; nq<Sqt_q00_x.size(); nq++) {
    std::ostringstream oss;
    switch(doWhat) {
      case 0: { oss<<"SQT_BC_"<<argv[8]<<"_nq"<<nq;  break; }
      case 1: { oss<<"SQT_SCW_"<<argv[8]<<"_nq"<<nq; break; }
      case 2: { oss<<"SQT_SCO_"<<argv[8]<<"_nq"<<nq; break; }
      case 3: { oss<<"SQT_CR_"<<argv[8]<<"_nq"<<nq;  break; }
      default: ;
    }
    ofs[nq].open(oss.str().c_str(),std::ios::out);
    switch(doWhat) {
      case 0: { ofs[nq]<<"# S(q,t)_BC_100 with L="<<L<<"  tc="<<tc<<std::endl;
                break; }
      case 1: { ofs[nq]<<"# S(q,t)_SCW_100 with L="<<L<<"  tc="<<tc<<std::endl;
                break; }
      case 2: { ofs[nq]<<"# S(q,t)_SCO_100 with L="<<L<<"  tc="<<tc<<std::endl;
                break; }
      case 3: { ofs[nq]<<"# S(q,t)_CR_100 with L="<<L<<"  tc="<<tc<<std::endl;
                break; }
      default: ;
    }
    ofs[nq]<<std::setw(15)<<std::left<<"# col1:nq"<<std::setw(15)<<std::left
           <<"col2:t"
           <<std::setw(15)<<std::left<<"col3:q00_x"
           <<std::setw(15)<<std::left<<"col4:q00_y"
           <<std::setw(15)<<std::left<<"col5:q00_z"
```

```
          <<std::setw(15)<<std::left<<"col6:q00_T"
          <<std::setw(15)<<std::left<<"col7:q00_m"
          <<std::endl;
  }
  for(unsigned int nq=0; nq<Sqt_q00_x.size(); nq++) {
    for(unsigned int t=0; t<tc+1; t++) {
      ofs[nq]<<std::setw(15)<<std::left<<std::setprecision(6)<<std::fixed
             <<nq<<std::setw(15)<<std::left<<std::setprecision(6)<<std::fixed
             <<t<<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
             <<Sqt_q00_x[nq][t]
             <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
             <<Sqt_q00_y[nq][t]
             <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
             <<Sqt_q00_z[nq][t]
             <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
             <<0.5*(Sqt_q00_x[nq][t]+Sqt_q00_y[nq][t])
             <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
             <<(Sqt_q00_x[nq][t]+Sqt_q00_y[nq][t]+Sqt_q00_z[nq][t])/3.
             <<std::endl;
    }
    ofs[nq].close();
  }
}

/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

std::cout<<"\n>>> Now FT on t and find s(q,w)......"<<std::endl;

std::ofstream ofs[Sqt_q00_x.size()];
for(unsigned int nq=0; nq<Sqt_q00_x.size(); nq++) {
  std::ostringstream oss;
  switch(doWhat) {
    case 0: { oss<<"SQW_BC_"<<argv[8]<<"_nq"<<nq;  break; }
    case 1: { oss<<"SQW_SCW_"<<argv[8]<<"_nq"<<nq; break; }
    case 2: { oss<<"SQW_SCO_"<<argv[8]<<"_nq"<<nq; break; }
    case 3: { oss<<"SQW_CR_"<<argv[8]<<"_nq"<<nq;  break; }
    default: ;
  }
  ofs[nq].open(oss.str().c_str(),std::ios::out);
  switch(doWhat) {
    case 0: { ofs[nq]<<"# S(q,w)_BC_100 with L="<<L<<"  tc="<<tc<<std::endl;
              break; }
    case 1: { ofs[nq]<<"# S(q,w)_SCW_100 with L="<<L<<"  tc="<<tc<<std::endl;
              break; }
    case 2: { ofs[nq]<<"# S(q,w)_SCO_100 with L="<<L<<"  tc="<<tc<<std::endl;
              break; }
    case 3: { ofs[nq]<<"# S(q,w)_CR_100 with L="<<L<<"  tc="<<tc<<std::endl;
```

```
              break; }
    default: ;
  }
  ofs[nq]<<std::setw(15)<<std::left<<"# col1:nq"<<std::setw(15)<<std::left
         <<"col2:w(J)"
         <<std::setw(15)<<std::left<<"col3:q00_x"
         <<std::setw(15)<<std::left<<"col4:q00_y"
         <<std::setw(15)<<std::left<<"col5:q00_z"
         <<std::setw(15)<<std::left<<"col6:q00_T"
         <<std::setw(15)<<std::left<<"col6:q00_m"
         <<std::endl;
}
std::ofstream ofs_nqs;
std::ostringstream oss;
if(printnqs==1) {
  switch(doWhat) {
    case 0: { oss<<"SQW_BC_"<<argv[8]<<"_nqs";  break; }
    case 1: { oss<<"SQW_SCW_"<<argv[8]<<"_nqs"; break; }
    case 2: { oss<<"SQW_SCO_"<<argv[8]<<"_nqs"; break; }
    case 3: { oss<<"SQW_CR_"<<argv[8]<<"_nqs";  break; }
    default: ;
  }
  ofs_nqs.open(oss.str().c_str(),std::ios::out);
  switch(doWhat) {
    case 0: { ofs_nqs<<"# S(q,w)_BC_100 with L="<<L<<"  tc="<<tc<<std::endl;
              break; }
    case 1: { ofs_nqs<<"# S(q,w)_SCW_100 with L="<<L<<"  tc="<<tc<<std::endl;
              break; }
    case 2: { ofs_nqs<<"# S(q,w)_SCO_100 with L="<<L<<"  tc="<<tc<<std::endl;
              break; }
    case 3: { ofs_nqs<<"# S(q,w)_CR_100 with L="<<L<<"  tc="<<tc<<std::endl;
              break; }
    default: ;
  }
  ofs_nqs<<std::setw(15)<<std::left<<"# col1:nq"<<std::setw(15)<<std::left
         <<"col2:w(J)"
         <<std::setw(15)<<std::left<<"col3:q00_x"
         <<std::setw(15)<<std::left<<"col4:q00_y"
         <<std::setw(15)<<std::left<<"col5:q00_z"
         <<std::setw(15)<<std::left<<"col6:q00_T"
         <<std::setw(15)<<std::left<<"col6:q00_m"
         <<std::endl;
}

/*
std::vector<psimag::Real>   expmhalftt(tc+1); // energy resolution convolution
unsigned int irt = atoi(argv[5]);
```

```
psimag::Real rt  = atof(argv[5]);
if(irt==0) {
  for(unsigned int t=0; t<=tc; t++) {
    expmhalftt[t]=1.;
  }
}
else {
  psimag::Real delta_w = rt/tc;
  for(unsigned int t=0; t<=tc; t++) {
    expmhalftt[t]=exp(-0.5*t*t*delta_w*delta_w);
  }
}
*/


/*
//Simpson's Rule (Only (100))
psimag::Real norm = dt/(2.*PI), w, dw=PI/(tc*dt*ratio), w_max = PI/(dt*ratio),
             even_q00_x, even_q00_y, even_q00_z,
             odd_q00_x,  odd_q00_y,  odd_q00_z,
             coswt_2;

for(unsigned int nq=0; nq<Sqt_q00_x.size(); nq++) {   // nq=0...L/2, q=0...pi
  for(unsigned int iw=0; iw<=tc/ratio; iw++) {
    w=iw*dw;
    even_q00_x=0.; even_q00_y=0.; even_q00_z=0.;
    odd_q00_x=0.;  odd_q00_y=0.;  odd_q00_z=0.;
    for(unsigned int t=1; t<=tc; t+=2) {
      coswt_2=cos(w*(t-1)*dt)*2.;
      even_q00_x += Sqt_q00_x[nq][t-1]*coswt_2;
      even_q00_y += Sqt_q00_y[nq][t-1]*coswt_2;
      even_q00_z += Sqt_q00_z[nq][t-1]*coswt_2;

      coswt_2=cos(w*t*dt)*2.;
      odd_q00_x += Sqt_q00_x[nq][t]*coswt_2;
      odd_q00_y += Sqt_q00_y[nq][t]*coswt_2;
      odd_q00_z += Sqt_q00_z[nq][t]*coswt_2;
    }
    coswt_2=cos(w*tc*dt)*2.;
    psimag::Real   sum_q00_x = (2.*even_q00_x-cos(w*0.*dt)*2.*Sqt_q00_x[nq][0]
                                +coswt_2*Sqt_q00_x[nq][tc])+4.*odd_q00_x,
                   sum_q00_y = (2.*even_q00_y-cos(w*0.*dt)*2.*Sqt_q00_y[nq][0]
                                +coswt_2*Sqt_q00_y[nq][tc])+4.*odd_q00_y,
                   sum_q00_z = (2.*even_q00_z-cos(w*0.*dt)*2.*Sqt_q00_z[nq][0]
                                +coswt_2*Sqt_q00_z[nq][tc])+4.*odd_q00_z;

    ofs[nq]<<std::setw(15)<<std::left<<std::setprecision(6)<<std::fixed
```

```
            <<nq<<std::setw(15)<<std::left<<std::setprecision(6)<<std::fixed
            <<w<<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
            <<sum_q00_x*norm
            <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
            <<sum_q00_y*norm
            <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
            <<sum_q00_z*norm
            <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
            <<0.5*(sum_q00_x+sum_q00_y)*norm
            <<std::endl;
    if(printnqs==1) {
      ofs_nqs<<std::setw(15)<<std::left<<std::setprecision(6)<<std::fixed<<nq
             <<std::setw(15)<<std::left<<std::setprecision(6)<<std::fixed<<w
             <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
             <<sum_q00_x*norm
             <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
             <<sum_q00_y*norm
             <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
             <<sum_q00_z*norm
             <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
             <<0.5*(sum_q00_x+sum_q00_y)*norm
             <<std::endl;
    }
  }
  ofs[nq].close();
}
if(printnqs==1) ofs_nqs.close();
*/

psimag::Real w, dw=PI/(tc*dt*ratio), w_max = PI/(dt*ratio);
psimag::Real norm = dt/(2.*PI);

for(unsigned int nq=0; nq<Sqt_q00_x.size(); nq++) {
  for(unsigned int iw=0; iw<=tc; iw++) {
    w=iw*dw;
    psimag::Real sum_q00_x = Sqt_q00_x[nq][0],
                 sum_q00_y = Sqt_q00_y[nq][0],
                 sum_q00_z = Sqt_q00_z[nq][0];
    for(unsigned int t=1; t<=tc; t++) {
      psimag::Real coswt_2 = 2.*cos(w*t*dt);
      sum_q00_x += Sqt_q00_x[nq][t]*coswt_2;
      sum_q00_y += Sqt_q00_y[nq][t]*coswt_2;
      sum_q00_z += Sqt_q00_z[nq][t]*coswt_2;
    }
    ofs[nq]<<std::setw(15)<<std::left<<std::setprecision(6)<<std::fixed<<nq
           <<std::setw(15)<<std::left<<std::setprecision(6)<<std::fixed<<w
           <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
```

```
            <<sum_q00_x*norm
            <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
            <<sum_q00_y*norm
            <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
            <<sum_q00_z*norm
            <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
            <<0.5*(sum_q00_x+sum_q00_y)*norm
            <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
            <<(sum_q00_x+sum_q00_y+sum_q00_z)*norm/3.
            <<std::endl;
    if(printnqs==1) {
      ofs_nqs<<std::setw(15)<<std::left<<std::setprecision(6)<<std::fixed<<nq
             <<std::setw(15)<<std::left<<std::setprecision(6)<<std::fixed<<w
             <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
             <<sum_q00_x*norm
             <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
             <<sum_q00_y*norm
             <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
             <<sum_q00_z*norm
             <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
             <<0.5*(sum_q00_x+sum_q00_y)*norm
             <<std::setw(15)<<std::left<<std::setprecision(9)<<std::fixed
             <<(sum_q00_x+sum_q00_y+sum_q00_z)*norm/3.
             <<std::endl;
    }
  }
  ofs[nq].close();
 }
 if(printnqs==1) ofs_nqs.close();

 switch(doWhat) {
   case 0: { std::cout<<"\n>>> FT on LC-BC has been finished!\n\n";  break; }
   case 1: { std::cout<<"\n>>> FT on LC-SCW has been finished!\n\n"; break; }
   case 2: { std::cout<<"\n>>> FT on LC-SCO has been finished!\n\n"; break; }
   case 3: { std::cout<<"\n>>> FT on LC-CR has been finished!\n\n";  break; }
   default: ;
 }

 return 0;
}
```