AN ABSTRACT OF THE THESIS OF

Seongweon Park for the degree of Doctor of Philosophy in Physics presented on December 17th, 2012.

Title: Monte Carlo Studies of Classical Heisenberg Spins on Face-Centered-Cubic Lattices: Effects of Strain, Interlayer Coupling, and Dilution of Lattice

Abstract Approved:

Tomasz M. Giebultowicz Guenter Schneider

This thesis presents the results from Monte Carlo calculations on classical vector spins in face-centered-cubic (FCC) lattices. The goal of the study was to understand the effect of interlayer coupling, dilution of magnetic atoms in the lattice, and symmetry-changing strain.

Experimental work by T. M. Giebultowicz et al. and J. A. Borchers et al. greatly inspired my work [1, 2]. J. A. Borchers's group studied NiO/CoO superlattices and observed that the magnetic order of CoO persisted above its Néel temperature due to the effect of interlayer coupling with NiO, which has a higher Néel temperature than CoO [1].

Simulating on a model of NiO/CoO bilayer reproduced the experimental results from Borchers et al. [1]. I concluded that exchange pinning on the NiO/CoO interface preserves the magnetic order of CoO above its Néel temperature significantly.

Building on this initial result, a ferromagnet/antiferromagnet/ferromagnet (FM/AFM/FM) trilayer model was studied, where the ferromagnetic (FM) layers were antiferromagnetically coupled. First, I calculated the strength of the AF coupling as a

function of the number of antiferromagnetic (AFM) spacer monolayers and concluded that the strength of AFM coupling decreases as the number of AFM spacer monolayers increases.

Secondly, I added a uniaxial anisotropy to the model and obtained magnetization curves which exhibited hysteresis-like features with an external field and a first order magnetic transition. Lastly, I diluted the AFM spacer layer in the FM/AFM/FM trilayer by replacing magnetic spins with zero spins in the model. The dilution of AFM spacer layer caused fluctuations in the magnetization curves with external field but the strength of AFM coupling decreases as the number of AFM monolayers increases as in the non-diluted cases.

The experimental results from T.M. Giebultowicz's group on MnSe/ZnTe superlattices by neutron scattering showed incommensurate helical spin order in MnSe, where MnSe layers were under tensile strain due to a small mismatching in the lattice parameter [2]. In addition, they observed that the pitch of the spin helix increased as the temperature increased [2]. I modeled the MnSe/ ZnTe system with Monte Carlo method and found that the pitch of the spin helix increased with temperature. In fact, the dependence of helix pitch on temperature was present regardless of the thickness of the sample, so I concluded that this pitch increase is not from the weakening of coupling of surface spins.

Monte Carlo Studies of Classical Heisenberg Spins on Face-Centered-Cubic Lattices: Effects of Strain, Interlayer Coupling, and Dilution of Lattice

by

Seongweon Park

A THESIS

Submitted to

Oregon State University

in partial fulfillment of

the requirements for the

degree of

Doctor of Philosophy

Presented December 17, 2012

Commencement June 2013

Doctor of Philosophy thesis of Seongweon Park

Presented on December 17, 2012

APPROVED:

Co-Major Professor, representing Physics

Co-Major Professor, representing Physics

Chair of the Department of Physics

Dean of the Graduate School

I understand that my thesis will become part of the permanent collection of Oregon State University libraries. My signature below authorizes release of my thesis to any reader upon request.

Seongweon Park, Author

ACKNOWLEDGEMENTS

I used Microsoft Word for editing this text with equation editing features.

I would like to thank Dr. Tomasz M. Giebultowicz for insights and numerous discussions on magnetic systems and Dr. Guenter Schneider for his advice on programming and for providing me with computational facilities. Also I would like to thank everybody in my graduate committee who have been very patient with my progress and generously offered help when I was in my "academic crisis". Lastly I thank my family, friends, teachers, and the staff members at the Oregon State University physics department for the warm encouragements and supports they provided when I needed them most. Definitely I would not have been here without everybody's supports.

December, 2012

Oregon State University

Seongweon Park

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

Page

TABLE OF CONTENTS (Continued)

Page

LIST OF TABLES

Table Page

LIST OF APPENDIX TABLES

Table Page

LIST OF APPENDICES

Appendix Page

LIST OF FIGURES

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF FIGURES (Continued)

LIST OF APPENDIX FIGURES

MONTE CARLO STUDIES OF CLASSICAL HEISENBERG SPINS
ON FACE-CENTERED-CUBIC LATTICES:
EFFECTS OF STRAIN, INTERLAYER COUPLING, AND DILUTION OF LATTICE

Chapter 1. INTRODUCTION

The goal of this work was to model and understand some effects of current interest in magnetic thin films using the numerical technique of Monte Carlo (MC) simulation of lattices of classical vector spins. A classical spin model is the classical limit of a quantum mechanical, localized spin model, which is known to approximate the properties of real lattices of quantum spins quite well, especially in the cases with larger values of the atomic quantum spin number S such as, for example, S=5/2 in spin lattices built up of Mn^{2+} magnetic ions, which are the component in many magnetic systems [3].

The work presented in this thesis consists of two major subprojects, which both focus on phenomena occurring in thin film structures. The first subproject is about "magnetic coupling" between two FM layers separated by a spacer layer that is made of a different material. The second subproject is focused on the effect of symmetry-changing strain on a frustrated lattice. The objective for the first subproject is to investigate a physical situation that may offer a new method of preparing thin multilayered films with controlled coupling strength between the constituent FM layers. The objective for the second subproject, the effect of strain, is to explain certain phenomena in a frustrated spin lattice under strain that have been observed in the past but have not been fully explained until today. The background for these projects is provided in the following sections.

Subproject one: interlayer coupling

1.1. Interlayer coupling and giant magnetoresistance

The effect of interlayer coupling (IC) or interlayer exchange coupling (IEC) was initially observed in Fe/Cr/Fe multilayer by P. Grünberg and A. Fert independently [4, 5, 6]. They observed that the magnetizations of the Fe layers separated by the Cr spacer "correlated" with each other and aligned parallel or anti-parallel depending on the thickness of the Cr spacer [4, 5, 6]. This result was surprising since the spacer material Chromium (Cr) is a nonmagnetic (NM) metal. Afterwards, Parkin et al. investigated the specimen by cross-section transmission electron microscopy (XTEM) and confirmed that indeed the correlation coefficient turned out to be an oscillating function of the NM spacer thickness [7]. This discovery enabled one to "manipulate" the sign of magnetization correlation (parallel or anti-parallel) of FM layers by changing the thickness of the spacer layer by a few atomic layers [8, 9].

However, a more exciting discovery A. Fert and P. Grünberg made in the Fe/Cr/Fe trilayer experiments than IC was an additional effect revealed by the experiments [4, 5, 6]. Namely, they found that the resistance of the trilayer depended strongly on the relative orientations of magnetizations of the FM layers [4, 5, 6]. The *resistance* R for the anti-parallel alignment of FM magnetizations was significantly higher (by 40-80%) than R for the parallel FM magnetizations [4, 5, 6]. If the magnetizations of the FM layers are initially anti-parallel, one can make them parallel by applying an external magnetic field **B** increasing gradually in intensity, which will

significantly change the resistance of the system [4, 5, 6]. This effect can be used in compact magnetic sensors since the magneto-transport behavior is very sensitive to the alignment of magnetizations in magnetic layers [10].

In fact, Lord Kelvin found that electric resistance of bulk metals and semiconductors depends on external magnetic field in 1856 and this effect is called magnetoresistance (MR) [11]. MR is relatively weak and therefore it has found few practical applications. The change of R in the Fe/Cr/Fe trilayer A. Fert and P. Grünberg measured with an external magnetic field **B** was stronger in orders of magnitude than the ordinary MR, so it was named Giant Magnetoresistance (GMR) [11]. Eventually sensors based on GMR led to a real revolution in computer hard-drives [11].

Since there are many other possible applications of GMR-based magnetic sensors, such as magneto-resistive random access memory (MRAM), the physical phenomena underlying GMR effects became the subject of vigorous theoretical and experimental research following P. Grünberg and A. Fert's discovery [10]. Experimental measurements on other thin metallic trilayers composed of ferromagnets and NM mid-spacer with varying thickness between them revealed an oscillatory behavior of IC and a strong dependence of R on $\vec{B}$ [8, 9]. It should be stressed that to use such systems as magnetic field sensors, it is important to have an initial anti-parallel ferromagnet alignment of the FM layers [12]. Needless to say, making a high quality magnetic field sensor requires a good understanding of the physical mechanism responsible for the oscillatory behavior or IC in such metallic trilayers.

Theoretical studies on IC in all-metal systems, most notably by P. Bruno and Barbara Jones, led to the conclusion that the magnetic interactions were conveyed by mobile conduction electrons in the metal spacer [13, 14]. The damped oscillatory behavior of IC was found to be caused by quantum effects closely related to those occurring in the Rudermann-Kittel-Kasuya-Yosida (RKKY) interactions, which explain the mechanism of coupling between magnetic ions in diluted magnetic alloys [13, 14, 15]. The coupling constant J for RKKY interactions is an oscillating function of the form $J(r) \sim \frac{\cos(2k_F r)}{r^3}$ with distance between two interacting ions r and a spherical Fermi surface of radius k_F [13, 14, 15].

The first explanation on strong dependence of R on **B** in GMR systems was given in terms of the two-current model, which views the total electron current as the sum of up and down spin currents [16, 17]. A polarized spin flows through a FM film more easily when it is parallel to the magnetization of the ferromagnets [16, 17]. If the magnetizations of ferromagnets are anti-parallel, both up and down currents are attenuated [16]. If the magnetizations of ferromagnets are parallel, one of the currents flows with little attenuation causing lower resistance [16, 17]. Therefore, GMR systems are the first generation of systems in which the spin state of the current is controlled in contrast to conventional systems that control the amount of current. The term spintronics was invented to refer to such novel electronic devices [10]. The exact mechanism of the phenomenon of GMR is beyond the scope of the present Ph.D. project - however, since the effects discussed later in this project play a major role in GMR systems, it is worth mentioning it in order to give some context of this project.

1.2. Spintronics: semiconductor-based systems

The strong interest in spintronics research after the success of GMR devices motivated new research efforts. For instance, semiconductor spintronics has been very attractive research area since one can combine the properties of semiconductors with those of magnetic materials [17].

The theoretical studies on semiconductor multilayers using the tight binding model showed that magnetic interactions in semiconductor systems were not conveyed by the mobile conduction electrons but by the electrons in the valence band and that the interactions cause an AFM coupling [18]. Experimental measurements of AFM IC in II-IV superlattices (e.g., EuS/PbS, EuS/YbSe) confirmed this theoretical result [19-24].

However, there was a big obstacle in all semiconductor spintronics research. It was the fact that he Curie temperature for semiconductor ferromagnet is low, which made it impossible to keep the semiconductor ferromagnet ordered at the room temperature [17]. Semiconductor GMR sensors has never been realized but currently research efforts on using diluted magnetic semiconductor systems and combining semiconductors with metals and other materials continue [17, 20].

1.3. Metallic trilayers with insulating NM spacer

The new generations of spintronics devices that have emerged in the last decade has required a renewed interest in insulating barriers. The novel sensors are also composed of two FM metallic films, with a NM spacer sandwiched in-between, but the spacer is non-metallic [25, 26]. It is a thin (~1 nm) layer of an insulating material, so electron current flows through the insulating barrier due to tunnel effect [28, 29]. These trilayers are called Magnetic Tunnel Junctions (MTJs) [26] As in GMR sensors, the effective resistance R of MTJs depends on the mutual orientations of magnetization vectors in the FM layers [26, 27]. This effect is called Tunnel Magnetoresistance (TMR), which was initially found by Michael Jullière [27]. TMR is the result of spin-polarized tunneling, which is different from the mechanism for GMR [27].

The R_{max} / R_{min} ratio in TMR sensors can be as high as several hundred %, which is much better in performance than that of the GMR sensors [28, 29]. For both TMR sensors and GMR devices, the initial orientations of FM magnetizations have to be anti-parallel [10]. In an experimental study by T. Katayama *et al*. on Fe/MgO/Fe trilayers it was demonstrated that IC between the FM layers may oscillate with the thickness of the spacer [28]. As shown by *ab initio* calculations, such behavior may result from the presence of oxygen vacancies in MgO [28]. It makes very thin layers of MgO AFM, even though the material is essentially a nonmagnetic insulator [28].

However, the concentration of oxygen vacancies in epitaxially grown materials is difficult to control. In most situations, MgO or Al_2O_3 (another material used to make

TMR junctions) do not act as efficient media for conveying the magnetic interactions between the FM layers [30]. Hence, other methods of ensuring an initial anti-parallel orientation of FM layers for TMR devices have been developed [26]. One of them is to use FM layers of different coercivities, either by using FM layers of different thicknesses or two different FM materials in the trilayer. Another method is to use the effect of *exchange bias* (EB), which is "shorthand" for an effect in which the magnetic hysteresis loops, normally centered at zero magnetic field, shifts toward a negative or positive field [26]. It should be noted that the EB effect in ferromagnet/antiferromagnet (FM/AFM) system, continues to be the subject of considerable interest and is currently studied by different groups both experimentally and theoretically [31, 32]. It will be described in more detail in Chapter 3.

The aim of the work described in Chapter 3 of the present thesis was to examine yet another possible method of imposing correlations between the magnetization vectors of the two FM layers separated by an insulating spacer – namely, by using an AFM spacer whose both interfaces are exchange-coupled to the FM blocks. Such an AFM spacer has to be composed of an *even* number of atomic monolayers (mnl) to ensure the anti-parallel alignment of the constituent FM layers. The application of an external magnetic field would impose opposite torques on the two FM layers, resulting in a "twist" of the AFM mid-layer, which would then act as a "torsion spring". If such system exhibits TMR properties, its resistance R will change as a function of the strength of the applied field $\vec{B}$.

1.4. Objectives for subproject one

In fact, the concept that an AFM spacer may act as a "connecting rod" with certain "magnetic stiffness constant" has been known for some time. As discussed in later in Chapter 3, an effect of this type was first considered by J. Slonczewski in 1995, in the context of IEC between two FM metallic layers separated by a metallic AFM spaces. Yet, in Slonczewski's original paper the "torsion spring" effect of the AFM layer was not presented as the principal effect responsible for the IEC, but rather as an "auxiliary" effect that could, depending on the circumstances, either strengthen or weaken the *principal* FM-FM interlayer interaction conveyed by conduction electrons [25, 33].

One of the objectives in the present project was to explore another physical scenario, in which the "torsion spring" effect in the AFM spacer would act as the *sole agent* transmitting the exchange coupling from one FM layer to another. The model seems straightforward then — yet, there are several intriguing questions and important aspects of such a model that need to be examined. What is the exact characteristic of the "torsion spring"? - does it exhibit a "Hooke's law-type" behavior, i.e., with the torque exerting by the "torsional spring" being approximately proportional to the angle between magnetization vectors of the two FM layer? Or, perhaps, is it significantly nonlinear or even showing a sharp discontinuity at certain threshold $\vec{B}$ value? Is there a hysteresis in the "spring" characteristic?

The MC modeling method is certainly a good theoretical tool getting closer insight into the above problems, as well as into other conceivable questions related to the

physics of such trilayers. One more important issue is the temperature behavior of the "AFM torsion spring". One's first thought may be that the "spring action" will cease exactly at the Néel point of the spacer material, i.e., at the temperature at which the AFM order in the material collapses. However, experimental neutron diffraction results from NiO/CoO superlattices reported by the J. A. Borchers's *et al.* [1] may make one to wonder whether it must be necessarily so. These authors observed that the magnetic order in the CoO component persisted well above its bulk Néel temperature when it was layered with NiO, a material with much higher Néel temperature than CoO (510 K vs. 291 K respectively) [1, 34].

They concluded that the persistence of magnetic order in CoO is due to the IC effects in the NiO/CoO interface [1]. Picturesquely speaking, it is as if the CoO spin monolayers, adjacent to the still well-ordered spin lattice of NiO, "borrow" the ordered state from it. And if the CoO layer is thin enough and is interposed between two NiO layers, this ordered state may spread across its entire width. Inspired by these findings, I decided to check whether an analogous effect would occur in FM/AFM/FM trilayers – namely, would AFM layers interposed between two *ferromagnetic* layers retain their ordered state even at temperatures higher than their bulk Néel temperature, and thus be still capable of maintaining interlayer coupling between the two FM blocks?

The last objective for the first subproject is to check whether a diluted AFM layer can convey the magnetic interactions between the FM blocks. The underlying idea for this was to propose new possible design schemes for TMR sensors – but I did not know whether there exist pure AFM materials suitable for using as spacers in TMR sensors.

MgO and Al_2O_3, which have been proven to be the most efficient spacer materials in TMR sensors, are not magnetic [28, 29]. But it is worth checking if *alloys* of MgO and MnO, or MgO and CoO would still offer a suitable spacer material. The numerical modeling work performed for this problem was supposed to answer the question of what MnO or CoO concentration would be needed for making the alloys capable of transferring interactions between the FM layers.

Subproject two: frustration in AFM lattice

The second subproject focuses on modeling of effects seen in thin films of an AFM material experiencing a symmetry-changing strain. The effects were observed in neutron diffraction experiments on artificial superlattices composed of alternating layers of two semiconductor materials, MnSe and ZnTe, where ZnTe is a non-magnet and MnSe is an antiferromagnet [2].

Frustration, which is also called geometrical frustration or topological frustration, in AFM spin lattices means that all spin neighbor pairs in the lattice cannot satisfy the energy minimizing AFM configuration [35]. Such frustration is a consequence of the lattice topology [35]. An example of a non-frustrated system is a square 2D lattice in which each spin is antiferromagnetically coupled with its four nearest neighbor (NN) spins. In contrast, if one considers a triangular 2D spin lattice with AFM coupling between the NN spins, all these bonds cannot be simultaneously minimized – 1/3 of the bonds remains “unsatisfied”, or “frustrated” [35, 36].

1.5. General overview on frustrated systems

The first theory of antiferromagnetism was principally developed by L. Néel the 1930s and initially it met with much skepticism, which radically changed in 1949 when C.G. Shull and J. S. Smart experimentally confirmed the existence AFM structures by neutron diffraction [37, 38]. In the 1950s, a number of neutron scattering laboratories were created worldwide. Many neutron diffraction studies of magnetic materials followed the pioneer work of Shull and Smart, and soon it was realized that a vast majority of all existing magnetically ordered crystalline substances are antiferromagnets [39]. Furthermore, the studies revealed that there is a rich variety of AFM ordering types, spanning from quite simple ones with only two magnetic sublattices, to quite "exotic" structures with many sublattices and helical periodicity not commensurate with that of the "parent" atomic crystal structure [40].

Frustrated AFM lattices are a class of spin systems which have been given a great deal of interest starting from the early days of research on antiferromagnets – and this interest continues until today. A peculiar property of such lattices is that their ground states consist of a large number of degenerate configurations [41]. Their degeneracy can be lifted by some additional symmetry-changing factors, either intrinsic (i.e., interactions between distant neighbors) or extrinsic (i.e., external factors such as external magnetic field or symmetry-breaking strain) [42, 43]. Such factors may stabilize a specific ground state configuration and thus produce a new unique kind of AFM ordering [42, 43].

The FCC spin lattice with AFM spin-spin interactions is a canonical example of a frustrated system. P.W. Anderson performed the first theoretical studies on this class of FCC AFM lattices and experimental studies followed [44, 45, 46]. Experimental studies on MnS, which became a "prototypical example" of a FCC antiferromagnet, revealed that there are two FCC structural modifications of this compound [47, 48]. In both the AFM Mn^{++}- Mn^{++} interactions are of the superexchange type, mediated by an intervening sulfur anion. In one modification, of zincblende (ZB) structure, the anions are located in the center of a tetrahedron formed by four NN cations [47, 48]. In the other, of NaCl type, the anions are positioned between NNN cations [47]. Accordingly, in the ZB modification the NN AFM interactions are the dominant ones. But it is "topologically impossible" to satisfy the AFM links of a Mn^{++} cation with all twelve of its NNs. One-third of these links must remain "unsatisfied"; hence, the ZB modification is often termed as "strongly frustrated" [43,47]. In the NaCl-type modification, in contrast, the dominant AFM interactions are between the NNNs, and *there is* a possible arrangement scheme in the FCC lattice in which all NNN AFM bonds are simultaneously satisfied. Some of the NN bonds still remain "unsatisfied", but since the NN interactions are now the weaker ones, the frustration effects play only a secondary role (such system are sometimes referred to as "moderately frustrated") [47].

The interest in frustrated spin system has remained strong until today. Over the first few decades following P. W. Anderson's pioneer work, theoretical studies were done primarily on systems consisting of classical vector spins [41, 49]. In fact, in most known examples of real frustrated systems in that era, the magnetic lattices were composed of

magnetic ions with higher spin quantum numbers (such as Mn^{2+} and Fe^{2+} with S=5/2 or Eu^{2+} and Gd^{3+} with S=7/2). Classical spins were therefore an appropriate approximation in theoretical modeling [3].

1.6. Current trend: magnetic frustration research

More recently, the attention started shifting more towards frustrated *quantum* antiferromagnets. This new trend was certainly a "side effect" of the great wave of excitement around research on high-temperature (high-T_C) superconductors in the years following the discovery of such materials [50]. Some families of the high-T_C compounds are often referred to as "cuprates" since they contain weakly-coupled copper oxide layer planes (CuO_2), in which the constituent Cu^{2+} ions are magnetic with S=1/2 and the interaction between them is AFM [50, 51, 52].

Due to the low spin quantum number, it is inadequate to use classical approximations for this system, so the copper ion planes are viewed as "quasi-2D quantum antiferromagnets" [51]. The development of techniques for cuprate preparation made it possible to produce a number of 1D and 2D AFM systems with S=1/2 magnetic ions [53]. The AFM lattices in many of the new 2D systems were "topologically frustrated", thus creating a strong motivation for new experimental and theoretical studies of "frustrated quantum antiferromagnets" [53, 54].

Even though the frustrated magnetism research is currently dominated by studies of quantum systems, there is still justification for doing more model work on frustrated

antiferromagnets using classical approximations. Some effects observed in the past in experiments on frustrated AFM lattices composed of spins with high quantum number *S* have not been fully understood. One of such problems was chosen for the second subproject. Namely, I refer to the results of neutron diffraction experiments on thin layers of frustrated FCC antiferromagnets performed in 1990s at the Center for Neutron Research of the National Institute of Technology (NCR NIST, Gaithersburg, MD) by T. M. Giebultowicz and the university of Notre Dame team [2].

As noted, a "canonical" example of a frustrated FCC antiferromagnet is the zincblende modification of MnS [47, 48]. Sulfur (S) is a representative of the family of chalcogens, and two other members of this group are Selenium (Se) and Tellurium (Te) [55]. Manganese Selenide (MnSe) and Telluride (MnTe) do not form ZB phases in bulk [55]. However, it has been found that these materials can be forced to form zinc-blend phases if they were interposed between layers of other ZB type materials with similar lattice dimensions — such as e.g., ZnTe, ZnSe, or CdTe [2]. This created an opportunity of investigating two entirely new frustrated AFM lattices [2].

As shown by P. W. Anderson in his aforementioned study on FCC antiferromagnets, if the spins are coupled only by AFM NN interactions, the system may form two different ground state configurations [56, 57]. Anderson called them "Type I" and "Type III" structures [56, 57]. Weak next-nearest neighbor (NNN) *ferromagnetic* interactions result in the formation of the Type I structure, whereas weak NNN *antiferromagnetic* interactions lead to the Type III ordering [56, 57]. Figure 1 shows Type I and Type III AFM order [56, 57].

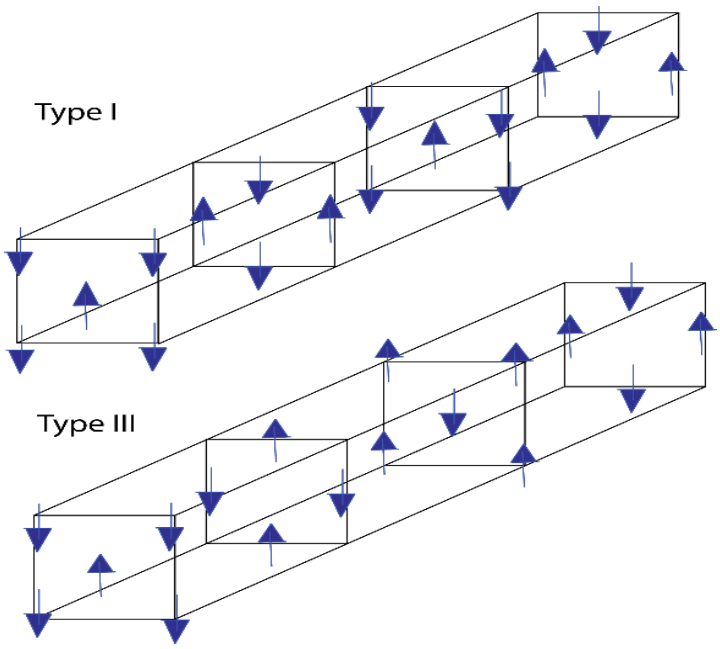


Figure 1: Type I and Type III of AFM order in FCC lattices [56]

As shown in Figure 1, the magnetic unit cell in the Type I structure is of the same size of "atomic" or "chemical" cubic unit cell in the FCC lattice [34]. In the Type III arrangement, on the other hand, the lattice periodicity is *doubled* along one of the principal cubic axes, so that the unit cell dimension in cubic coordinates is *(a, a, 2a)* [34]. These two ordering types could be easily distinguished as they produce different peak patterns in neutron diffraction [34]. Such measurements performed on the ZB modification of MnS clearly indicated the Type III order, indicating the NNN interaction is AFM [46, 47].

The results of neutron diffraction experiments on the MnSe and MnTe layers embedded, respectively, in MnSe/ZnSe and MnTe/ZnTe superlattices revealed essentially the same AFM structure as that seen in the ZB MnS [42, 43, 55]. But the results obtained

from MnSe/ZnTe structures were quite surprising, as they indicated an ordering whose periodicity was not doubled, but nearly tripled along the cubic axis [2]. The investigation of several samples showed that the change of periodicity was slightly different for each specimen [2]. In general, it could be described as (a, a, Λa), with the observed Λ values for individual samples between 2.92 and 3.42 [2]. Clearly, the data indicated the formation of an incommensurate helical AFM structure, not yet seen in any other FCC antiferromagnet [2].

The results of experiments were reported in a 1992 paper, together with an analysis based on mean-field theory (MFT) arguments [2]. Simple calculations in terms of MFT showed convincingly that a distortion of the FCC lattice caused by a *tensile* stress experienced by the MnSe layers in MnSe/ZnTe structures introduces anisotropy in the NN interactions [2]. The anisotropy lifts the degeneracy of the frustrated FCC lattice. It minimizes the energy of a helical arrangement whose periodicity is longer than the original doubling of the FCC unit cell in an undistorted lattice. However, the simple model could not explain several facts revealed by the measurement [2]. The mean-field model also suggested that another relevant factor influencing the magnitude of Λ might be the layer *thickness*. However, there was not enough basis for resolving which one of the two – the magnitude of strain or the layer thickness – plays a greater role. Another important fact revealed by the measurements was the temperature (T) dependence of Λ [2]. The $\Lambda(T)$ graphs shown in the paper reveal that, starting from T values equal roughly one-half of the Néel temperature of MnSe, the $\Lambda(T)$ curves are rapidly increasing [2]. Yet, mean-field arguments – which, in fact, are trustworthy only when considering

situations just above the ground state – could not offer much clue for understanding the observed strong dependence of Λ on T when approaching the phase transition point. Cleary, more modeling studies on the system were needed using more sophisticated tools than simple mean-field considerations.

Chapter 2. Interlayer coupling in a NiO/CoO Superlattices

2.1. Persistence of magnetic order

Using neutron diffraction, J. A. Borchers et al. studied artificial superlattices of $[(NiO)_p/(CoO)_q]_N$ (NiO/CoO), in which layers of NiO, an antiferromagnet with Néel temperature of 520K, are alternated with layers of CoO, an antiferromagnet of the same crystallographic structure but with a decisively lower Néel temperature ($T_{Néel,CoO} = 291K$) [1, 34]. They observed that the magnetic order in CoO persisted up to temperatures as high as 30% above its bulk Néel temperature [1]. Apparently the effect occurred due to interlayer exchange pinning of CoO spins by NiO spins [1, 58].

Similar effects have been reported on different magnetic interfaces [58, 59, 60]. P.V. van der Zaag et al. found using neutron diffraction that in Fe_3O_4/CoO bilayers the ordering temperature of the CoO layers was enhanced above the CoO bulk Néel temperature when the CoO layer thicknesses $t \leq 100$Å due to the proximity of magnetic Fe_3O_4 layers [60]. Fe_3O_4 is a ferrimagnet and the Néel temperature of CoO increased as the thickness of the CoO layer decreased in the Fe_3O_4/CoO bilayer [60]. Lenz et al. studied the ordering temperature of an AFM thin films layered with a FM layer using magneto-optic Kerr effect (MOKE) measurements and concluded that the FM layer substantially influences the ordering temperature of the AFM layer due to a proximity effect [59]. Fe_3O_4/NiO superlattices were also studied experimentally by J.A. Borchers et al. by neutron diffraction [58]. They reported that the AFM order of which, to some

simplification, can be treated as a "wave-like" arrangement, propagated through the ferrimagnet [58].

The goal for this chapter is twofold: (1) to obtain a microscopic model to gain insight into the effect occurring at the NiO/CoO interface and (2) to test the quality of the MC model. Both NiO and CoO form Type II AFM order and they are insulators so the electrons are localized [61]. AFM properties of transition metal oxides such as CoO, MnO, and NiO are known to arise from superexchange interaction mediated by the oxygen bonds [61, 45, 46].

AFM oxides and salts, which are most often insulators, are known to form four different types of collinear spin structures [61]. In the absence of carrier which mediates exchange interaction between distant spins, the relevant interaction in insulating materials in FCC structure is usually limited to the first and second nearest neighbors. Different types of magnetic order in the FCC lattice are the results of the interplay of competing first (or NN) and second-nearest neighbor (or NNN) interactions J_1 and J_2 [62].

In oxides, the NN interaction is either from direct exchange or superexchange and NNN interaction is from superexchange [61]. Mediation of oxygen anions is effective, therefore, AFM NNN coupling is dominant and this gives rise to the Type II magnetic order, which consists of antiferromagnetically-coupled FM sheets arranged along the (111)-direction [61, 63, 64]. Each set of FM sheets are called the sublattices of the Type II lattice [63]. The magnetic sublattices of Type II magnetic order are shown in Figure 2.

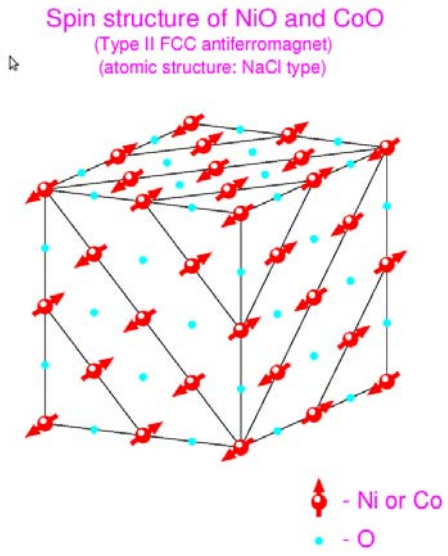


Figure 2: Spin structure of NiO and CoO: antiferromagnetically coupled ferromagnetic sheets are arranged along the (111) direction for Type II magnetic order [63]. The spins of one sublattice are antiferromagnetically coupled to those in the other sublattice [63].

Exchange interaction can be described by the Heisenberg exchange Hamiltonian, which is shown in equation 2.1 [61].

2.2. Model for Monte Carlo calculation

The MC calculations were performed using the Metropolis Algorithm [65, 66]. The size of each spin array was 50×50×50 resulting in 125000 vector spins. The lattice boundaries were set to be periodic, which is common in magnetism simulations [65, 66]. There were three different cases of $[(NiO)_p/(CoO)_q]_N$ superlattices: (1) p = 8, q = 17, (2) p = 11, q = 14, and (3) p = 13, q = 12.

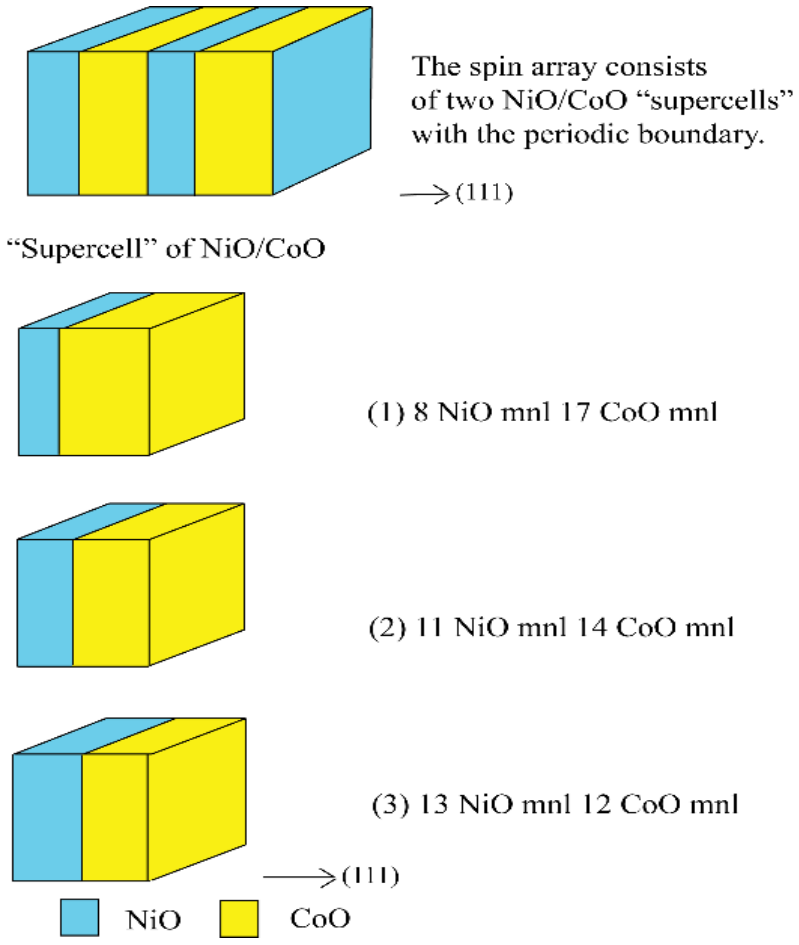


Figure 3: Three different configurations of NiO/CoO superlattices are shown above: (1) $[(NiO)_8/(CoO)_{17}]_N$, (2) $[(NiO)_{11}/(CoO)_{14}]_N$, and (3) $[(NiO)_{13}/(CoO)_{12}]_N$.

The exchange Hamiltonian H_i for a single spin $\boldsymbol{S_i}$ in the lattice is

$$H_i = \boldsymbol{S_i} \cdot \{J_{1in} \sum_{j \in NN} \boldsymbol{S_j} + J_{1out} \sum_{n \in NN} \boldsymbol{S_n} + J_{2in} \sum_{k \in NNN} \boldsymbol{S_k}\}, \tag{2.1}$$

where J_{1in}, J_{1out}, and J_{2in} are exchange constants for in-plane NN interaction, in-plane NNN interaction, and out-of-plane NNN interaction respectively. Then the exchange Hamiltonian for the whole lattice is the sum of the single spin exchange Hamiltonians for all the spins in the lattice.

$$H_{total} = \sum_{i \in all\ lattice} H_i \tag{2.2}$$

To achieve a Néel temperature ratio similar to the real one, the exchange constants for NiO were set $J_{1,NiO}$ = - 0.5 and $J_{2,NiO}$ = 1.0 and those for CoO were set $J_{1,CoO}$ = - 0.25 and $J_{2,CoO}$ = 0.5. At the interface, I set $J_{2interface}$ = 0.75, which is the average value of the two exchange constants $J_{2,NiO}$ and $J_{2,CoO,}$ which was suggested by J. A. Borchers's et al. [1].

Figure 4 shows the NiO/CoO superlattice model $[(NiO)_{11}(CoO)_{14}]_N$. The spin sheets are arranged along the (111) direction with the lattice basis *B* (eq. A.1 in section A1 of Appendix A). A NiO/CoO bilayer was repeated once to reduce the possibility of the boundary imposing a periodicity to the lattice. The free boundary condition is another option but it requires a larger spin array to allow long-range order.

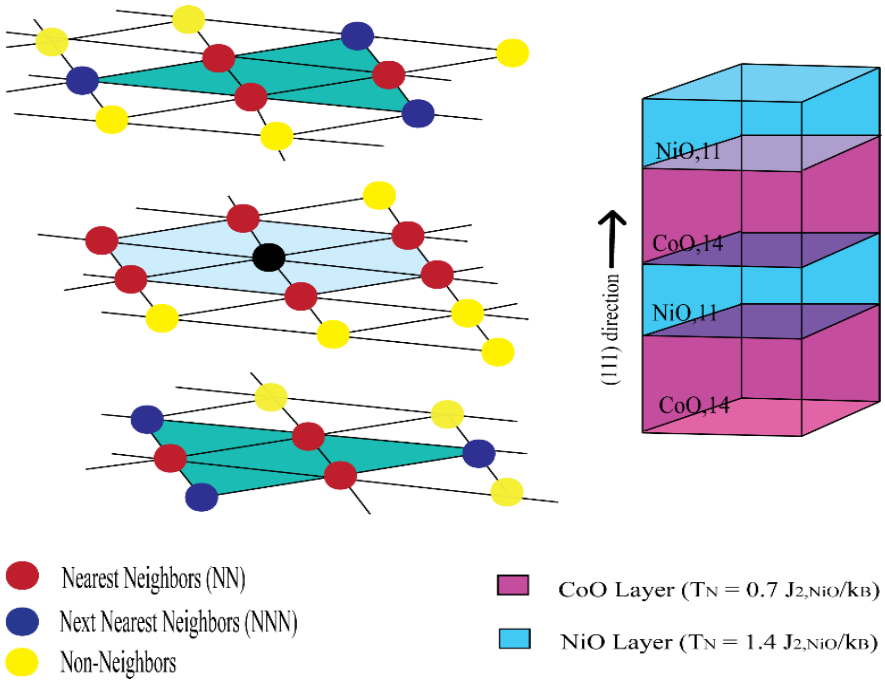


Figure 4: NiO/CoO superlattice model with 11 NiO monolayers and 14 CoO monolayers with Hamiltonian (in equation 2.2), which results in the Néel temperatures $T_{N,NiO}$ = 1.4 $J_{2,NiO}/k_B$ and $T_{N,CoO}$ = 0.7 $J_{2,NiO}/k_B$.

2.3. Data analysis

The variables calculated for NiO/CoO superlattice model include energy, magnetization (**M**), and reduced magnetization ($\mathbf{M}_r$). The (normalized) magnetization of the system **M** is defined as the sum of all the spins in the lattice divided by the total number of spins in the lattice.

$$\mathbf{M} = \frac{\sum_{i \in all\ lattice} \boldsymbol{S_i}}{N_i}, \tag{2.3}$$

where N_i is the total number of spins in the system. I was interested in investigating magnetic order of CoO layer above its bulk Néel temperature. To measure magnetic order in AFM systems, it is important to know their magnetic structures. In principle, magnetic structures including collinear antiferromagnets are completely determined by specifying the magnetic unit cell [16, 38]. Magnetic atoms are linked by distances equal to multiples of the magnetic lattice spacing with parallel orientations[1] [16, 38]. If one adds other atoms with the same spin orientations to these, linked to the original atoms by translations **within** the magnetic cell, then the resultant set of spins is called a "magnetic" sublattice [38]. In ordinary AFM substances the lattice of paramagnetic ions is divided into two sublattices [38]. For an AFM system, magnetic order is measured in terms of the reduced magnetization. The reduced magnetization $\mathbf{M}_r$ is defined as

$$\mathbf{M}_r = \frac{\sum_{j \in sublattice} \overrightarrow{S_j}}{N_j}, \tag{2.4}$$

where N_j is the total number of spins in each sublattice of the superlattice.

In the AFM state, spins on one sublattice are parallel to each other and antiparallel to spins on the other sublattice [67]. A sublattice can be specified by the wave vector **k**, where the magnetic moment **m**(**r**) of the ion at the position $\boldsymbol{r}$ is related to the Fourier components $\boldsymbol{m_k}$ by the expression of equation 2.5 [67, 68].

$$\mathbf{m}(\mathbf{r}) = \sum_k \boldsymbol{m_k} \exp(2\pi i \boldsymbol{k} \cdot \boldsymbol{r}) \quad (2.5)$$

There are structures with more than one pair of sublattices and they are called the multi-**k** structure, where the vector **k** is identified with the propagation of a specified component of magnetic moment **m**(**r**) [67]. To measure magnetic order in an AFM lattice, an appropriate choice of sublattices is required [67].

The transition metal oxides NiO, MnO, CoO, and FeO crystallize in the rock-salt structure and order with a Type II AFM structure characterized by the wave vector $\mathbf{k} = (\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$ [67]. The reduced magnetization for sublattices along the (111)-direction is calculated for each layer for different temperatures since both NiO and CoO exhibit Type II magnetic order, as described in Figure 2. The total reduced magnetization calculation on the bulk CoO and bulk NiO model confirmed that the magnetic AFM order was formed along the (111) direction. The detailed procedures on the reduced magnetization calculation are described in Section A2 of Appendix A.

2.4. Results and discussion

Figures 5 and 6 show the reduced magnetization plotted as a function of temperature for bulk CoO and NiO model respectively. The Néel temperature corresponds to the temperature at which the reduced magnetization goes to zero [38].

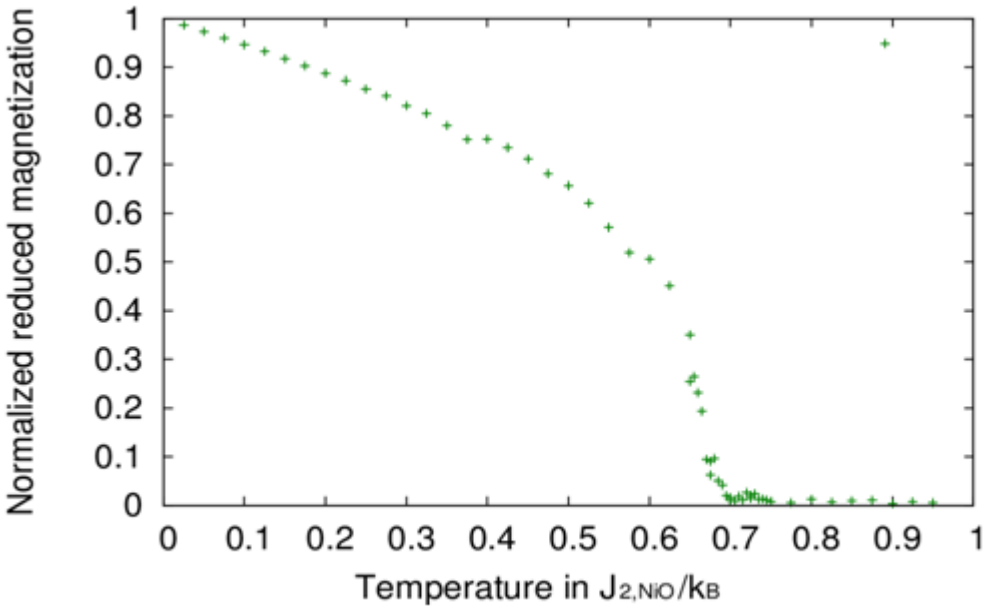


Figure 5: Reduced magnetization vs. temperature for bulk CoO model: The Néel temperatures is approximately $T_{N,CoO} = 0.7\ J_{2,NiO}/k_B$.

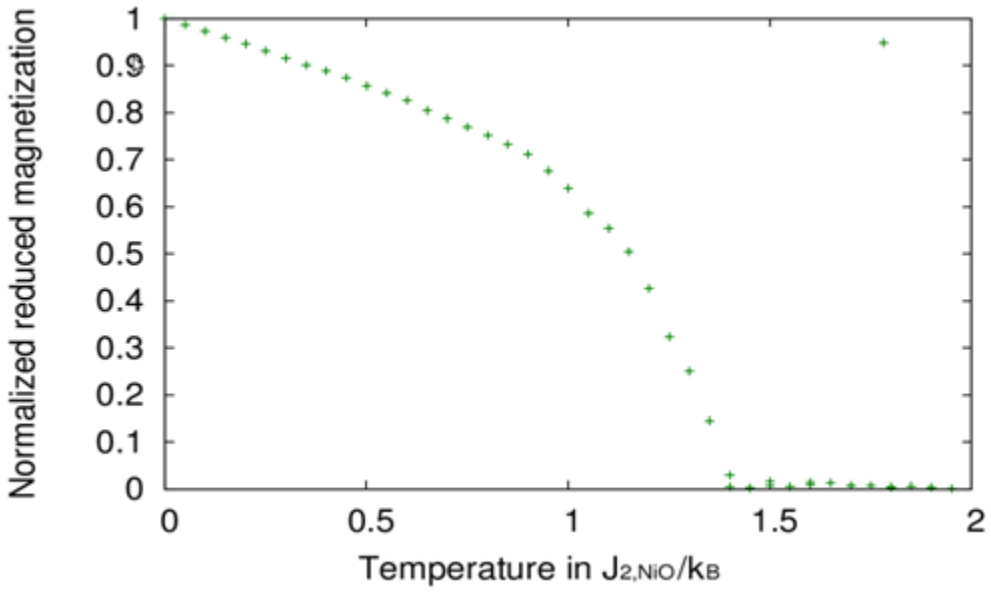


Figure 6: Reduced magnetization vs. temperature for bulk NiO model: the Néel temperatures is approximately $T_{N,NiO} = 1.4\ J_{2,NiO}/k_B$.

With the exchange parameters described above, the bulk calculations resulted in the Néel temperatures for NiO and CoO $T_{N,NiO} = 1.4\ J_{2,NiO}/k_B$ and $T_{N,CoO} = 0.7\ J_{2,NiO}/k_B$ respectively. After estimating the Néel temperatures for bulk CoO and NiO models, I calculated the normalized reduced magnetization of the NiO/CoO superlattice as a function of temperature.

Figure 7 shows the normalized reduced magnetization as a function of temperature for the NiO/CoO superlattice model with varying number of NiO and CoO monolayers. Reduced magnetization vs. temperature curves for bulk CoO and NiO models are plotted together for comparison. Figure 8 is included to compare the simulation result to the experimental result by J.A. Borchers's group [1]. Figure 8 is the plot of the reduced moment (which is equivalent to the normalized reduced magnetization in my simulation) vs. temperature measured by J.A. Borchers et al. [1]. The curves for

the superlattices in Figure 7 look similar to those from J.A. Borchers's experimental results in Figure 8 [1].

As I increase the number of NiO monolayers (in Figure 7), the "dip" in the reduced magnetization curve above the Néel temperature of the bulk CoO gets "shallower", which is reasonable, since increasing the thickness of NiO and decreasing the thickness of CoO should "strengthen" the magnetic order of the system.

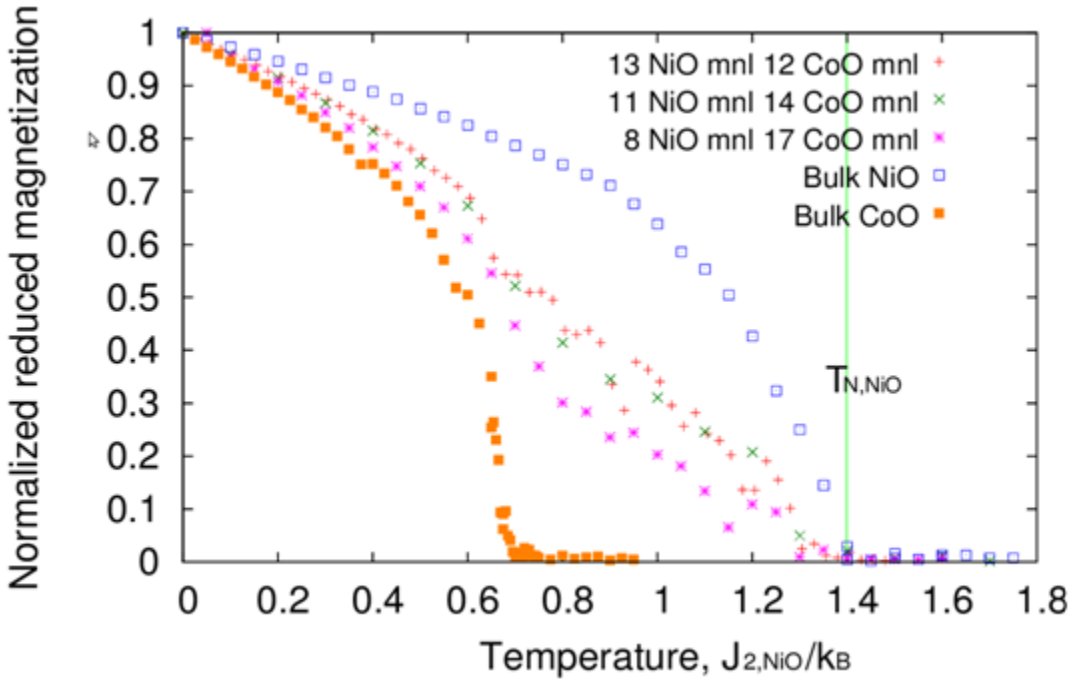


Figure 7: Reduced magnetization of NiO/CoO lattices for three cases: (1) 13 NiO monolayers and 12 CoO monolayers, (2) 11 NiO monolayers and 14 CoO monolayers, and (3) 8 NiO monolayers and 17 CoO monolayers.

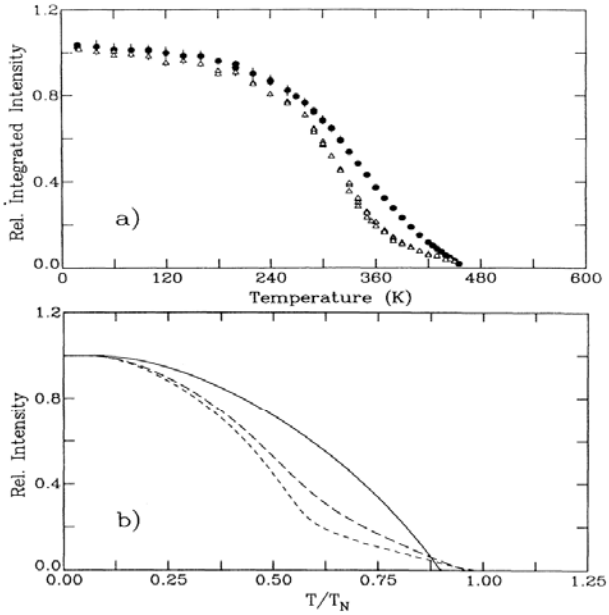


FIG. 2. (a) Temperature dependence of the integrated intensity of the central $(\frac{1}{2}\frac{1}{2}\frac{1}{2})$ magnetic peak for [NiO(21 Å)|CoO(15 Å)]$_{145}$ (dark circles) and [NiO(43 Å)|CoO(29 Å)]$_{100}$ (open triangles). (b) Mean field calculation of the relative intensity of the $(\frac{1}{2}\frac{1}{2}\frac{1}{2})$ peak as a function of temperature for NiO/CoO bilayers consisting of three (solid line), eight (long dashes), and fifteen (short dashes) atomic planes of both Ni and Co. The temperature is scaled by T_N for NiO.

Figure 8: Experimental results from J.A. Borchers's group [1]; the relative intensity of the magnetic peak is equivalent to the reduced magnetization in my calculations.

To investigate the magnetic order of each CoO monolayers in the superlattice model, I plotted the reduced magnetization per individual monolayer $\mathbf{M}_{rl}$ for different temperatures, which is defined as:

$$\mathbf{M}_{rl} = \frac{\sum_{j\in sublattice,l} \boldsymbol{S_{jl}}}{\mathrm{N_{jl}}}, \tag{2.5}$$

where l is the spin monolayer index, and they are plotted as a function of the monolayer index in Figures 9, 10, and 11. It should be also noted that the bulk Néel temperature of CoO model is $T_{N,CoO}$=0.7 $J_{2,NiO}/k_B$ for Figures 9, 10, and 11. In Figure 9, the reduced magnetization does not become zero at the bulk Néel temperature of CoO. The magnetic

order of CoO persists significantly at temperatures up to $0.9J_{2,NiO}/k_B$, which is approximately 28% above the Néel temperature of bulk CoO. As I increase the number of NiO monolayers, the magnetic order of CoO persists up to higher temperatures. Also the magnetic order grows stronger as the monolayers gets closer to the NiO/CoO interface in all the studied cases of NiO/CoO superlattices.

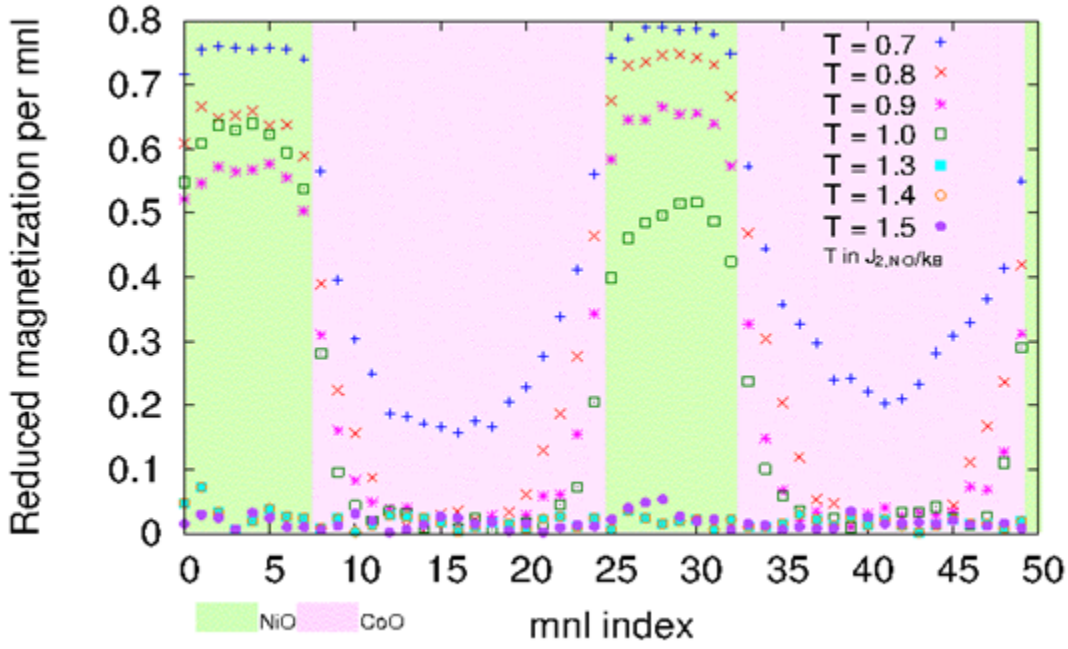


Figure 9: (Normalized) Reduced magnetization per monolayer (mnl) for superlattice with 8 NiO monolayers and 17 CoO monolayers for different temperatures, where $T_{N,CoO} = 0.7\ J_{2,NiO}/k_B$ and $T_{N,NiO} = 1.4\ J_{2,NiO}/k_B$

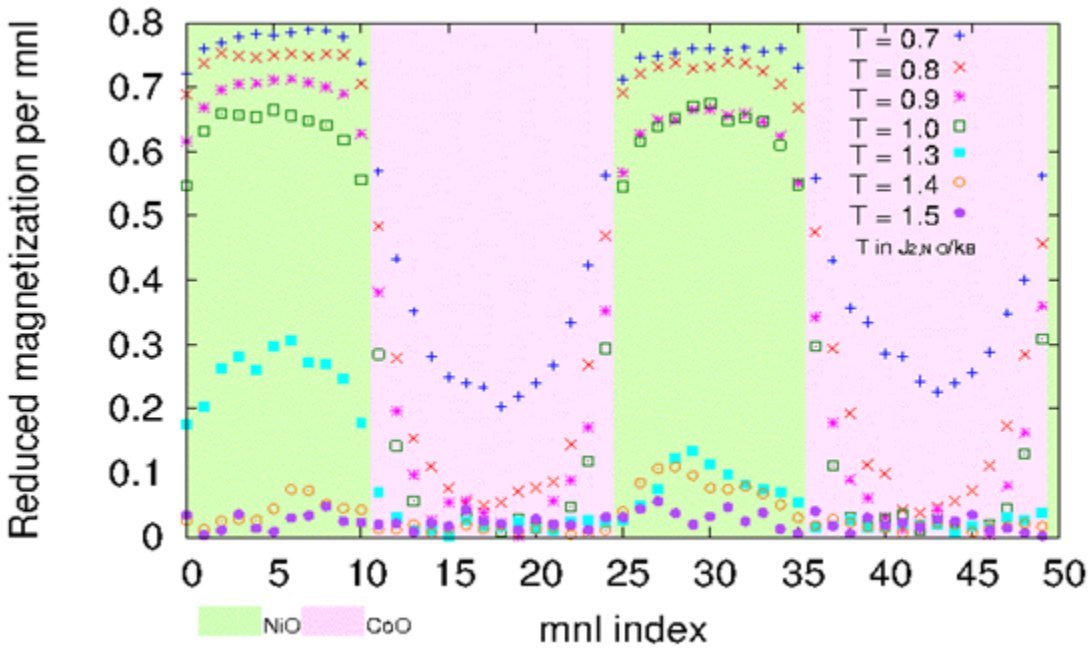


Figure 10: (Normalized) Reduced magnetization per monolayer for 11 NiO monolayers and 14 CoO monolayers; the Néel temperatures are $T_{N,CoO} = 0.7\ J_{2,NiO}/k_B$ and $T_{N,NiO} = 1.4\ J_{2,NiO}/k_B$ for CoO and NiO respectively.

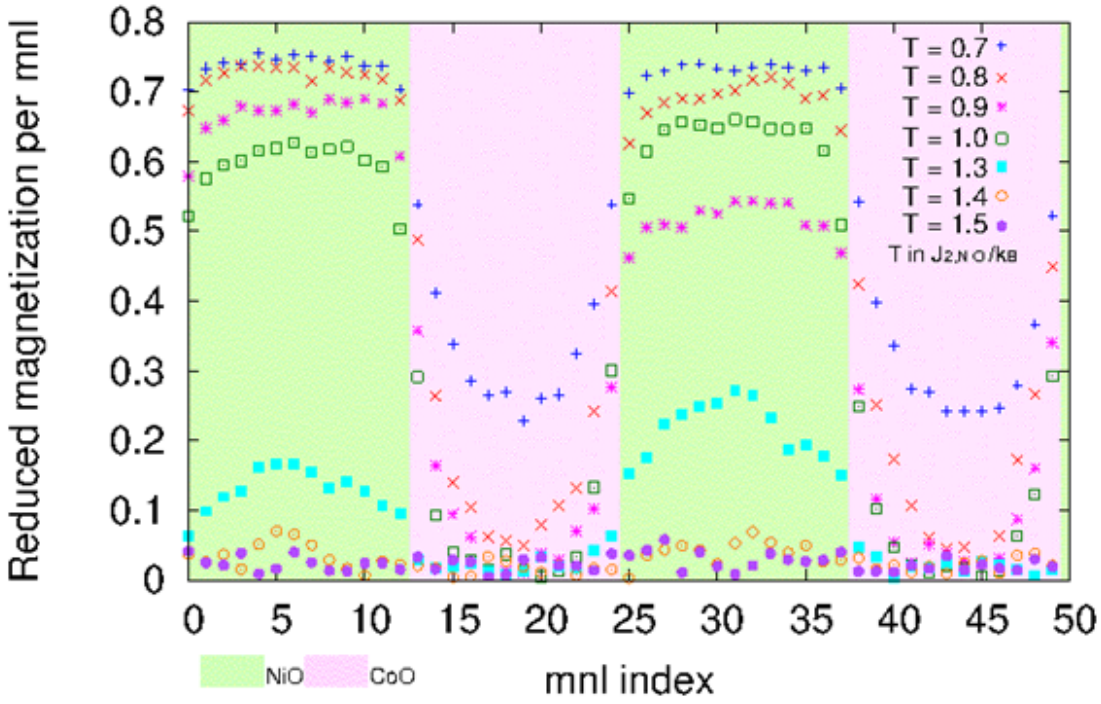


Figure 11: (Normalized) Reduced magnetization per monolayer (mnl) for 13 NiO monolayers and 12 CoO monolayers; the Néel temperatures are $T_{N,CoO} = 0.7\ J_{2,NiO}/k_B$ and $T_{N,NiO} = 1.4\ J_{2,NiO}/k_B$ for CoO and NiO respectively.

2.5. Summary and comments

The results from the MC calculations indicate that the magnetic order of the CoO layer persists at temperatures above its bulk Néel temperature. This suggests that CoO spins on the NiO/CoO interface are effectively "pinned" to NiO spins by the exchange interaction on the interface [1]. Increasing the number of NiO spin sheets "strengthens" the magnetic order in the CoO layer. In all the studied cases, magnetic order disappeared for temperatures higher than the Néel temperature of NiO model. Therefore, it can be

concluded that the results from the MC calculations are in a reasonable agreement with the experimental results from J. A. Borchers's group [1].

It should be emphasized that the purpose of this chapter is to model and test the MC model by comparing the calculation results to the experimental results. Further investigation on superlattices composed of antiferromagnets only will be a useful project in that one can apply a more complex model to antiferromagnet-only systems and obtain a more-detailed model, but it is outside the scope of my Ph.D. project. In the next chapter, this MC model is used to perform MC calculations on FM/AFM/FM trilayer systems, which does not have associated experimental results.

Chapter 3. Interlayer coupling in FM/AFM/FM trilayer lattice

In the preceding chapter, I discussed effects occurring in artificial layered structures composed of two different AFM materials of different Néel temperatures. The effect seen in the CoO/NiO superlattices is interesting from the perspective of fundamental studies in the physics of magnetism. The results obtained from neutron scattering experiments of such superlattices are valuable for the present project as they offer excellent material for testing a MC model.

However, except for their role in fundamental studies of magnetism, systems consisting exclusively of AFM components have a relatively limited impact on research areas in which application-oriented research and design work on artificial systems is more actively conducted. The reason is that antiferromagnetism is essentially a phenomenon that can be observed only by research tools operating on a truly microscopic level [11, 25, 69]. Because the net magnetic moment of an antiferromagnet is zero, it produces no macroscopic effects readily detectable by standard apparatus that is used for investigating magnetic materials [11, 25]. For instance, one typically measured "macro" characteristic is the magnetic susceptibility of a given material vs. temperature, $\chi(T)$ [11]. With the onset of AFM order (i.e., below the Néel temperature), the $\chi(T)$ curve starts deviating from the Curie Law obeyed by ordinary paramagnets – but the deviation is often very weak and detecting it may require using ultra-sensitive magnetic measuring techniques (such as, e.g., SQUID magnetometry) [70]. Because of their very weak reaction to external magnetic fields, AFM systems did not find any practical applications

in technology – neither bulk materials, nor artificial nanostructures prepared from combinations of AFM-only compounds.

On the other hand, artificial structures in which AFM materials *are combined with ferromagnets* have become the object of great interest since the mid-1990s, of both physicists and technologists. The reason was that such systems emerged as crucial components in novel generations of nanoscale magnetic field sensors. Most notably, in the so-called "spin valves" — i.e., sensors based on the effect of either GMR or TMR [10].

The simplest GMR device is a trilayer composed of two FM layers sandwiching a NM metal [6, 10]. The resistivity of the device changes as the relative orientation of the magnetizations of two FM films changes, responding to an applied external magnetic field **B** [6, 10]. The spacer layer must be made from a material that can convey RKKY exchange interactions between the two FM films. The sign of the RKKY interaction oscillates depending on the distance between the interacting spins [13, 14, 15]. Therefore, the thickness of the spacer layer should be such that it leads to an *antiparallel* orientation of the magnetization vectors in the two FM layers [12].

One can view the RKKY interaction as one in a "torsional spring" connecting the two magnetization vectors ($\mathbf{M_1}$, $\mathbf{M_2}$) in the FM layers and maintaining their antiparallel orientation. Then an external magnetic field **B** applied to the system exerts a torque on each magnetization vector ($\mathbf{M_1}$, $\mathbf{M_2}$), inclining them to be parallel. As the angle between $\mathbf{M_1}$ and $\mathbf{M_2}$ vectors gradually changes with increasing **B**, so does the electric resistance of the trilayer (the MR is maximum from antiparallel magnetization vectors, and is the

smallest when they are completely parallel) [4, 5, 6]. However, such a system is sensitive only to the **magnitude** of **B, not to its direction**.

In many practical applications the information on the magnetic field direction may be far more important than knowing its magnitude (for instance - in digital magnetic recording) [10]. To make a GMR sensor direction-sensitive, the orientation of the magnetization vector in one of the FM films should be *immobilized* or "pinned down" to the substrate on which the trilayer system is prepared [10, 26].

One of the methods to pin down or immobilize one of the FM spins is to use the phenomenon of exchange bias (EB) [71]. EB was initially observed by Miklejohn and Bean in their study of the CoO-Co interfaces [72]. They found that the magnetic hysteresis loop for oxidized Co nanoparticles was shifted from the origin [72]. They explained that this observation was the result of exchange coupling between the Co spins in the Co/CoO interface region – i.e., between those in the metallic cobalt and those that are built in the CoO lattice [72]. CoO is an antiferromagnet which in fact, by itself, makes its spin lattice very insensitive to an external field [61]. In addition, in CoO, the "intrinsic" reluctance of the AFM sublattices to react to external B fields is augmented by a strong magneto-crystalline anisotropy [61]. Thus the direction of the Co^{2+} spins in the atomic plane of CoO adjacent to the metallic Co lattice remains fixed. The spins in metallic Co in the interface region remain "pinned" to the immobile CoO lattice during the magnetization-demagnetization cycle when the magnetic hysteresis of the system is measured [72]. It creates a "memory effect" which causes the Co magnetization to return to its original orientation after the field is removed. It also manifests itself as an

asymmetry in the hysteresis loop [72].

Until early 1990s the EB effect was primarily the subject of interest of scientists working on fundamental aspects of magnetism. A dramatic increase of interest in EB began when it turned out to be crucial for building sensors for read-heads of hard drives – the type of sensor commonly called a “spin valve” [10, 73].

However, the effect occurring at the FM/AFM interface in spin valves seemed to be completely unrelated to the principal physical mechanism underlying the phenomenon of GMR itself - i.e., to the interlayer coupling between the constituent FM films, The phenomenon of interlayer coupling is crucial for the operation of *any* MR sensor, with or without an FM film immobilized [10, 48]. In a "canonical" GMR, the two FM films interact via a *NM* spacer material [10]. Hence, in the years following the discovery of GMR, the attention of researchers was focused primarily on how *electrons – mobile ones* in metals, and *valence electrons* in certain cases when the spacer material was insulating – mediate or convey the interaction across the spacer [13-15, 18]. The spacer itself was treated as a mostly passive “blank sheet” for the electrons rather than active players.

No other mechanisms received any significant attention until in 1995 J. C. Slonczewski pointed out that there may yet be another way of conveying interlayer exchange across the spacer – a way not involving the electrons directly. Rather, the interactions would be transmitted via magnetic forces acting within the spacer [33]. He stressed that some spacer materials used in GMR studies (e.g., Cr, Mn and their alloys) were not “NM”, but actually were *AFM* — in other words, they were NM in the sense of *not having a net magnetic moment*. But such an AFM spacer, as Slonczewski argued,

would no longer be “magnetically neutral”. If the FM films were magnetically coupled to it from both sides via interfacial interactions, they would thus be able to modify the condition within the thin AFM spacer through an effect that he called *proximity magnetism* [33]. If the $\mathbf{M_1}$ and $\mathbf{M_2}$ vectors in the FM layers were not ideally parallel, the AFM structure would be “twisted” and it would act like a “torsional spring”, producing torque that would combine with the action of the electrons [33].

In fact, in the Slonczewski “magnetic proximity model” there is no “entirely new physics”, it’s rather the application of a known concept in an entirely *new role*. “Spin springs” of the same type are known to occur in domain walls in antiferromagnets [63, 74]. By forming a domain structure, the antiferromagnet may minimize its magnetic energy. The process is not exactly the same as in the case of domain structure formations in ferromagnets [63]. The ordered spin structure in a ferromagnet is the source of a macroscopic *magnetic field* whose energy is a significant part of the total magnetic energy [74]. By forming a multitude of differently oriented FM domains the system effectively reduces the macroscopic field and the related energy to zero or nearly zero [74]. The price for that is that the formation of the “Bloch walls” between requires expending some energy by the system – however, the energy expended for it is less than the “energy gain” coming from reducing the macroscopic field [74].

In contrast to ferromagnets, atomic moments of AFM structures do not produce any macroscopic field but they use other physical mechanisms, which can make the formation of domains energetically favorable (e.g., magnetoelastic effects) [75]. As in ferromagnets, the AFM domain walls have certain energy and they produce a *torque*

tending to align the AFM spin structures in the domains on both sides. With some simplification, it can be said that Slonczewski in his model "found a new possible role for the AFM domain walls. by placing them not in between two AFM domains, but between two FM layers coupled to them by exchange forces [33].

The original paper by Meiklejohn and Bean (M&B) on exchange bias is now a "classic", with numerous citations and the 1995 Slonczewski's paper on the "magnetic proximity effect" has also become a "classic" [33, 72]. When I was searching for articles focusing on effects occurring in the interface regions of exchange-coupled FM and AFM films, I came across many papers that deal with exchange bias, interlayer coupling in $(AFM/FM)_N$ superlattices and FM/AFM trilayers [39, 76-82]. In many of the papers I read, the M&B paper was quoted and in many others the Slonczewski's paper was – yet, what seemed somewhat intriguing to me, those two "classics" were almost never cited in the same paper. Since both these effects occur in FM/AFM interfaces, it seemed somewhat surprising to me that researchers investigating phenomena occurring in such interfaces focused exclusively on one effect or the other, as if they were completely unrelated phenomena.

The reason why authors reporting research on exchange bias seemed to "ignore" the Slonczewski's "proximity effects", and vice versa, is probably caused by the fact that the basic models describing the situations in systems exhibiting "exchange bias", and those exhibiting "proximity effects", describe *two extremes* in exchange-coupled FM/AFM interfaces [33, 72, 83, 84]. In the exchange bias models the AFM material is usually pictured as "extremely rigid", whereas it is the FM material that yields to the

external field **B** and forms a "spin helix" adjacent to the AFM "rigid wall" [85].

In contrast, in Slonczewski's basic model of "magnetic proximity" it is rather the antiferromagnet that is "soft" and forms a "sping-like" helix coupling two FM films that incline toward the external field **B**, but spin structure on each of the two films remain collinear, not affected by **B** [33]. The above is illustrated in Figure 12, which shows (a) a "perfectly rigid" AFM film exchange-coupled to a "soft" FM layer, in zero and non-zero external field B, and (b) and a "soft" AFM layer exchange-coupled on both sides to "perfectly rigid" FM films, in zero and non-zero **B**.

The "prototypical situation" in the planned modeling seemed to be an "AFM spin spring" as discussed in the Slonczewski's model [33, 81]. Then a question that naturally arises from such a physical picture is: *Are all ferromagnetic materials indeed much stiffer than antiferromagnets they can be exchange coupled with?* Of course, there are no "perfectly rigid" FM or AFM spin lattices.

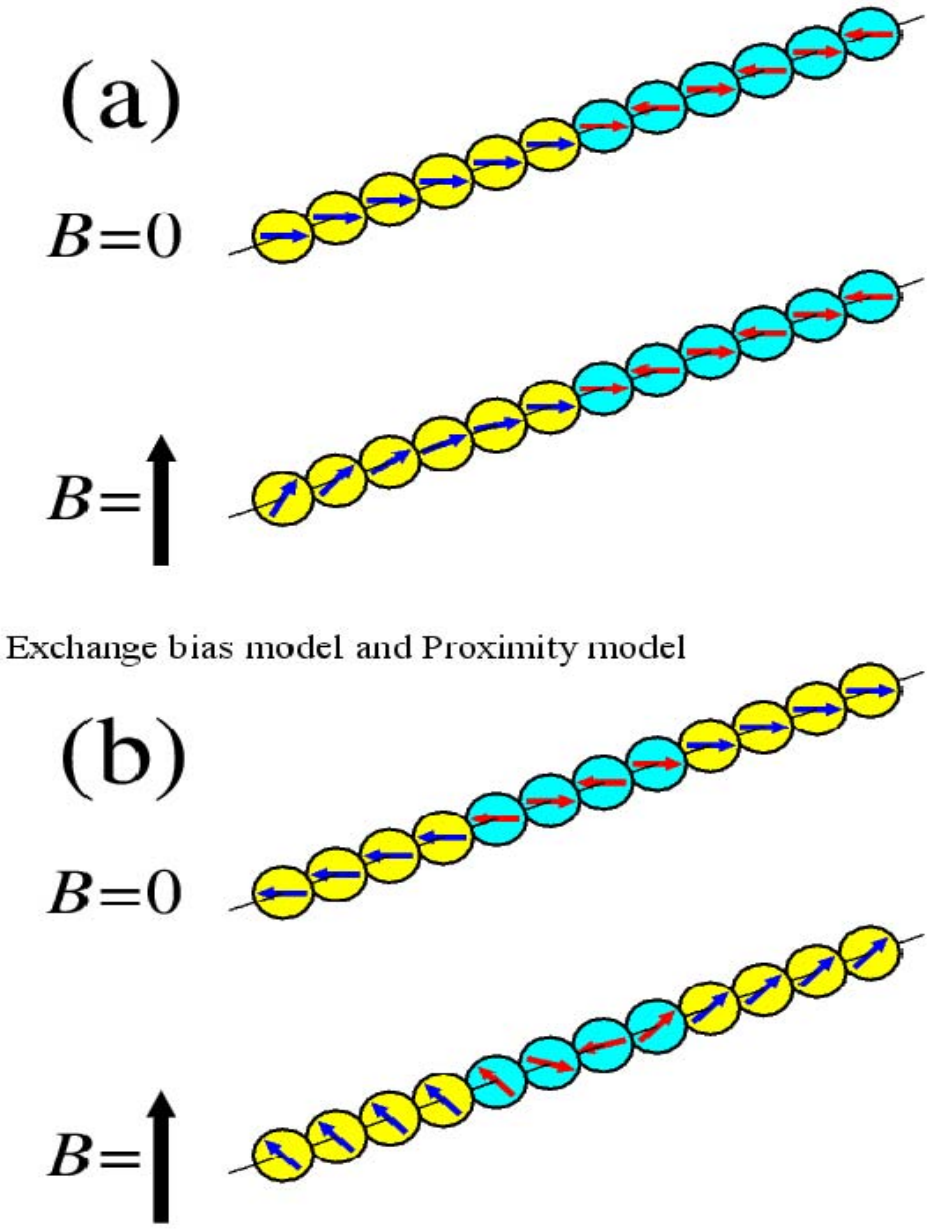


Figure 12: Illustrations of behaviors of FM/AFM systems implied in (a) exchange bias model and (b) the proximity model: (a) shows a FM/AFM bilayer with a rigid AFM layer and (b) shows a FM/AFM/FM trilayer with a rigid FM layers.

A reasonable assumption is therefore that real systems are positioned somewhere in between the two extreme cases, represented by the basic versions of the "magnetic

proximity" and the "exchange bias" models. In the model investigated, neither material was assumed to be "perfectly rigid". A torque exerted by an external magnetic field would produce a helical twist in both materials — and the magnitude of the "twist effect" in each material could be controlled by the input parameters in the model.

Further, some thoughts were given to the meaning of the word "proximity". In the original Slonczewski's paper, it was the "proximity" of FM layers that overcomes the natural tendency of a simple AFM spin lattice to form a *collinear* structure [33]. However, is it the only possible modification of the behavior of AFM lattice caused by the proximity of another magnetic lattice? What comes to one's mind in this moment is the behavior of the CoO layers in the CoO/NiO superlattices discussed in Chapter 2. Here, the behavior observed by Borchers *et al.* – namely, the persistence of the AFM spin structure in CoO well above its bulk Neel temperature – was definitely caused by the *proximity* of the NiO layers which retain their AFM structure to much higher a temperature than the CoO ones [1]. Would such a "proximity effect" also occur if the AFM layer in a FM/AFM/FM trilayer is the AFM layer is already above its bulk Néel temperature, but the FM layers are still in the ordered state? Would such proximity increase the effective Néel temperature of the AFM layer?

Thirdly, there is yet another conceivable "proximity effect" that one can think of. Namely, temperature is one factor capable of destroying spin order in an antiferromagnet. Another one is *magnetic dilution*. Substitution of some part of magnetic atoms in the AFM lattice lowers the Néel temperature, and if it exceeds a certain threshold value, it completely destroys the AFM order [86, 87]. So, an interesting question may be, does the

"proximity" of ordered FM lattice reduce the tendency of the AFM's order to collapse? Or, conversely, would a diluted AFM lattice be capable of maintaining interlayer coupling between FM films? The latter question may have some practical importance: namely, a material widely used as a spacer in TMR devices is MgO, which readily "mixes" with CoO or MnO to form diluted AFM lattices (such "mixed" oxides can be obtained, e.g., by sintering methods) [75, 88].

The original goal in this project was to investigate the interaction between two FM films conveyed by an AFM spacer with MC methods. As noted, this represents a new physical situation relative to the much-investigated FM-FM exchange coupling mediated by electrons (mobile, or valence-band) in non-magnetic spacers. The fact that such coupling is participating in the overall exchange coupling in the systems, showing the "magnetic proximity effect", definitely makes it conceivable that the "spring effect" may be the principal player in FM-FM interlayer coupling in systems in which the AFM layer is *insulating* [33, 89]. (The candidate materials are AFM insulators such as CoO, NiO, and the strength of interlayer coupling dependent on the materials) When FM layers are separated by an AFM insulator, RKKY model is excluded from the picture, since the RKKY interaction couples the magnetic moments in FM layers through the conduction electrons of a NM metal spacer [61]. And such systems may definitely be of interest from the viewpoint of magnetic field sensors utilizing the phenomenon of TMR, where the spacer must be an insulator [27-29].

In sum, the tasks in the simulation studies essentially focused on investigating a variety of "proximity" effects. The coupling of two FM layers just by the "spin spring" in

the AFM layers is the closest to the original concept from Slonczewski – minus the coupling effects of the electrons, which are not taken into account in the present work.

Two other types of possible "proximity effects" are investigated – the persistence of the AFM order above the bulk Néel point, and weakening the effects of magnetic dilution of the AFM lattice. In addition to that, the model is certainly more realistic than the highly idealized system discussed by Slonczewski in his original 1995 paper, in which the term "magnetic proximity effects" was coined [33, 90]. The model in the present work takes much from the "exchange bias" theory, allowing the FM components, not only the AFM ones, to form "spin springs".

I also like to point out that there has been a growing interest in the effect of interlayer coupling between FM layers mediated by AFM spacers while the present work was progressing. Experimental and theoretical papers have been published on FM/AFM/FM trilayer structures [89, 91-95].

3.1. FM/AFM/FM trilayer with no anisotropy

3.1.1. Model for Monte Carlo calculations

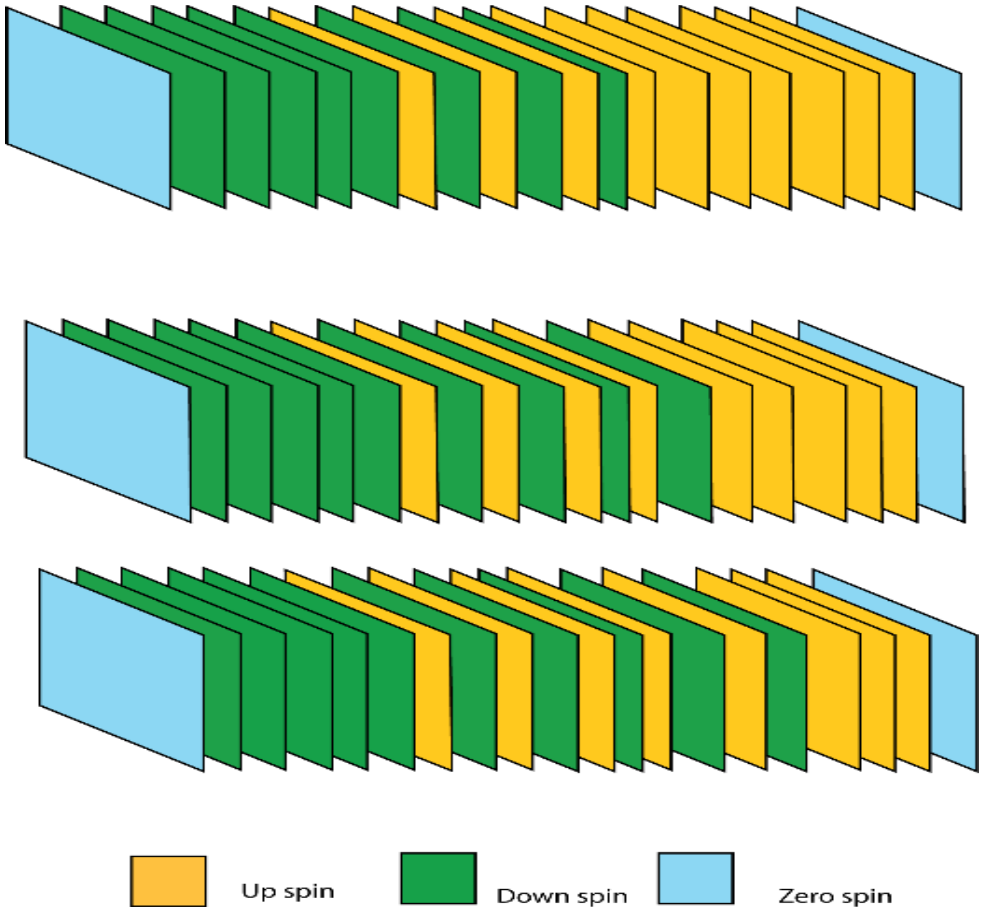


Figure 13: FM/AFM/FM trilayer lattice without external magnetic field **B**; the boundary is periodic for each sheet and there are zero spin sheets on both sides. The monolayers are arranged along the (111) direction.

In this section I discuss a method to control the strength of magnetic correlation between the FM layers influenced by the thickness of AFM spacer layer and the strength of AFM coupling between the two FM layers when there is no anisotropy in the system. Figure 13 shows the FM/AFM/FM trilayer model used for the MC calculations. I used a

$50 \times 50 \times 50$ lattice, which consists of 125000 classical vector spins. The spin monolayers were on the xy-plane and the z-axis was perpendicular plane (growth axis). The first and last spin monolayers were set to be zero as a convenient way of imposing free boundary conditions in the direction corresponding to the z-direction. Periodic boundary was used in the xy-direction and the number of monolayers in the AFM mid-layer was varied. Calculations were performed for three different AFM layer thicknesses: (1) $n_{AFM} = 4$ AFM monolayers, (2) $n_{AFM} = 6$ AFM monolayers, and (3) $n_{AFM} = 8$ AFM monolayers. The number of FM monolayers on each side was set identical: $\frac{48-n_{AFM}}{2}$. In all cases, the basis for the lattice vectors was identical to those used in NiO/CoO superlattice model. The Hamiltonian H_i for the ith spin in the lattice defined as:

$$H_i = \boldsymbol{S_i} \cdot (J_{1in} \sum_{j \in NN,in} \boldsymbol{S_j} + J_{1out} \sum_{j' \in NN,out} \boldsymbol{S_{j'}} + J_{2out} \sum_{k \in NNN,out} \boldsymbol{S_k} - \mathbf{B}) \quad (3.1)$$

This gives the Hamiltonian H of the lattice

$$H = \sum_{i \in lattice} H_i \quad (3.2)$$

The exchange constants for NN interaction in both the AFM layer and FM layer were set to be $J_{1FM} = J_{1AFM} = -0.5$. The NNN exchange constants were set $J_{2AFM} = 1.0$ and $J_{2FM} = -1.0$ for the AFM layer and the FM layer respectively. These exchange constants with the Hamiltonian in eq. 3.2 resulted in the Curie temperature $T_C = 3.50\ J_2/k_B$ of the bulk ferromagnet, where J_2 is defined as $J_2 = |J_{2FM}| = |J_{2AFM}|$. The Néel temperature of the bulk antiferromagnet was $T_C = 1.41 J_2/k_B$. For the interface, I used $J_{1interface} = -1.0$ and $J_{2interface} = 0.5$. The sign of $J_{2interface}$ is not expected to change the result of the calculation since both positive and negative $J_{2interface}$ (which correspond to AFM and FM neighbor interactions respectively) will cause AFM coupling of the FM layers.

3.1.2. Data analysis

A naïve approach to calculate AFM coupling with external fields is measuring the magnetic moment (or magnetization **M**) of the lattice induced by an external magnetic field (**B**) and extract the coupling coefficient by plotting **M** against **B.** Before the magnetic field is applied, the net magnetic moment of the lattice is approximately zero since the number of FM monolayers on each side is the same and the number of AFM monolayers is even. As an external magnetic field increases in intensity, the magnetic moment of the trilayer is expected to change monotonically until it reaches saturation when there is no anisotropy, which is shown in Figure 14.

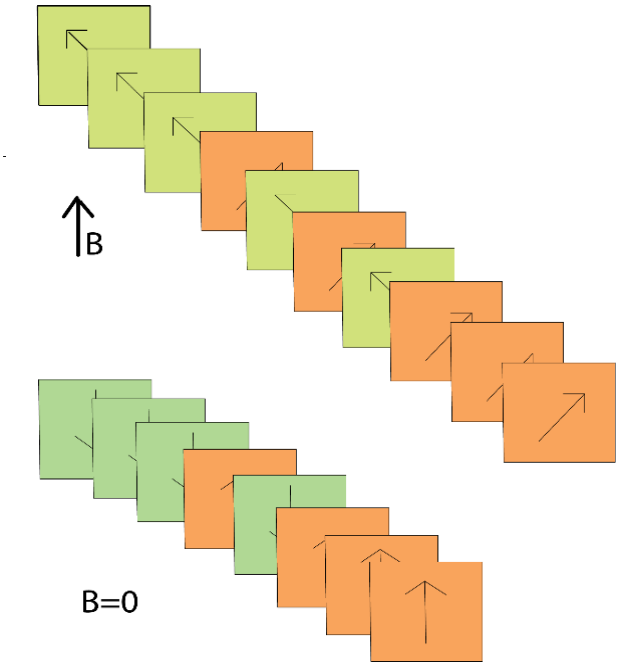


Figure 14: Spins on each sheet are expected to move toward the direction of the external magnetic field in a uniform fashion. There are two orientations of FM sheet magnetization vectors denoted by green and orange colors.

The trilayer monolayers are arranged along the (111) direction and I can view each individual monolayer as FM. As the magnetic field increases, I assume that the magnetization per each monolayer curls toward the direction of the magnetic field uniformly. As a result, there appear two types of FM monolayers, which are denoted by green and orange colors in Figure 14. The total magnetization of the system induced by the external magnetic field (which I call the induced magnetization $\mathbf{M}_{\text{induced}}$) is the sum of the magnetizations of the two types ($\mathbf{M_1}$ from monolayers in orange and $\mathbf{M_2}$ from monolayers in green in Figure 14) of spin sheets.

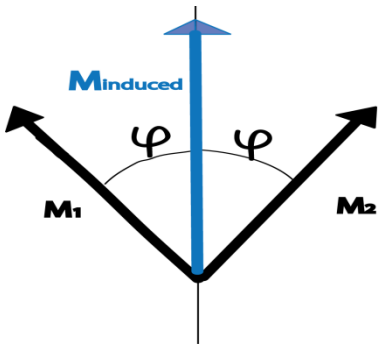


Figure 15: Expected induced magnetization with nonzero magnetic field: $\mathbf{M_1}$ and $\mathbf{M_2}$ are the magnetizations from two types of FM sheets respectively. The magnetizations will curl toward the direction of the external field and the blue arrow will be the result of the sum. $\mathbf{M}_{\text{induced}} = \mathbf{M_1} + \mathbf{M_2}$. (In this case, in the direction of the induced magnetization is the same as that of the external magnetic field.)

It is expected that the slope of the $\mathbf{M}_{\mathbf{induced}}$ vs. $\mathbf{B}$ will be larger as there is a weaker AFM coupling between the FM layers. Therefore, I can define the AFM coupling coefficient A by plotting $\mathbf{M}_{\text{induced}}$ against $\mathbf{B}$ and taking the slope u of the plot. Then A is defined as $A = |\frac{1}{u}|$. Larger coefficient A means a stronger AFM coupling between the FM layers.

3.1.3. Results and discussion

Figure 16 shows the normalized magnetization of the bulk FM model vs. temperature, which gives the Curie temperature for the bulk FM of approximately $3.5J_2/k_B$.

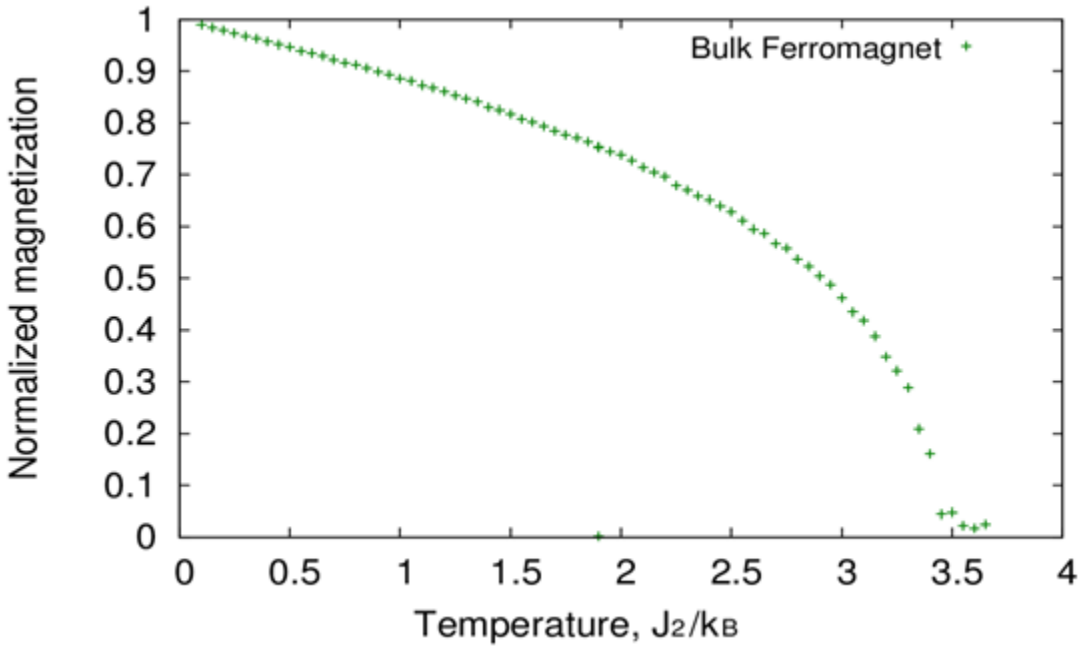


Figure 16: Magnetization vs. temperature for bulk ferromagnet using equation 3.1: The Curie temperature is approximately $T_{Curie} = 3.5\ J_2/k_B$.

The next step is to plot the induced magnetization against the external magnetic field **B** for different temperatures to obtain the AFM coupling coefficient. Figure 17 is an example of $\mathbf{M}_{induced}$ vs. **B** plots. In this case, $\mathbf{M}_{induced}$ is the component of the total magnetization of the trilayer that is in the direction of the magnetic field **B**.

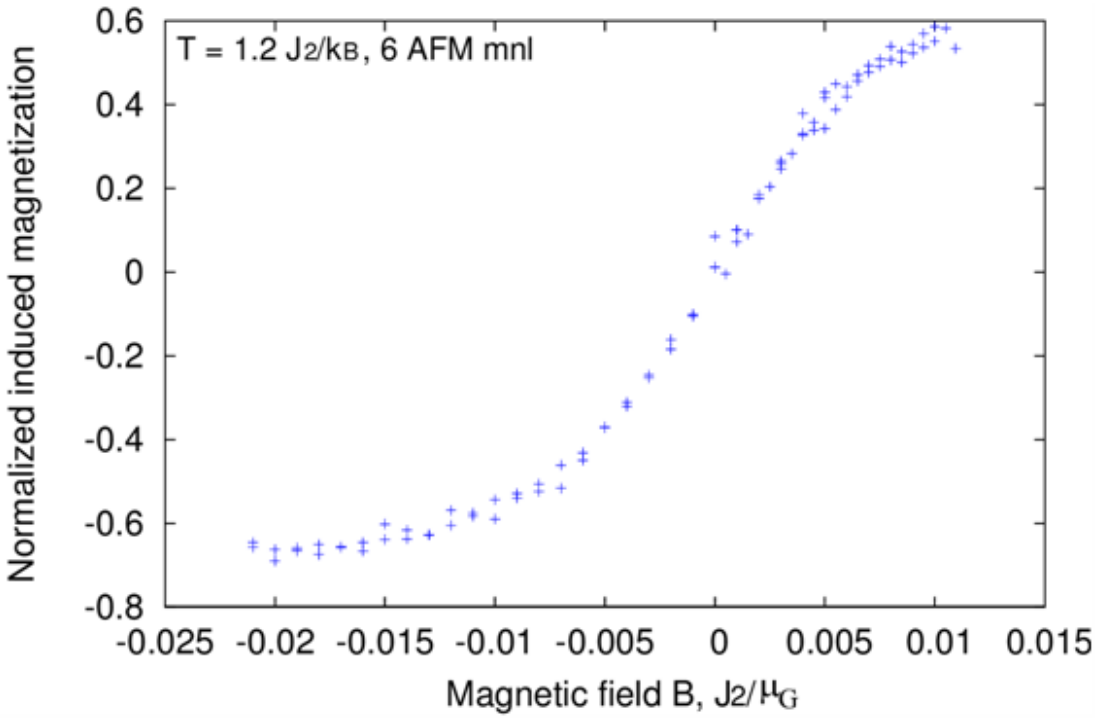


Figure 17: Normalized induced magnetization in the direction of the magnetic field **B** vs. the external magnetic field **B** for the lattice model with 6 AFM monolayers at T = 1.2 J_2/k_B. The derivation of units for the variables in the calculations is included in Appendix H.

Section A3 of Appendix A describes basis vector set up schemes for FCC lattices and two different sets of basis vectors for the lattice setup (B_1 and B_2, which corresponds to equations A.2 and A.3 respectively) were used for comparison. MC calculations with the two different basis sets B_1 and B_2 produced results with no significant difference, which is shown in Figure 18. This implies that the choice of bases (B_1 or B_2) does not influence the result of the calculations.

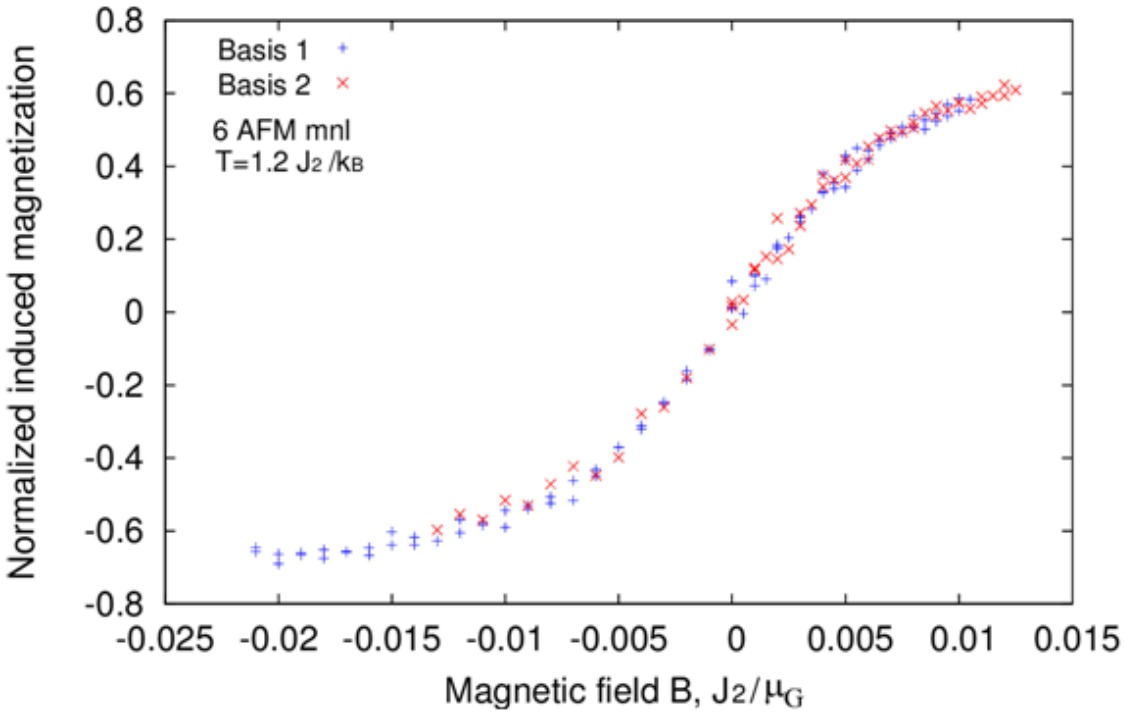


Figure 18: Normalized $\mathbf{M}_{induced}$ vs. magnetic field (**B**, in J_2/μ_G) at T = 1.2 J_2/k_B for the lattice vectors of bases B_1 and B_2 (6 AFM monolayers) described in section A3 of Appendix A

In fact, the scheme for calculating the strength of AFM using the induced magnetization $\mathbf{M}_{induced}$ did not turn out to be a good method since the spins on each monolayer did not respond to **B** in the way as I assumed. Examining the magnetization per monolayer showed that there is a non-uniform curling of the spins depending on the location of spin monolayer in the lattice. The spins in the FM layers respond to **B** differently depending on the location of the FM monolayer. There are two interactions that are in competition: one is the magnetic interaction (with an external field **B**) and the other is the exchange pinning of the spins on the AFM/FM interface. As the spin gets farther from the AFM/FM interface the influence from exchange pinning from the AFM

layer gets weaker and the influence from **B** increases in significance. As a result, spins form a "helix-like" alignment, which is similar to a Bloch domain wall (See Figure 19) [96]. The assumption on uniform sheet FM monolayer is not correct, which makes Figure 14 depicting the spin alignment on each monolayer in the trilayer incorrect.

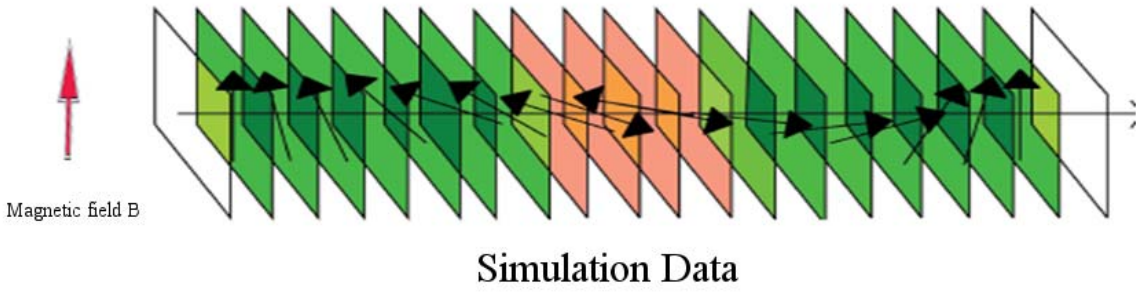


Figure 19: Magnetization per monolayer (mnl) in the trilayer: magnetizations on FM (green) monolayer are curling more toward the direction of the external magnetic field as the monolayer gets farther from the center (AFM layer, orange).

To avoid using $\mathbf{M}_{\text{induced}}$ for calculating the strength of AFM coupling between two FM layers, I redefined the AFM coupling coefficient using spin torque approach. The idea is based on a torsion balance. The "twisting" of the AFM layer is measured in terms of the spin torque. If the angle between the magnetizations of the first FM layer and that of the second FM layer is 180°, the twist (torque) should be reduced if there is a strong AFM coupling between the FM layers. Therefore, I define $\tau = \frac{1}{A}\theta$, where τ is the spin torque (twist), θ is the angle between the FM magnetizations and A is the AFM coupling coefficient. Large A means that AFM coupling is strong. I calculate the spin torque by the formula

$$\tau = \frac{1}{2}(|\mathbf{M_1} \times \mathbf{B}| + |\mathbf{M_2} \times \mathbf{B}|), \tag{3.3}$$

where $\mathbf{M_1}$ and $\mathbf{M_2}$ denote the magnetization of each FM layer respectively. **B** is the external magnetic field and θ is the angle between the sheet magnetizations $\mathbf{m_1}$ and $\mathbf{m_2}$ FM monolayer interfacing the AFM layer. Figure 20 shows $\mathbf{M_1}$, $\mathbf{M_2}$, $\mathbf{m_1}$, and $\mathbf{m_2}$ for the torque calculation.

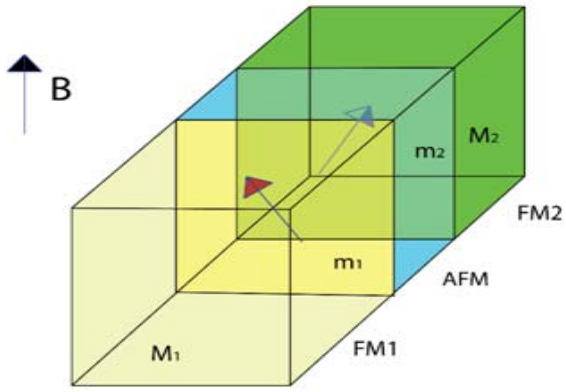


Figure 20: Spin torque ($\boldsymbol{\tau}$) calculation: $\mathbf{M_1}$ and $\mathbf{M_2}$ are the total magnetizations of FM layer 1 and FM layer 2 respectively. $\mathbf{m_1}$ band $\mathbf{m_2}$ are the magnetizations of FM monolayers interfacing the AFM layer.

To calculate the coupling coefficient A, I plot the spin torque over the angle between the sheet magnetizations on the two FM monolayers interfacing the AFM spacer. The torque decreases as the magnetic field decreases. As I increase magnetic field in both directions, the angle between the spins on the two FM monolayers increases. To obtain the AFM coupling coefficient, I need to remove the "V" shape of the plot (Figure 21) by considering the relative angle between $\mathbf{m_1}$ and $\mathbf{m_2}$ and select a linear region. The absolute value of the slope for torque vs. angle plot gives $\frac{1}{A}$.

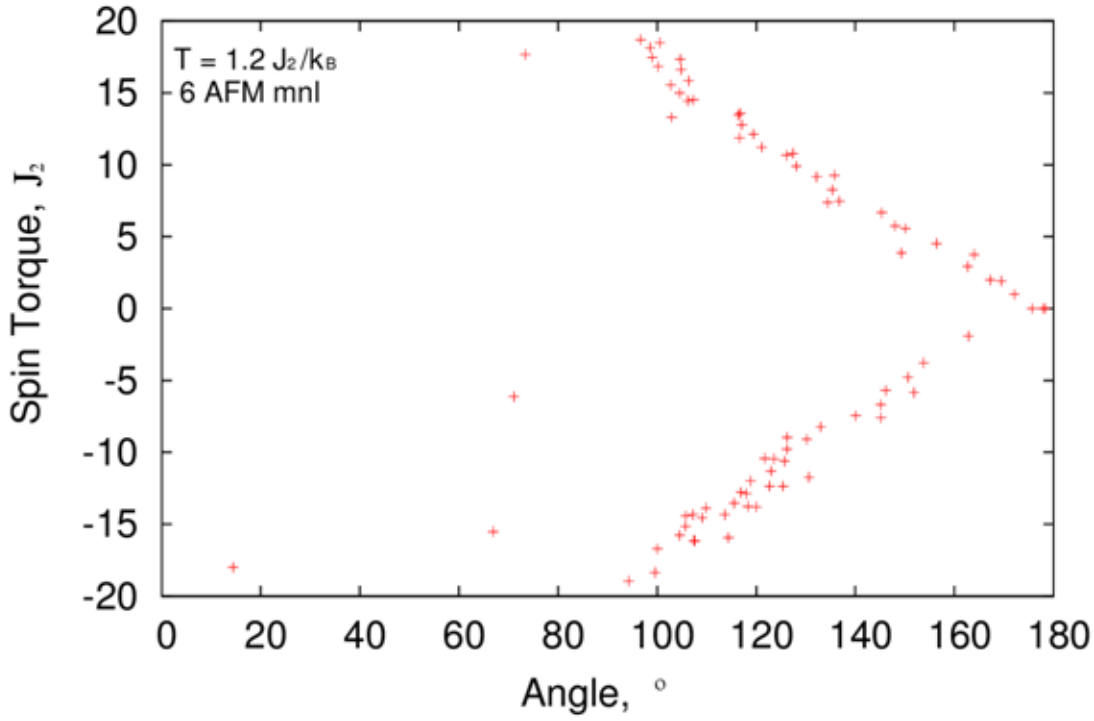


Figure 21: Spin torque vs. the angle between the two FM monolayers: the "V" shape can be removed by reflecting the upper curve about $\theta = 180^{\circ}$.

Figure 22 shows the AFM coupling coefficient A for trilayers with 4, 6, and 8 AFM monolayers calculated with varying temperature. The AFM coupling coefficient decreases as temperature increases. As I reduce the number of AFM spacer monolayers, the AFM coupling coefficient A increases. The AFM coupling is significant above the Néel temperature of the AFM layer.

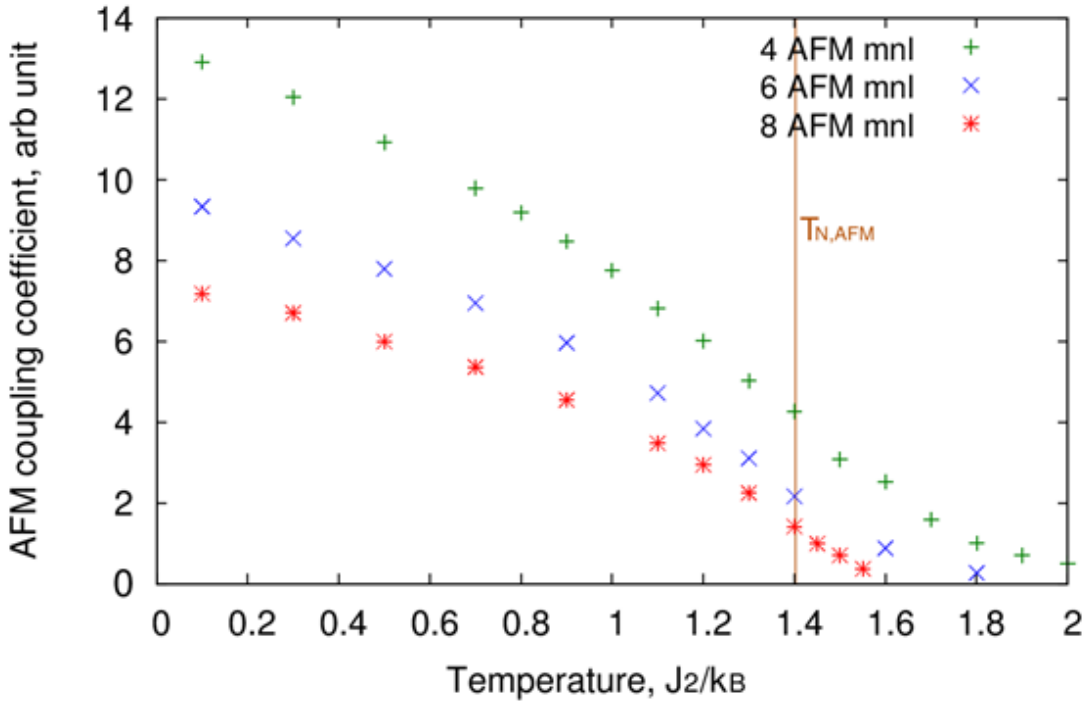


Figure 22: AFM coupling constant A vs. temperature: as the number of AFM monolayers increases, the AFM coupling strength decreases. Note that the magnetic order persists above the Néel temperature $T_N = 1.41\ J_2/k_B$ of the antiferromagnet.

3.2. FM/AFM/FM trilayer with anisotropy

In the previous section, the MC model did not include magnetic anisotropy. In reality, however, proper magnetic anisotropy is always required for any type of applications of magnetic systems [97]. One of the underlying effects giving rise to magnetic anisotropy is a relativistic phenomenon: spin-orbit coupling [97, 98]. It is well-known that single crystals of FM and AFM substances are magnetically anisotropic even when extrinsic origin of anisotropy such as the shape effect is removed [99].

Also it is possible to introduce anisotropy by magnetic annealing, plastic deformation, and irradiation [96]. Anisotropy of FM metals is not well understood, but anisotropy in ionic compounds, either FM or AFM, has been extensively studied [99]. There are mainly three classes in the origin of anisotropy energy [99]. One class is aforementioned anisotropic interactions between spins of two ions which arise from the spin-orbit coupling and various electrostatic interactions between the ions [99, 100]. The second class is the classical magnetic dipolar energy, and the second is energies which depend on spin states only [99, 100]. There is a temperature dependence of anisotropy constants, which is usually associated with the temperature dependence of magnetization [101, 102].

On ultrathin film and multilayer magnetism, understanding the nature of the anisotropy in a given structure and how to control it became an issue [96]. The anisotropy energy controls the orientation of magnetization in the ground state of the ultrathin film [96]. The effective magnetic anisotropy energy is associated with two factors: the spin-

orbit coupling (which is called the "magnetocrystalline anisotropy") and the magnetostatic dipole-dipole interaction (which is also called the "shape anisotropy") [98-100, 102]. Magnetic anisotropy can be affected by factors such as film thickness, strain, structure, and broken symmetry at the surface and interface [99, 102, 103].

3.2.1. Model for Monte Carlo calculation

Types of anisotropy need to be considered before introducing anisotropy to the system. When the structure is composed of ferromagnets and other material, there are competing anisotropy contributions, such as the composition of nonmagnetic spacer layers, lattice distortions at the interface, and perpendicular uniaxial and exchange anisotropies [96]. The uniaxial anisotropy was used for this study since it is observed most commonly [96]. The following Hamiltonians H_1, H_2, and H_3 have uniaxial anisotropy along x-, y-, and z-direction respectively.

$$H_i = \boldsymbol{S_i} \cdot (J_1 \sum_{j \in NN} \boldsymbol{S_j} + J_2 \sum_{k \in NNN} \boldsymbol{S_k} - \boldsymbol{B}) \quad (3.4)$$

$$H_0 = \sum_{i \in lattice} H_i \quad (3.5)$$

$$H_1 = H_0 + \mathrm{K_F} \sum_{l \in FM} {S_{lx}}^2 + \mathrm{K_A} \sum_{l\prime \in AFM} {S_{l\prime x}}^2 \quad (3.6)$$

$$H_2 = H_0 + \mathrm{K_F} \sum_{l \in FM} {S_{ly}}^2 + \mathrm{K_A} \sum_{l\prime \in AFM} {S_{l\prime y}}^2 \quad (3.7)$$

$$H_3 = H_0 + \mathrm{K_F} \sum_{l \in FM} {S_{lz}}^2 + \mathrm{K_A} \sum_{l\prime \in AFM} {S_{l\prime z}}^2 \quad (3.8)$$

Since the magnetic field is set to be in the *x*-direction, Hamiltonian H_1 (eq. 3.6) was chosen. The anisotropy parameter for FM and AFM layers are denoted K_F and K_A respectively. If K_F, $K_A < 0$, it represents an easy-axis anisotropy along the *x*-direction and if K_F, $K_A > 0$, it represents a hard-axis anisotropy along the x-direction (i.e., an easy-plane anisotropy on the yz - plane) [104]. All the details on the trilayer model except for the Hamiltonian were set to be identical to those on the trilayer with no anisotropy.

3.2.2. Data Analysis

In order to study the effect of magnetic anisotropy, the total induced magnetization was calculated and plotted vs. magnetic field **B** for different temperatures. The induced magnetization curve ($\mathbf{M}_{\mathbf{induced}}$) with a varying external magnetic field **B** with hard-axis anisotropy, where $K_F = 0.0063$ and $K_A = 0.0125$, did not show a significant difference from the $\mathbf{M}_{\text{induced}}$ vs. **B** curve for the case without anisotropy. Figure 23 shows the induced magnetization as a function of the magnetic field **B**, where there is no visible hysteresis. The unit of magnetic field B is J_2/μ_G, where μ_G is the gyromagnetic ratio for the magnetic ion of interest. The details on the units used in this project are described in Appendix H.

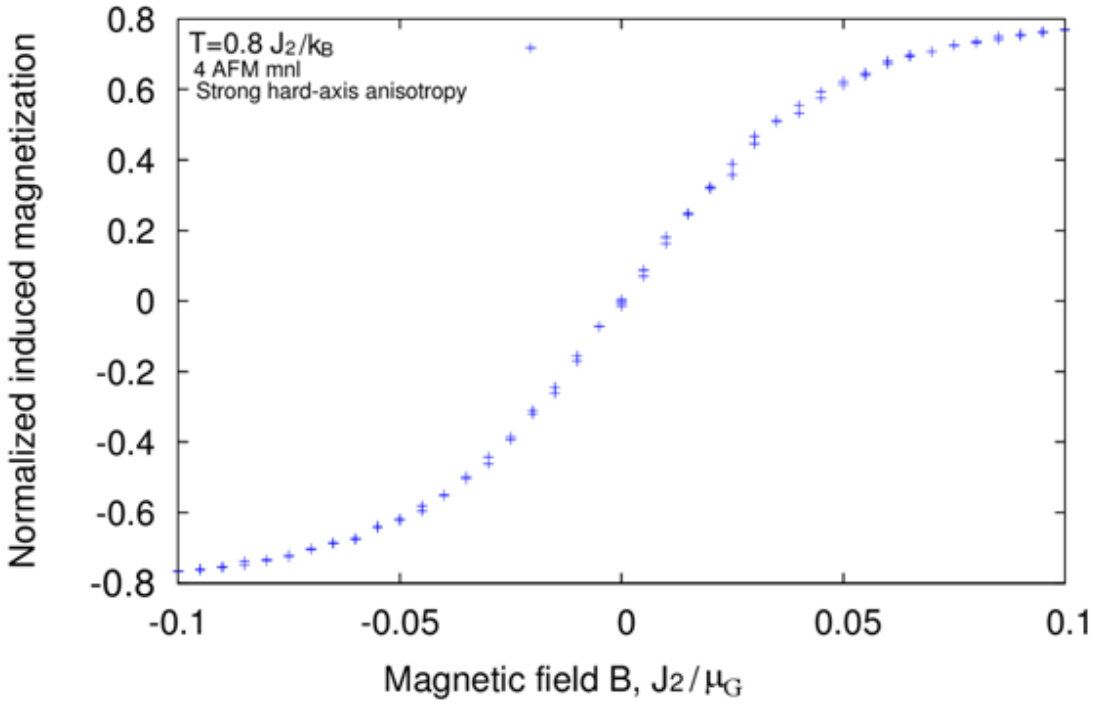


Figure 23: Induced magnetization component parallel to the direction of the magnetic field ($\mathbf{M}_{\mathbf{induced}}$) vs. magnetic field **B** (in the unit of J_2/μ_G) with strong hard axis anisotropy along the magnetic field direction

On the other hand, the magnetization vs. **B** with easy-axis anisotropy showed a significant difference. There was a hysteresis in the magnetization and the magnetic transition seems to be of the first order. There was a step-like transition. However, I should note that there is no experimental result to verify this result.

With $K_F = -0.0063$ and $K_A = -0.0125$, I could observe a fairly symmetric looking magnetization vs. magnetic field curve for each temperature. The external magnetic field value for the magnetic transition decreased as the temperature increased.

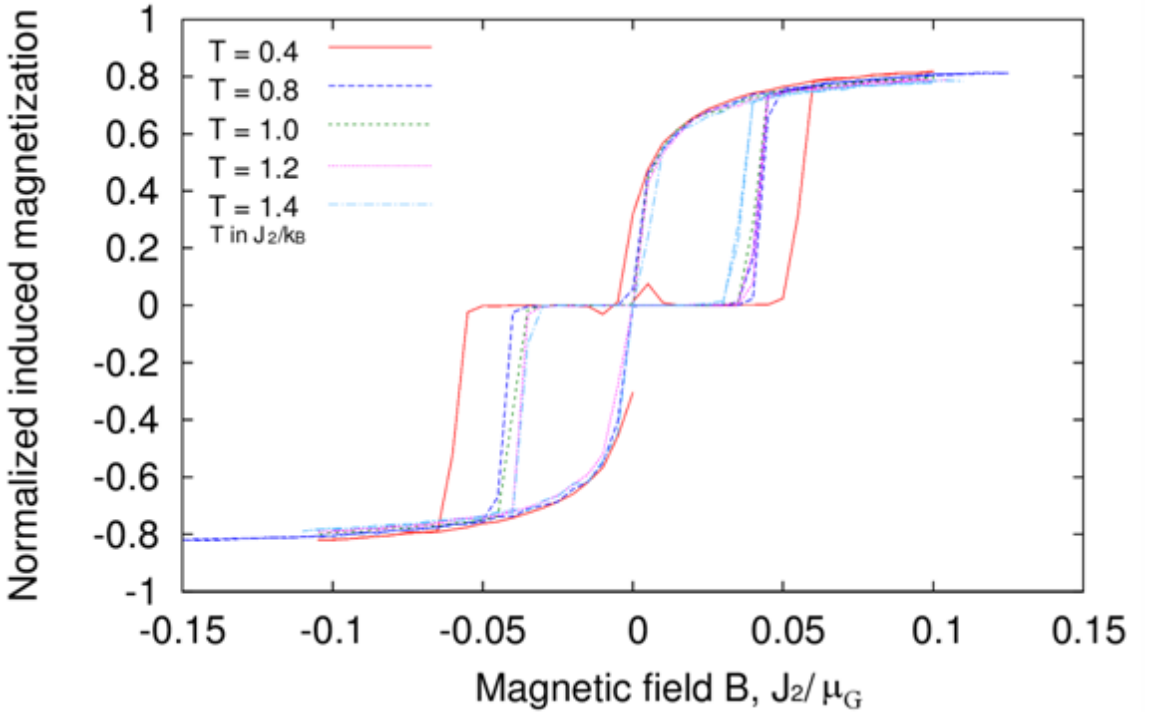


Figure 24: Normalized induced magnetization component parallel to the direction of the magnetic field vs. magnetic field with temperature K_A = - 0.0125, K_F = - 0.0063

As the temperature increases, the gap in the hysteresis-like loop decreased. This is reasonable since thermal fluctuation can cause the onset of magnetic transitions. As the temperature increases there are more thermal fluctuations. Therefore, the threshold magnetic field for the magnetic transition decreases. Once the temperature reaches T = 1.2 J_2/k_B, the gap does not decrease as much with increasing temperature.

3.2.3. Results and discussion

With hard-axis anisotropy, the induced magnetization vs. magnetic field curve did not significantly differ from the curve for the case without anisotropy. However, the induced magnetization curve showed a hysteresis with easy-axis anisotropy. This result is consistent with Stoner-Wolfarth theory, where there is hysteresis when there is easy-axis anisotropy only. [105]. However, it should be noted that the there is currently no experimental result for FM/AFM/FM trilayer systems to compare the simulation results to [106, 107].

3.3. FM/AFM/FM trilayer with diluted AFM spacer layer

3.3.1. Preliminary study: diluting bulk AFM

Before running calculations on a trilayer lattice, how the Néel temperature of a bulk AFM model lattice is influenced by the concentration of magnetic sites was investigated. The purpose of this calculation is to measure "the persistence" of magnetic order in an AFM lattice with different concentrations of magnetic ions.

In principle, in order to determine the Néel temperature, a fit of the function $(\mathrm{T} - \mathrm{T_{N\acute{e}el}})^{\gamma}$ can be applied to sublattice magnetization vs. T data: Heisenberg systems usually obey this rule, with the γ value close to $\frac{1}{3}$. However, in diluted systems the sublattice (or reduced) magnetization $\mathbf{M_r}(\mathrm{T})$ is no longer "well-behaved". T. M Giebultowicz et al., who studied $Co_pMg_{1-p}O$ using neutron diffraction, ($Co_pMg_{1-p}O$ is a diluted FCC antiferromagnet and it has the original structure of CoO) found by fitting power-law dependence functions to the sublattice magnetization data from systems with $p < 1.0$ that such systems exhibit "smeared" AFM phase transition [108, 109]. Not surprisingly, attempts of fitting $\mathbf{M_r}(\mathrm{T})$ data to simple power-dependence functions failed. As dilution increases, the magnetic order disappears at lower temperatures [109]. In other words, the Néel temperature decreases as dilution increases.

3.3.1.1. Model for Monte Carlo calculation

The size of the lattice is 50×50×50, which is composed of 125000 spins. I set the first and last monolayers empty to have zero spins to simulate the free boundary. Each spin monolayer had the periodic boundary on the xy-direction. The spin lattice model is identical to the AFM model described previously except that there are zero spins on random sites in the AFM monolayers. To produce a diluted lattice I initialized the array with zero spins then I randomly chose spin sites without repeat and put nonzero spins according to a given nonzero spin concentration. This method does not affect the method of calculating the energy but the MC steps need to skip zero spins after checking if the magnitude of the spin is nonzero. There are other methods such as "marking" the zero spins with another array element to reduce the number of "if" statements to increase computation speed, but I did not use such method since it complicates the program significantly.

The Hamiltonian I used was

$$H = (J_1 \sum_{i,j \in NNN} \boldsymbol{S_i} \cdot \boldsymbol{S_j} + J_2 \sum_{i,k \in NN} \boldsymbol{S_i} \cdot \boldsymbol{S_k})_{AFM}, \quad (3.9)$$

which gives the transition temperature of AFM $T_{N,AFM} = 1.4\ J_2/k_B$.

3.3.1.2. Data analysis

The reduced magnetization plot (Figure 25) shows that magnetic order exists at concentration 45%, though it disappears at about 0.4 J_2/k_B. This result is consistent with

the experimental results from T.M. Giebultowicz et al. [108, 109]. If I like to know if the neutron diffraction peaks are present, I can choose to calculate the structure factor with the spin output files from the MC programs as I did in Chapter 4.

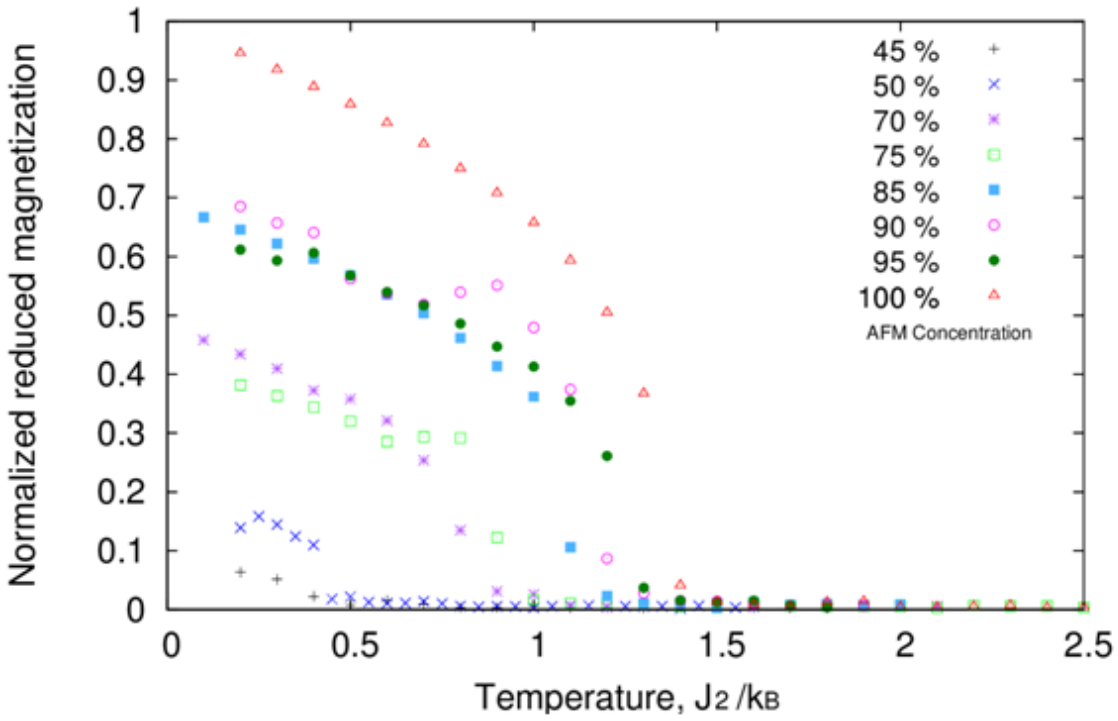


Figure 25: Normalized reduced magnetization vs. temperature: bulk AFM model for different concentrations of magnetic ions. The nondiluted bulk AFM results in the Néel temperature $T_{N,AFM} = 1.4\ J_2/k_B$.

3.3.2 Dilution of AFM spacer layer in FM/AFM/FM trilayer

The main goal of this part is to investigate the behavior of FM/AFM/FM trilayer lattices with different magnetic ion concentrations in the AFM spacer layer. The main question on this system is how much the behavior of FM/AFM/FM trilayer systems with diluted mid-layer resembles the behavior of FM/AFM/FM lattices with non-diluted mid-layer. No other complicating factor (such as anisotropy) was added to the system.

3.3.2.1. Model for Monte Carlo calculations

The Hamiltonian I used was identical to the non-diluted trilayer case except that there are some zero spins on the lattice. The Hamiltonian for a single spin H_i is written as

$$H_i = \boldsymbol{S_i} \cdot \left(J_{1mat} \sum_{j \in NN} \boldsymbol{S_j} + J_{2mat} \sum_{k \in NNN} \boldsymbol{S_k} - \mathbf{B}\right). \quad (3.10)$$

This gives the the Hamiltonian for the whole system, which is written as

$$H = \sum_{i \in lattice} H_i, \quad (3.11)$$

, where $J_{1\text{mat}}$, and $J_{2\text{mat}}$ denote the NN and NNN exchange constants for a material respectively.

Monte Carlo calculation parameters such as exchange constants are set to be identical to the previous non-diluted cases. The concentration of the AFM spacer layer model was varied from 40% to 100%. After initializing the spins in the array with zero, spin sites were selected randomly and set to be a nonzero value. Once the MC program reads in the spin values, it skips all the zero spins.

3.3.2.2. Data analysis

To understand the effect of dilution of the AFM spacer layer on the AFM coupling, the induced magnetization of the whole lattice was plotted as a function of external magnetic field. In this case, calculating the spin torque becomes inadequate due to the fact that hysteresis is expected in the result. The magnetic interactions are conveyed by the AFM layer.

3.3.2.3. Results and discussion

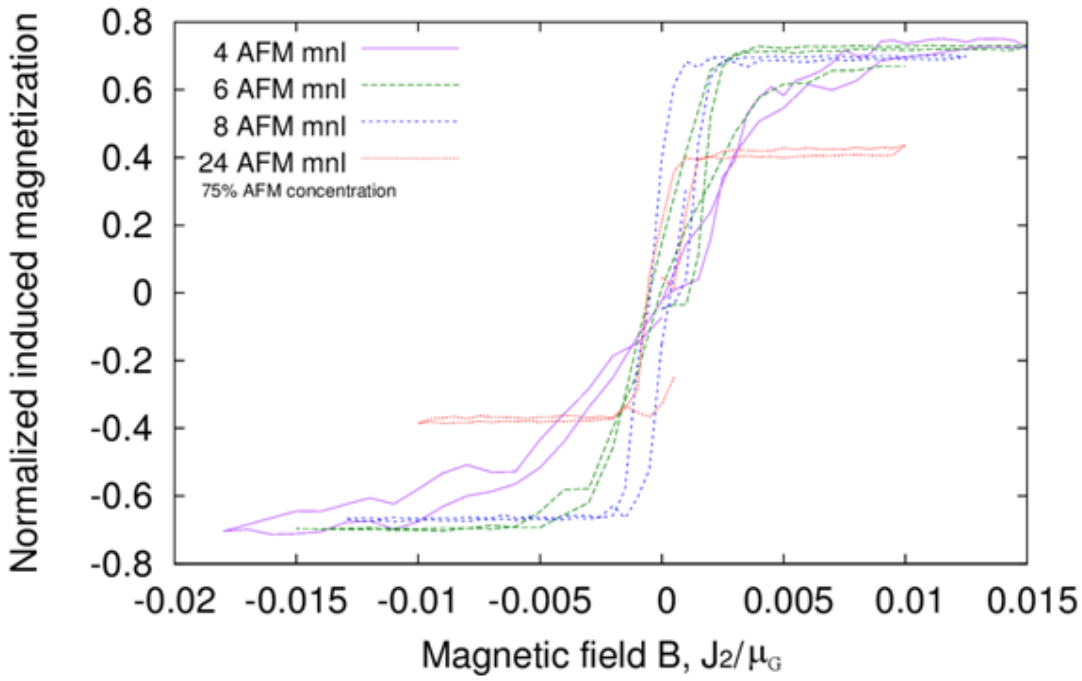


Figure 26: Induced magnetization vs. magnetic field for AFM magnetic concentration 75%, T = 1.2 J_2/k_B 4, 6, 8, 24 AFM monolayers (mnl): as I increase the number of AFM monolayers, the saturation magnetization decreases and the slope of the curve increases indicating the strength of AFM coupling decreases.

Figure 26 above shows the induced magnetization vs. an external magnetic field with 75 % NiO concentration. The slope of $\mathbf{M}_{induced}$ vs. **B** gets steeper as the number of the AFM spacer monolayers increases, which indicates that AFM coupling between the FM layers increases when the thickeness of AFM mid-layer increases. For all the given cases of AFM layer thicknesses, the induced magnetization showed hysteresis.

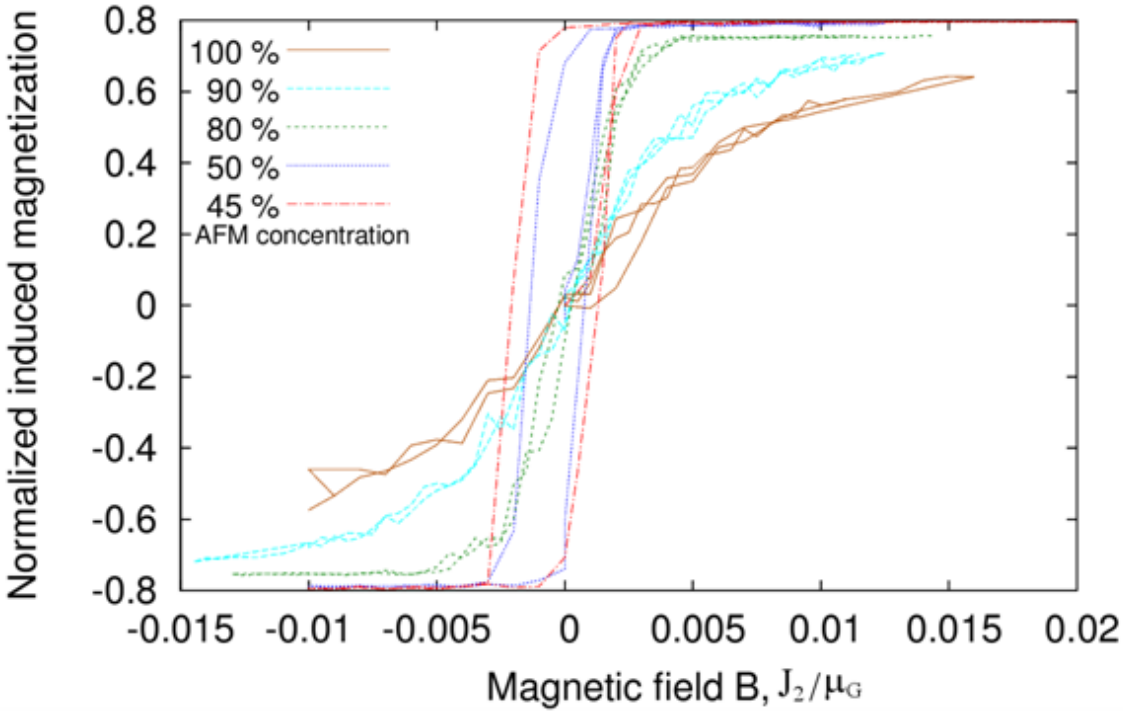


Figure 27: Normalized induced magnetization vs. **B** for 6 AFM monolayers (mnl) at T = 1.2 J_2/k_B for magnetic ion concentrations 45%, 50%, 80%, 90% and 100%: the gap of the hysteresis loops increases as I decrease the AFM concentration. Also it should be noted that the magnetic hysteresis loop is not centered about zero magnetic field.

Figure 27 shows the $\mathbf{M}_{induced}$ vs. **B** curve with different concentrations of magnetic ions in the AFM spacer layer. The general trend is that the slope of the $\mathbf{M}_{induced}$ vs. **B** increases as the dilution of magnetic ions increases. This is reasonable since there

will be less magnetic correlations with higher dilution rate of the AFM mid-layer. Another feature to note is that the gap of the hysteresis loop in each curve increases as the dilution of the AFM mid-layer increases. This might be due to the fact that diluted lattices are more likely to have more magnetic domains [74].

Note that the hysteresis loops are not centered about zero magnetic field and that as the gap of the hysteresis loops increases the center of the hysteresis loops move away from the zero magnetic field. There were studies on the effect of dilution (of AFM layer) in systems with exchange bias and they showed that there is an enhancement of exchange bias in diluted systems [110-114].

Lastly the saturation magnetization value is increases for the lattices when dilution of magnetic ions in the AFM spacer layer increases. As I dilute the AFM spacer layer more, I expect that there will be more "isolated" spins, which will respond to an external field paramagnetically. Therefore, it is not unreasonable to have the value of saturation magnetization increasing with dilution.

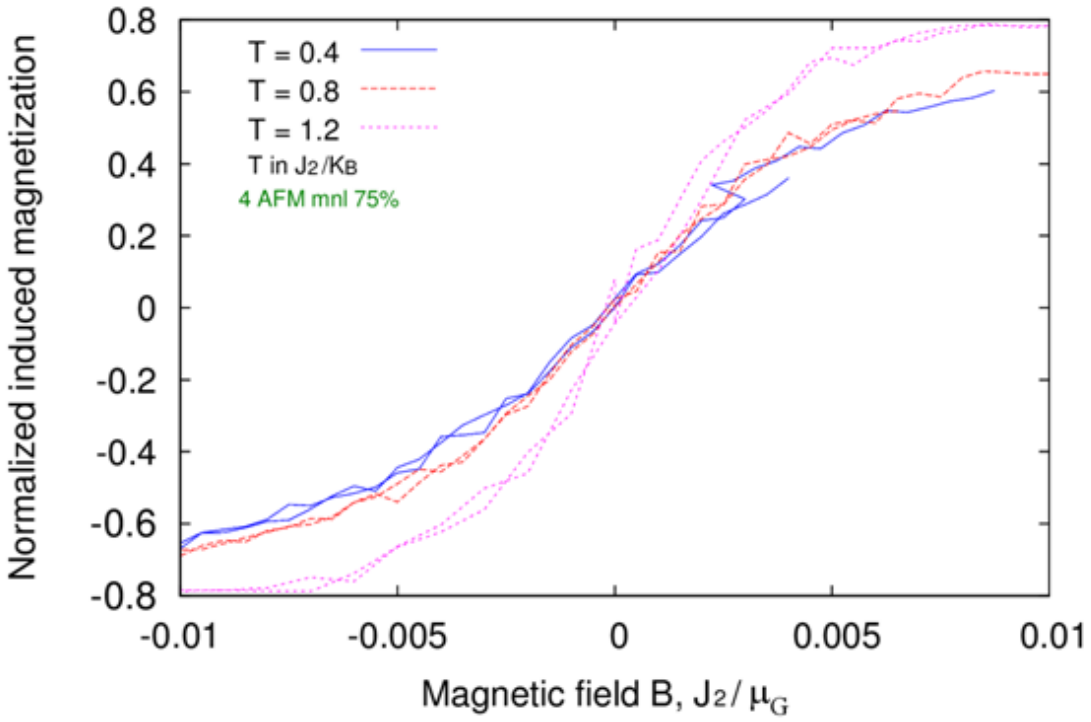


Figure 28: Normalized induced magnetization vs. external magnetic field **B** for 4 AFM monolayers (mnl) with varying temperature and 75% magnetic concentration in the AFM layer, the slope of the curve increases as the temperature increases.

Figure 28 shows the dependence of the slope of the $\mathbf{M}_{\mathbf{induced}}$ vs. B curve on temperature. As in the FM/AFM/FM systems without diluted layers, the slope of the $\mathbf{M}_{\mathbf{induced}}$ vs. B curve increases as the temperature increases. Therefore, I can conclude that FM/AFM/FM systems with diluted AFM mid-layer exhibit behaviors that are similar to the behaviors of the system without dilution but that the weakening of magnetic correlation in the diluted AFM mid-layer causes hysteresis and weaker AFM coupling between the FM layers.

Chapter 4. Incommensurate Helical spin order in MnSe/ZnTe superlattice

4.1. Motivation for studying magnetic order in MnSe/ZnTe superlattices

T. M. Giebultowicz et al. studied MnSe/ZnTe superlattices using neutron scattering and observed an incommensurate helical spin order whose pitch varied with temperature [2]. It was surprising since the Type I order was observed in MnTe/ZnSe, which is a system similar to MnSe/ZnTe superlattice [113]. The reason for the different behavior of the spin systems in those layered structures was successfully explained by considering strain effects. Namely, due to a small difference in the lattice parameters, the lattice experienced a tensile strain and got distorted [2].

Since the strength of the spin-spin exchange interaction J is a sensitive function of the spin-spin distance, the small distortion of the FCC lattices resulted in meaningful changes in the pattern of interaction of spins with their 12 nearest neighbors [116]. In the case of the compressive layer strain, the resulting changes in J_{NN} further stabilized the Type III order – but, as was shown based on mean-field-theory arguments, the anisotropy in the J_{NN} pattern resulting from tensile strain led to an energy minimum for an AFM *helical arrangement* of the spins. This model also indicated that the helix period Λ should depend monotonically on the lattice distortion magnitude – and since there were some differences between the strains in the samples, it explained the differences in Λ values obtained from individual samples [2].

4.2. Previous simulation work and objective

An attempt to get more theoretical insight into the problem was undertaken by M. Collins and W. Saslow in 1995 [117]. They used the numerical technique of MC simulation [117]. For simplicity they used the so-called "XY model" in which vector spins are confined to a plane, not 3D Heisenberg interactions, [117]. A sensitive issue in MC simulations of helical magnetism is the choice of boundary conditions [119]. Periodic boundary conditions are used most commonly in MC modeling of magnetic systems, but they have obvious limitations: they are "safe" in the case of Ising spins, and for XY and fully 3D models they are appropriate only for systems with ***collinear*** spin order [117]. If such boundary conditions were used for modeling *incommensurate helical* spin structures, they would impose periodicity on them, so that the Λ values obtained from such models would not be reliable [35]. One possible solution is to do simulation on very large "spin supercells" created in the computer memory, with "free boundaries". Physically, such a situation is certainly even closer to that occurring in a real system than when using periodic boundary conditions. Yet, it requires a very large "spin array" – certainly, a reasonable criterion for the size would be more than an order of magnitude longer than the expected value of Λ. At the time when M. Collins and W. Saslow performed their work, it was a requirement far exceeding the potential of the computational resources, which imposed them on creating their own boundary condition [117].

The work from M. Collins and W. Saslow was definitely a partial success - it confirmed that the MC modeling, which is a technique enabling one to simulate the behavior of systems at finite temperatures, showed that the spin order obtained from the mean-field model is stable at finite temperatures (in the case of complicated spin structures, the mean-field approach can offer only reasonable predictions concerning the ground state configuration, but is no longer a good tool for predicting the behavior of the system in the $T>0$ region [117]. However, as the authors admitted, their work still had some "weak spots" from the viewpoint of MC methodology: the use of the XY model instead of the more realistic 3D Heisenberg interactions, and boundary conditions which did not offer a 100% guarantee of *not introducing* artifacts [117]. In addition, not all relevant questions concerning the *physical mechanisms* underlying the effects seen in the experiments were not addressed; the Λ (T) dependence in the spin structures modeled had not been systematically investigated, so that it could not be checked whether the shape of the $\Lambda(T)$ characteristics, which is certainly an important aspect of the spin ordering forming in the real system, was correctly reproduced by the numerical model [117]. Also, the influence of the layer thickness on Λ was not investigated [117].

Considering the above issues discussed above, it was determined that undertaking renewed efforts of numerical modeling of the phenomena seen in the MnSe/ZnTe would be a useful project. Advances in computer technology enabled me to complete calculations with a full 3D Hamiltonian for large spin arrays whose linear sizes correspond to 30-50Λ within a realistic time. Therefore, in the present project I decided to use vector spins with unrestricted freedom of turning, and "the free boundary

conditions". Also the $\Lambda(T)$ dependence in the systems modeled was systematically investigated, as well as the influence of the layer thickness on the value of Λ - including simulations of hypothetical "bulk" MnSe with its FCC lattice distorted in an analogous fashion as in the MnSe layers experiencing a tensile strain.

One of the factors that can induce anisotropy into a magnetic system is mechanical strain [113, 117]. To study the effect of strain on AFM magnetic order, T.M. Giebultowicz et al. studied MnSe/ZnTe superlattices with neutron scattering [2]. ZnTe is non-magnetic and MnSe is AFM [2]. Single crystals of ZnTe and MnSe have zincblende structure and Mn^{2+} ions are arranged in an FCC lattice [2, 40, 113]. In bulk, MnSe does not form the zincblende structure but In MnSe/ZnSe superlattices MnSe resembles the structure of the substrate [2, 40, 113] MnSe layer experiences a compressive strain and Type III magnetic order was observed [42]. In the case of MnSe/ZnTe superlattices T.M. Giebultowicz et al. found an incommensurate helical spin order [2]. In addition, they found that the pitch of the helical order increased with temperature [2].

The MnSe/MnTe superlattice is characterized by the lattice constant a(MnSe) = a(ZnTe) = a in the direction of the layer plane and c(MnSe) ≠ c(ZnTe) in the direction perpendicular to the layer plane [2]. Due to this difference in the lattice parameters Mn^{2+} ions experience a tensile strain [2].

As mentioned in Chapter 1, M. Collins and W. Saslow performed Monte Carlo calculations using an XY-spin model and they concluded that the pitch of helical spin order increased with temperature and that the AFM phase depended on two parameters α and η, where $\alpha = \frac{J_{1in} - J_{1out}}{J_{1in}}$ and $\eta = \frac{J_2}{J_1}$ [117]. α is a measure of how much the lattice is

distorted and η is a measure of the relative strengths of NN and NNN interactions [117]. To replicate this result without a boundary condition that "assists" the formation of a spin helix, Monte Carlo calculations were performed with Heisenberg spins with different boundary conditions.

4.3. Mean field theory calculation

When the lattice is under a tensile strain, one can imagine the lattice is compressed, twisted, or stretched depending on the type of the strain. In the case of a tensile strain, I consider a lattice that is stretched as shown in Figure 29. It has been found that the coupling is a function of distance between the magnetic ions [116, 120]. Therefore, the lattice distortion will affect the strengths of spin-spin interactions [120].

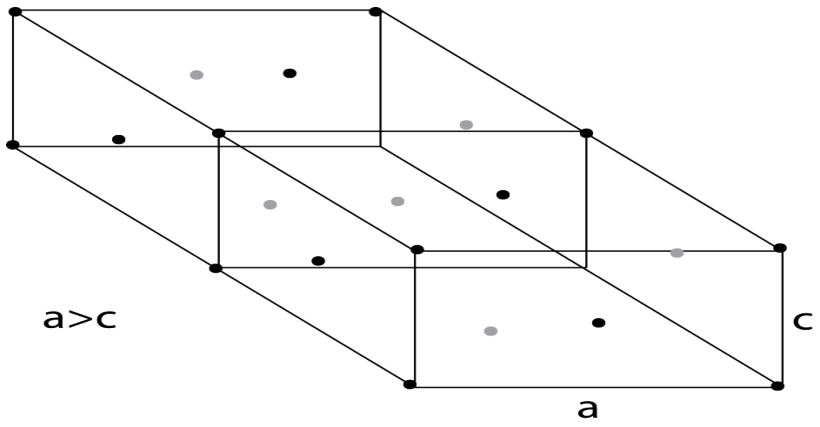


Figure 29: FCC lattice spin model under tensile strain: the lattice parameters *a* and *c* and not equal to each other but $a > c$.

Let the Hamiltonian be $H = -2J_{kl}\boldsymbol{S_k} \cdot \boldsymbol{S_l}$ for each spin neighbor pair and let the growth axis and the lattice parameter in the growth direction be [001] and *c* respectively [120]. Also I denote the lattice parameter in the layer plane by *a*, and the number of monolayers in a single MnSe layer by N [120]. I only include the NN interaction and the NNN interaction [120]. Both NN interaction and NNN interaction are AFM in MnSe, so the exchange constants $J_1 = J_{NN} < 0$ and $J_2 = J_{NNN} < 0$ [120]. The exchange constants J_1 and J_2 are the same everywhere in an undistorted lattice since the lattice parameters *a* and *c* are equal to each other [120].

Once a tensile strain is introduced, the lattice parameter ratio $\frac{c}{a} < 1$ the distance between the "in-plane" neighbors (denoted by $R_{1\parallel}$) is slightly larger than that between the "out-of-plane" neighbors ($R_{1\perp}$). As a result, the AFM coupling between the "in-plane" NNs is weaker than the coupling between the "out-of-plane" NNs (i.e., $|J_{1\perp}| > |J_{1\parallel}|$) [120]. Now let $\Delta J_1 = J_{1\perp} - J_{1\parallel}$. The same rule can be applied to $J_{2\parallel}$ and $J_{2\perp}$ as well I set $|J_{2\perp}| > |J_{2\parallel}|$ and $\Delta J_2 = J_{2\perp} - J_{2\parallel}$.

Figure 1 on page 15 shows the Type I and Type III magnetic order; both structures show (100)-type AFM planes. Each spin is antiferromagnetically coupled with its four nearest neighbors on the same plane [120]. The exchange coupling is very sensitive to the distance between interacting magnetic ions [114]. There are also four next nearest neighbors in the plane. The resulting in-plane coupling energy per spin is $E_0 = -8J_1 + 8J_2$. Since $|J_1| > |J_2|$, E_0 is negative.

An important feature of the FCC lattice is that the energy of interaction with the eight nearest neighbors on the adjacent planes always sums up to zero. So the coupling

energy contribution from the neighbors in adjacent planes is always zero [120].The only nonzero term in the total magnetic energy is from the interactions with the two NNNs on the second-nearest spin planes , which can be written $E_{2ndplane} = \pm 4J_{2\parallel}$ [120]. The sign of $E_{2ndplane}$ depends on the orientation of the spin on the second adjacent plane. Due to the effective "decoupling" of the adjacent planes, the spins on adjacent planes are not necessarily collinear [120].

If I stretch the lattice to model the MnSe lattice under tensile strain, $c / a < 1$ and $\Delta J_1 \neq 0$ [18]. Then the spin interaction energy between adjacent planes is no longer zero and the ground state configuration becomes an incommensurate helical order [120].

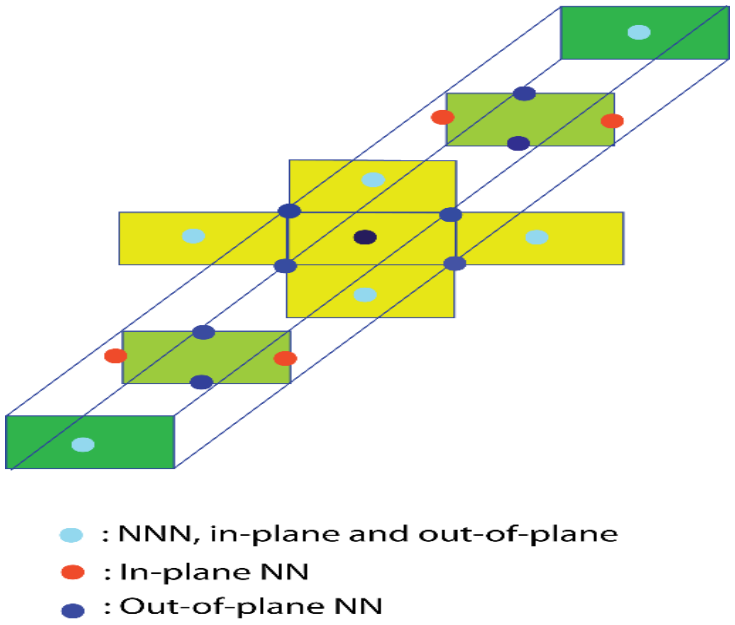


Figure 30: Stretched MnSe lattice model: spin neighbors: the out-of plane neighbors are closer than the in-plane neighbors [120]

The helical spin configuration can be derived from the Type I order by successive rotation of the spin planes at φ, 2φ, 3φ, … The total magnetic energy per spin become

$$U = E_0 + 8\Delta J_1 cos\varphi - 4J_{2in} cos2\varphi \quad (4.1)$$

To calculate the energy minimum assuming E_0 is constant, differentiate eq. 4.1 with respect to φ.

$$\frac{dU}{d\varphi} = -8\Delta J_1 sin\varphi + 8J_{2\parallel} sin2\varphi = -8\Delta J_1 sin\varphi + 16J_{2\parallel} cos\varphi sin\varphi = 0 \quad (4.2)$$

Solving eq.4.2 gives $\varphi = \arccos\left(\frac{\Delta J_1}{2J_{2\parallel}}\right)$ and $U_{min} = E_0 + 4J_1\left[1 + \left(\frac{\Delta J_1}{J_2}\right)^2\right]$ for $0 \leq \varphi \leq 90°$ [120].

At this point, I assume $J_{2\parallel} = J_{2\perp} = J_2$ since the minimum energy configuration depends only on $J_{2\parallel}$. Then the angle φ is a function of $\frac{\Delta J_1}{2J_{2\parallel}} = \frac{\Delta J_1}{2J_2}$ [120]. If I plot both U-E_0 and φ as functions of $\frac{\Delta J_1}{2J_2}$, I find that Type I and Type III structures occur when the lattice is undistorted and the helical configuration leads to a lower magnetic energy than Type I and Type III configuration for $0 \leq \frac{\Delta J_1}{2J_2} \leq 4$ [120]. Now let the modulation period or the pitch of spin helix $\Lambda = \frac{\pi}{\varphi} a = \frac{1}{1-q}$. If I decrease $\frac{c}{a}$, $\frac{\Delta J_1}{2J_2}$ increases and the helix angle φ decreases. When the $\frac{\Delta J_1}{2J_2}$ is 4, φ becomes zero and $\Lambda = \infty$, which corresponds to a collinear Type I order [34].

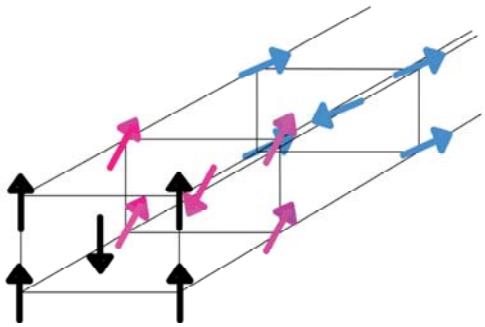


Figure 31: Incommensurate helical spin order, the spins on each consecutive spin plane gets rotated by an angle.

In fact, the spins on the boundaries do not have a full set of neighboring spins. If the thickness of the sample is small, then the spins on the surface will affect the magnetic energy of the system significantly [120]. Table 1 below shows the set of neighbors for spins on the surface [120].

Table 1. Neighbors for inner and out-of-plane spins

Spin layer index	Nearest and next nearest neighbors to a given spin located in (100)-type AFM planes which are:		
	In-plane	Adjacent planes	Next nearest
1 or N	$2NN_{\perp} + 2NNN_{\parallel} + 1NNN_{\perp}$ (missing $2NN_{\perp} + 1NNN_{\perp}$)	$4NN_{\parallel} + 2NN_{\perp}$ (missing $2NN_{\perp}$)	Full set
2 or N-1	$4NN_{\perp} + 2NNN_{\parallel} + 1NNN_{\perp}$ (missing $1NNN_{\perp}$)	Full set	Full set
3 to N-2	Full set	Full set	Full set

The full set of neighbors consists of $4NN_{\perp} + 2NNN_{\parallel} + 2NNN_{\perp}$ in plane and $4NN_{\parallel} + 4NN_{\perp}$ in the two adjacent planes and $2NNN_{\parallel}$ out of plane. Then the interaction energy is modified to

$$U_{1\,or\,N} = E_0 + (4J_{1\perp} - 8J_{1\parallel})cos\varphi - 4J_2 cos2\varphi$$

$=E_0 + (8\Delta J_1 - 4J_{1\perp})cos\varphi - 4J_2 cos2\varphi$ for the layers 1 and N.

For layers 2 and N-1, the interaction energy is

$$U_{N-1\ or\ 2} = E_0 + (8J_{1\perp} - 8J_{1\parallel})cos\varphi - 4J_2 cos2\varphi = E_0 + 8\Delta J_1 - 4J_2 cos2\varphi.$$

Then take the "weighted" average of the energy values.

$$\overline{U(\varphi)} = \frac{2}{N}U(\varphi)|_{n=1,N} + \frac{N-2}{N}U(\varphi)|_{n=2,\dots,N-1} = (8\Delta J_1 - \frac{8}{N}J_{1\perp})cos\varphi - 4J_2 cos2\varphi)$$

Again, differentiate this average energy by φ to minimize it and obtain

$$\varphi|_{\overline{U_{min}}} = \arccos\left(\frac{\Delta J_1 - \frac{1}{N}}{2J_2}\right)$$

$$\varphi_{helix} = arccos\left(\frac{\Delta J_1 - \frac{1}{N}J_{1\perp}}{2J_2}\right)$$

Since the ΔJ_1 is small compared to J_1, one can approximate $J_{1\perp} \approx J_1$ and obtain

$\varphi_{helix} = arccos\left(\frac{\Delta J_1 - \frac{1}{N}J_{1\perp}}{2J_2}\right)$. As $N \to \infty$, $\varphi_{helix} \to arccos\left(\frac{\Delta J_1}{2J_2}\right)$.

It is known that MnSe has the exchange constant J_1 that is 5 or 10 times larger than J_2 [49]. Then one can conclude that the increase in the layer thickness will affect the pitch of the helix pitch Λ [120].

Then I need to think of a way to explain the relaxation of the spin helix with temperature. One possible explanation can be that the spins on the boundary/surface have incomplete set of neighbors and the exchange coupling of those spins gets weaker as temperature increases due to thermal fluctuations. To test this idea, it is necessary to have the layer thickness as the control variable.

4.4. Monte Carlo calculations

4.4.1. Model for Monte Carlo calculations

In fact, T.M. Giebultowicz and J.K. Furdyna studied Type III AFM lattice with MC methods, where the alternating ferromagnetic sheets are arranged along the layer growth direction [49].With the exchange parameters $J_{NN} = 1.0$ and $J_{NNN} = 0.1$, they found the Néel temperature to be $T_N = 0.47J_{NN}/k_B$ and the transition to be of the first order [49]. For the MC calculations I used Heisenberg spins with NN and NNN spin-spin interaction only. I assume that the tensile strain "stretches" the lattice so that the lattice parameter of the out-of-plane direction becomes larger than that of in-plane direction. This causes the exchange parameters scale accordingly, i.e., $J_{1in} < J_{1out}$ and $J_{2in} < J_{2out}$. For simplicity I assumed $J_{1in} \neq J_{1out}$ and $J_{2in} = J_{2out} = J_2$. J_{1in} and J_{1out} and J_2 are in-plane and out-of-plane NN exchange constants and the NNN exchange constant respectively.

The Hamiltonian for a single spin $\boldsymbol{S_i}$ in the lattice is

$$H_i = \boldsymbol{S_i} \cdot [J_{1in} \sum_{j \in NN} \boldsymbol{S_j} + J_{1out} \sum_{j' \in NN} \cdot \boldsymbol{S_{j'}} + J_2 \sum_{,k' \in NNN} \boldsymbol{S_{k'}}] \quad (4.3)$$

To simulate a system without a strain, I used $J_{1in} = J_{1out} = J_1 = 1.0$ and $J_{2in} = J_{2out} = 0.1$ as the exchange constants for in-plane and out-of-plane NN interaction and in-plane and out-of-plane NNN interaction respectively. For a lattice with strain, exchange constants that are used are as follows. $J_{2in} = J_{2out} = 0.1$ was used for all calculations. Each case includes both increasing and decreasing temperature directions.

Table 2. In-plane and out-of-plane nearest neighbor exchange constants

Trial	$J_{1\text{in}}$	$J_{1\text{out}}$
1	0.925	1.075
2	0.9125	1.0875
3	0.95	1.05
4	0.94	1.06
5	0.9625	1.0375
6	0.96	1.04
7	0.9	1.1
8	0.965	1.035
9	0.913397	1.086602
10	0.923395	1.076604

I set $J_{1\text{in}} = 1 - \alpha$ and $J_{1\text{out}} = 1 + \alpha$, where $0 < \alpha < 1$, which sets $\alpha = \frac{\Delta J_1}{2}$.

For the lattice with 1-dimensional spin array, I used lattice with three different sizes and two boundary conditions; the free boundary and the periodic boundary condition.

4.4.2. Data analysis: simulated neutron diffraction

Experimentally neutron diffraction is used to measure the magnetic order of the system [121, 122]. The scattering vector q is used to describe neutron scattering process [122, 121]. The neutron diffraction intensity is associated with the structure factor. [4] The structure factor is defined as $f(\mathrm{q}) = \int f(r) e^{-i\mathbf{q}\cdot\boldsymbol{r}} d\mathrm{q}$, where f(r) is the neutron scattering intensity. [121,122]. The scattering cross section is proportional to $|f(q)|^2$. To calculate magnetic scattering intensity, the impact parameter is replaced by the spin vector, which will give an extra "periodicity [122]. $f(\mathrm{q}) = \sum b\, e^{-i\mathbf{q}\cdot\boldsymbol{r}}$ for general scattering and $f(\mathrm{q}) = \vec{S} e^{-i\mathbf{q}\cdot\boldsymbol{r}}$ for magnetic scattering [121,122].

To simulate neutron diffraction, I calculate the structure factor F_{hkl} for each q-vector q = (h k l).The total non-magnetic scattering per unit cell is given by

$E(\varphi) = \frac{\pi N_0}{2\kappa^2}\sum_{hkl} 4\pi {F_{hkl}}^2 d_{hkl}$, where d_{hkl} is the spacing between the atoms and ${F_{hkl}}^2$ is the square of the structure factor and N_0 is the number of unit cells per unit volume [122]. The square of the structure factor is written

${F_{hkl}}^2 = \left|\sum_{hkl} b exp\{ (2\pi i(\frac{hx_n}{a} + \frac{ky_n}{b} + \frac{lz_n}{c}))\}\right|^2$,

where b is called the scattering amplitude and it taken at each atomic position [122] for a non-magnetic scattering [122].

The magnetic structure factor is expressed by $\boldsymbol{F_{mag}} = \frac{e^2\gamma}{mc^2}\sum_n \mathbf{S}_n \mathbf{q}_n \exp\{2\pi i(\frac{hx_n}{a} + \frac{ky_n}{b} + \frac{lz_n}{c})\}$, where $\mathbf{q} = \mathbf{K} - \boldsymbol{\varepsilon}\,(\boldsymbol{\varepsilon} \cdot \mathbf{K})$ [4] and $\boldsymbol{\varepsilon}$ is the unit scattering vector [4]. Replacing $\mathbf{q}$ by $\mathbf{K} - \boldsymbol{\varepsilon}\,(\boldsymbol{\varepsilon} \cdot \mathbf{K})$ and $\mathbf{K}$ by $\boldsymbol{S}_\mathbf{n}$ gives

$\boldsymbol{F_{mag}} = \frac{e^2\gamma}{mc^2}\sum_n [\boldsymbol{S}_\mathbf{n} - \boldsymbol{\varepsilon}(\boldsymbol{\varepsilon} \cdot \boldsymbol{S}_\mathbf{n})] \exp\{2\pi i(\frac{hx_n}{a} + \frac{ky_n}{b} + \frac{lz_n}{c})\}$.

Note that $\boldsymbol{F_{mag}}$ is a vector quantity and its squared magnitude gives the magnetic scattering intensity [122].

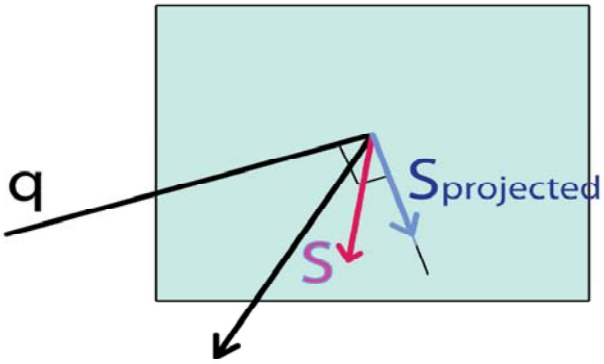


Figure 32: Magnetic Scattering vector q

Any **q** vector can be written $\vec{q} = (q_x, q_y, q_z) = (h, k, l) = (\frac{2\pi}{a}h, \frac{2\pi}{a}k, \frac{2\pi}{a}l)$, wherer *a*, *b*, and *c* are lattice parameters [122].

$$F_{hkl} = \left|\sum_n S_n \exp\left[2\pi i\left(\frac{x_n}{a}h + \frac{y_n}{b}k + \frac{z_n}{c}l\right)\right]\right| = \left|\sum_n S_n \exp\left[2\pi i\left(\frac{x_n}{a}h + \frac{y_n}{b} + \frac{z_n}{c}\right)\right]\right|$$

The helix pitch is defined $\Lambda = \frac{1}{1-q}$, where q is the scattering vector. For each temperature and each set of exchange constants, spin images are output and the structure factor is calculated for each spin output along different q-vector values. For a Type I spin order, the diffraction peak should be formed at about q = (0, 0, $\frac{1}{2}$) since the system is AFM [122].

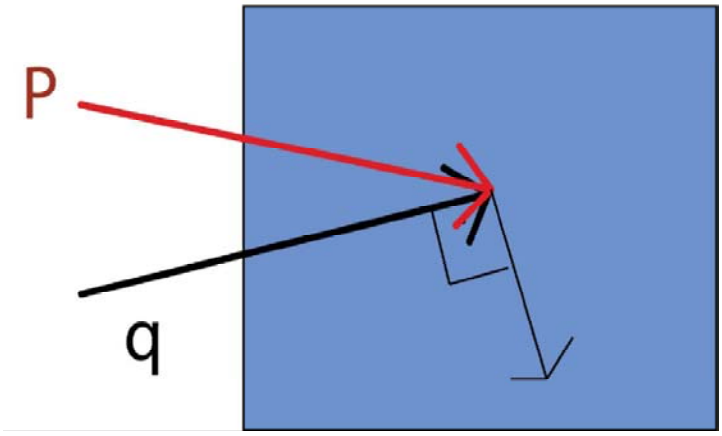


Figure 33: Calculating the magnetic structure factor

The algorithm for calculating the structure factor is in the section A4 of Appendix A.

4.5. Results and discussion

4.5.1. Testing the structure factor program

To test the program that calculates the structure factor, I used model inputs. FM and AFM spin arrays of 320 nonzero spin produced diffraction peaks in the correct positions. The pattern of diffraction peaks for an FCC ferromagnet exist for q = (1, 1, 1) and q = (2, 2, 2), which was shown in Figure 47 [34]. For the case of a Type I antiferromagnet, it gives a diffraction peak at q = (001) as suggested by T.M. Giebultowicz et al. in their studies using neutron scattering [34].

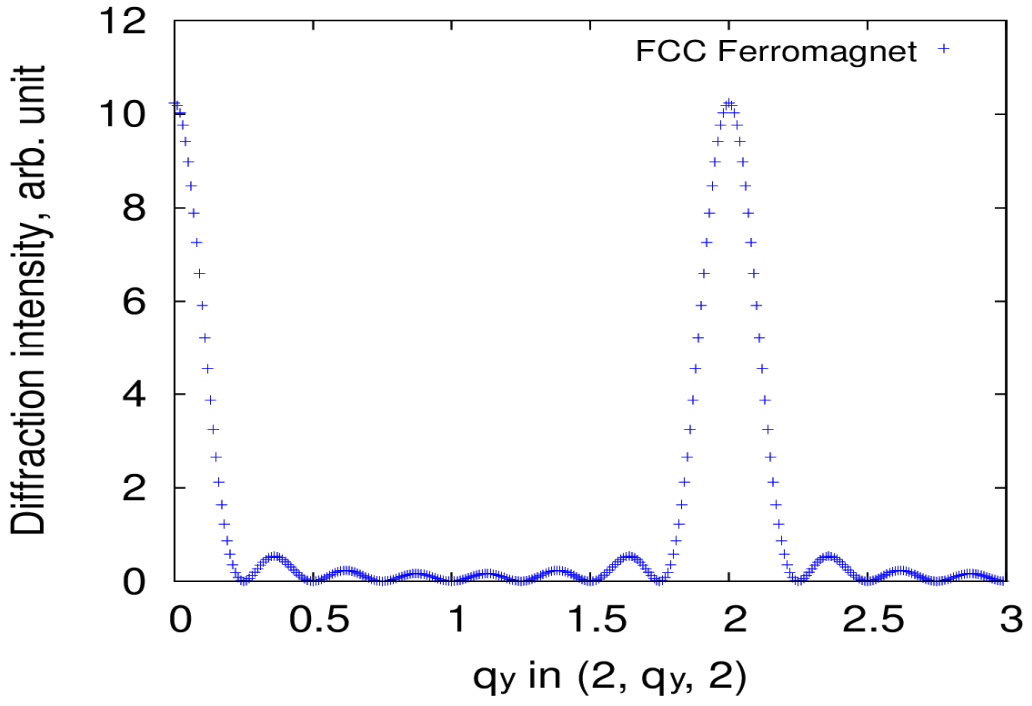


Figure 34: Structure factor for an FCC ferromagnet: the diffraction peaks are located at q = (222).

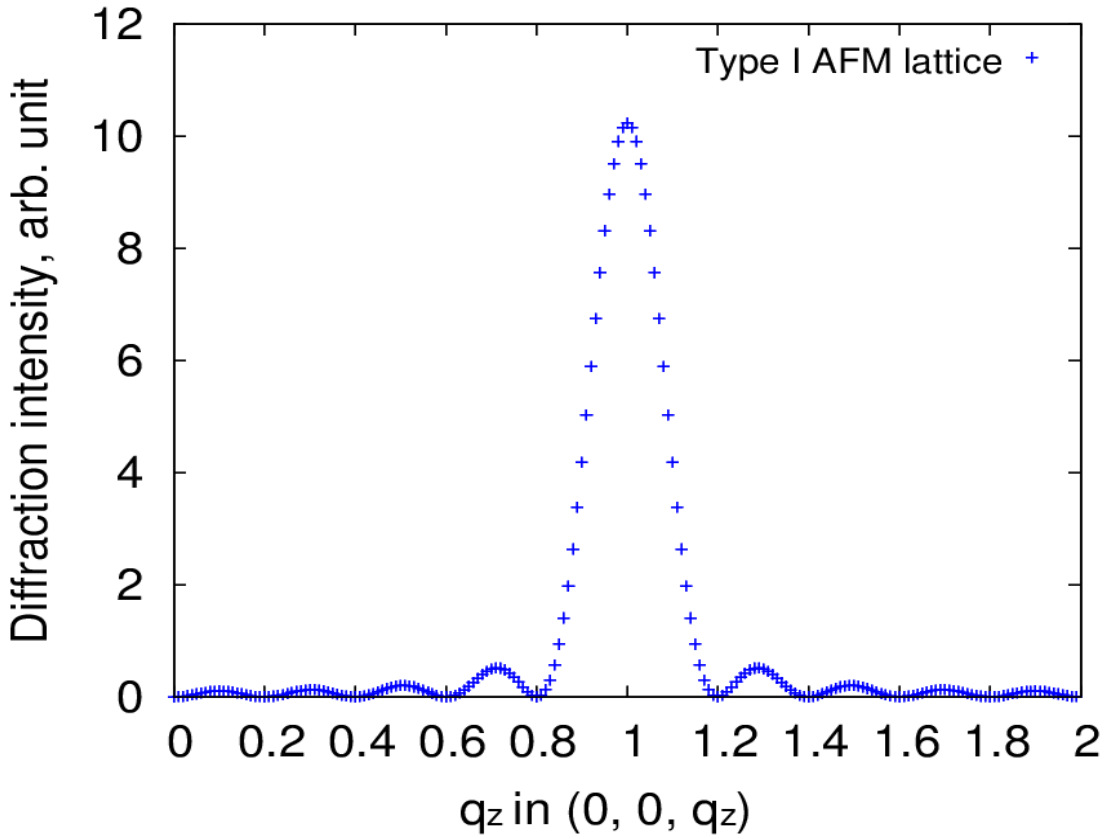


Figure 35: Structure factor for a Type I FCC antiferromagnet: the diffraction peak is located at q = (0 0 1).

4.5.2. Results of structure factor calculations

To verify that my MC simulation program produces the correct output, I calculated the spin outputs for $J_{1in} = J_{1out} = 1.0$ and $J_{2in} = J_{2out} = 0.1$, which is supposed to produce the Type III AFM spin lattice. The diffraction peak is expected to be at $(1, \frac{1}{2}, 0)$ and I obtained the peak at the correct position, which is shown in Figure 36 [40].

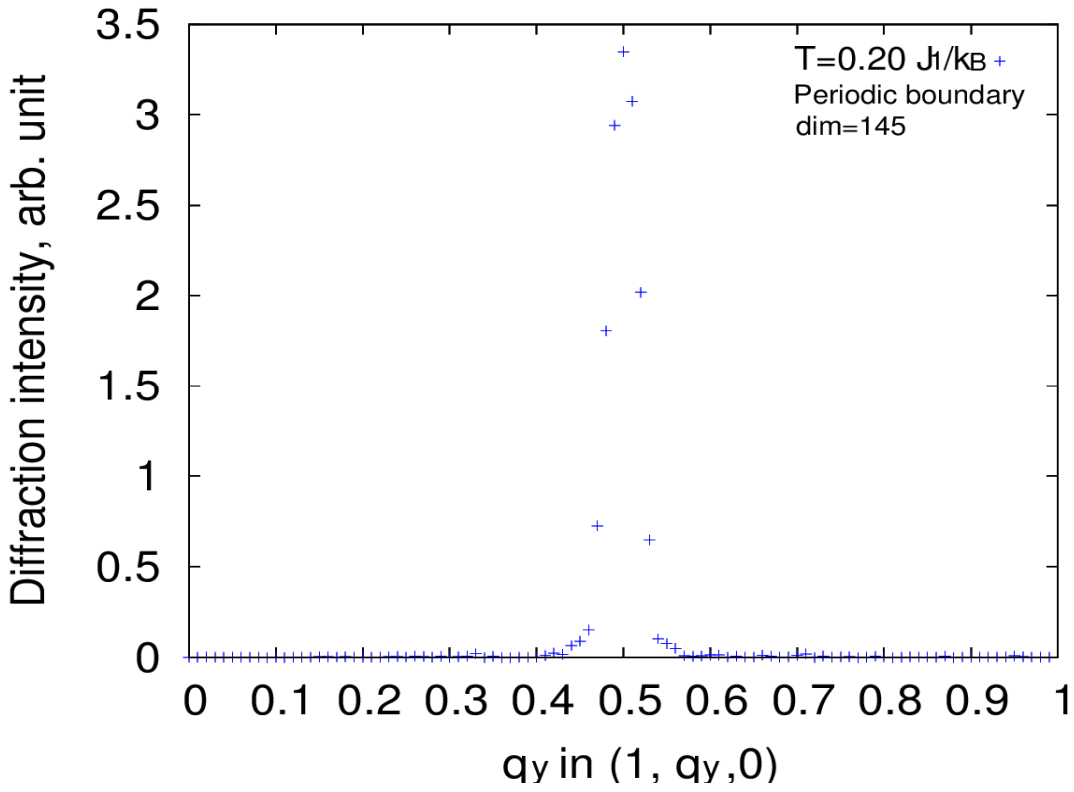


Figure 36: Neutron diffraction peak at temperature T = 0.20 J_1/k_B for the 1-dimensional spin (i = 145), Type III lattice: the peak is located at q = (1 $\frac{1}{2}$ 0).

The next step is to calculate the structure factors for MnSe/ZnTe lattice models for different exchange constant ratios. I used the q-vectors in (q_x, 0, 1) to analyze the data. For J_{1in} = 0.925 and J_{1out} = 1.075, the neutron diffraction scan along (q_x, 0, 1) are located about q_x = 0.35 (See Figure 37). As the temperature increases, the diffraction peak gets shifted to a lower q_x value and the diffraction intensity decreases. The diffraction peaks are relatively well-defined compared to the other neutron diffraction plots I produced with different J_{1in} / J_{1out} ratios.

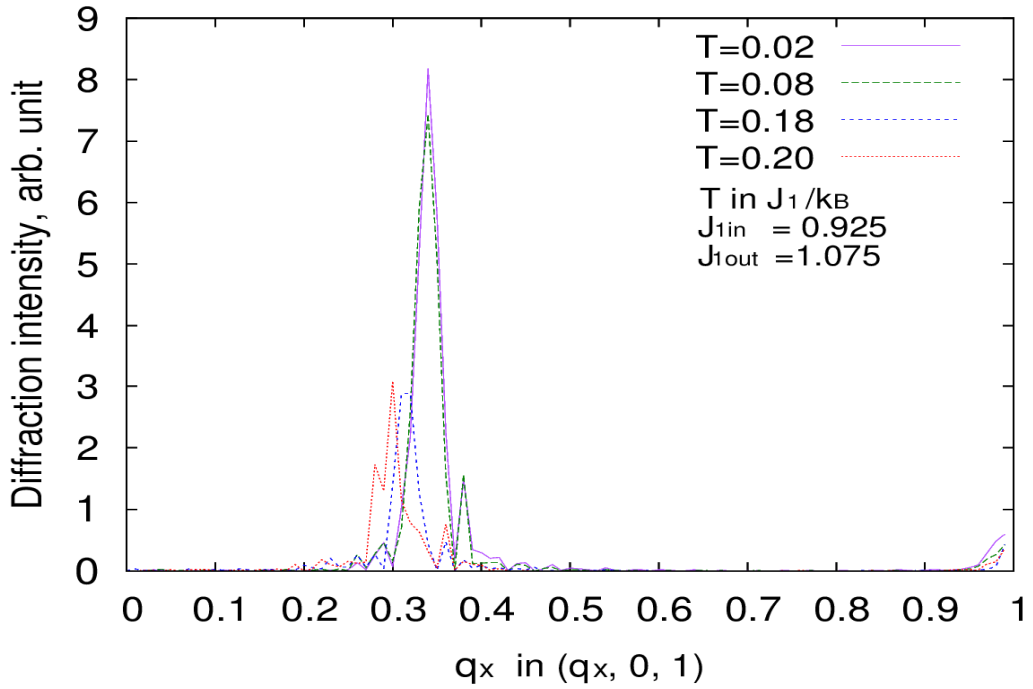


Figure 37: Diffraction peaks for $J_{1\text{in}} = 0.925$ and $J_{1\text{out}} = 1.075$, 1-dimensional spin array (i = 145) with the free boundary condition

The peak intensity decreased as the temperature increased, which is reasonable since magnetic order gets weaker with temperature increase. Another factor influencing the peak intensity is that the orientation of helical spin order shifts since it is not coupled to the lattice, which was confirmed when the spin lattices were examined. Also it is necessary to note that the spin projection value changes systematically as I change the q-vector value since neutron diffraction simulation only sums up the value of the spin component projected onto the plane perpendicular to the q-vector.

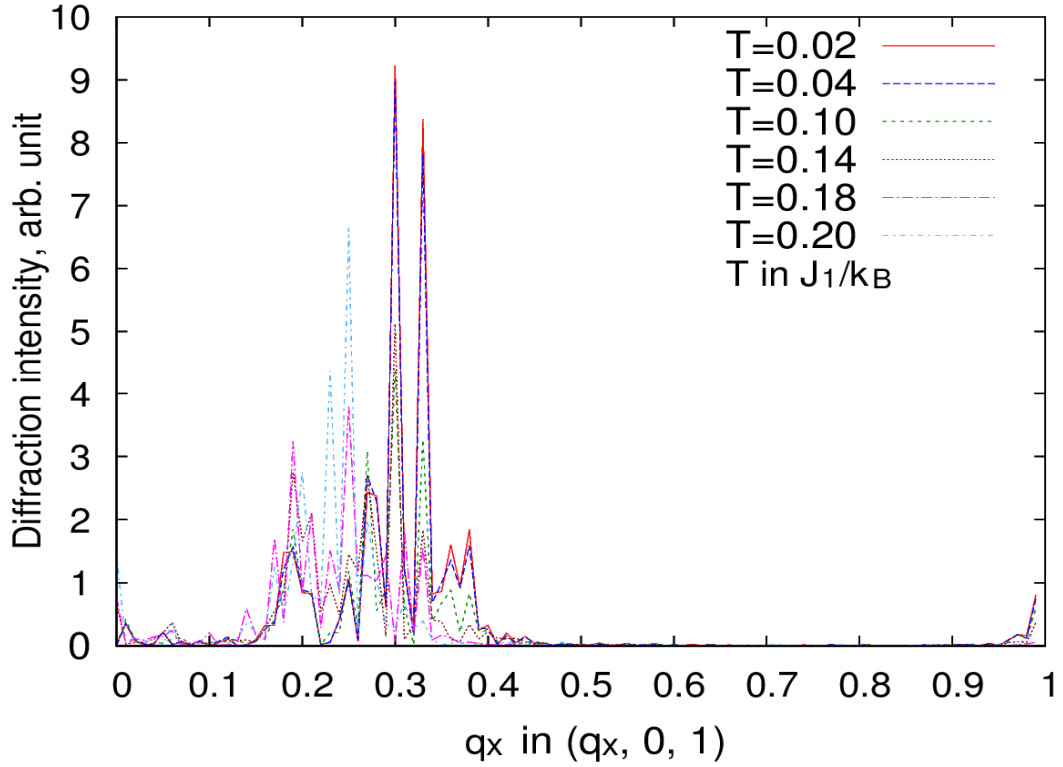


Figure 38: Diffraction intensities vs. q_x, 1- dimensional spin array with the free boundary condition (i = 145) for $J_{1\text{in}} = 0.9125$ and $J_{1\text{out}} = 1.0875$

In the case of calculations with $J_{1\text{in}} = 0.9125$ and $J_{1\text{out}} = 1.0875$, many diffraction peaks are visible, which might indicate there are multiple helical phases in the lattice. As I increase the temperature, the group of diffraction peaks appears to move toward a smaller q_x value. However, it is impossible to make a conclusion on the motion of the helix peaks from Figure 38 due to the issue of identifying diffraction peaks.

The periodic boundary case with $J_{1\text{in}} = 0.925$ and $J_{1\text{out}} = 1.075$ does not produce a well-defined single diffraction peaks. There is no significant shift of the position of the diffraction peaks (Figure 39).

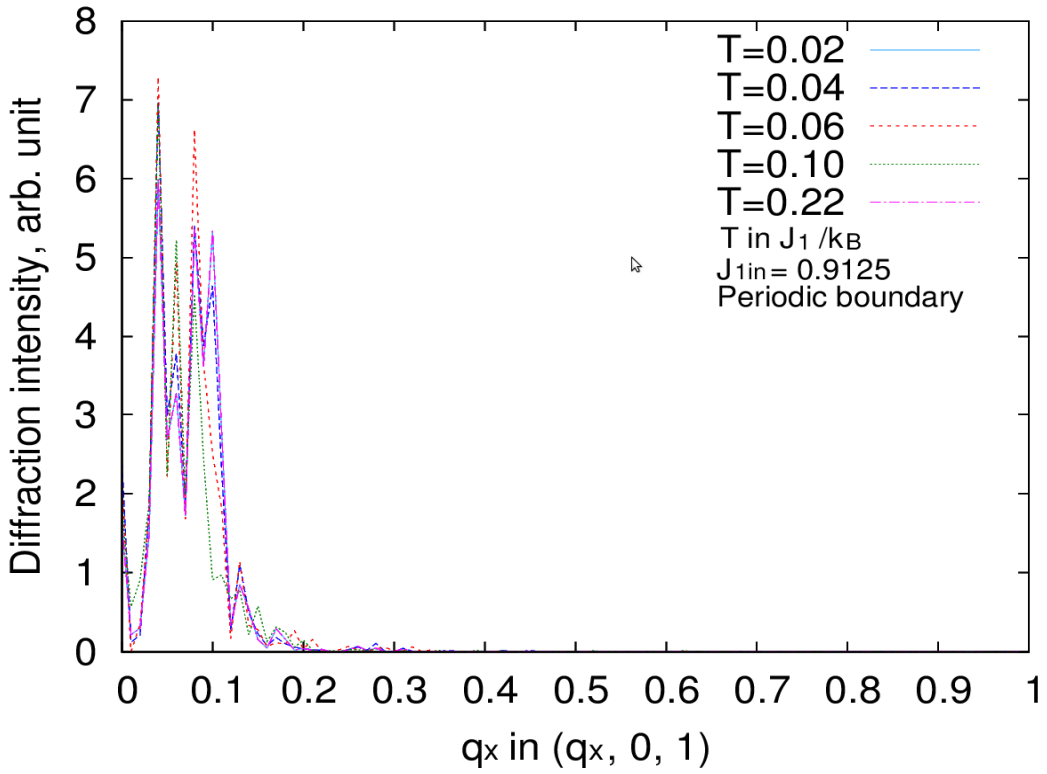


Figure 39: Periodic boundary: Diffraction intensity vs. (q_x, 0, 1): 1-dimensional spin array (i=145) for $J_{1\text{in}} = 0.9125$ and $J_{1\text{out}} = 1.0875$

With the periodic boundary and $J_{1\text{in}} = 0.9125$ and $J_{1\text{out}} = 1.0875$, the neutron diffraction scan for q = (q_x, 0, 1) for the 1-dimensional periodic lattice with size I = 145 looks like Figure 39. There are many diffraction peaks accumulated in the region $q_x \in [0.01, 0.2]$ and the peak positions do not get shifted as much as the case of the free boundary as the temperature increases.

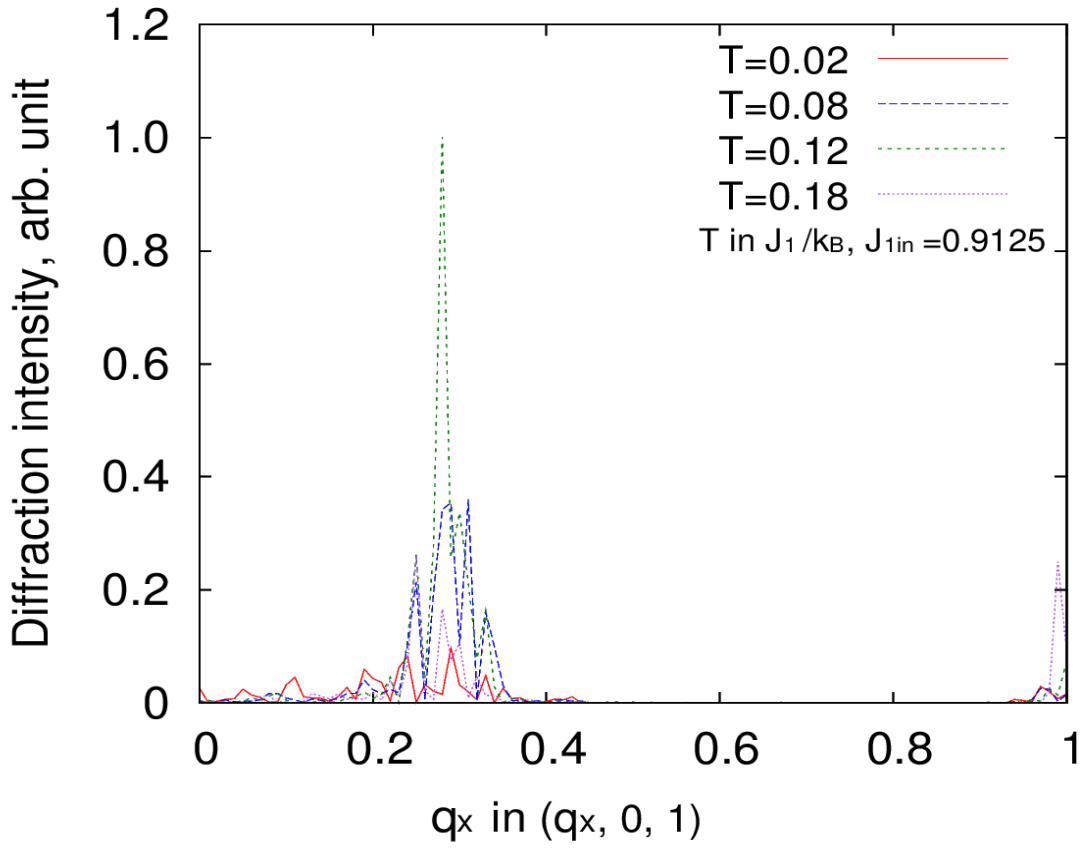


Figure 40: 3D spin array $J_{1in} = 0.9125$, 3-dimensional spin array with the free boundary condition (the dimensions were $n_x = n_y = 100$, $n_z = 10$)

For a 3-dimensional spin array with $J_{1in} = 0.9125$ and the free boundary condition, the diffraction peak positions were located in $q_x \in [0.2,0,4]$, which is similar to the range of q_x for the 1-dimensional spin array case. High peaks are near $q_x = 0.3$ and there are many other peaks in the plot. I notice that the neutron diffraction peaks did not completely visible at temperatures 0.02 J_1/k_B, 0.08 J_1/k_B, and 0.18 J_1/k_B. From the plots, it is difficult to make any conclusions on the positions of diffraction peaks.

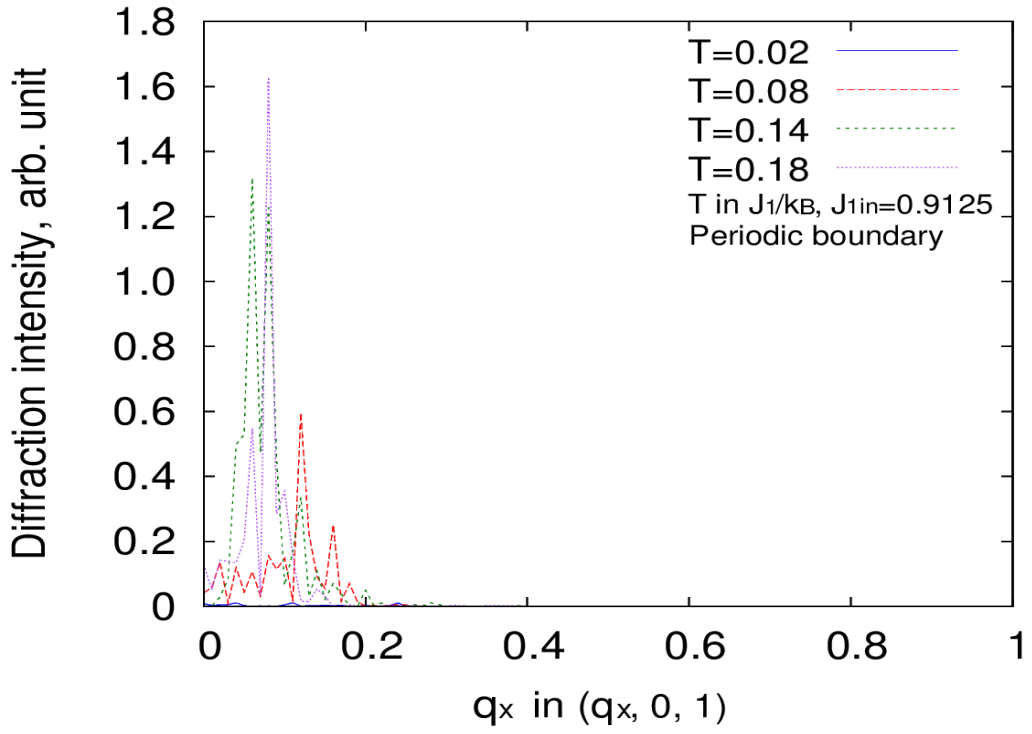


Figure 41: 3D spin array $J_{1\text{in}} = 0.9125$ periodic boundary $n_x = n_y = 100$, $n_z = 10$

With the periodic boundary the diffraction peaks are located at $q_x \in [0.0, 0.2]$, where the highest peaks are at about $q_x = 0.1$. The location of the diffraction peaks gets shifted as the temperature increases, but the direction of the shift is not identifiable. For example, the diffraction peaks for T = 0.08 J_1/k_B are at a larger q_x value than those for T = 0.02 J_1/k_B. The location of peaks for this case is similar to that for the 1D spin array case with the periodic boundary. The quality of data I obtained from my calculations was not good enough to make judgments.

Since the periodic boundary condition seems to "impose" a periodicity on the lattice edges, I concluded that it is inadequate to use periodic boundaries for simulating incommensurate magnetic ordering, as it was mentioned in Chapter 1 [35]. Rough

estimates from simulation data show that the diffraction peaks are fluctuating in a fixed range of q_x values.

The last step for the simulation was to investigate how the size of the system influences the helical pitch, I performed MC calculations of both 1-dimensional and 3-dimensional spin arrays of different sizes.

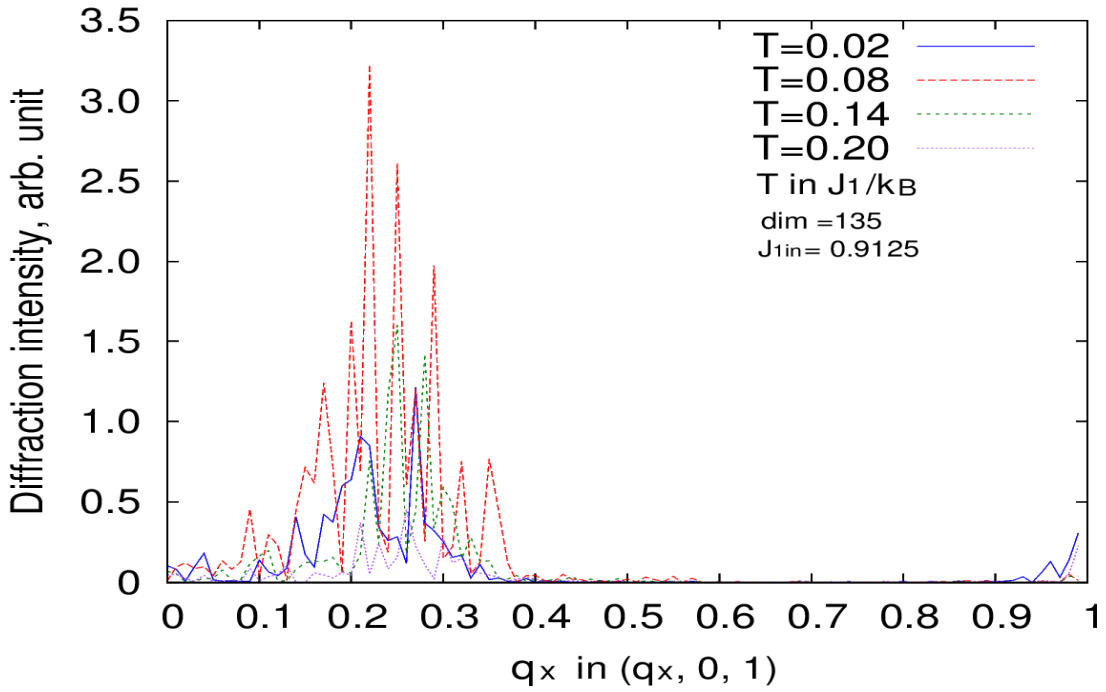


Figure 42: Neutron diffraction intensities for 1-dimensional spin array with free boundary (i = 135) for $J_{1\text{in}} = 0.9125$ and $J_{1\text{out}} = 1.0875$

Figure 42 shows the simulated neutron diffraction peaks for 1-dimensional spin array of size i = 135. Multiple peaks are observed for most of the temperatures. For both 3-dimensional and 1-dimensional spin arrays, the neutron peak formation gets more difficult as I decrease the size of the lattice with the same number of MC iteration per spin.

The lattice size can influence the formation of long range order, whose idea comes from the arguments for Mermin-Wagner theorem [123, 124]. Mermin-Wagner theorem states that there is no long range order in 1-dimensional or 2-dimensional lattice [124]. M. Manojlović et al. reported that Mermin-Wagner theorem can be extended to 3-dimensional isotropic spin lattices after studying La_2CuO_4-type compound using an isotropic spin model [124]. I might be able to apply this idea to a more general case, but it is beyond the scope of this project.

4.5.3. Discussion on the quality of data

In the previous section, I stated that it is difficult to make conclusions on the positions of neutron diffraction peaks since in many cases the peaks were not completely formed. One of the difficulties in using the Metropolis algorithm was that equilibration at low temperatures is highly inefficient [65]. At temperature $0.02J_1/k_B$ the neutron diffraction peak was not formed at all in many cases. In the trilayer studies in Chapter 3, the temperature range was above $T_N = 1.2\ J_{2,NiO}/k_B$ with H in eq. 3.1, but the transition temperature for simulating helical spin order is $0.28\ J_1/k_B$ [49].

I concluded that the system was not fully equilibrated with the Metropolis algorithm due to its low spin flip rate at low temperatures. The programs started calculations from a random spin configuration and it should have made the equilibration more difficult. Another issue is that using a variable-sized array significantly slowed

down the calculations. In sum, the result suggested that revising the scheme for simulating a thick lattice is required.

4.6. Resolving the issues with the MC calculations

4.6.1. Walker and Walstedt Algorithm

There are several options to improve the quality of the data. One is to run the current MC program for a longer period of time to check if a stable single helix gets formed. Another is to use a simpler program, possibly with a fixed-sized spin array to speed up the calculation. A third is to use an algorithm that does not lose efficiency at low temperatures. The fourth is to add easy-plane anisotropy to "facilitate" the formation of helix on a specific plane.

The Metropolis algorithm becomes inefficient at low temperatures since the spin flip rate is low [65, 66]. To improve the results, I used an alternative MC algorithm, which is called Walker and Walstedt (W & W) algorithm. It was developed by L. R. Walker and R. E. Walstedt and it was discussed in their article published in 1980 [125]. In parallel to the W & W algorithm with fixed-sized spin arrays, I wrote a program that uses the Metropolis algorithm with a fixed-sized spin array to cross-examine the result.

It is quite straightforward to implement the W & W algorithm with 3-dimensional classical vector spins. The key to running MC simulation is to generate an appropriate random set of states according to the Boltzmann probability distribution and

the two major requirements that have to be satisfied for Monte Carlo calculations are ergodicity and the detailed-balance principle [65, 66]. Ergodicity means that there must be a nonzero probability between any two states that are picked and the detailed balance means the rate at which the system makes transitions into and out of any state must be equal [65, 66]. In other words, the detailed balance principle states

$$P(State\ 1 \rightarrow State\ 2) = P(State\ 2 \rightarrow State\ 1).$$

The Boltzman probability distribution is $P(\mathcal{E}) = Ce^{\frac{-\mathcal{E}}{k_B T}}$, where C is a normalization constant and $\mathcal{E}$ is the total magnetic energy. Without anisotropy the Hamiltonian is

$$H = -J_1 \sum_i \sum_{j \in nb} \boldsymbol{S_i} \cdot \boldsymbol{S_j} - J_2 \sum_i \sum_{k \in nnb} \boldsymbol{S_i} \cdot \boldsymbol{S_k}. \quad (4.4)$$

I define the total effective field as

$$\boldsymbol{E}_{eff} = -J_1 \sum_j \boldsymbol{S_j} - J_2 \sum_k \boldsymbol{S_k} \quad (4.5)$$

and rewrite the Hamiltonian

$$H = \sum_i \boldsymbol{E}_{eff} \cdot \boldsymbol{S_i} = \sum_i E_{eff} S_i cos(\theta_i), \quad (4.6)$$

where θ_i is the angle between the two vectors $\boldsymbol{E}_{eff}$ and $\boldsymbol{S_i}$. The probability density is $P(\mathcal{E}, \mathcal{E} + d\mathcal{E}) = p(\mathcal{E})d\mathcal{E} = Ce^{\frac{-\mathcal{E}}{k_B T}}d\mathcal{E}$ and $P(\mathcal{E}_{min}, \mathcal{E}_{max}) = 1$ [125].

Putting $\boldsymbol{E_{eff}}$ in the direction of the z-component of the spin S_Z makes

$$H = \sum_i \sum_i \boldsymbol{E}_{eff} \cdot \boldsymbol{S_i} = \sum_i S_z E_{eff}.$$

$$P(\mathcal{E}) = C \int_{\mathcal{E}_{min}}^{\mathcal{E}} e^{\frac{-\mathcal{E}}{k_B T}} d\mathcal{E} = C(-k_B T e^{\frac{-\mathcal{E}}{k_B T}}|_{-\mathcal{E}_o}^{\mathcal{E}}) = C(k_B T e^{\frac{\mathcal{E}_o}{k_B T}} - k_B T e^{\frac{-\mathcal{E}}{k_B T}})$$

For normalization, $C = \frac{1}{P(\mathcal{E}_{min}, \mathcal{E}_{max})} = \frac{1}{(kTe^{-\frac{\mathcal{E}_{min}}{k_B T}} - kTe^{\frac{-\mathcal{E}_{max}}{k_B T}})}$.

Set the maximum total energy $\mathcal{E}_{max} = \mathcal{E}_o$ and the minimum total energy $\mathcal{E}_{min} = -\mathcal{E}_o$, since the maximum and minimum energy values have the same absolute value.

$$P(-\mathcal{E}_o, \mathcal{E}) = \frac{\frac{k_B T}{\mathcal{E}_o} e^{\frac{\mathcal{E}_o}{kT}} - \frac{k_B T}{\mathcal{E}} e^{\frac{-\mathcal{E}}{k_B T}}}{k_B T e^{\frac{\mathcal{E}_o}{k_B T}} - k_B T e^{\frac{-\mathcal{E}_o}{k_B T}}} = \frac{e^{\frac{\mathcal{E}_o}{k_B T}} - e^{\frac{-\mathcal{E}}{k_B T}}}{e^{\frac{\mathcal{E}_o}{k_B T}} - e^{\frac{-\mathcal{E}_o}{k_B T}}} = \frac{1 - e^{\frac{-2\mathcal{E}}{k_B T}}}{1 - e^{\frac{-2\mathcal{E}_o}{k_B T}}} = R, where\ R \in [0,1].$$

Solve the above equation for energy $\mathcal{E}$ [125].

$$e^{\frac{-2\mathcal{E}}{k_B T}} = R e^{\frac{-2\mathcal{E}_0}{k_B T}} - (R - 1) \tag{4.7}$$

$$\mathcal{E} = -k_B T \ln \left[R e^{\frac{-2\mathcal{E}_0}{k_B T}} - (R - 1)\right] \tag{4.8}$$

Take another random number $R' = 1 - R$. Since $R \in [0,1], R' \in [0,1]$.

$$\mathcal{E} = -k_B T \ln \left[(R' - 1) e^{\frac{-2\mathcal{E}_0}{k_B T}} - R'\right] \tag{4.9}$$

The equation written by L. R. Walker et al. is of the form $\mathcal{E} = -k_B T \ln \left[(R' - 1) e^{\frac{-\mathcal{E}_0}{k_B T}} - R' e^{\frac{\mathcal{E}_0}{k_B T}}\right]$, which is equivalent to eq. 4.9 [125].

For each Monte Carlo step the previous value of the spin is not considered. Instead a new spin that satisfies the Boltzmann probability distribution is generated and replaces the old spin value [125]. For each Monte Carlo step, pick a random number R' and calculate energy by the equation

$$\mathcal{E} = -k_B T \ln \left[(R^{'} - 1) e^{\frac{-2\mathcal{E}_0}{k_B T}} - R'\right]. \tag{4.10}$$

Set the magnitude of the z-component of the spin to be $\mathcal{E}$ and set $\boldsymbol{S_z}$ to be parallel to the z-component of the effective field vector $\boldsymbol{E_{effz}}$.

Pick a random spin site i and generate a random number $R \in [0,1]$ and set $S_{iz} = -kT \ln\left[(R-1)e^{\frac{-2\varepsilon_0}{kT}} - R\right]$, where $\varepsilon_o = E_{eff} = \max\{E_{eff} S_i cos(\theta_i)\}$; this ensures that the exchange energy of the chosen spin satisfies the Boltzmann probability distribution.

Then generate the S_{ix} and S_{iy} randomly by picking a random angle φ and setting $S_{ix} = \frac{\sqrt{1-{S_{iz}}^2}\cos\,\varphi}{\sqrt{{E_{effx}}^2+{E_{effy}}^2+{E_{effz}}^2}}$ and $S_{iy} = \frac{\sqrt{1-{S_{iz}}^2}\sin\,\varphi}{\sqrt{{E_{effx}}^2+{E_{effy}}^2+{E_{effz}}^2}}$. Update the spin and energy and other calculated variables.

The detailed balance principle states

$P(State\ 1 \rightarrow State\ 2) = P(State\ 2 \rightarrow State\ 1)$. [65].

For the construction I have above, the detailed balance principle is satisfied since $P(old\ spin \rightarrow new\ spin) = P(new\ spin \rightarrow old\ spin) = Ce^{-\varepsilon_o/kT}$.

In fact, the method described above is not the only method that I can use to generate a random spin. For example, I can choose to use the rotation matrix with Euler angles; I decided not to use it since it takes more operations and slows down my program.

Also it should be noted that W & W algorithm works well only for Heisenberg spins since it is more difficult to develop a method for generating a random XY-spin since there is no analytical solution to the integral $P(\mathcal{E}) = C\int_{\mathcal{E}_{min}}^{\mathcal{E}} e^{\frac{-\mathcal{E}}{k_B T}} d\mathcal{E}$, as the energy is not equal to SzE_{eff}. For detailed computational procedures, see Appendix C.

4.6.2. Results and discussion

The spin flipping rate for the Metropolis algorithm at the temperature T = 0.02 J_1/k_B was about 1% for the Metropolis algorithm, which is fairly low. I checked that before the system gets equilibrated there is a phase of multiple helices in the system that causes multiple neutron diffraction peaks. Eventually a single peak "survives" as the number of MC steps increases. The MC program with the Metropolis algorithm with fixed-sized spin arrays produced results similar to the results from the W&W algorithm.

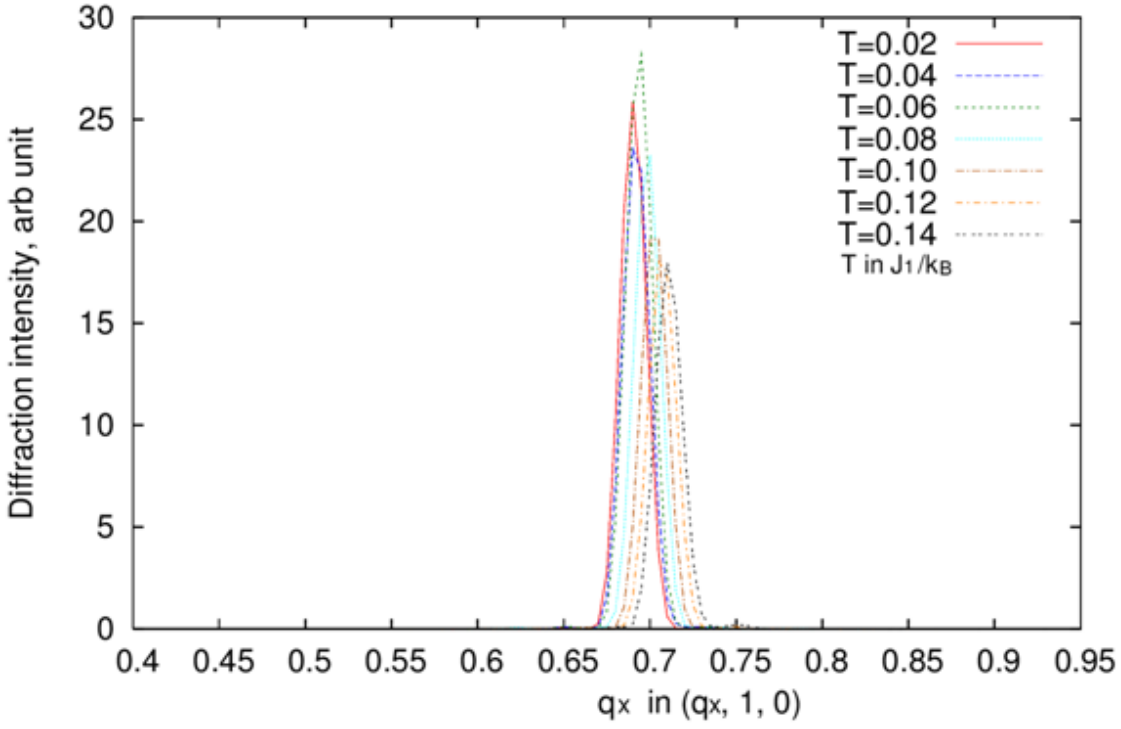


Figure 43: Diffraction intensity vs. q_x in (q_x,1,0) with J_{1in} = 0.9125, J_{1out} = 1.0875 with W&W algorithm

As in Figure 43, the value of q_x increases with temperature for the case with J_{1in}=0.9125 and J_{1out}=1.0875. Note that the scan direction is different from my previous plots, which was (q_x, 0, 1). If I scan the spin lattice along the (q_x,0,1), the peaks will move to lower q_x value as I increase the temperature. The helix pitch $\Lambda = \frac{1}{q_x}$ for (q_x, 0, 1) scan $\Lambda = \frac{1}{1-q_x}$ for (q_x, 1, 0).

Figures 44 and figure 45 show the helix pitch Λ as a function of temperature for thin and thick films. For a thick film the pitch changes faster than the thin film case regardless of the direction of the temperature change. Also it is notable that the value of the pitch Λ depends on whether the temperature is increasing or decreasing. The helix pitch does not change linearly with the temperature. At low temperature region, the pitch changes more slowly with temperature than the high temperature region.

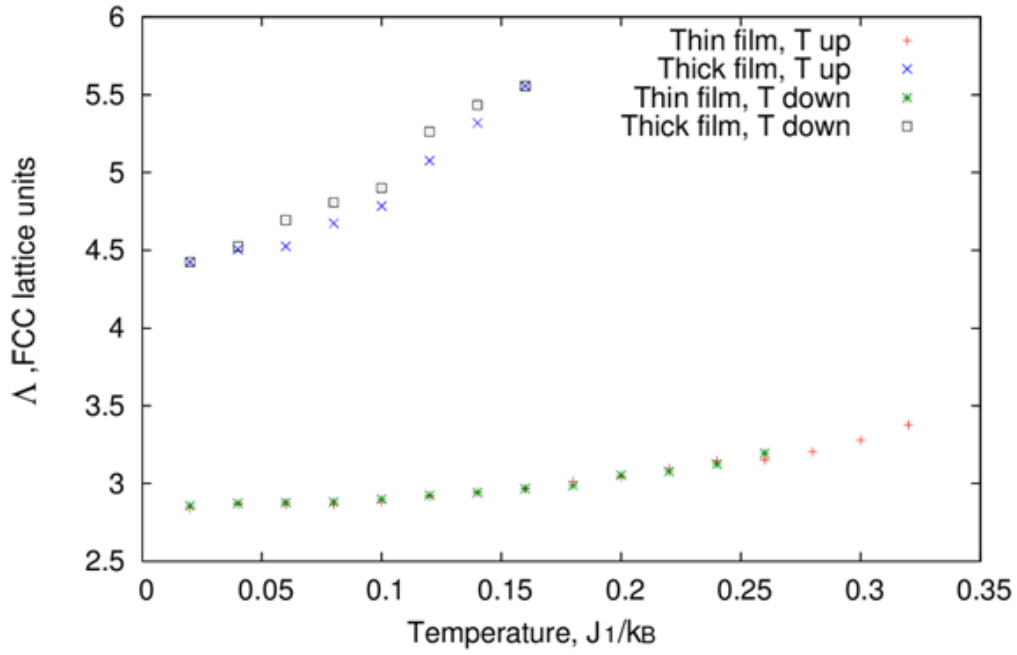


Figure 44: Helix pitch Λ vs. temperature T for MnTe/ZnSe model with J_{1in} = 0.925 and J_{1out} = 1.075 for thin and thick films

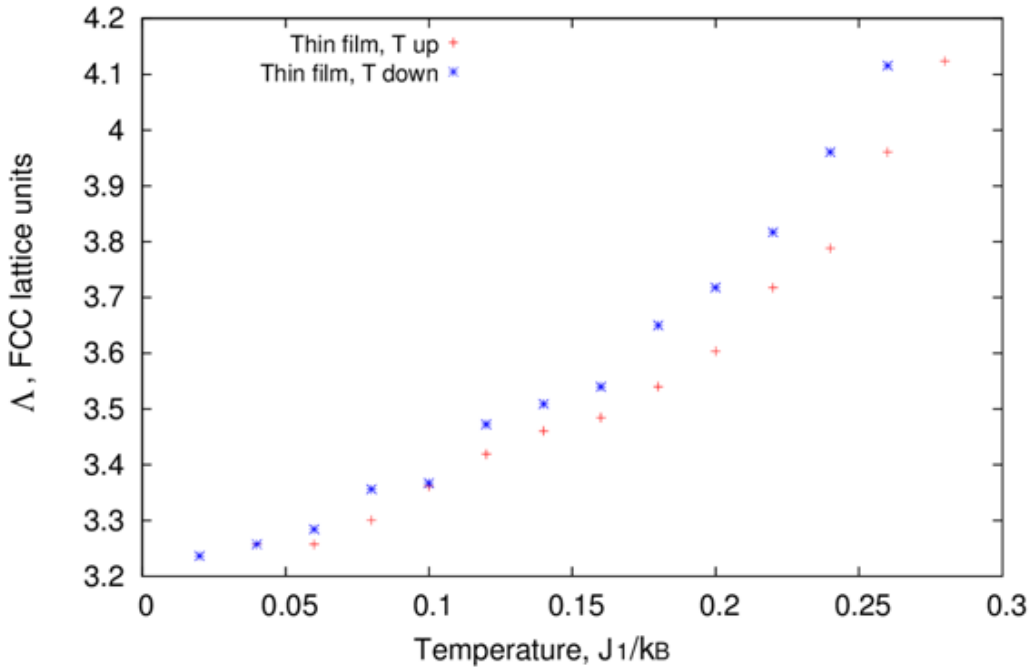


Figure 45: Helix pitch Λ vs. temperature T for $J_{1\text{in}} = 0.9125$ and $J_{1\text{out}} = 1.0875$ for thin and thick films

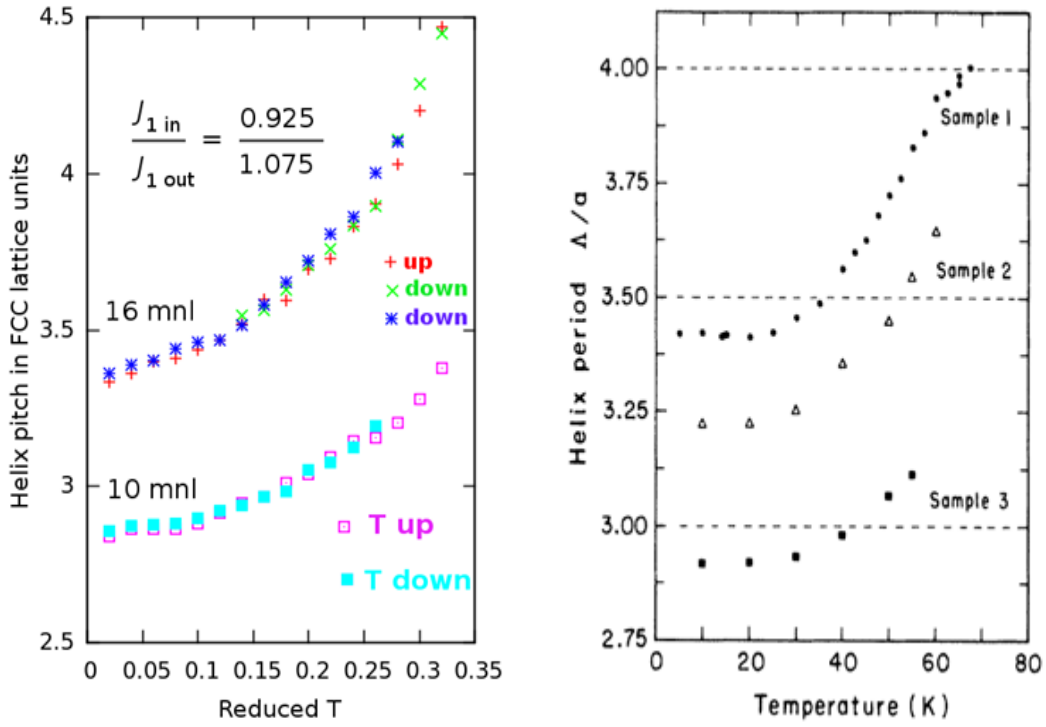


Figure 46: Comparison to experimental result: the slope of the helix period vs. temperature plot: the experimental helix period starts "flat" and it changes to a "steep curve" as the temperature increases. Compared to the experimental result, the calculated helix pitch exhibits a curve whose slope increases as the temperature increases, but the plot is not "flat" at the lower temperature region.

Figure 46 shows the plots of the pitch of spin helix vs. temperature from simulation data (left) and from experimental data (right) from T. M. Giebultowicz et al. [2]. Experimental data show that the helix pitch Λ stays rather flat until a threshold temperature and starts increasing after the threshold temperature [2]. Simulation plot also shows an increase of helix pitch with temperature. However, there is no "flat region" in the Λ vs. temperature curve from the simulation, though the slope of simulated Λ curve is smaller at low temperatures. It is probable that the discrepancy is due to the fact that the MC model used is not simulating long-range order at low temperatures.

4.6.3. Summary and comments

The calculation of structure factor from the spin outputs from the Monte Carlo calculations yielded diffraction peaks whose pitch increases with temperature as in experimental results [2]. As the thickness of the lattice gets larger, the pitch of the spin helix increased. For the free boundary conditions, regardless of the lattice size, the diffractions peaks were located near $q_x = 0.35$. For the periodic boundary cases, the peaks were concentrated at about $q_x = 0.15$, which is due to the fact that the boundary condition imposes a periodicity to the lattice.

The initial idea was that the change in the pitch of spin helix with temperature might be caused by the weakening of exchange coupling of surface spins at high temperatures due to the fact that surface spins have incomplete neighbor sets. This possibility is completely ruled out since the pitch changed with temperature for regardless of the thickness of the lattices. In this case, consideration of the free energy of the system is necessary to understand the mechanism that underlies the change in pitch of spin helix since the stabilization of magnetic structures will depend on the free energy [65].

Chapter 5. Concluding remarks

In the beginning of this project, a MC model for NiO/CoO superlattice systems was developed to simulate experimental results from J.A. Borchers's group, which indicates that the magnetic order of an antiferromagnet persists above its Néel temperature when it is layered with another antiferromagnet of a higher Néel temperature due to exchange pinning on the interface [1]. The simulated behavior of the superlattice was in a good qualitative agreement with the experimental data [1]. It was an important test for the MC model that was used in the next stages of the project.

Then I applied the MC model developed for NiO/CoO systems to simulation of FM/AFM/FM trilayer systems. The two FM layers in the trilayer were antiferromagnetically coupled by placing an AFM spacer with an even number of AFM monlayers. The simplest case was a FM/AFM/FM model without complicating factors such as anisotropy or disorder in the lattice. The simulation result showed that the AFM coupling between the two FM layers increases as the thickness of the AFM mid-layer decreases. It also indicated that AFM coupling between the FM layers is present at temperatures above the Néel temperature of the antiferromagnet and a helix-like spin arrangement similar to a "Bloch wall" is formed in the AFM/FM interface region [74]. The simulated magnetization vs. external magnetic field curve exhibits no hysteresis.

After studying the simplest FM/AFM/FM trilayers, I added anisotropy to the system. Two different anisotropy models were used: (1) hard-axis anisotropy and (2) easy-axis anisotropy along the direction of the external magnetic field. In the case of

hard-axis anisotropy, the magnetization curve vs. external magnetic field exhibited behaviors identical to those without anisotropy. On the other hand, with easy-axis anisotropy along the direction of the magnetic field, there was a hysteresis-like feature in the induced magnetization vs. magnetic field curve. As the temperature increased, the threshold magnetic field for the transition decreased. In all hysteresis curves, the magnetic transition was of the first order.

Lastly for FM/AFM/FM systems, the AFM spacer layer in the trilayer lattice was diluted. Before the trilayer calculations, magnetic order in a bulk AFM lattice was calculated with various concentrations of magnetic ions. The calculation result showed that magnetic order is present up until 50% concentration of magnetic ions in the AFM lattice, which is consistent with experimental results from T. M. Giebultowicz's group [108, 109]. The MC calculation for trilayers with diluted AFM mid-layer showed that the induced magnetization vs. external magnetic field curve exhibits hysteresis and that the gap in the hysteresis curve increases as the magnetic concentration in the AFM mid-layer decreases. The hysteresis curve was not centered at the zero magnetic field.

The last part of this project involved modeling helical spin order in MnSe/ZnTe superlattices, which is an FCC lattice under tensile strain [2]. Assuming that the distance between the magnetic ions influences the strength of exchange interaction, I set $J_{1\text{in-plane}} < J_{1\text{out-of-plane}}$ and used a free boundary condition, which is not supposed to "facilitate" the formation of helical spin order [117]. For both thin and thick film models, the pitch of the spin helix increased as the temperature increased, which implies that the increasing pitch

is not associated with the thickness of the lattice. This means that there might be other mechanisms that influence the pitch of the helix and it requires a further study.

There are several items to note on the project. One is that is no experimental data for FM/AFM/FM superlattices to verify the relationship between the strength of AFM coupling of the FM layers and the thickness of the AFM mid-layer since the focus of many groups who study FM/AFM/FM systems is mainly on the effect of exchange bias [91, 94, 95]. Persistence of AFM coupling of the FM layers at temperatures above the Néel temperature is consistent with Lenz et al.'s observation of magnetic order in FM/AFM systems [59].

Another is that The behavior of the trilayer system with anisotropy is consistent with Stoner-Wohlfarth model, which shows hysteresis only for easy-axis anisotropy [105]. Anisotropy can be controlled experimentally by adjusting parameters when one is fabricating a multilayer, so it is possible to investigate this phenomenon experimentally [103, 126].

A third is that diluting the AFM mid-layer in the FM/AFM/FM trilayers increases the gap of the magnetic hysteresis loop, which seems to promote the effect of exchange bias by moving the center of the hysteresis loop further from zero field. This can be studied experimentally since diluted systems give more feasibility to experimental studies since diluted magnetic layers can be fabricated with current technology [108, 109].

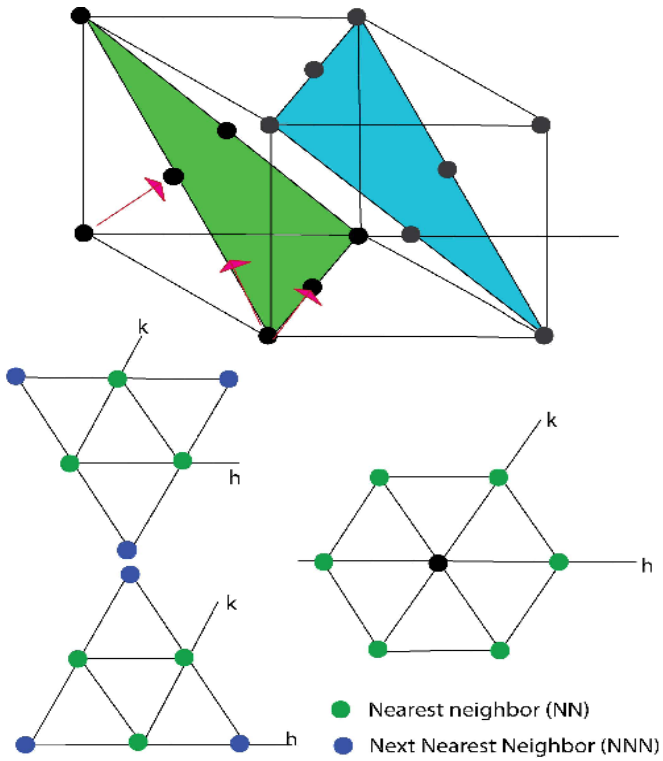
The last is regarding the MnSe/ZnTe superlattice model and it is that the free energy of the system should be calculated to understand the behavior of the system, which was not performed for this project [65].

Appendix A. Spin loading and miscellaneous calculations

A1. Basis of the lattice vectors for NiO/CoO and FM/AFM/FM lattices

Calculations for both NiO/CoO superlattices and FM/AFM/FM trilayer lattices were based on the same basis vectors.

$$B = \left\{\left(-\frac{1}{2},\frac{1}{2},0\right),\left(-\frac{1}{2},0,\frac{1}{2}\right),\left(0,\frac{1}{2},\frac{1}{2}\right)\right\}. \quad \text{(A.1)}$$



.

Figure A.1: Lattice vectors with the basis B: The lattice vectors are $\boldsymbol{h} = \left(0,\frac{1}{2},\frac{1}{2}\right), \boldsymbol{k} = \left(-\frac{1}{2},\frac{1}{2},0\right), \boldsymbol{l} = \left(-\frac{1}{2},0,\frac{1}{2}\right)$, denoted by red arrows. The spin on the lower right figure

(black dot) has 6 in-plane NNs (green dots) and 3 NNs (green dots) and 3 NNNs (blue dots) on each adjacent plane.

Each spin has six in-plane NNs, six out-of-plane NNs, and six out-of-plane NNNs. The spin neighbor coordinates are in Table A.1.

Table A. 1. Neighbor coordinates for spin at (h, k, l): NiO/CoO superlattices

Layer	Nearest neighbors	Next nearest neighbors
Upper layer (l+1)	(h,k,l+1), (h, k-1, l+1), (h-1,k,l+1)	(h-1,k-1,l+1), (h+1, k-1, l+1), (h-1, k+1, l+1)
In-plane (l)	(h+1, k ,l), (h-1, k ,l), (h, k-1, l),(h+1,k-1,l), (h, k+1,l), (h-1,k+1,l)	None
Lower layer (l-1)	(h, k, l-1), (h+1, k, l-1), (h, k+1, l-1)	(h+1, k+1, l-1), (h-1, k+1, l-1), (h+1, k-1, l-1)

Let $n_{x,}$ n_y, and n_y be the dimensions of the 3-dimensional spin array. To load spins, read in the spin and figure out the spin coordinates. The neighbor coordinates are calculated as the following for the spin at (h, k, l). To identify the periodic boundary, I set the coordinates as the following, where

$hup = (h + 1) \% n_x$

$hdn = (h – 1 + n_z) \% n_x$

$kup = (k + 1) \% n_y$

$kdn = (k – 1 + n_y) \% n_y$

$lup = (l + 1) \% n_z$

$ldn = (l - 1 + n_z) \% n_z$

A2. Reduced magnetization calculation procedure for NiO/CoO model

To calculate the reduced magnetization, I sum up the spins according to the following method.

(1) Read in a spin.

(2) Calculate the coordinates for the spin (h,k,l) by the formula in the Monte Carlo setup section.

(3) Define the sign of the lattice by the formulas; there are two sublattices of spins inter-penetrating each other and opposite in the spin orientations. Here, % is the modulo-arithmetic operator.

sign1 = (h % 2)*2-1

sign2 = (k % 2)*2-1

sign3 = (l % 2)*2-1

sign4 = ((h + k + l) % 2)*2-1

In this case, sign4 corresponds to the sublattices along the (111) direction.

(4) Add the spins multiplied by the sign to the reduced magnetization sum, which was initially set to be zero.

Reduced magnetization = Reduced magnetization + sign1*spin(h,k,l)

(5) Repeat this until the end of spin array is reached.

(6) For the layer reduced magnetization, sum up the spins per each layer.

For the layer reduced magnetization, I follow the same steps but sum up the spins per each layer.

A3. Spin loading: CoO/NiO superlattice and FM/AFM/FM trilayer lattice model

The lattice vectors used in the FM/AFM/FM trilayer lattices are similar to those that were used for the NiO/CoO superlattice model. In fact, the basis used for NiO/CoO superlattice model calculations is one of the six possible bases. There are three choices for the out-of-plane lattice vector $\boldsymbol{l} = \left(0, \frac{1}{2}, \frac{1}{2}\right), \boldsymbol{l} = \left(\frac{1}{2}, 0, \frac{1}{2}\right), or\ \boldsymbol{l} = \left(\frac{1}{2}, \frac{1}{2}, 0\right)$ and there are two options for choosing the in-plane vectors $\boldsymbol{h}$ and $\boldsymbol{k}$ for each $\boldsymbol{l}$. Each set of lattice vectors $\boldsymbol{h}$, $\boldsymbol{k}$, and $\boldsymbol{l}$ will result in different neighbor coordinates. Figure A.2 below shows the lattice vector options.

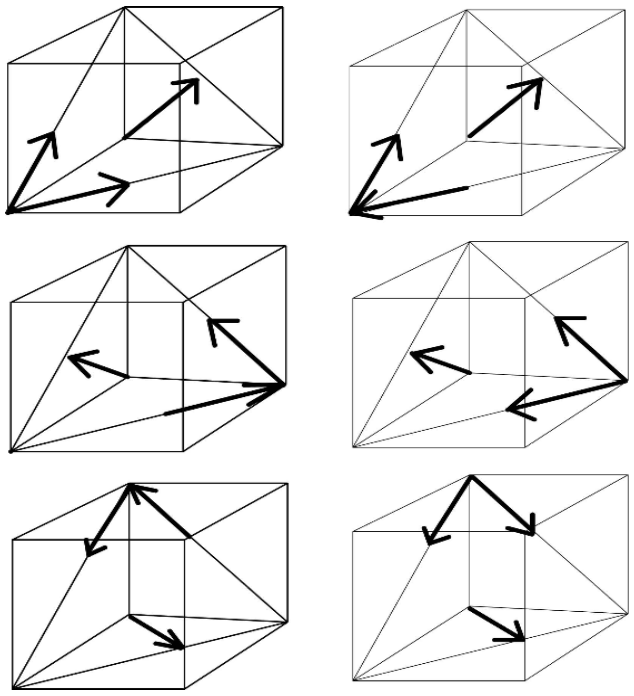


Figure A.2: Options for choosing basis vectors: there are six options since each out-of-plane lattice vectors will have two options for choosing the in-plane vectors.

However, the choice of lattice vector basis should not affect the result of the calculations in principle. For example, I can pick

$$B_1 = \left\{\left(-\frac{1}{2},\frac{1}{2},0\right),\left(-\frac{1}{2},0,\frac{1}{2}\right),\left(0,\frac{1}{2},\frac{1}{2}\right)\right\} \quad \text{(A.2)}$$

$$B_2 = \left\{\left(-\frac{1}{2},\frac{1}{2},0\right),\left(-\frac{1}{2},0,\frac{1}{2}\right),\left(\frac{1}{2},0,\frac{1}{2}\right)\right\} \quad \text{(A.3)}$$

$$B_3 = \left\{\left(-\frac{1}{2},\frac{1}{2},0\right),\left(-\frac{1}{2},0,\frac{1}{2}\right),\left(\frac{1}{2},\frac{1}{2},0\right)\right\} \quad \text{(A.4)}$$

If I choose the basis $B_1 = \left\{\left(-\frac{1}{2},\frac{1}{2},0\right),\left(-\frac{1}{2},0,\frac{1}{2}\right),\left(\frac{1}{2},0,\frac{1}{2}\right)\right\}$, I get the neighbor coordinates in the Table 2 and they are consistent with Figure A.3.

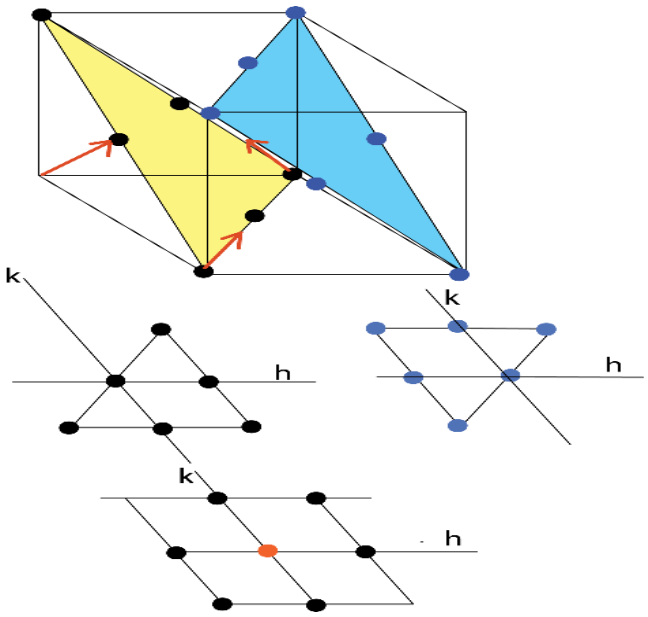


Figure A.3: Lattice vectors for $B_1 = \left\{\left(-\frac{1}{2},\frac{1}{2},0\right),\left(-\frac{1}{2},0,\frac{1}{2}\right),\left(\frac{1}{2},0,\frac{1}{2}\right)\right\}$

Table A.2. Neighbor coordinates for spin at (h, k, l) for basis B_1: FM/AFM/FM trilayer

Layer	Nearest neighbors	Next nearest neighbors
Upper layer (l+1)	(h,k,l+1), (h+1, k, l+1), (h,k-1,l+1)	(h+1,k+1,l+1), (h-1, k-1, l+1), (h+1, k-1, l+1)
In-plane (l)	(h-1, k ,l), (h+1, k ,l), (h, k-1, l),(h-1,k-1,l), (h, k+1,l), (h+1,k+1,l)	None
Lower layer (l-1)	(h, k, l-1), (h-1, k, l-1), (h, k+1, l-1)	(h-1, k+1, l-1), (h+1, k+1, l-1), (h-1, k-1, l-1)

To verify that the choice of this vector does not affect the simulation result, I included both B_1 and B_2 (equations (3.3) and (3.4)) in my calculations. The spin neighbor coordinates corresponding to B_2 are listed in Table 3.

Table A.3. Neighbor coordinates for spin at (h, k, l) for basis B_2: FM/AFM/FM trilayer

Layer	Nearest neighbors	Next nearest neighbors
Upper layer (l+1)	(h,k,l+1), (h+1, k, l+1), (h,k-1,l+1)	(h-1,k-1,l+1), (h+1, k-1, l+1), (h+1, k+1, l+1)
In-plane (l)	(h+1, k ,l), (h-1, k ,l), (h, k-1, l),(h-1,k-1,l), (h, k+1,l), (h+1,k+1,l)	None
Lower layer (l-1)	(h, k, l-1), (h-1, k, l-1), (h, k+1, l-1)	(h+1, k+1, l-1), (h-1, k+1, l-1), (h-1, k-1, l-1)

The method for spin loading and spin neighbor coordinate calculation is included in the next sections. The procedure for the Metropolis algorithm is described in the Appendix B.

A4. MnSe/ZnTe superlattice model

1-dimensional spin array with free boundary

I start with the size of the array, denoted by *i*, which is variable but an odd integer. Then I define the double spin index *n* of the lattice and it marks all possible spin sites on the

cubic lattice. Since the lattice is FCC, only the half of the lattice is occupied with real spins. Therefore, I define another variable N, which is the single spin index $N = \frac{n}{2}$, and it stores the count of the real spins in the lattice. Any double spin site index is written $n = h + ki + li^2$, where (h, k, l) are spin coordinates on the lattice. The actual number of real spins stored in the array including the zero spins on the boundaries is $N_{total} = \frac{n_{total}}{2}$, where n_{total} is the total number of sites in the cubic lattice.

Then I calculated the number of nonzero spins per monolayer $Numperlayer = \frac{(i-5)^2}{2}$ and the number of spins per row $Numperrow = \frac{i-5}{2}$. Spins are loaded according to the following formulas. As each spin is read out of the spin input file, I track the spin count (denoted as count) and calculate n using the value of each input spin count.

$l = (count / Numperlayer) + 2$

$k = (count \% numperlayer) / numperrow + 2$

$h = 2(count \% numperrow + 1) + (l + k) \% 2$

$n_{count} = h + ki + li^2$

The single index of a nonzero spin is $N_{nzspin} = \frac{n_{count}}{2}$

For each N_{count}, the spin values are stored. The size of the spin array is $N_{total} = \frac{1}{2}[(number\ of\ nonzero\ layers + 4]i^2 = \frac{n_{total}}{2}$

To perform the Monte Carlo spin flip, it is necessary to retrieve the spin coordinate information. I choose a random site by multiplying N_{total} by a random number r such that $r \in [0,1]$ then calculate $n_{pick}=2N_{pick}$. i.e., $N_{pick} = rN_{total}$.

Subsequently I calculate

$h_{pick} = 2(n_{pick} \% \text{ numperlayer}+1) + (h_{pick}+k_{pick})\%2$,

$k_{pick} = (n_{pick} \% \text{ numperrow}) +2$,

and $l_{pick} = n_{pick} / \text{numperlayer} + 2$.

For example, if I take a spin lattice with dimension $i = 145$ with the number of nonzero spin monolayers is numnzlayer = 10, there are $numperlayer = \frac{(i-5)^2}{2} = \frac{140^2}{2} = 9800$ spins per monolayer and $\frac{i-5}{2} = \frac{140}{2} = 70$ nonzero spins per each row on the spin monolayer. Each monolayer has 140 rows of 70 nonzero spins. The actual spin array has 14 monolayers since it includes the two empty monolayers on each layer boundary. The spin coordinates are calculated as follows.

l = count / 9800 + 2

k = (count % 9800)/70 + 2

h = 2(count%7 + 1) + (l + k) % 2

n = h + (145)k + (21025)l

The total number of real spin sites is $N_{Total} = \frac{145^2(10+4)}{2}$=147175 and the total number of nonzero spins is 9800 (number of nonzero spins per monolayer).

Each spin has four nearest neighbors and four next nearest neighbors in plane and eight nearest neighbors and four next nearest neighbors out of plane. With the orthogonal spin lattice vectors

B = {(1, 0, 0), (0, 1, 0), (0, 0, 1)}, the spin neighbor coordinates are as follows.

In each Monte Carlo step, I pick a spin from the set of nonzero spins so that I do not have to check if the spin is zero using an "if" statement. I convert the index of the chosen spin into (h, k, l) coordinates.

$N_{pick} = r \times 98000$, where r is a random number such that $r \in [0,1]$.

Then I set $n_{pick} = 2N_{pick}$.

$h_{pick} = 2[(n_{pick} \%numperlayer)+1]+(h+k)\%2$,

$k_{pick} = (n_{pick}\%numperlayer)/numperrow+2$,

$l_{pick} = n_{pick}/numperlayer+2$

The 1-dimensional neighbor coordinates are listed in Table A.4.

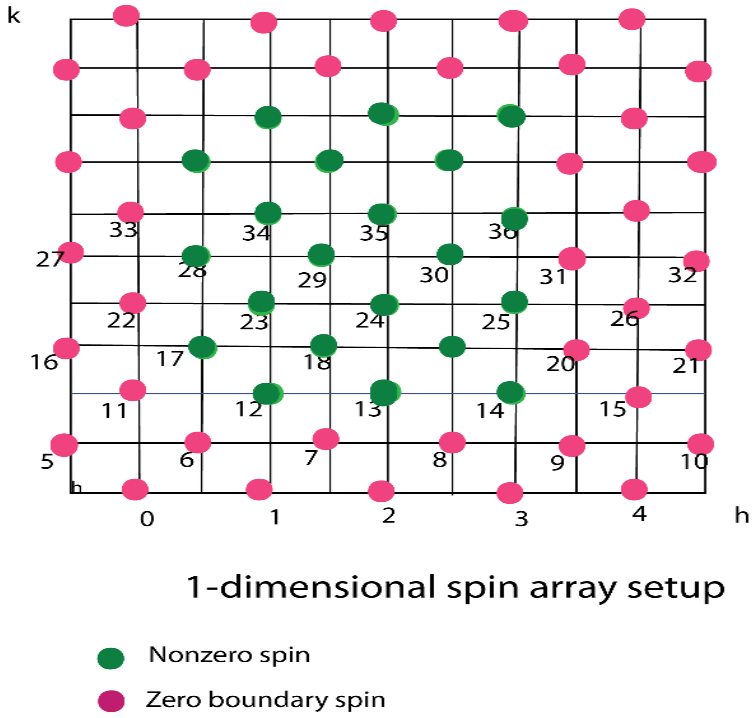


Figure A.4: 1-dimensional spin array with the free boundary

Table A.4. 1D spin array with basis B = {(1, 0, 0), (0, 1, 0), (0, 0, 1)}: for the spin at N

Plane	Nearest neighbors	Next nearest neighbors
Upper plane (l+2)	None	$N + [(h-2)+(k-2)i+(l+2)i^2]/2$
Upper-plane (l+1)	$N+[h+ ki + i^2 (l+2)] / 2$, $N+[(h-1)+ ki+(l+1)i^2]/2$,	None

	$N+[h+ (k-1)i+(l+1)^2]/2$, $N+[(h-1)+ (k-1)i+(l+1)i^2]/2$	
In plane (l)	$N+[(h+1)+ ki+ li^2]/2$, $N +[h+(k+1)i+ li^2]/2$, $N + [(h-1)+ ki+li^2)/2$ $N + [h+(k-1)i+li^2]/2$	$N + [(h+1)+(k+1)i+ li^2]/2$, $N + [(h-1)+(k+1)i+ li^2]/2$, $N + [(h+1)+(k-1)i+li^2]/2$, $N + [(h-1)+(k-1)i+ li^2]/2$
Lower plane (l-1)	$N + [h+ki+(l-1)i^2]/2$, $N + [(h+1)+ ki+(l-1)i^2]/2$, $N + [h+(k+1)i+(l-1)i^2]/2$, $N +[(h+1)+(k+1)i+(l-1)i^2]/2$	None
Lower plane (l-2)	None	$N+ [h+2+(k+2)i+(l-2)i^2]/2$

Periodic spin lattice with 1-dimensional spin array

To compare the system with the free boundary I also set up Monte Carlo calculations on the FCC lattice with the periodic boundary condition. It is reasonable to consider a spin lattice with no spin with incomplete neighbor set. My question in this case would be whether a similar helical spin order will be formed in the periodic lattice and whether the pitch of the spin helix will change with temperature.

I consider a 1-dimensional spin array similar to the case with the free boundary but I identify the neighbor spins on the boundary edges with the ones on the other boundary edge. The spin coordinates have to be modified accordingly. With this formulation there is not supposed to be free boundary spins on the edge, but I kept some "technical" spin sites on the edge since to ensure that the double spin index n is always even.

I load the spins as I track the spin count and use the spin count to figure out the double spin index n in the 1-dimensional array. Note that the size of the spin array is half

of the n_{total}, where n_{total} is the total number of spin sites (both real and technical) in the cubic lattice with dimension i.

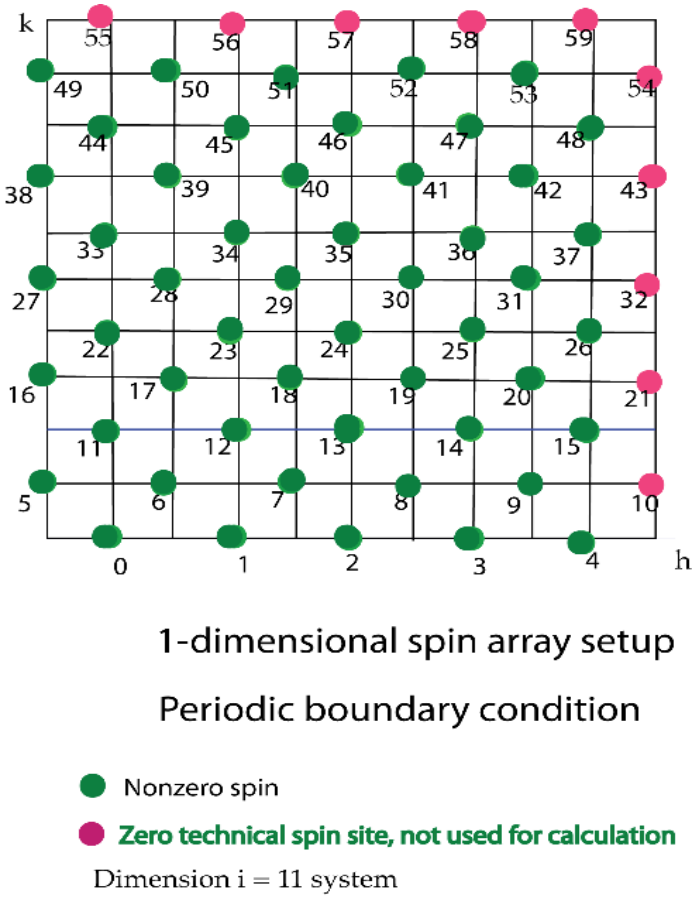


Figure A.5: 1-dimensional spin array with the periodic boundary

The figure above is an example of the spin lattice model I used for the periodic boundary case. In the figure, there are $i \times i = 11 \times 11 = 121$ sites on the cubic sheet, where i is the dimension (size) of the lattice. The formulas for setting the coordinates are similar but now the empty spins are on the upper edge and the right edge only.

The number of spins per monolayer becomes $numperlayer = \frac{(i-1)^2}{2}$.

The number of spins per row is $numperrow = \frac{(i-1)}{2}$.

I load the spin according to the following.

$$l = \frac{count}{numperlayer}$$

$$k = \frac{count\%numperlayer}{numperrow},$$

$$h = 2(count\%numperrow) + (l + k)\%2$$

$$n = h + ki + li^2,$$

$$N = \frac{n}{2}.$$

The spin values are stored for each N. The size of the spin array is $N_{total} = \frac{(numnzlayers)i^2}{2}$ since there is no free boundary layer.

Table A.5. Neighbor coordinates for a spin at N

Plane	Nearest neighbors	Next nearest neighbors
Upper plane (l+2)	None	(h-2, k-2, l+2)
Upper-plane (l+1)	(h,k,(l+1)) (h-1, k, l+1), (h,k-1, l+1), (h-1, k-1, l+1)	None
In plane (l)	(h+1, k, l), (h, k+1, l), (h-1, k , l), (h, k-1, l)	(h+1, k+1, l), (h-1, k+1, l), (h+1, k-1, l), (h-1, k-1, l)
Lower plane (l-1)	(h, k, l-1), (h+1, k, l-1), (h,k+1, l-1), (h+1, k+1, l-1)	None
Lower plane(l-2)	None	(h+2, k+2, l-2)

As in the case with the free boundary, each 3-dimensional coordinates are converted into the 1-dimensional coordinate by the formula

$N_{coordinate} = \frac{h_{coordinate}+k_{coordinate}i+l_{coordinate}i^2}{2}$.

In the Monte Carlo step, I pick a random index N_{pick}, which is the product of number of nonzero spins and r, where r is a random number such that $r \in [0,1]$. Then multiply N_{pick} by 2 to obtain n_{pick} and extract the coordinates ($h_{pick,}$ $k_{pick,}$ l_{pick}) for the picked index using the following formula.

$n_{pick} = 2N_{pick}$

$l_{pick} = \frac{n_{pick}}{numperlayer}$

$k_{pick} = \frac{n_{pick}\%numperlayer}{numperrow}$,

$h_{pick} = 2(n_{pick}\%numperrow) + (l + k)\%2$.

If the picked coordinates are (h_{pick}, k_{pick}, l_{pick}) for instance, the corresponding 1-dimensional coordinate is $N_{pick} = \frac{h_{pick}+ik_{pick}+i^2l_{pick}}{2}$.

3-dimensional spin array with free boundary

To eliminate the possibility of geometrical effects of spin lattice on the simulation result, I included calculations with the basis $B = \left\{\left(\frac{1}{2},\frac{1}{2},0\right),\left(-\frac{1}{2},\frac{1}{2},0\right),\left(0,\frac{1}{2},\frac{1}{2}\right)\right\}$. The system size is defined as $N_{total} = (n_x + 4)(n_y + 4)(n_z + 4)$, where n_x, n_y, n_z are the nonzero spin layer dimensions; N_{total} includes zero spins on the free boundary spins. Since the basis vectors are "slanted", the spin sheets are shifted up as the layers propagate on the growth direction. I store the spins in a 3-dimensional array.

Figures A.6 and A.7 show the spin layer configurations with the basis B.

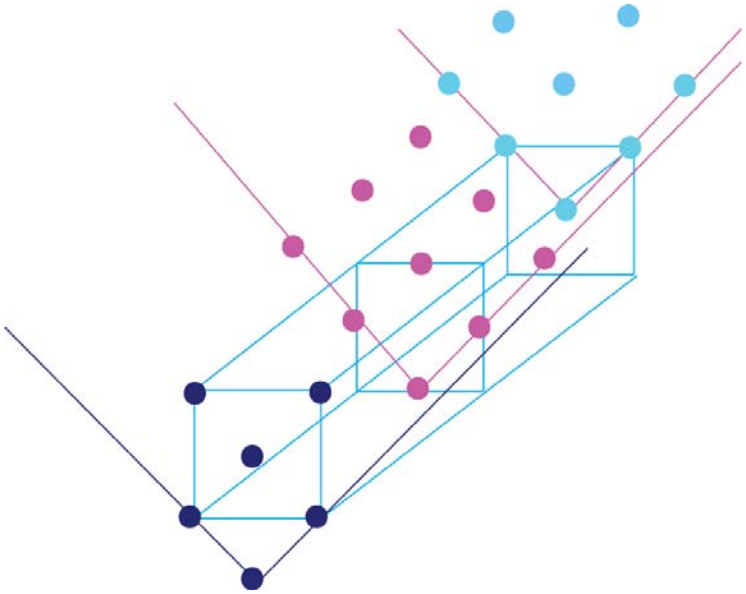


Figure A.6: 3-dimensional spin planes

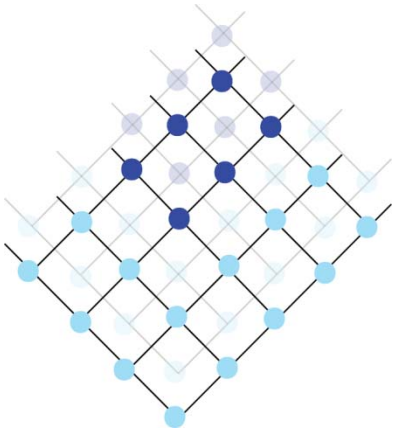


Figure A.7: 3-dimensional spin array with the free boundary

For the system with the free boundary I leave the edge spins empty. The system size parameter n_x, n_y, and n_z are known, so spin loading function uses them. The size of the spin array is $(n_x+4)\times(n_y+4)\times (n_z+4)$, where $n_x\times n_y\times n_z$ is the inner nonzero spin lattice size.

For each input spin count, the spin loading formula becomes as the following.

$h = count \% n_x + 2$

$k = (count \% (n_x n_y)) / n_x + 2$

$l = count / (n_x n_y) + 2$

I add 2 to each coordinate since there are zero boundary spins. In the Monte Carlo spin flip step, I only choose from the pool of nonzero spins to avoid "if" statements.

For the Monte Carlo step, let the picked spin index be N_{pick} and the total number of nonzero spins be $N_{nonzero}$. $N_{nonzero}=n_x n_y n_z$ and $N_{pick} = rN_{nonzero}$, where $r \in [0,1]$ is a nonzero random number. Then the 3-dimensional coordinates are calculated from the formulas below.

$h_{pick} = N_{pick} \% n_x + 2$

$k_{pick} = (N_{pick} \% (n_x n_y)) / n_x + 2$

$l_{pick} = N_{pick} / (n_x n_y) + 2$

For both energy calculation and the Monte Carlo steps, neighbor coordinates have to be configured. Let hup = h+1, hup2 = h+ 2, hdn = h-1, hdn2 = h-2, kup = k+1, kup2 = k + 2, kdn=k-1, kdn2 = k – 2, lup = l+1, lup2 = l + 2, ldn = l-1, and ldn2 = l – 2, then the neighbor coordinates are as in the table A.6.

Table A.6. Neighbor coordinates for a spin at (h, k, l)

Plane	Nearest neighbors	Next nearest neighbors
Upper plane (l+2)	None	(hdn2, kdn2, lup2)
Upper-plane (l+1)	(h, k, lup), (hup, k, lup), (h, kdn, lup), (hdn, kdn, lup)	None
In plane (l)	(hup, k, l),(h, kup, l), (hdn, k ,l), (h, kdn, l)	(hup, kup, l), (hdn, kup, l), (hup, kdn, l),(hdn, kdn, l)
Lower plane (l-1)	(h, k, ldn), (hup, k, ldn), (h,kup,1dn),(hup, kup, ldn)	None

Lower plane(l-2)	None	(hup2, kup2, ldn2)

3-dimensional spin array with periodic boundary

In the case of the periodic boundary I remove all free boundary spins and identify edge spins. The total size of the system is $N_{total} = n_x \times n_y \times n_z$, where n_x, n_y, n_z are lattice dimensions. The diagram below shows the spin lattice.

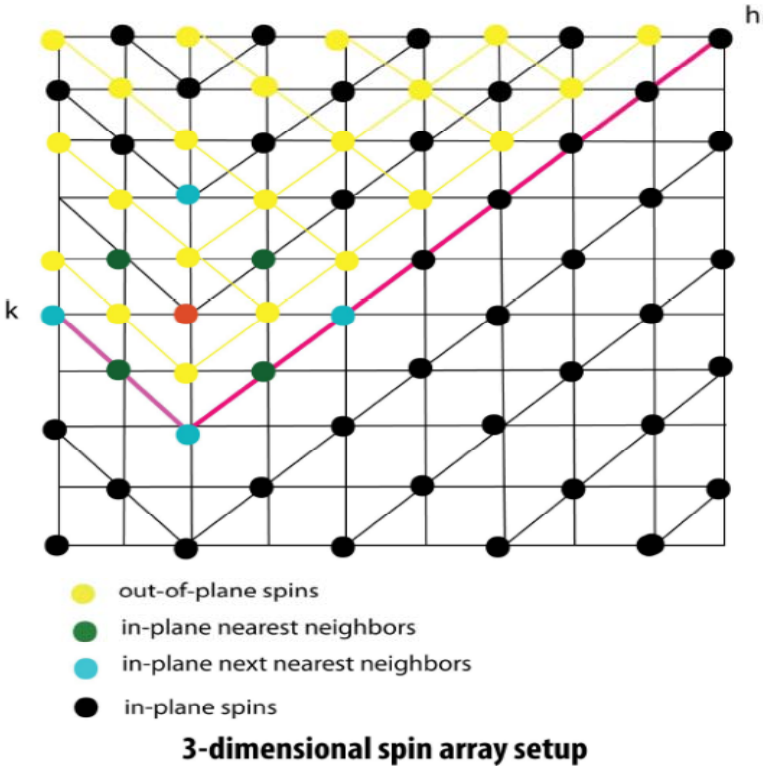


Figure A.8: 3-dimensional spin array with periodic boundary with the basis vector $\boldsymbol{B} = \{\left(\frac{1}{2},\frac{1}{2},0\right),\left(-\frac{1}{2},\frac{1}{2},0\right),(0,\frac{1}{2},\frac{\mathbf{1}}{\mathbf{2}})\}$

Since there is no zero spin, spin loading formulas gets simplified to

h = count % n_x

k = count % ($n_x n_y$) / n_x

l = count / ($n_x n_y$), where count is the loaded spin count.

The neighbor coordinates are identical except that the boundary coordinates will be identified with the spin sited on the other edge using modulo arithmetic.

lup = (l + 1) % n_z, ldn = (l – 1+n_z) % n_z, lup2 = (l + 2) % n_z, ldn2 = (l – 2+n_z) % n_z, kup = (k + 1) % n_y, kdn = (k – 1+n_y) % n_y, kup2 = (k + 2) % n_y, kdn2 = (k – 2+n_y) % n_y, hup = (h + 1) % n_x, hdn = (h – 1+n_x) % n_x ,hup2 = (h + 2) % n_x, hdn2 = (h – 2+n_x) % n_x.

There are other ways to simulate the periodic boundary. One method I considered was to copy the nonzero edge spins on the boundary; the drawback of this method is that I need to update the image of the edge spin whenever I update an edge spin, which will slow down the calculation.

To speed up the calculation, I used a neighbor table; I stored the list of neighbor coordinates so that both energy calculation function and Monte Carlo spin flip function can access it.

Structure factor algorithm

The algorithm for calculating the structure factor is as the following.

(1) Read the spin input file.

(2) Calculate the coordinates $\vec{r_i} = (x_i, y_i, z_i)$ of each spin $\vec{S_i}$ in the FCC lattice.

(3) Set the q-vector $\vec{q} = (q_x, q_y, q_z)$.

(3) Using the q-vector and the spin vector, calculate the vector $\vec{P}$, where $\vec{P}$ is a vector that represents the spin projected on the plane perpendicular to

(4) Take the cross product: $\vec{C} = \vec{P} \times \vec{q}$. Now for each q-vector value, I take the dot product of $\vec{C}$ and the spin $\overrightarrow{S_i}$ to obtain the component of the spin that is perpendicular to the q-vector and to the vertical axis vector, i.e., $\overrightarrow{I_i} = \vec{C} \cdot \overrightarrow{S_i}$. Also I calculate the dot product $\overrightarrow{I_v} = \vec{P} \cdot \overrightarrow{S_i}$ to obtain the component of the spin vector that is in the same direction as the vertical axis vector. Then I sum up the value of the exponential $\overrightarrow{I_i}\exp(2\pi i\vec{q} \cdot \vec{r_i})$ and $\overrightarrow{I_v}\exp(2\pi i\vec{q} \cdot \vec{r_i})$. Computationally I calculate the real part and the imaginary part of the exponential and sum up each of them separately using the Euler formula $e^{i\theta} = cos\theta + isin\theta$.

(5) Sum up the squared imaginary and the squared real part.

(6) Plot the sum as a function of the q-vector

Appendix B. The Metropolis algorithm

The following describes MC calculation procedures with the metropolis algorithm. Before the MC calculation starts, calculate the energy of the spin lattice using the Hamiltonian given for the model. Then perform MC spin flips, each of which is described below.

(1) Pick a random spin site in the lattice.

(2) Generate a random spin value for the picked spin site and calculate the energy difference between the new and old spin configuration.

(3) Check if the energy difference is negative or not.

(4) If the energy difference is negative, accept the new spin value (flip the spin).

(5) If the energy difference is not negative, generate a random number $r \in [0,1], where\ r \in \mathbb{R}$ and compare it to $\exp(\frac{-\Delta E}{kT})$.

(6) If the random number *r* is smaller than $\exp(\frac{-\Delta E}{kT})$, accept the new spin value.

(7) If not, keep the old value of the spin.

Repeat the MC step as many times as necessary to equilibrate the system. In low temperatures, the acceptance rate is generally low so the system evolves quite slowly [65].

Appendix C. Implementation of W & W algorithm: calculation procedures

Only the Monte Carlo step will be affected by changing from the Metropolis algorithm to the W & W algorithm. This algorithm was implemented in a 3-dimensional spin array with a fixed array sizes since variable-sized spin arrays for simplicity; fixing the array sizes significantly speeds up the program. The spin array size used was 10 monolayers of 100×100 spins. Let the total number of spin sites be numsites and the index of a randomly picked spin be ransite. The effective field $\boldsymbol{E_{eff}}$ is defined as the sum of coupling of the spin neighbors. Note that there is no external magnetic field is applied to the system.

$$\boldsymbol{E_{eff}} = \sum_{j \in NN\ in} J_{1in}\boldsymbol{S_j} + \sum_{j' \in NN\ out} J_{1out}\boldsymbol{S_{j'}} + \sum_{k \in NNN\ in} J_{2in}\boldsymbol{S_k} + \sum_{k' \in NNN\ out} J_{1out}\boldsymbol{S_{k'}}$$

(1) Pick a random spin site. ransite = random*numsites

(2) Obtain the effective field $\boldsymbol{E_{eff}}$ for the selected spin site.

(3) Pick a random number r_1 and set $S_z = kTln((r_1 - 1)\exp\left(-\frac{2E_{eff}}{kT}\right) - r_1)$.

(4) Pick a random angle $\varphi \in [0, \pi]$ and set $S_x = \sqrt{1 - {S_z}^2}cos\varphi$ and $S_y = \sqrt{1 - {S_z}^2}sin\varphi$.

(5) Select two vectors $\boldsymbol{J}$ and $\boldsymbol{k}$ that are perpendicular to the $\boldsymbol{E_{eff}}$ by taking the cross products $\boldsymbol{J} = \boldsymbol{E_{eff}} \times \boldsymbol{S_i}$ and $\boldsymbol{k} = \boldsymbol{E_{eff}} \times \boldsymbol{J}$

(6) Project S_x and S_y onto vectors $\boldsymbol{J}$ and $\boldsymbol{k}$.

(7) Update the spin value to (S_x, S_y, S_z).

(8) Repeat.

Appendix D. Comments on the boundary condition

In studying spin lattices with MC methods, the choice of the boundary condition can influence the behavior of the system. For example, the periodic boundary condition eliminates the boundary effect but it suffers the finite size effect [26]. If the size of the lattice is L, the correlation length becomes $\frac{L}{2}$ and the resultant property of the system differs from those of the corresponding infinite lattice [26]. Finite sizes can affect the order of phase transition [26].

A way to reduce finite size effect is to introduce an effective field which acts only on the boundary spins and which is adjusted to keep the magnetization of the boundary

equal to the mean magnetization of the bulk; this method is called the mean field boundary condition [26]. There are other boundary conditions such as screw periodic boundary conditions and anti-periodic boundary condition [26]. To study surface effects, the free boundary condition is a reasonable choice [26].

W.A. Saslow et al. used the self-determined boundary condition, which seems "facilitating" the formation of spin helix since at the boundary a random helical phase was chosen [117]. To study the effect of the thickness of the material one can either increase the size of the lattice and study the effect or fix the spins on the boundaries so that the spins on the layers near the edge are pinned to the fixed spins.

Appendix E. Notes on random number generators

The programs were written using the BOOST random number generator as well as with xorshift random number generator [127, 128]. For the BOOST random number generator, it should be noted that the sequence of random numbers started from the same seed number. Another option for the random number generator was the xorshift random number generator by Georege Marsaglia, which uses a set of five random unsigned integers as the seed numbers [128]. There was no significant difference in the results from both random number generators.

Appendix F. Comments on programming

I used the Matrix library in the C++ programming language to construct variable-sized multi-dimensional spin arrays [129]. Dynamic arrays in C++ allow variable-sized arrays but they are limited to building 1-dimensional arrays [129]. Matrix library allows up to 3-dimensional spin arrays of a variable size [129].

On average 40000 MC spin flips per spin were performed to equilibrate the system and data was taken after performing extra 20000 flips per spin. Note that the equilibration time was not calculated in my projects. To estimate the equilibration time, the time-displaced auto-correlation function is calculated for a variable (i.e., magnetization, energy, and etc [65]. The time-displaced auto-correlation function for magnetization is defined

$$X(t) = \int dt'[m(t') - < m >][m(t'+t) - < m >]$$

$$= \int dt'[m(t')m(t'+t) - < m >^2] \qquad \text{(F. 1)}$$

The auto-correlation function is expected to fall exponentially. $X(t) \sim e^{-t/\tau}$, where τ is defined as the correlation time and a form of discrete sum is used to calculate this quantity in simulations [65].

Also it should be noted that the free energy of the system was not calculated. The challenge is to calculate the entropy of the system. It is also possible to measure entropy

using the Metropolis MC algorithm by using the specific heat *C* by the following formula , which is the equation [65]

$$S(T) = \int_0^T \frac{C}{T} dT \text{ , where } S(T_0=0)=0 \text{ is assumed.} \quad \text{(F. 2)}$$

However, it requires numerical integration techniques that add a significant complexity to the problem [65].

Appendix G. Comments on magnetic interactions

Two magnetic dipoles $\boldsymbol{\mu_1}$ and $\boldsymbol{\mu_2}$ separated by $\mathbf{r}$ have energy equal to

$$E = \frac{\mu_0}{4\pi r^3}\left[\boldsymbol{\mu_1}\cdot\boldsymbol{\mu_2} - \frac{3}{r^2}(\boldsymbol{\mu_1}\cdot\boldsymbol{r})(\boldsymbol{\mu_2}\cdot\boldsymbol{r})\right]. \quad \text{(G.1) [11]}$$

The order of magnitude of this effect for the moments each of $\mu = 1\ \mu_B$ separated by 1 Å is approximately $\frac{\mu^2}{4\pi r^3} \sim 10^{-23}$J which is equivalent to 1 K in temperature [11]. Since many materials are in order at much higher temperatures, the magnetic dipolar interaction is too weak to account for the ordering of magnetic materials [11]. This is associated with the long range order, which is related to the formation of domain walls [11]

Exchange interactions are electrostatic interactions which arise because charges of the same sign cost energy when they are close together and they save energy when they are apart [23]. The Pauli's exclusion principle is behind this [11, 37].

There are also other mechanisms such as direct exchange, superexchange, double exchange, anisotropic exchange interaction (Dzyaloshinsky-Moriya interaction), and RKKY interaction [45, 46, 74, 130].

Appendix H. Units used for the MC calculations

The units used in the calculations are not the standard SI units. Therefore, it is worth showing how the units for MC simulations were configured. A reference unit variable was chosen and the units for the other variables were adjusted to the reference variable. In this case, the reference variable J_{ref} is the exchange constant for the most dominant energy term in the Hamiltonian, which is either J_1 or J_2 depending on which one has a larger magnitude.

Let the reference variable be $|J_{ref}|$. Note that the absolute value is taken for J_{ref} to eliminate the possibility of having a negative temperature unit. The reference exchange constants are $|J_{2,NiO}|$, $|J_2|$, and $|J_1|$ for NiO/CoO, FM/AFM/FM, and MnTe/ZnSe systems respectively. First I set $|J_{ref}| = 1.0$. Then I consider a general form of the Hamiltonian H used for the MC calculations.

$$H = \sum_i \boldsymbol{S_i} \cdot (J_{ref} \sum_{j \epsilon neighbor1} \boldsymbol{S_j} + J_{nonref} \sum_{k \in neighbor2} \boldsymbol{S_k} - \mathbf{B}), \qquad \text{(H. 1)}$$

where J_{ref} and J_{nonref} are reference and non-reference exchange constants respectively. The unit for any Hamiltonian is energy. Let the unit of a spin be $[S]$ and the unit of energy $[H]$, then the unit for the exchange constants J_{ref} and J_{nonref} is $\frac{[H]}{[S]^2}$. Then the unit for the Hamiltonian is $[H] = |J_{ref}|[S]^2$.

The magnetic interaction term is included to derive the unit for the magnetic field **B**. The magnetic interaction energy (Zeeman energy) E_{mag} for an electron is defined

$$E_{mag} = -\mu_B \mathbf{m} \cdot \mathbf{B}, \tag{H.2}$$

where **B**, **m**, and μ_B are magnetic field, magnetic moment of the material, and the Bohr magneton respectively [131]. However, since our system is composed of magnetic ions, μ_B has to be replaced by the gyromagnetic ratio μ_G for a magnetic ion which is not an electron. The gyromagnetic ratio of an atom or an ion $\mu_G = g\mu_B$, where g is the g-factor for the particle of interest [131].

In other words, magnetic interaction energy for a magnetic ion is $E_{mag}{}'$, where $E'_{mag} = -\mathbf{m} \cdot \mathbf{B} = -\mu_G \sum_i \boldsymbol{S_i} \cdot \mathbf{B}$. The unit of the magnetic field is

$$[\boldsymbol{B}] = \frac{[E_{mag}{}']}{\mu_G[S]} = \frac{[H]}{\mu_G[S]} = \frac{J_{ref}[S]^2}{\mu_G[S]} = \frac{J_{ref}[S]}{\mu_G}. \tag{H.3}$$

The unit for the reduced temperature comes from the following equation.

$$[H] = k_B[\mathrm{T}] = J_{ref}[S]^2 \tag{H. 4}$$

Rearranging the terms gives the unit of temperature $[T] = \frac{J_{ref}[S]^2}{k_B}$.

To obtain the unit of magnetic torque, I use the equation $\boldsymbol{\tau} = \mathbf{m} \times \mathbf{B}$, the magnetic moment has the unit of spin. $[\boldsymbol{\tau}] = [\mathbf{m}][\mathbf{B}] = \frac{J_{ref}\mu_G[S]^2}{\mu_G} = J_{ref}[S]^2$.

I assume that the spin is normalized (i.e., normalized) for the MC calculations. Therefore, the unit of the spin [*S*] = 1. If I substitute [*S*] with unity, I obtain the reduced units for magnetic field, magnetic torque, and temperature.

$$[\mathrm{T}] = \frac{J_{ref}}{k_B}$$

$$[\boldsymbol{\tau}] = J_{ref}$$

$$[\mathbf{B}] = \frac{J_{ref}}{\mu_G},$$

where k_B is the Boltzmann constant and μ_G is the gyromagnetic ratio of the magnetic ion of interest. For the simulation of NiO/CoO and FM/AFM/FM superlattices, the reference variable $|J_{ref}| = |J_2| = J_2$, where $J_2 > 0$. For the MnSe/ZnTe system, $|J_{ref}| = |J_1| = J_1$, since $J_1 > 0$.

Appendix I. Bibliography

1. J.A. Borchers, M. J. Carey, R. W. Erwin, C. F. Majkrazak, and A. E. Berkowitz, Spatially modulated antiferromagnetic order in CoO/NiO superlattices, Phys. Rev. Lett. 70, 1878 (1993)

2. T.M.Giebultowicz, N. Samarth, H. Luo, J. K. Furdyna, and P. Kłosowski, Strain-engineered incommensurability in epitaxial Heisenberg antiferromagnets, Phys Rev B, 46, 12076 (1992)

3. Ulrich Nowak, "Handbook of Magnetism and Advanced magnetic materials", John Wiley & Sons (2007)

4. P. Grünberg, R. Schreiber, P. Yang, M. B. Brodsky, H. Sowers, Layered magnetic structures: Evidence for antiferromagnetic coupling of Fe Layers across Cr interlayers, Phys. Rev. Lett. 57, 2442 (1986)

5. G. Binasch, P. Grünberg, F. Saurenbach, and W. Zinn, Enhanced magnetoresistance in layered magnetic structures with antiferromagnetic interlayer exchange, Phys. Rev. B 39, 4828 (1989)

6. M. N. Baibich, J. M. Broto, A. Fert, F. Nguyen Van Dau, and F. Petroff, Giant Magnetoresistance of (001) Fe/(001) Cr Magnetic Superlattices Phys. Rev. Lett. 61, 2472 (1988)

7. S. S. P. Parkin, N. More, and K. P. Roche, Oscillations in exchange coupling and magnetoresistance in metallic superlattice structures, Phys. Rev. Lett. 64, 2304 (1990)

8. M. T. Johnson, S. T. Purcell, N. W. E. McGee, R. Coehoorn, J. aan de Stegge, and W. Hoving, Structural Dependence of the oscillatory interaction across Cu layers, Phys. Rev. Lett. 68, 2688 (1992)

9. S. T. Purcell, M. T. Johnson, N. W. E. McGee, R. Coehoorn, and W. Hoving, Two monolayer oscillations in the antiferromagnetic exchange coupling through Mn in Fe/Mn/Fe sandwich structures, Phys. Rev. B 45, 13064 (1992)

10. S.A. Wolf, D. D. Awschalom, R. A. Buhrman, J. M. Daughton, S. von Molnár, M. L. Roukes, A. Y. Chtchelkanova, D. M. Treger, Spintronics: a spin-based electronics vision for the future, Science 294, 1488 (2001)

11. D. L. Mills and J.A.C. Bland, R. E. Camley eds., "Nanomagnetism", Contemporary concepts of condensed Matter science, Elsevier (2006)

12. A. D. Giddings, T. Jungwirth, anad B. L. Gallagher, (Ga,Mn)As based superlattices and the search for antiferromagnetic interlayer coupling, Phys. Rev. B 78, 165312 (2008)

13. P. Bruno and C. Chappert, Ruderman-Kittel theory of oscillatory: interlayer exchange coupling, Phys. Rev. B 46, 261 (1992)

14. Barbara A. Jones and C. B. Hanna, Contribution of quantum-well states to the RKKY coupling in magnetic multilayers, Phys. Rev.Lett. 71, 4253 (1993)

15. Karol Szałoski, Tadeusz Balcerzak, Antiferromagnetic interlayer coupling in diluted magnetic thin films with RKKY interaction, Phys. Rev. B 79, 214430 (2009)

16. Stephen Blundell, "Magnetism in condensed matter", Oxford University Press (2001)

17. Albert Fert, Nobel Lecture: origin, development, and future of spintronics, Rev. Mod. Phys. 80, 1517 (2008)

18. J. Blinowski and P Kacman, Interlayer exchange coupling mediated by valence-band electrons, Phys. Rev. B 64, 145302 (2001)

19. H. Kępa, C. F. Majkrzak, A. Sipatov, A. G. Fedorov, T. A. Samburskaya, and T. M. Giebultowicz, Interlayer coupling in EuS/SrS, EuS/PbSe and EuS/PbTe magnetic semiconductor superlattice, J. Phys.: Condens. Matter 21, 124207 (2009)

20. H. Kępa, J. Kutner-Pielaszek, J. Blinowski, A. Twardowski, C. F. Majkrzak, T. Story, P. Kacman, R. R. Galązka, K. Ha, H. J. M. Swagten, W. J. M. De Jonge, A. Yu. Sipatov, V. Volotbuev, and T. M. Giebultowicz, Antiferromagnetic interlayer coupling in ferromagnetic semiconductor EuS/PbS(001) superlattices, Europhys. Lett. 56, 54 (2001); for review, see T.M. Giebultowicz and H. Kępa, "Introduction to the physics of diluted magnetic semiconductors", Jan A. Gaj and Jacek Kossut eds., Springer (2010) Chap. 12

21. T.M. Giebultowicz, H. Kępa, J. Bilnowski, and P. Kacman, Neutron diffraction and reflectivity studies of interlayer correlations in magnetic semiconductor superlattices, Physica E 10, 411 (2001)

22. H. Kępa and T. M. Giebultowicz, Studies of interlayer magnetic coupling in all-semiconductor superlattices by means of neutron scattering techniques, Acta Physica Polonica A 102, 21 (2002)

23. L. Kowalczyk, S. Wrotek, P. Dziawa, V. Osinniy, M. Szot, T. Story, A. Yu. Sipatov, V. V. Volobuev, Interlayer exchange coupling in semiconductor EuS-PbS ferromagnetic wedge multilayers, Acta Physica Polonica 110, 225 (2006)

24. P. Sankowki and P. Kacman, Interlayer exchange coupling in (Ga, Mn)As-based superlattices, Phys. Rev. B 71, 201303 (2005)

25. J.C. Slonczewski, Conductance and exchange coupling of two ferromagnets separated by a tunneling barrier, Phys. Rev. B 39, 6995 (1989)

26. J. S. Moodera, Lisa R. Kinder, Terrilyn M. Wong, and R. Meservey, Large magnetoresistance at room temperature in ferromagnetic thin film tunnel junctions, Phys. Rev. Lett. 74, 3273 (1995)

27. M. Jullière, Tunneling between ferromagnetic Films, Phys. Lett. 54A, 225 (1975)

28. T. Katayama, S. Yuasa, J. Velev, M. Ye. Zhuravlev, S. S. Jaswal, and E. Y. Tsymbal, Interlayer exchange coupling in Fe/MgO/Fe magnetic tunnel junctions, Appl. Phys. Lett. 89, 112503 (2006)

29. S. Yuasa, T. Nagahama, A. Fukushima, Y. Suzuki, and K. Ando, Giant room-temperature magnetoresistance in single-crystal Fe/MgO/Fe magnetic tunnel junctions, Nature Materials 3, 868 (2004)

30. I. I. Oleinik, E. Y. Tsymbal, D. G. Pettifor, Structural and electronic properties of $Co/Al_2O_3/Co$ magnetic tunnel junction from first principles, Phys. Rev. B 62, 3952 (2000)

31. Mikhail Feygenson, Yuen Yiu, Angela Kou, Ki-Sub Kim, and Meigan C. Aronson, Controlling the exchange bias field in Co core/CoO shell nanoparticles, Phys. Rev. B 81, 195445 (2010)

32. S. Giri, M Patra and S Majumdar, Exchange bias effect in alloys and compounds, *J. Phys.: Condens. Matter* 23, 073201 (2011)

33. J. C. Slonczewski, Overview of interlayer exchange theory, J. Mag. and Mag. Mater. 150, 13 (1995)

34. A. Oleś, F. Kajzar, M. Kucab, and W. Sikora, "Magnetic Structures determined by neutron diffraction", Państwowe Wydawnictwo Nakowe (PWN) (1976)

35. Hikaru Kawamura, Universality of phase transitions of frustrated antiferromagnets, J. Phys.: Condens. Matter 10, 4707 (2008)

36. V. Hemmati, M. L. Plumer, and J. P. Whitehead, and B. W. Southern, Monte Carlo simulations of magnetic ordering in the fcc kagome lattice, Phys. Rev. B 86, 104419 (2012)

37. C.G. Schull and J.S. Smart, Detection of antiferromagnetism by Neutron Diffraction, Phys. Rev. 76, 1256 (1949)

38. S. Smart "Effective field theories of magnetism", W.B. Saunders Company (1966)

39. Charles F. Squire, Antiferromagnetism in some manganese compounds, Phys. Rev. 56, 922 (1939)

40. Jerzy Kocinsky, The incommensurate helical spin structure in MnSe/ZnTe superlattice, IEEE transactions in Magnetics 29, 3075 (1993)

41. M.V. Gvozdikova and M. E. Zhitomirsky, A Monte Carlo study of the first-order transition in a Heisenberg FCC antiferromagnet, JETP Letters 81 (2005)

42. N. Samarth, P. Kłosowski, H. Luo, T. M. Giebultowicz, and J. K. Furdyna, J. J. Rhyne, B. E. Larson, N. Otsuka, Antiferromagnetism in ZnSe/MnSe strained-layer superlattice, Phys. Rev. B 44, 4701 (1991)

43. T. M. Giebultowicz, P. P. Kłosowski, N. Samarth, H. Luo, and J. K. Furdyna, Neutron diffraction studies of zinc-blende MnTe epitaxial films and MnTe/ZnTe superlattices: the effect of strain and dilution on a strongly frustrated fcc antiferromagnet, Phys. Rev. B 48, 12817 (1993)

44. P. W. Anderson, An approximate quantum theory of the antiferromagnetic ground state, Phys. Rev. 86, 694 (1952)

45. P. W. Anderson, Antiferromagnetism. theory of superexchange interaction, Phys. Rev. 79, 350 (1950)

46. P. W. Anderson, New approach to the theory of superexchange interactions, Phys. Rev. 115, 2 (1959)

47. Lester Corliss, Norman Elliott, and Julius Hastings, Magnetic structure of the polymorphic forms of manganese sulfide, Phys. Rev. 104, 924 (1956)

48. J. M. Hastings L. M. Corliss, and W. Kunnmann, Characterization of the magnetic phase transition in cubic β-MnS, Phys. Rev. B 24, 1388 (1981)

49. T. M. Giebultowicz and J. K. Furdyna, Monte Carlo simulation of fcc Heisenberg antiferromagnet with nearest- and next-nearest-neighbor interactions, J. Appl. Phys. 57, 3312 (1985)

50. D. Vaknin, E. Caignol, P. K. Davies, and J. E. Fischer, Antiferromagnetism in ($Ca_{0.85}Sr_{0.15}$) CuO_2, the parent of the cuprate family of superconducting compounds, Phys. Rev. B 39, 9122 (1989)

51. Andrey Chubakov, First-order transition in frustrated quantum antiferromagnets, Phys. Rev. B 44, 392 (1991)

52. N. P. Armitage, F. Ronning, D. H. Lu, C. Kim, A. Damascelli, K. M. Shen, D. L. Feng, H. Eisaki, Z.-X. Shen, P. K. Mang, N. Kaneko, M. Greven, Y. Onose, Y. Taguchi, and Y. Tokura, Doping dependence of an *n*-type cuprate superconductor investigated by angle-resolved photoemission spectroscopy, Phys. Rev. Lett 88, 257001 (2002)

53. Andreas Hackl and Matthias Vojta, Nernst-effect anisotropy as a sensitive probe of Fermi-surface distortions from electron-nematic order, Phys. Rev. B 80, 220514 (R) (2009)

54. K. Ishida, Y. Kitaoka, Y. Tokunaga, S. Matsumoto, and K. Asayama, M. Azuma, Z. Hiroi, and M. Takano, Spin correlation and spin gap in quasi-one-dimensional spin-1/2 cuprate oxides: A ^{63}Cu NMR study, Phys. Rev. B 53, 2827 (1996)

55. P. Kłosowski, T. M. Giebultowicz, J. J. Rhyne,N. Samarth, H. Luo, and J. K. Furdyna, Antiferrqmagnetism in epilayers and superlattices containing zincblende MnSe and MnTe, J. of Appl. Phys. 70, 6221 (1991)

56. P. W. Anderson, Generalization of the Weiss molecular field theory of antiferromagnetism, Phys. Rev. 79, 705 (1950)

57. J. Samuel Smart, Molecular field treatment of ferromagnetism and antiferromagnetism, Phys. Rev. 86, 968 (1952)

58. J.A. Borchers, R.W. Erwin, S.D. Berry, D. M. Lind, J. F. Ankner, E. Lochner, K. A. Shaw, and D. Hilton, Long-range magnetic order in Fe_3O_4/NiO superlattices, Phys. Rev. B 51, 8276 (1995)

59. K. Lenz, S. Zander, and W. Kuch, Magnetic proximity effects in antiferromagnet / ferromagnet bilayers: the impact on the Néel temperature, Phys. Rev. Lett. 98, 237201 (2007)

60. P. V. van der Zaag, Y. Ijiri, J. A. Borchers, L. F. Feiner, R. M. Wolf, J. M. Gaines, R. W. Erwin, and M. A. Verheijen, Difference between blocking and Néel temperatures in the exchange biased Fe_3O_4/CoO system, Phys. Rev. Lett. 84,6102 (2000)

61. Lamberto Duò, Marco Finazzi, and Franco Ciccacci eds., "Magnetic Properties of Antiferromagnetic Oxide Materials: Surfaces, Interfaces, and Thin Films" Wiley-VCH (2010)

62. W. Minor and T. M. Giebultowicz, Studies of FCC Heisenberg antiferromagnets by Monte Carlo simulation on large spin arrays, J. De Physique, Colloques 49 C8-1551 (1988)

63. W. L. Roth, Neutron and optical studies of Domains in NiO, J. of Appl. Phys. 31, 2000 (1960)

64. F. U. Hillebrecht, H. Ohldag, N. B. Weber, C. Bethke, and U. Mick, Magnetic moments at the surface of antiferromagnetic NiO(100),Phys. Rev. Lett. 86, 3419 (2001)

65. M. E. J. Newman and G. T. Barkima, "Monte Carlo methods in statistical physics", Oxford University Press (1999)

66. David P. Landau and Kurt Binder, "A guide to Monte Carlo simulations in statistical physics", 2nd edition, Cambridge University Press (2000)

67. D. Harrmann-Ronzaud, P. Burlet , and J. Rossat-Mignod, Equivalent type II magnetic structures: CoO, a collinear antiferromagnet, J. Phys. C: Solid State Physics 11, 2123 (1978)

68. M. J. Longfield, J. A. Paixão N. Bernhoeft, and G. H. Lander, Resonant x-ray scattering from multi-k magnetic structures, Phys. Rev. B 66, 054417 (2002)

69. E. N. Abarra, K. Takano, F. Hellman, and A. E. Berkowitz, Thermodynamic measurements of magnetic ordering in antiferromagnetic superlattices, Phys. Rev. B 77, 3451 (1996)

70. Fausto Fiorillo, "Measurement and characterization of magnetic materials", Academic Press Elsevier (2004)

71. Sarbeswar Sahoo, Srinivas Polisetty, Yi Wang, Tathagata Mukherjee, Xi He, Sitaram S Jaswal, and Christian Binek, Asymmetric magnetoresistance in an exchange bias Co/CoO bilayer, J. Phys. Condens. Matter 24, 096002 (2012)

72. W.H. Miklejohn and C. P. Bean, New anisotropy, Phys. Rev. 102, 1413 (1956)

73. J. Salafranca, M. J. Calderón, and L. Brey, Magnetoresistance of an all-manganite spin valve: a thin antiferromagnetic insulator sandwiched between two ferromagnetic metallic electrodes, Phys. Rev. B 77, 014441 (2008)

74. Charles Kittel, Physical theory of ferromagnetic domains, Rev. Mod. Phys. 21, 541 (1949)

75. Shoji Ikeda, Jun Hayakawa, Young Min Lee, Fumihiro Matsukura, Yuzo Ohno, Takahiro Hanyu, and Hideo Ohno, Magnetic tunnel junctions for spintronic memories and beyond, IEEE Trans. Elec. Dev. 54, 991 (2007)

76. Y. Ijiri, J.A. Borchers, R. W. Erwin, and S.-H. Lee, Role of the antiferromagnet in exchange-biased Fe_3O_4/CoO superlattices, J. of Appl. Phys. 83, 6882 (1998)

77. S. Couet, K. Schlage, Th. Diederich, R. Rüffer, K. Theis-Bröhl, B. P. Toperverg, K. Zhernenkov, H. Zabel, and R. Röhlsberger, The magnetic structure of coupled Fe/CoO multilayers revealed by nuclear resonant and neutron scattering methods, New J. of Phys. 11, 013038 (2009)

78. Sato, Takashi et al., Orientational dependence of exchange anisotropy of Mn-Ir/Co-Fe epitaxial bilayers, J. of Applied Phys. 95, 7513-7515 (2004)

79. R. Morales, Zhi-Pan Li, J. Olamit, Kai Liu, J. M. Alameda, and Ivan K. Schuller, Role of the antiferromagnetic bulk spin structure on exchange bias, Phys. Rev. Lett. 102, 097201 (2009)

80. M. R. Fitzsimmons, C. Dufour, K. Dumesnil, Jian Dou, and Michael Pechan, Mechanisms of exchange bias in $DuFe_2/YFe_2$ exchange-coupled superlattices, Phys. Rev. B 79, 144425 (2009)

81. P. J. Jensen, H. Dreyssé, and M. Kiwi, Magnetic reordering in the vicinity of a ferromagnetic/antiferromagnetic interface, Eur. Phys. J. B 46, 541 (2005)

82. J. A. Borchers, Y. Ijiri, S.-H. Lee, C. F. Majkrzak, and G. P. Felcher, K. Takano, R. H. Kodama, and A. E. Berkowitz, Spin-flop tendencies in exchange-biased Co/CoO thin films, J. of Appl. Phys. 83, 7219 (1998)

83. Miguel Kiwi, Origin of the magnetic proximity effect, Mat. Res. Soc. Symp. Proc. 746, Q5.2.1 (2003)

84. A. P. Malozemoff, Mechanisms of exchange anisotropy, J. Appl. Phys**. 63**, 3874 (1988)

85. Shan-Ho Tsai, D. P. Landau, Thomas C. Schulthess, Effect of interfacial coupling on the magnetic ordering in ferro-antiferromagnetic bilayers, J. of Applied Physics, 93, 8612 (2003)

86. T. Giebultowicz, B. Lebech, B. Buras, W. Minor, H. Kepa, R. R, Galazka, Neutron scattering studies of $Cd_{1-x}Mn_x$ Te, J. Appl. Phys. 55, 2305 (1984)

87. J. M. D. Coey, Jerome T. Mlack, M. Venkatesan, and P. Stamenov, Magnetization process in dilute magnetic oxides, IEEE Trans. On Mag. 46, 2501 (2010)

88. Hong, Nguyen Hoa et al., Role of defects in tuning ferromagnetism in diluted magnetic oxide films, Phys. Rev. B 72, 045336 (2005)

89. A. Brambilla, P. Sessi, M. Cantoni, M. Finazzi, N. Rougemaille, R. Belkhou, P. Vavassori, L. Duò, and F. Ciccacci, Frustration-driven micromagnetic structure in Fe/CoO/Fe thin film layered systems, Phys. Rev. B 79, 172401 (2009)

90. Ki-Suk Lee, Young-Sang Yu, and Sang-Kook Kim, An interface-proximity model for switchable interfacial uncompensated antiferromagnetic spins and their role in exchange bias, Appl. Phys. Lett. 86, 192512 (2005)

91. J. Wu, J. Choi, A. Scholl, A. Doran, E. Arenholz, Y. Z. Wu, C. Won, Chanyong Hwang, and Z. Q. Qiu, Element-specific study of the anormalous magnetic interlayer coupling across NiO Spacer layer in Co/NiO/Fe/Ag(001) using XMCD and XMLD, Phys. Rev. B 80, 012409 (2009)

92. A. I. Morosov and I. A. Morosov, Magnetic phase diagram of spin-valve structure with an antiferromagnetic oxide layer, Physics of the solid state 53, 59 (2011)

93. Christian Binek, Xi He, and Srinivas Polisetty, Temperature dependence of the training effect in a Co/CoO exchange-bias layer, Phys. Rev. B 72, 054408 (2005)

94. D. T. Pierce, A. D. Davies, J. A. Stroscio, D. A. Tulchinsky, J. Unguris, and R. J. Celotta, Non-collinear exchange coupling in Fe/Mn/Fe(001) insight from scanning tunneling microscopy, J. of Mag. and Mag. Mat. 222, 13 (2000)

95. R. Bali, B. B. Nelson-Cheeseman, A. Scholl, E. Arenholz, Y. Suzuki, and M. G. Blamire, Competing magnetic anisotropies in antiferromagnet-ferromagnet-antiferromagnet trilayer, J. Appl. Phys. 106, 113925 (2009)

96. S. Mangin, C. Bellouard, and H. Fritzsche, Observation of a well characterized 180° domain wall by polarized neutron reflectometry, Physica B 276, 558 (2000)

97. J.C. Slonczewski, Origin of magnetic anisotropy of Cobalt-substituted magnetite, Phys. Rev. 110, 1341 (1958)

98. H. J. F. Jansen, Magnetic anisotropy in density-functional theory, Phys. Rev. B 59, 4699 (1999)

99. Andrew Putnis, "Introduction to mineral science", Cambridge University Press (1992)

100. Ding-sheng Wang, Ruqian Wu, and A. J. Freeman, First-principles theory of surface magnetocrystalline anisotropy and the diatomic-pair model, Phys. Rev. B 47, 14932 (1993)

101. Sunghyun Yoon and Kannan M. Krishnan, Temperature dependence of magnetic anisotropy constant in manganese ferrite nanoparticles at low temperature, J. Appl. Phys. 109, 07B534 (2011)

102. M. T. Johnson, P. J. H. Bloemen, F. J. A. den Broedery and J. J. de Vries, Magnetic anisotropy in metallic multilayers, Rep. Prog. Phys. 59 ,1409 (1996)

103. D. H. Wei, X. Y. Xu, L. F. Yin, G. S. Dong, and X. F. Jin, Oscillatory magnetic anisotropy tuned by interlayer coupling, Phys. Rev. B 80, 092403 (2009)

104. Alexandre Tamion, Edgar Bonet, Florent Tournus, Cécile Raufast, Arnaud Hillion, Oksana Gaier, and Véronique Dupuis, Efficient hysteresis loop simulations of nanoparticle assemblies beyond the uniaxial anisotropy, Phys. Rev. B 85, 134430 (2012)

105. E. C. Stoner and E. P. Wohlfarth, A mechanism of magnetic hysteresis in heterogeneous alloys, E P, Phil. Trans. Roy. Soc. A240, 599 (1948)

106. Wei Zhang and Kannan M. Krishnan, Spin-flop coupling and rearrangement of bulk antiferromagnetic spins in epitaxial exchange-biased Fe/MnPd/Fe/IrMn multilayers, Phys. Rev. B 86, 054415 (2012)

107. A. N. Dobrynin and D. Givord, Exchange bias in a Co/CoO/Co trilayer with two different ferromagnetic-antiferromagnetic interfaces, Phys. Rev. B 85, 014413 (2012)

108. T. M. Gibultowicz et al., Neutron diffraction in $Co_pMg_{1-p}O$ solid solutions, J. de Physique Colloques 49 C8-1105 (1988)

109. T. M. Giebultowicz, P. Klosowski, N. Samarth, and H. Luo, Antiferromagnetic phase transition in $Cd_{1-x}Mn_xSe$ epilayers, Phys. Rev. B 42, 2582 (1990)

110. Jung-il Hong, Titus Leo, David J. Smith, and Ami E. Berkowitz, Enhancing exchange bias with diluted antiferromagnets, Phys. Rev. Lett. 96, 117204 (2006)

111. A. P. Malozemoff, Random field model of exchange anisotropy at rough ferromagnetic-antiferromagnetic interfaces, Phys. Rev. B 35, 3679 (1987)

112. Xiaozhi Zhan, Zhongquan Mao, Xiao Xu, and Xi Chen, Spin disorder dependence of the exchange bias effect, Phys. Rev. B 86, 020407 (R) (2012)

113. R. F. L. Evans, D. Bate, and R. W. Chantrell, Influence of interfacial roughness on exchange bias in core-shell nanoparticles, Phys. Rev. B 84, 092404 (2011)

114. C. Fleischmann, F. Almeida, J. Demeter, K. Paredis, A. Teichert, R. Steitz, S. Brems, B. Opperdoes, C. Van Haesendonck, A. Vantomme, and K. Temst, The influence of interface roughness on the magnetic properties of exchange biased CoO/Fe films, J. of Appl. Phys. 107, 113907 (2010)

115. T. M. Giebultowicz, Hong Luo, Nitin Samarth, J. K. Furdyna, and J. J. Rhyne, Strain-engineered magnetic phenomena in MnSe/ZnSe, MnTe/ZnTe, and MnSe/ZnTe superlattices, IEEE Trans. Mag. 29,3382 (1993)

116. Zachary Q. Wiren, Ph.D. Thesis, Oregon State University (2008)

117. M. Collins and W. M. Saslow, Temperature-dependent pitch and phase diagram for incommensurate *XY* spins in a slab geometry, Phys. Rev. B 53, 8533-8538 (1996)

118. M. Kolb, Symmetry and boundary condition of planar spin systems, Phys. Rev. B 31, 7494 (1985)

119. Trinanjan Datta and Dao-Xin Yao, Effects of magnetic field, anisotropy, and biquadratic interactions in type IIA fcc antiferromagnets studied by linear spin-wave theory, Phys. Rev. B 85, 054409 (2012)

120. T.M. Giebultowicz, "Notes on incommensurate spin order", unpublished

121. Izyyumov et al., “Magnetic Neutron Diffraction”, Plenum Press (1970)

122. G. E. Bacon “Neutron Diffraction”, 3rd ed. Oxford University Press (1976)

123. N.D. Mermin and H. Wagner, Absence of ferromagnetism or antiferromagnetism in one- or two-dimensional isotropic Heisenberg models, Phys. Rev. Lett. 17, 1133 (1966)

124. M. Manojlović, M. Pavkov, M. Škrinjar, M. Pantić, D. Kapor, and S. Stojanović, Mermin-Wagner theorem analogous treatment of the long-range order in La_2 CuO_4-type compound spin models, Phys. Rev. B 71, 132510 (2005)

125. L. R. Walker and R. E. Walstedt, Numerical simulation of spin-glass transition phenomena, J. Appl. Phys. 53, 7985 (1982)

126. I. C. Infante, J. O. Ossó, F. Sánchez, and J. Fontcuberta, Tuning in-plane magnetic anisotropy in (110) $La_{2/3}Ca_{1/3}MnO_3$ films by anisotropic strain relaxation, Appl. Phys. Lett. 92, 012508 (2008)

127. BOOST random number generator http:// www.boost.org/

128. George Marsaglia, Random number generators, J. of Mod. Appl. Stat. Software 8, Issue 14, Jul 2003

129. Bjarne Stroustrup, “Programming: principles and Practice Using C++”, Addison-Wesley (2009)

130. Tôru Moriya, Anisotropic superexchange interaction and weak ferromagnetism, Phys. Rev. 120, 91 (1960)

131. Fujia Yang and Joseph H. Hamilton ,“Modern atomic and nuclear physics”, World Scientific (2010)

Appendix J. Some programs used in the project

J1. MC program for diluted AFM lattice

```
#include "Matrix.h"
#include "MatrixIO.h"
#include <iostream>
#include <fstream>
#include <math.h>
#include <iomanip>
#include <sstream>
#include <string>

// Diluted layer so that I can use this for a bulk simulations

// xorshift RNG -renamed from niocoo_h.cpp on April 8, 2012

            /*
             The basis vectors h=(-0.5, 0.5, 0),k=(-0.5, 0, 0.5),l=(0,
0.5, 0.5)


                        j1[0][nz]: upper layer neighbor
                        j1[1][nz]: same layer neighbor
                        j1[2][nz]: lower layer neighbor

                        j2[0][nz]: upper layer neighbor
                        j2[1][nz]: same layer neighbor
                        j2[2][nz]: lower layer neighbor

                        Free boundary layers on both ends


                    For NiO/CoO system, the proposed parameter file:
param_nc.dat

                    nx              ny              nz
                    itemp           deltemp         numtemp
                    ibext           delbext         numbext
                    MC0             MC1             MC2
                    j1NiO           j1inNiO         j1outNiO
                    j2NiO           j2inNiO         j2outNiO
                    j1CoO           j1inCoO         j1outCoO
                    j2CoO           j2inCoO2        j2outCoO
```

```
                numNiO          numCoO


             The final scaling factor is decided by

                   j1[0][nz]= j1mat1*j1inmat1
                   j2[0][nz]= j2mat1*j2inmat1

                   j1[0][nz]: upper layer
                   j1[1][nz]: current layer
                   j1[2][nz]: lower layer

          */


           using namespace std;
           using namespace Numeric_lib;//for Matrices


        void cal_mag_op(Matrix<double,2>& multk, Matrix<double,1> &
m ,Matrix<double, 2> & ml, Matrix<double,3>& spinx, Matrix<double,3>&
spiny, Matrix<double, 3>& spinz,
                              Matrix<double,3>& multkl,double
temperature, double bextx);

        void optest_l(Matrix<double,2>& multk,Matrix<double,3>&
spinx, Matrix<double,3>& spiny, Matrix<double, 3>& spinz,double
temperature, double bextx,
                                   Matrix<double,3>& multkl);


        void namegen(double itemp, double deltemp, int numtemp,
double ibext,
                     double delbext, int numbext, string series);

        void spinoutput( Matrix<double,3>& spinx, Matrix<double,3>&
spiny, Matrix<double, 3>& spinz,char filename[]);
        int spinload(Matrix <double, 3>& spinx, Matrix <double, 3>&
spiny, Matrix <double,3>& spinz, char filename[]);


        void neighborcheck(Matrix <double,3>& spinx, Matrix
<double,3>& spiny, Matrix <double,3>& spinz);
        void spinprint(Matrix <double,3>& spinx, Matrix <double,
3>& spiny, Matrix <double, 3>& spinz);

        //Monte Carlo calculation and energy calculation

        void randomspin(double &sx, double &sy, double &sz);//drand
function
        void randomspin_xorshift(double &sx, double &sy, double
&sz);//drand function
```

```
            void scale_NiOCoO( Matrix <double, 2>& j1scale, Matrix
<double,2>& j2scale
                                    ,double j1inNiO,double j1outNiO,
double j2inNiO,double j2outNiO
                                    ,double j1inCoO,double j1outCoO,
double j2inCoO,double j2outCoO,
                                     int numNiO,int numCoO );


            double energycal_scale(Matrix<double,3> &spinx,
Matrix<double,3>& spiny, Matrix<double, 3>& spinz,
                                                    Matrix <double, 2>&
j1scale, Matrix <double,2>& j2scale,
                                                    double bextx,
Matrix<double,1>& m,Matrix<double,2> & multk);

            //not only the energy, this function calculates
magnetization and reduced magnetization;


            void MonteCarlo_niocoo_dil(Matrix<double,2>& multk,
Matrix<double,3>& spinx, Matrix<double,3>& spiny,
                                          Matrix<double, 3>&
spinz,Matrix <double, 2>& j1scale, Matrix <double,2>& j2scale, double
bextx,
                                          Matrix<double,1>& m, double
&energy,double temperature,int MC1,int MC2, Matrix<double,3>& multkl);

            //this function also performs equilibration runs

        //xorshift RNG
            static  unsigned long xxx,yyy,zzz,www,vvv;
            //xorshift random number generator

            unsigned long xorshift(void);


            // Global variables

            int nx, ny, nz; //dimension
            int numperlayer;
        int numspin;
        int nzspin; //number of nonzero spins

            int main()
            {

                  //for the xorshift RNG
                  unsigned int ran;
                  char filename3[15]="fiverands.dat";

                  ifstream xo;
                  xo.open(filename3);
                  xo >> vvv >> www >> xxx >> yyy >> zzz;
```

```
        //RNG


int i, ii,iii; //general indices
    cout.setf(ios::fixed);
    cout.precision(10);


    double invtemp; // 1.0/temperature
    double temperature; // temperature
    double bextx; // magnetic field

    //spin arrays
    Matrix<double, 1> kmag(4); //magnitude of the k vectors

    //spin variables

    char spinfile[25]="afm.out";
    char filename[25]= "postspin.out";
    char filename2[25]="prespin.out";
    int h,k,l;
int count=0;

    double sx,sy,sz;
    double bext(0.0);
    double energy;

    int br; //break point



    ifstream param;
    param.open("param_nc.dat");

    double itemp,deltemp;
    double ibextx,delbextx;
    int numbextx,numtemp;

    int  MC0,MC1,MC2;
    string series;
    double  j1NiO,j1inNiO,j1outNiO;
    double  j2NiO,j2inNiO,j2outNiO;
    double  j1CoO,j1inCoO,j1outCoO;
    double  j2CoO,j2inCoO,j2outCoO;
    int numNiO,numCoO;

    param >> series;
    param >>  nx     >> ny       >> nz;
    param >>  itemp  >> deltemp  >> numtemp;
    param >>  ibextx  >> delbextx >> numbextx;
    param >>  MC0    >> MC1      >> MC2;
    param >>  j1NiO  >> j1inNiO  >> j1outNiO;
    param >>  j2NiO  >> j2inNiO  >> j2outNiO;
```

```
            param >>  j1CoO  >> j1inCoO  >> j1outCoO;
            param >>  j2CoO  >> j2inCoO >> j2outCoO;
            param >>  numNiO >> numCoO;

            cout << series << endl;
            cout <<  nx     <<"  "  << ny      << "  " <<nz << endl;
            cout <<  itemp  <<"  " << deltemp  << "  " <<numtemp <<
endl;
            cout <<  ibextx  <<"  " << delbextx  << "  " << numbextx
<< endl;
            cout <<  MC0    <<"  "<< MC1       << "  " << MC2 <<
endl;
            cout <<  j1NiO  <<"  "<< j1inNiO  << "  " << j1outNiO <<
endl;
            cout <<  j2NiO  <<"  "<< j2inNiO  << "  " << j2outNiO <<
endl;
            cout <<  j1CoO  <<"  "<< j1inCoO  <<"  " << j1outCoO <<
endl;
            cout <<  j2CoO  <<"  "<< j2inCoO <<"  "  << j2outCoO <<
endl;
            cout <<  numNiO <<"  "<< numCoO <<"  "   << endl;

            param.close();

            Matrix<double,3>spinx(nx,ny,nz); //create the spin matrix
            Matrix<double,3>spiny(nx,ny,nz);
            Matrix<double,3>spinz(nx,ny,nz);



            Matrix <double,2> ml(3,nz);
            Matrix <double,3> multkl(3,4,nz);


            Matrix <double,1> m(3); //magnetization
            Matrix <double,2> multk(3,4); //reduced magnetization

            //scaling factors

            Matrix<double, 2>j1scale(3,nz);
            Matrix<double, 2>j2scale(3,nz);


        numperlayer=nx*ny;
        numspin=nx*ny*nz;

            //NiO: antiferromagnet

             j1inNiO = j1NiO*j1inNiO;
             j2inNiO = j2NiO*j2inNiO;
             j1outNiO = j1NiO*j1outNiO;
             j2outNiO = j2NiO*j2outNiO;
```

```
            //CoO: antiferromagnet

                j1inCoO= j1CoO*j1inCoO;
                j2inCoO = j2CoO*j2inCoO;
                j1outCoO = j1CoO*j1outCoO;
                j2outCoO = j2CoO*j2outCoO;


                //generate file names
                namegen(itemp,deltemp, numtemp, ibextx, delbextx,
numbextx, series);


            //set the scaling factors


            scale_NiOCoO(j1scale,j2scale,j1inNiO,j1outNiO,
j2inNiO,j2outNiO
                                      ,j1inCoO,j1outCoO,j2inCoO,
j2outCoO,numNiO,numCoO);

             invtemp = 1.0/itemp;
             cout << "The inverse temperature is " << invtemp <<
endl;

             count = spinload(spinx, spiny,spinz,spinfile);

             // for debugging, print spins
          // spinprint(spinx,spiny,spinz);

             cout<<"There are "<<count<<" spins total."<<endl;
             if(count!=nx*ny*nz) cout << "The spin count is wrong."
<< endl;

             //neighborcheck(spinx,spiny,spinz);


            //this has to be changed to the energy with scales


            //assume that k=k1+k2+k3+k4, where ki are 3-dimensional
vectors
            energy = energycal_scale(spinx, spiny, spinz, j1scale,
j2scale, ibextx, m, multk);

            cout << "The energy per spin is" << energy /
double(count) << endl;
            cal_mag_op(multk, m ,ml,spinx,spiny,spinz, multkl, itemp,
ibextx);
            optest_l(multk,spinx, spiny, spinz,itemp, ibextx,
multkl);
                  for(iii=0;iii<2;iii++)
             {
                 kmag(0)=kmag(0)+multk(iii,0)*multk(iii,0);
```

```
                    kmag(1)=kmag(1)+multk(iii,1)*multk(iii,1);
                    kmag(2)=kmag(2)+multk(iii,2)*multk(iii,2);
                    kmag(3)=kmag(3)+multk(iii,3)*multk(iii,3);
                }

                  kmag(0)=sqrt(kmag(0))/double(count);
                  kmag(1)=sqrt(kmag(1))/double(count);
                  kmag(2)=sqrt(kmag(2))/double(count);
                  kmag(3)=sqrt(kmag(3))/double(count);


                  cout <<"The reduced magnetization is " <<"("<< kmag(0)
<<"  " << kmag(1) <<"  "<<kmag(2) <<"  "<<kmag(3)<<")" << endl;


                   //  spinoutput(spinx,spiny,spinz,filename2);

                     ifstream file;
                     file.open("filenams.dat");


                     for(ii=0;ii<numtemp;ii++)
                    {   //there is no magnetic field loop; should I add
one? Maybe not.
                 for(iii=0;iii<numbextx;iii++)
                 {
                    file>>filename;
                    bextx=ibextx + double(iii)*delbextx;
                          temperature=itemp+ii*deltemp;
                          cout<<"The current temperature is " <<
temperature << endl;
                    cout<<"The current magnetic field is  " << bextx <<
endl;
                    cout<<"The current file name is "<<filename<<endl;
                    MonteCarlo_niocoo_dil(multk, spinx, spiny, spinz,
j1scale, j2scale, bextx, m, energy, temperature,MC1,MC2,multkl);
                      cal_mag_op(multk, m ,ml,spinx,spiny,spinz, multkl,
temperature, bextx);
                      optest_l(multk,spinx, spiny, spinz,temperature,
bextx, multkl);
                      spinoutput(spinx,spiny, spinz,filename);
                }

                  }

              return 0;

            }



            void randomspin(double &sx, double &sy, double &sz)
            {
```

```
  //generates a random spin

  double r1,r2,r3;
  double angle, height;
  double pi=3.141592653589793238462643;

  r1=drand48();
  r2=drand48();
  angle=2.0*pi*r1;

  sx= 2.0*r2-1.0;
  height=sqrt(1.0-sx*sx);

  sy=height*cos(angle);
  sz=height*sin(angle);

  return;
}


void randomspin_xorshift(double &sx, double &sy, double
&sz)
{

  //generates a random spin using Boost random number
generator

  unsigned int ran1,ran2,ran3;
  double r1, r2,r3;
  double angle, height;
  double pi=3.141592653589793238462643;

  ran1=xorshift();
  r1 = ran1*2.3283064e-10;
  ran2=xorshift();
  r2 = ran2*2.3283064e-10;

  angle=2.0*pi*r1;

  sx= 2.0*r2-1.0;
  height=sqrt(1.0-sx*sx);

  sy=height*cos(angle);
  sz=height*sin(angle);

  return;

}

void neighborcheck(Matrix <double,3>& spinx, Matrix
<double,3>& spiny, Matrix <double,3>& spinz)
{
  //this program checks for the neighbors
```

```
            int h,k,l;
            int flag=0;
            int hp,hm,kp,km,lp,lm;


            while(flag==0)

                {
                   cout<<"The neighbor check!"<<endl;
                   cout <<"Please enter the spin coordinates."<<endl;

                   cout<<"h:";
                   cin >> h;
                   cout<<"k:";
                   cin>> k;
                   cout<<"l:";
                   cin>>l;

                  cout <<"You entered ("<<h<<", "<<k<<",
"<<l<<")"<<endl;
                  cout <<"The spin values are
(sx,sy,sz)="<<spinx(h,k,l)<<", "<<spiny(h,k,l)<< ",
"<<spinz(h,k,l)<<")"<<endl;

                                  hp = (h + 1)%nx;
                                  hm = (h + nx-1)%nx;
                                  kp = (k + 1)%ny;
                                  km = (k + ny-1)%ny;
                                  lp = (l + 1)%nz;
                                  lm = (l + nz-1)%nz;



                  cout<<"The neighbor coordinates:"<<endl;

                  cout<<"The same layer<<"<<endl;
                  cout<< "(" << hp << "," << k << "," << l << ")" <<
" (" << spinx(hp,k,l) << "," << spiny(hp,k,l) << "," << spinz(hp,k,l)
<< ")" << endl;
                  cout<< "(" << h << "," << kp << "," << l << ")" <<
" (" << spinx(h,kp,l) << "," << spiny(h,kp,l) << "," << spinz(h,kp,l)
<< ")" << endl;
                  cout<< "(" << hm << "," << kp << "," << l << ")" <<
" (" << spinx(hm,kp,l) << "," << spiny(hm,kp,l) << "," <<
spinz(hm,kp,l) << ")" << endl;
                  cout<< "(" << hm << "," << k << "," << l << ")" <<
" (" << spinx(hm,k,l) << "," << spiny(hm,k,l) << "," << spinz(hm,k,l)
<< ")" << endl;
                  cout<< "(" << hp << "," << km << "," << l << ")" <<
" (" << spinx(hp,km,l) << "," << spiny(hp,km,l) << "," <<
spinz(hp,km,l) << ")" << endl;
```

```
                    cout<< "(" << h << "," << km << "," << l << ")" <<
" (" << spinx(h,km,l) << "," << spiny(h,km,l) << "," << spinz(h,km,l)
<< ")" << endl;


                    cout<<"Different layers"<<endl;


                    cout<< "("<<h<<","<<k<<","<<lp<<")"<<"
("<<spinx(h,k,lp)<<","<<spiny(h,k,lp)<<","<<spinz(h,k,lp)<<")"<<endl;
                    cout<< "("<<h<<","<<km<<","<<lp<<")"<<"
("<<spinx(h,km,lp)<<","<<spiny(h,km,lp)<<","<<spinz(h,km,lp)<<")"<<endl
;
                    cout<< "("<<hm<<","<<k<<","<<lp<<")"<<"
("<<spinx(hm,k,lp)<<","<<spiny(hm,k,lp)<<","<<spinz(hm,k,lp)<<")"<<endl
;

                    cout<< "("<<hp<<","<<km<<","<<lp<<")"<<"
("<<spinx(hp,km,lp)<<","<<spiny(hp,km,lp)<<","<<spinz(hp,km,lp)<<")"<<e
ndl;
                    cout<< "("<<hm<<","<<kp<<","<<lp<<")"<<"
("<<spinx(hm,kp,lp)<<","<<spiny(hm,kp,lp)<<","<<spinz(hm,kp,lp)<<")"<<e
ndl;
                    cout<< "("<<hm<<","<<km<<","<<lp<<")"<<"
("<<spinx(hm,km,lp)<<","<<spiny(hm,km,lp)<<","<<spinz(hm,km,lp)<<")"<<e
ndl;



                    cout<< "("<<h<<","<<k<<","<<lm<<")"<<"
("<<spinx(h,k,lm)<<","<<spiny(h,k,lm)<<","<<spinz(h,k,lm)<<")"<<endl;
                    cout<< "("<<hp<<","<<k<<","<<lm<<")"<<"
("<<spinx(hp,k,lm)<<","<<spiny(hp,k,lm)<<","<<spinz(hp,k,lm)<<")"<<endl
;
                    cout<< "("<<h<<","<<kp<<","<<lm<<")"<<"
("<<spinx(h,kp,lm)<<","<<spiny(h,kp,lm)<<","<<spinz(h,kp,lm)<<")"<<endl
;

                    cout<< "("<<hp<<","<<km<<","<<lm<<")"<<"
("<<spinx(hp,km,lm)<<","<<spiny(hp,km,lm)<<","<<spinz(hp,km,lm)<<")"<<e
ndl;
                    cout<< "("<<hm<<","<<kp<<","<<lm<<")"<<"
("<<spinx(hm,kp,lm)<<","<<spiny(hm,kp,lm)<<","<<spinz(hm,kp,lm)<<")"<<e
ndl;
                    cout<< "("<<hp<<","<<kp<<","<<lm<<")"<<"
("<<spinx(hp,kp,lm)<<","<<spiny(hp,kp,lm)<<","<<spinz(hp,kp,lm)<<")"<<e
ndl;



                    cout<<"If you want to repeat, please enter 0:";
                    cin >>flag;

                }
```

```
            cout<<"End of neighbor print."<<endl;
            return;

          }


          void spinoutput( Matrix<double,3>& spinx, Matrix<double,3>&
spiny,
                       Matrix<double, 3>& spinz, char filename[])
          {
                 int h,k,l;
                 ofstream outspin;
                 outspin.open(filename);

                 ofstream spincheck;
                 spincheck.open("spincheck.dat"); //for debugging

                 cout<<"Spin output function." << endl;

                 for(l=0;l<nz;l++)
                 {
                   for(k=0;k<ny;k++)
                   {
                         for(h=0;h<nx;h++)
                         {

                                outspin <<spinx(h,k,l)<<"
"<<spiny(h,k,l)<<" "<<spinz(h,k,l)<<endl;
                                spincheck << "hkl: "<<
h<<","<<k<<","<<l <<"," << spinx(h,k,l)<<", "<<spiny(h,k,l)<<",
"<<spinz(h,k,l)<<endl;

                             }
                   }
                }

            outspin.close();
        spincheck.close();

        return;

          }

          int spinload(Matrix <double, 3>& spinx, Matrix <double, 3>&
spiny, Matrix <double,3>& spinz, char filename[])
          {
                //for debugging
                ofstream loadcheck;
                loadcheck.open("load.dat");


                //initialize the spin variables
```

```
                    spinx=0.0;
                    spiny=0.0;
                    spinz=0.0;

                    ifstream inspin;

                    double sx, sy,sz;
                    int h,k,l;
                    int count=0;
                    nzspin=0;

                    int br; //break point
                    numperlayer=nx*ny;

                    inspin.open(filename);

                    cout << "Loading spins."<<endl;

                     while(inspin >> sx >> sy >> sz)
                     {

                       h = count%nx;
                       k = (count%numperlayer)/nx;
                       l = count/numperlayer;


                       loadcheck << "count: "<< count <<" (hkl)="<<h<<"
"<< k<<" "<<l<<endl;

                       spinx(h,k,l)=sx;
                       spiny(h,k,l)=sy;
                       spinz(h,k,l)=sz;

                       loadcheck << "The spin is
"<<spinx(h,k,l)<<","<<spiny(h,k,l)<<","<<spinz(h,k,l)<<endl;


if(sqrt(spinx(h,k,l)*spinx(h,k,l)+spiny(h,k,l)*spiny(h,k,l)+spinz(h,k,l
)*spinz(h,k,l)) > 0.99) nzspin = nzspin + 1;

                       count=count+1;

                      // cout <<"the spin is input"<<endl;
                }

                inspin.close();
                cout << " There are " << count << " spins total according
to the loading function." << endl;
           cout << "There are "<< nzspin << "nonzero spins according to
the loading function." << endl;

                loadcheck << " There are " << count << " spins total
according to the loading function." << endl;
```

```
              loadcheck << " There are " << nzspin << " spins total
according to the loading function." << endl;
              loadcheck.close();

              return count;
            }



            void spinprint(Matrix <double,3>& spinx, Matrix <double,
3>& spiny, Matrix <double, 3>& spinz)
            {
              int h,k,l;

              // cout<<"spin load test"<<endl;
               cout <<"(h,k,l) and spinx spiny spinz"<<endl;

              for(l=0; l<nz; l++)
                  {
                    for(k=0 ;k< ny; k++)
                  {
                         for(h=0;h< nx;h++)
                    {
                        cout<<"("<<h<<","<<k<<","<<l<<"),("
<<spinx(h,k,l)<<","<<spiny(h,k,l)<<","<<spinz(h,k,l)<<")"<<endl;

                    }
                     }
                  }

              return;

            }


            double energycal_scale(Matrix<double,3> &spinx,
Matrix<double,3>& spiny, Matrix<double, 3>& spinz,
                                                Matrix <double, 2>&
j1scale, Matrix <double,2>& j2scale,
                                                double bextx,
Matrix<double,1>& m,Matrix<double,2> & multk)
            {

               double denergy,energy;
               int h,k,l;
               int hp,hm,kp,km,lp,lm;


               double sum_1,sum_2,sum_3,sum_4,b_sum;


               Matrix <double,1> eff(3); //effective field

               //initialization of variables
```

```
                int sign1,sign2,sign3,sign4;


                   sum_1=0.0;
                   sum_2=0.0;
                   sum_3=0.0;
                   sum_4=0.0;
                   b_sum=0.0;

                   m=0.0;  //magnetization
                   multk=0.0; //order parameter

                //magnetization and reduced magnetization can be
calculated from m and op
              // cout << "Energy  function calculation starts here."
<<endl;


                    for(l=0;l<nz;l++)
                    {
                          for(k=0;k<ny;k++)
                          {
                               for(h=0;h<nx;h++)
                               {
                                 //implement the periodic boundary
condition

                                    hp = (h+1)%nx;
                                    hm = (h+nx-1)%nx;
                                    kp = (k+1)%ny;
                                    km = (k+ny-1)%ny;
                                    lp = (l+1)%nz; //upper layer
                                    lm = (l+nz-1)%nz; //lower layer

                                    eff(0)=0.0;
                                    eff(1)=0.0;
                                    eff(2)=0.0;

                                    //maybe I add some break points?

                                    // cout<<"h k l: "<<h<<" "<<k<<"
"<<l<<endl;
                                    // check << "hkl: "<< h << "," << k <<
"," << l << endl;
                        // check <<"lm j1scale(0,l):
"<<j1scale(0,l)<<endl;
                        // check <<"l j1scale(1,l):
"<<j1scale(1,l)<<endl;
                        // check <<"lp j1scale(2,l):
"<<j1scale(2,l)<<endl;
                        // check <<"lm j2scale(0,l):
"<<j2scale(0,l)<<endl;
```

```
                        // check <<"l j2scale(1,l):
"<<j2scale(1,l)<<endl;
                        // check <<"lp j2scale(2,l):
"<<j2scale(2,l)<<endl;

                         //Do the neighbor sums; there are 18 neighbors

                                  eff(0) = eff(0)+
j1scale(1,l)*(spinx(hp,k,l)+spinx(h,kp,l)+spinx(hm,kp,l)+spinx(hm,k,l)

+spinx(hp,km,l)+spinx(h,km,l)); //current layer neighbors

                                  eff(1) = eff(1)+
j1scale(1,l)*(spiny(hp,k,l)+spiny(h,kp,l)+spiny(hm,kp,l)+spiny(hm,k,l)

+spiny(hp,km,l)+spiny(h,km,l));

                                  eff(2) = eff(2)+
j1scale(1,l)*(spinz(hp,k,l)+spinz(h,kp,l)+spinz(hm,kp,l)+spinz(hm,k,l)

+spinz(hp,km,l)+spinz(h,km,l));

                                   //check << "in-plane j1 eff: " <<
eff(0) << "," << eff(1) << "," << eff(2) << endl;

                                   // lower layer

                                  eff(0) = eff(0) +
j1scale(2,l)*(spinx(h,k,lp) + spinx(h,km,lp) + spinx(hm,k,lp));//lower
layer neighbors
                                  eff(1) = eff(1) +
j1scale(2,l)*(spiny(h,k,lp) + spiny(h,km,lp) + spiny(hm,k,lp));
                                  eff(2) = eff(2) +
j1scale(2,l)*(spinz(h,k,lp) + spinz(h,km,lp) + spinz(hm,k,lp));

                                  // check << "lp j1 eff: " << eff(0) <<
"," << eff(1) << "," << eff(2) << endl;

                                  eff(0) = eff(0) +
j2scale(2,l)*( spinx(hp,km,lp) + spinx(hm, kp, lp) + spinx(hm, km,
lp)); // lower layer neighbors
                                  eff(1) = eff(1) +
j2scale(2,l)*( spiny(hp,km,lp) + spiny(hm, kp, lp) + spiny(hm, km,
lp));
                                  eff(2) = eff(2) +
j2scale(2,l)*( spinz(hp,km,lp) + spinz(hm, kp, lp) + spinz(hm, km,
lp));

                                   // check << "lp j2 eff: " << eff(0) <<
"," << eff(1) << "," << eff(2) << endl;


                                  //upper layer
```

```
                                eff(0) = eff(0)+
j1scale(0,l)*(spinx(h,k,lm)+spinx(hp,k,lm)+spinx(h,kp,lm));//upper
layer neighbors
                                eff(1) = eff(1)+
j1scale(0,l)*(spiny(h,k,lm)+spiny(hp,k,lm)+spiny(h,kp,lm));
                                eff(2) = eff(2)+
j1scale(0,l)*(spinz(h,k,lm)+spinz(hp,k,lm)+spinz(h,kp,lm));


                                // check << "lm j1 eff: " << eff(0) <<
"," << eff(1) << "," << eff(2) << endl;


                               eff(0) = eff(0)+ j2scale(0,l)*(spinx(hp,
km, lm) + spinx(hm, kp, lm) + spinx(hp, kp, lm));//upper layer
neighbors
                               eff(1) = eff(1)+ j2scale(0,l)*(spiny(hp,
km, lm) + spiny(hm, kp, lm) + spiny(hp, kp, lm));
                               eff(2) = eff(2)+ j2scale(0,l)*(spinz(hp,
km, lm) + spinz(hm, kp, lm) + spinz(hp, kp, lm));


                                // check << "lm j2 eff: " << eff(0) <<
"," << eff(1) << "," << eff(2) << endl;

                                //02-16-2010 magnetic field term
                                b_sum=b_sum + spinx(h,k,l)*bextx;

                                // check <<"b_sum: "<<b_sum<<" and the
magnetic field increment is "<<spinx(h,k,l)*bextx << endl;


                                m(0)= m(0)+ spinx(h,k,l);
                                m(1)= m(1)+ spiny(h,k,l);
                                m(2)= m(2)+ spinz(h,k,l);

                                //order parameter

                                sign1=(h%2)*2-1;
                                sign2=(k%2)*2-1;
                                sign3=(l%2)*2-1;
                                sign4=((h+k+l)%2)*2-1;


multk(0,0)=multk(0,0)+double(sign1)*spinx(h,k,l);

multk(1,0)=multk(1,0)+double(sign1)*spiny(h,k,l);

multk(2,0)=multk(2,0)+double(sign1)*spinz(h,k,l);


multk(0,1)=multk(0,1)+double(sign2)*spinx(h,k,l);

multk(1,1)=multk(1,1)+double(sign2)*spiny(h,k,l);
```

```
multk(2,1)=multk(2,1)+double(sign2)*spinz(h,k,l);


multk(0,2)=multk(0,2)+double(sign3)*spinx(h,k,l);

multk(1,2)=multk(1,2)+double(sign3)*spiny(h,k,l);

multk(2,2)=multk(2,2)+double(sign3)*spinz(h,k,l);


multk(0,3)=multk(0,3)+double(sign4)*spinx(h,k,l);

multk(1,3)=multk(1,3)+double(sign4)*spiny(h,k,l);

multk(2,3)=multk(2,3)+double(sign4)*spinz(h,k,l);


                                denergy= denergy +
spinx(h,k,l)*eff(0)+spiny(h,k,l)*eff(1)+spinz(h,k,l)*eff(2);
                                //check <<"The nb energy increment is
"<<  spinx(h,k,l)*eff(0)+spiny(h,k,l)*eff(1)+spinz(h,k,l)*eff(2)<<endl;
                                //check <<"The magnetic field increment
is " <<b_sum<<endl;
                           }

                  }

                    energy= denergy*0.5 + b_sum;

                    //check<<"The b_sum is " << b_sum << endl;
                    //check<<"The double energy is "<< denergy << " and
the halved energy is " << energy << endl;
                    //Maybe I need to check this program again

                  }
                  // check.close();

                   return energy;
            }


            void namegen(double itemp,double deltemp, int
numtemp,double ibextx, double delbextx, int numbextx, string series)
            {
                  //this function generates spin output file names
                  int q,r;
                  string te="t";
                  string be="b";
                  string tempnum,bfieldnum;
                  double temp,bfield;
                  ofstream filenam;
                  string ofilename;
                  string extension=".out";
```

```
                filenam.open("filenams.dat");

              for(q=0;q<numtemp; q++)
              {
                  for(r=0;r<numbextx;r++)
                  {
                       temp=itemp+q*deltemp;
                       bfield = ibextx+ r*delbextx;
                       ostringstream ss0;
                       ostringstream ss1;

                       ss0.setf(ios::fixed);
                       ss0.precision(4);
                       ss0 << temp;
                       tempnum =ss0.str();

                       ss1.setf(ios::fixed);
                       ss1.precision(4);
                       ss1 << bfield;
                       bfieldnum =ss1.str();

                       ofilename= series +te+
tempnum+be+bfieldnum+extension;
                       cout << ofilename << endl;
                       filenam << ofilename << endl;
                }
              }

              filenam.close();
              return;
          }


          void cal_mag_op(Matrix<double,2>& multk, Matrix<double,1> &
m ,Matrix<double, 2> & ml, Matrix<double,3>& spinx, Matrix<double,3>&
spiny, Matrix<double, 3>& spinz,
                                    Matrix<double,3>& multkl,double
temperature, double bextx)
          {
                //combined magnetization and reduced magnetization
calculations
                //normalization has to be done as well; =D

                 Matrix<double, 1> kmag(4);
                 Matrix<double,2> kmagl(4,nz);
                 Matrix <double,1> mmagl(nz); //layer magnetization

                 int count = nx*ny*nz; //number of spins in the
system
                 int iii,ii;
                 int h,k,l;
                 int numl;
                 double sign1,sign2,sign3,sign4;
```

```
                    double mmag; //magnetization


                    //Normalized variables

                    double nmmag;
                    Matrix<double,1> nmmagl(nz);
                    Matrix<double,2> nkmagl(4,nz);
                    Matrix<double,1> nkmag(4);


                    multk = 0.0; //order parameter
                    multkl = 0.0;//layer order parameter
                    m=0.0;//magnetization
                    ml=0.0;//layer magnetization

                    cout<<"The number of spins is "<< numspin << endl;

                //magnetization and reduced magnetization can be
calculated from m and op

                   for(l=0;l<nz;l++)
                     {
                      for(k=0;k<ny;k++)
                      {
                          for(h=0;h<nx;h++)
                         {
                                 //implement the periodic boundary
condition
                                  //order parameter

                                     sign1=double(h%2)*2.0-1.0;
                                     sign2=double(k%2)*2.0-1.0;
                                     sign3=double(l%2)*2.0-1.0;
                                     sign4=double((h+k+l)%2)*2.0-1.0;

                                     if(sign1!=1.0 && sign1!=-
1.0)cout<<"wrong sign!"<<endl;

                                     //regular order parameter
calculation


      multk(0,0)=multk(0,0)+sign1*spinx(h,k,l);//x-comp

      multk(1,0)=multk(1,0)+sign1*spiny(h,k,l);//y-comp

      multk(2,0)=multk(2,0)+sign1*spinz(h,k,l);//z-comp


      multk(0,1)=multk(0,1)+sign2*spinx(h,k,l);

      multk(1,1)=multk(1,1)+sign2*spiny(h,k,l);
```

```
multk(2,1)=multk(2,1)+sign2*spinz(h,k,l);

multk(0,2)=multk(0,2)+sign3*spinx(h,k,l);

multk(1,2)=multk(1,2)+sign3*spiny(h,k,l);

multk(2,2)=multk(2,2)+sign3*spinz(h,k,l);

multk(0,3)=multk(0,3)+sign4*spinx(h,k,l);

multk(1,3)=multk(1,3)+sign4*spiny(h,k,l);

multk(2,3)=multk(2,3)+sign4*spinz(h,k,l);
                          //layer order parameter calculation

multkl(0,0,l)=multkl(0,0,l)+sign1*spinx(h,k,l);

multkl(1,0,l)=multkl(1,0,l)+sign1*spiny(h,k,l);

multkl(2,0,l)=multkl(2,0,l)+sign1*spinz(h,k,l);

multkl(0,1,l)=multkl(0,1,l)+sign2*spinx(h,k,l);

multkl(1,1,l)=multkl(1,1,l)+sign2*spiny(h,k,l);

multkl(2,1,l)=multkl(2,1,l)+sign2*spinz(h,k,l);

multkl(0,2,l)=multkl(0,2,l)+sign3*spinx(h,k,l);

multkl(1,2,l)=multkl(1,2,l)+sign3*spiny(h,k,l);

multkl(2,2,l)=multkl(2,2,l)+sign3*spinz(h,k,l);

multkl(0,3,l)=multkl(0,3,l)+sign4*spinx(h,k,l);

multkl(1,3,l)=multkl(1,3,l)+sign4*spiny(h,k,l);

multkl(2,3,l)=multkl(2,3,l)+sign4*spinz(h,k,l);

                             m(0)= m(0)+spinx(h,k,l);
                             m(1)= m(1)+spiny(h,k,l);
                             m(2)= m(2)+spinz(h,k,l);

                             ml(0,l)=ml(0,l)+spinx(h,k,l);
                             ml(1,l)=ml(1,l)+spiny(h,k,l);
                             ml(2,l)=ml(2,l)+spinz(h,k,l);
```

```
                        }
                    }
                }

                        //regular order parameter

                        kmag(0) = multk(0,0)*multk(0,0) +
multk(1,0)*multk(1,0) + multk(2,0)*multk(2,0);
                        kmag(1) = multk(0,1)*multk(0,1) +
multk(1,1)*multk(1,1) + multk(2,1)*multk(2,1);
                        kmag(2) = multk(0,2)*multk(0,2) +
multk(1,2)*multk(1,2) + multk(2,2)*multk(2,2);
                        kmag(3) = multk(0,3)*multk(0,3) +
multk(1,3)*multk(1,3) + multk(2,3)*multk(2,3);

                        //layer order parameter


                for(ii=0; ii<nz; ii++)
                        {

                                kmagl(0,ii)=
multkl(0,0,ii)*multkl(0,0,ii)+multkl(1,0,ii)*multkl(1,0,ii)+multkl(2,0,
ii)*multkl(2,0,ii);
                                kmagl(1,ii)=
multkl(0,1,ii)*multkl(0,1,ii)+multkl(1,1,ii)*multkl(1,1,ii)+multkl(2,1,
ii)*multkl(2,1,ii);
                                kmagl(2,ii)=
multkl(0,2,ii)*multkl(0,2,ii)+multkl(1,2,ii)*multkl(1,2,ii)+multkl(2,2,
ii)*multkl(2,2,ii);
                                kmagl(3,ii)=
multkl(0,3,ii)*multkl(0,3,ii)+multkl(1,3,ii)*multkl(1,3,ii)+multkl(2,3,
ii)*multkl(2,3,ii);
                                cout<<"multkl(0,2): " <<
multkl(0,2,ii) << "multkl(1,2): " << multkl(1,2,ii) << " multkl(2,2): "
<< multkl(2,2,ii) << endl;
                                cout<< " kmagl(0,"<<ii<<"): " <<
kmagl(0,ii) << " kmagl(1,"<< ii <<"): " <<kmagl(1,ii) << "
kmagl(2,"<<ii<<"): " << kmagl(2,ii) << " kmagl(3,"<<ii<<"): " <<
kmagl(3,ii) << endl;
                                mmagl(ii)=
ml(0,ii)*ml(0,ii)+ml(1,ii)*ml(1,ii)+ml(2,ii)*ml(2,ii);
                        }

                //normalization

        //total order parameter

                        //cout << "Before the normalizaion, the
k vectors are:"<<endl;
                        //cout << "(k0):" << sqrt(kmag(0)) <<
endl;
```

```
                                  //cout << "(k1):" << sqrt(kmag(1)) <<
endl;
                                  //cout << "(k2):" << sqrt(kmag(2)) <<
endl;
                                  //cout << "(k3):" << sqrt(kmag(3)) <<
endl;

                                nkmag(0)=sqrt(kmag(0))/double(count);
                                nkmag(1)=sqrt(kmag(1))/double(count);
                                nkmag(2)=sqrt(kmag(2))/double(count);
                                nkmag(3)=sqrt(kmag(3))/double(count);

                               numl=nx*ny; //number of spins per layer;

                            //layer order parameter

                            for(ii=0;ii<nz;ii++)
                             {

      nkmagl(0,ii)=sqrt(kmagl(0,ii))/double(numl);

      nkmagl(1,ii)=sqrt(kmagl(1,ii))/double(numl);

      nkmagl(2,ii)=sqrt(kmagl(2,ii))/double(numl);

      nkmagl(3,ii)=sqrt(kmagl(3,ii))/double(numl);
                                     //nml(ii)=mmagl(ii)/double(count);
                              }

                    ofstream dataout1;
                    ofstream dataout2;
                    ofstream dataout3;
                    ofstream dataout4;

                    dataout1.open("mncl.dat",ios::app);
                    dataout2.open("mnc.dat",ios::app);
                    dataout3.open("opnc.dat",ios::app);
                    dataout4.open("opncl.dat",ios::app);

               //layer magnetization and layer reduced magnetization

               nmmag=sqrt(m(0)*m(0)+m(1)*m(1)+m(2)*m(2))/double(count);
               for(ii=0;ii<nz;ii++)
               {
                  dataout1 << temperature <<"   "<< bextx << "  " << ii
<<"   "<< ml(0,ii) << "   " << ml(1,ii) << "  " << ml(2,ii) << "  " <<
mmagl(ii) << endl;
                  dataout4 << temperature  <<"  "<< bextx << "  " << ii
<<"   "<< nkmagl(0,ii) << "  " << nkmagl(1,ii)<<"  "<< nkmagl(2,ii) <<
"  " << nkmagl(3,ii) << endl;
               }

               dataout2 << temperature << "   " << bextx << "  " << m(0)
<< "   " << m(1) << "  " << m(2) << "  " << nmmag << endl;
```

```
            dataout3 << temperature << "  " << bextx << "  " <<
nkmag(0) << "  " << nkmag(1) << "  " << nkmag(2) << "  " <<  nkmag(3)
<< endl;


            dataout1.close();
            dataout2.close();
            dataout3.close();
            dataout4.close();



                return;
              }



            void scale_NiOCoO(Matrix <double, 2>& j1scale, Matrix
<double,2>& j2scale
                                    ,double j1inNiO,double j1outNiO,
double j2inNiO,double j2outNiO
                                ,double j1inCoO,double j1outCoO,
double j2inCoO,double j2outCoO,
                                    int numNiO,int numCoO)
            {

                               //generation of scaling factors;NiO/CoO
bilayer system
                               //in fact there are five interfaces,
where I need to use the average exchange interaction]

                            int index1,index2,index3;
                            double j1inave,j2inave,j1outave,j2outave;
                            int numNiO2,numCoO2;

                           int  interface0_2,  interface1_1,interface1_2,
interface2_1,interface2_2, interface3_1,interface3_2;


                           //If I use the periodic boundary condition,
I get five interfaces


                              j1inave=0.5*(j1inNiO+j1inCoO);
                              j2inave=0.5*(j2inNiO+j2inCoO);
                              j1outave=0.5*(j1outNiO+j1outCoO);
                              j2outave=0.5*(j2outNiO+j2outCoO);


                               numCoO2=numCoO;

                               cout << "The total number of layers is "
<< nz << endl;
```

```
                                numNiO2= nz-(numCoO2+numCoO+numNiO);

                                cout << "The system size:"<<endl;
                                cout << "The NiO1:"<< numNiO<<endl;
                                cout << "The CoO1:"<< numCoO<<endl;
                                cout << "The second
NiO:"<<numNiO2<<endl;
                                cout << "The second
CoO:"<<numCoO2<<endl;
                                cout << "The total layer number is "<<
numNiO2 + numCoO2 + numCoO + numNiO << endl;

                                cout << "J1inNiO:"<< j1inNiO<<endl;
                                cout << "J1outNiO:"<< j1outNiO<<endl;
                                cout << "J2inNiO:"<<j2inNiO<<endl;
                                cout << "J2outNiO:" <<j2outNiO<<endl;

                                cout << "J1inCoO:"<< j1inCoO<<endl;
                                cout << "J1outCoo:"<< j1outCoO<<endl;
                                cout << "J2inCoO:"<<j2inCoO<<endl;
                                cout << "J2outCoO:" <<j2outCoO<<endl;

                                cout << "J1inave:"<< j1inave<<endl;
                                cout << "J1outave:"<< j1outave<<endl;
                                cout << "J2inave:"<<j2inave<<endl;
                                cout << "J2outave:" <<j2outave<<endl;

                            if (numNiO2 + numCoO2 + numCoO + numNiO !=
nz) cout<<"Wrong layer sum."<<endl;

                            //upper layer j1scale[0][nz]
                            //the first layer interface 0

                                        j1scale[0][0]=j1outave;//CoO-NiO
interface
                                        j1scale[1][0]=j1inNiO;
                                        j1scale[2][0]=j1outNiO;
                                        j2scale[0][0]=j2outave;
                                        j2scale[1][0]=j2inNiO;
                                        j2scale[2][0]=j2outNiO;


                            for(index1=1;index1<(numNiO-1);index1++)

                                {//Inner region I: NiO

                                        j1scale[0][index1]=j1outNiO;
                                        j1scale[1][index1]=j1inNiO;
                                        j1scale[2][index1]=j1outNiO;
                                        j2scale[0][index1]=j2outNiO;
                                        j2scale[1][index1]=j2inNiO;
                                        j2scale[2][index1]=j2outNiO;

                                }
```

```
                                        interface1_1=numNiO-1;
                                        interface1_2=numNiO;

                                //two interlayer-coupled layers -interface I

j1scale[0][interface1_1]=j1outNiO;

j1scale[1][interface1_1]=j1inNiO;

j1scale[2][interface1_1]=j1outave;

j2scale[0][interface1_1]=j2outNiO;

j2scale[1][interface1_1]=j2inNiO;

j2scale[2][interface1_1]=j2outave;

                                        //first CoO layer

j1scale[0][interface1_2]=j1outave;

j1scale[1][interface1_2]=j1inCoO;

j1scale[2][interface1_2]=j1outCoO;

j2scale[0][interface1_2]=j2outave;

j2scale[1][interface1_2]=j2inCoO;

j2scale[2][interface1_2]=j2outCoO;

                                        for(index2 = interface1_2 + 1; index2
< (numNiO+numCoO-1);index2++)
                                              {//second CoO layer through the
second to the last layer of CoO; inner region II

                                               j1scale[0][index2]=j1outCoO;
                                                 j1scale[1][index2]=j1inCoO;
                                                 j1scale[2][index2]=j1outCoO;
                                                 j2scale[0][index2]=j2outCoO;
                                                 j2scale[1][index2]=j2inCoO;
                                                 j2scale[2][index2]=j2outCoO;
                                              }

                                    //The second set of the interface layers

                                    //two interlayer-coupled layers;
interface II

                                              interface2_1= numNiO+numCoO-1;
```

```
                                        interface2_2=interface2_1+1;

j1scale[0][interface2_1]=j1outCoO;

j1scale[1][interface2_1]=j1inCoO;

j1scale[2][interface2_1]=j1outave;

j2scale[0][interface2_1]=j2outCoO;

j2scale[1][interface2_1]=j2inCoO;

j2scale[2][interface2_1]=j2outave;
                                    //first CoO layer

j1scale[0][interface2_2]=j1outave;

j1scale[1][interface2_2]=j1inNiO;

j1scale[2][interface2_2]=j1outNiO;

j2scale[0][interface2_2]=j2outave;

j2scale[1][interface2_2]=j2inNiO;

j2scale[2][interface2_2]=j2outNiO;


for(index1=numCoO+numNiO+1;index1<(numNiO+numCoO+numNiO2-1);index1++)
                                  {  // the secont set of NiO monolayers;
inner region III

                                        j1scale[0][index1]=j1outNiO;
                                        j1scale[1][index1]=j1inNiO;
                                        j1scale[2][index1]=j1outNiO;
                                        j2scale[0][index1]=j2outNiO;
                                        j2scale[1][index1]=j2inNiO;
                                        j2scale[2][index1]=j2outNiO;

                                  }

interface3_1=numNiO+numCoO+numNiO2-1;

interface3_2=numNiO+numCoO+numNiO2;
                                       // two interlayer-coupled layers


j1scale[0][interface3_1]=j1outNiO;
```

```
j1scale[1][interface3_1]=j1inNiO;

j1scale[2][interface3_1]=j1outave;

j2scale[0][interface3_1]=j2outNiO;

j2scale[1][interface3_1]=j2inNiO;

j2scale[2][interface3_1]=j2outave;

                                // first CoO layer


j1scale[0][interface3_2]=j1outave;

j1scale[1][interface3_2]=j1inCoO;

j1scale[2][interface3_2]=j1outCoO;

j2scale[0][interface3_2]=j2outave;

j2scale[1][interface3_2]=j2inCoO;

j2scale[2][interface3_2]=j2outCoO;


                                for(index3 = interface3_2+1; index3 <
(numNiO + numCoO + numNiO2 + numCoO2-1); index3++)
                                        {//second CoO layer through
the second to the last layer of CoO: inner region IV

                                        j1scale[0][index3]=j1outCoO;
                                        j1scale[1][index3]=j1inCoO;
                                        j1scale[2][index3]=j1outCoO;
                                        j2scale[0][index3]=j2outCoO;
                                        j2scale[1][index3]=j2inCoO;
                                        j2scale[2][index3]=j2outCoO;

                                        }
                                    //the last layer
                                    interface0_2= numNiO + numCoO +
numNiO2 + numCoO2-1;

                                    cout<<"The index for the last layer
is " << interface0_2<<endl;


      j1scale[0][interface0_2]=j1outCoO;

      j1scale[1][interface0_2]=j1inCoO;

      j1scale[2][interface0_2]=j1outave;
```

```
      j2scale[0][interface0_2]=j2outCoO;

      j2scale[1][interface0_2]=j2inCoO;

      j2scale[2][interface0_2]=j2outave;


                                        ofstream scaleout;
                                        scaleout.open("scaleout.dat");

                                    for(index1=0; index1 <
nz ;index1++)
                                    {


scaleout<<"Layer:"<<index1<<endl;

                                        scaleout << "Upper layer J1:"<<
j1scale[0][index1] << endl;
                                        scaleout << "Current layer
J1:"<< j1scale[1][index1] << endl;
                                        scaleout << "Lower layer J1:"<<
j1scale[2][index1] << endl<<endl;
                                        scaleout << "Upper layer J2:"<<
j2scale[0][index1] << endl;
                                        scaleout << "Current layer
J2:"<< j2scale[1][index1] << endl;
                                        scaleout << "Lower layer J2:"<<
j2scale[2][index1]<<endl<<endl;
                                      }

                                 scaleout.close();

                  return;
            }

            //xorshift random number generator

            unsigned long xorshift(void)
            {
                        // Define a random number generator and
initialize it with a reproducible seed.
                       // (The seed is unsigned, to avoid trouble with
some generators.)

                     unsigned long ttt;

                     ttt   = xxx ^ (xxx >> 7);
                     xxx   = yyy;
                     yyy   = zzz;
                     zzz   = www;
                     www   = vvv;
```

```
                    vvv   = (vvv ^ (vvv << 6)) ^ (ttt ^ (ttt << 13));

                    return (yyy + yyy + 1) * vvv;


            }


            void MonteCarlo_niocoo_dil(Matrix<double,2>& multk,
Matrix<double,3>& spinx, Matrix<double,3>& spiny,
                                        Matrix<double, 3>&
spinz,Matrix <double, 2>& j1scale, Matrix <double,2>& j2scale, double
bextx,
                                        Matrix<double,1>& m, double
&energy,double temperature,int MC1,int MC2, Matrix<double,3>& multkl)
             {

                    //Dilution only affects the Monte Carlo function

                          int br; //break point

                              // boost random number generator
                              // this function includes both
equilibration and data run
                              // Prior to this function, the order
parameter has to be recalculated?

                               int pick;
                               double r1,r2;
                               unsigned int ran1, ran2;
                               double refenergy;

                               double delsx,delsy,delsz;
                               double sxnew,synew,sznew;
                               double hp,hm,kp,km,lp,lm;
                               double invtemp;
                         double sign1,sign2, sign3,sign4;
                         double b_sum; //magnetic field sum

                                // Statistical calculations


                            // add the standard deviation of the energy,
magnetization, and the reduced magnetization

                        //in the for loop


                         double ensum=0.0;
                         double ensqsum=0.0;

                         double magsq;
                         double magsqsum=0.0;
```

```
                        double msumx=0.0;
                        double msumy=0.0;
                        double msumz=0.0;

                        double kmags0,kmags1,kmags2,kmags3; //squared
magnitudes

                        double ksqsum0=0.0;
                      double ksqsum1=0.0;
                      double ksqsum2=0.0;
                      double ksqsum3=0.0;

                      double k0sumx=0.0;
                      double k0sumy=0.0;
                      double k0sumz=0.0;

                      double k1sumx=0.0;
                      double k1sumy=0.0;
                      double k1sumz=0.0;

                      double k2sumx=0.0;
                      double k2sumy=0.0;
                      double k2sumz=0.0;

                        double k3sumx=0.0;
                      double k3sumy=0.0;
                      double k3sumz=0.0;

                                //outside the for loop

                      double mmags; //squared magnitude

                      double envar, k0var,k1var,k2var,k3var;
                      double mvar;

                      //do I want to normalize it; normalization
requires dividing the variance by the square of the number of nonzero
spins?

                                invtemp=1.0/temperature;
                                ofstream data;

                                data.open("datav.dat",ios::app);

                               // cout<<"Real data run."<<endl;

                                int i,j,ii;
                                int s=0;
                                int s1=0;
                                int n(0),nn(0),n1(0),nn1(0); //flip non-
flip counters for equilibration and data collection
                                int h,k,l; //selected coordinates
```

```
                        //number of spins per layer

                                Matrix<double,1> eff(3);


                                double efff;

                         cout<<"The initial energy is "<<energy<<endl;
                                cout<<"Equilibration"<<endl;


                           for(i=0;i<MC1;i++)
                             { //repeat this for the number of
preruns*number of spins

                             for(ii=0;ii<numspin;ii++)
                        {
                                        ran1 = xorshift();
                                        r1 = ran1*2.3283064e-10;
                                        pick = r1*numspin;
                                        h = pick % nx;
                                        k = (pick % numperlayer)/nx;
                                        l = pick / numperlayer;


if(sqrt(spinx(h,k,l)*spinx(h,k,l)+spiny(h,k,l)*spiny(h,k,l)+spinz(h,k,l
)*spinz(h,k,l))>0.99)
                                {
                                                        //random spin
generation


randomspin_xorshift(sxnew,synew,sznew);


                        //calculate the efffective field

                                                        delsx=sxnew-
spinx(h,k,l);
                                                        delsy=synew-
spiny(h,k,l);
                                                        delsz=sznew-
spinz(h,k,l);


                                                        // cout<<" sxdel: "
<<sxnew-spinx(hpick,kpick,lpick)<<endl;
                                                        // cout<<" sydel: "
<<synew-spiny(hpick,kpick,lpick)<<endl;
                                                        // cout<<" szdel: "
<<sznew-spinz(hpick,kpick,lpick)<<endl;

                                                             hp = (h +
1)%nx;
```

```
                                                        hm = (h + nx-
1)%nx;
                                                        kp = (k +
1)%ny;
                                                        km = (k + ny-
1)%ny;
                                                        lp = (l +
1)%nz;
                                                        lm = (l + nz-
1)%nz;

                                                        eff(0)=0.0;
                                                        eff(1)=0.0;
                                                        eff(2)=0.0;

                                                        eff(0) =
eff(0)+
j1scale(1,l)*(spinx(hp,k,l)+spinx(h,kp,l)+spinx(hm,kp,l)+spinx(hm,k,l)

+spinx(hp,km,l)+spinx(h,km,l)); //current layer neighbors
                                                        eff(1) =
eff(1)+
j1scale(1,l)*(spiny(hp,k,l)+spiny(h,kp,l)+spiny(hm,kp,l)+spiny(hm,k,l)

         +spiny(hp,km,l)+spiny(h,km,l));
                                                        eff(2) =
eff(2)+
j1scale(1,l)*(spinz(hp,k,l)+spinz(h,kp,l)+spinz(hm,kp,l)+spinz(hm,k,l)

         +spinz(hp,km,l)+spinz(h,km,l));


                                        //upper layer

                                                        eff(0) = eff(0)
+ j1scale(2,l)*(spinx(h,k,lp)+spinx(h,km,lp)+spinx(hm,k,lp));//upper
layer neighbors
                                                        eff(1) = eff(1)
+ j1scale(2,l)*(spiny(h,k,lp)+spiny(h,km,lp)+spiny(hm,k,lp));
                                                        eff(2) = eff(2)
+ j1scale(2,l)*(spinz(h,k,lp)+spinz(h,km,lp)+spinz(hm,k,lp));


                                        eff(0) = eff(0) +
j2scale(2,l)*(spinx(hp,km,lp)+spinx(hm,kp,lp)+spinx(hm,km,lp));//upper
neighbor neighbors
                                                        eff(1) = eff(1)
+ j2scale(2,l)*(spiny(hp,km,lp)+spiny(hm,kp,lp)+spiny(hm,km,lp));
                                                        eff(2) = eff(2)
+ j2scale(2,l)*(spinz(hp,km,lp)+spinz(hm,kp,lp)+spinz(hm,km,lp));
```

```
                                                        eff(0) =
eff(0)+j1scale(0,l)*(spinx(h,k,lm)+spinx(hp,k,lm)+spinx(h,kp,lm));//low
er layer neighbors
                                                        eff(1) =
eff(1)+j1scale(0,l)*(spiny(h,k,lm)+spiny(hp,k,lm)+spiny(h,kp,lm));
                                                        eff(2) =
eff(2)+j1scale(0,l)*(spinz(h,k,lm)+spinz(hp,k,lm)+spinz(h,kp,lm));


                                                        eff(0) =
eff(0)+j2scale(0,l)*(spinx(hp,km,lm)+spinx(hm,kp,lm)+spinx(hp,kp,lm));/
/lower layer neighbors
                                                        eff(1) =
eff(1)+j2scale(0,l)*(spiny(hp,km,lm)+spiny(hm,kp,lm)+spiny(hp,kp,lm));
                                                        eff(2) =
eff(2)+j2scale(0,l)*(spinz(hp,km,lm)+spinz(hm,kp,lm)+spinz(hp,kp,lm));

b_sum=delsx*bextx;

                                                        //cout<<"The
magnetic field sum: " << b_sum << endl;

                                                        efff=
delsx*eff(0)+delsy*eff(1)+delsz*eff(2)+b_sum; //to remove the double-
count

                                                        //cout << "The
energy is before update is" << energy << endl;
                                                        //cout << "The
effective field is "<< efff << endl;


                                                       //efff is the
energy difference

                                                        if(efff < 0.0)
                                                        {
                                                              //update
the spin, magnetization, and energy
                                                              //it is
possible to update the magnetization too

      energy = energy + efff;


      m(0)=m(0)+delsx;

      m(1)=m(1)+delsy;

      m(2)=m(2)+delsz;


      spinx(h,k,l) = sxnew;
```

```
      spiny(h,k,l) = synew;
      spinz(h,k,l) = sznew;

                                                    //sign has to be
implemented for order parameter calculation

sign1=(h%2)*2-1;
sign2=(k%2)*2-1;
sign3=(l%2)*2-1;
sign4=((h+k+l)%2)*2-1;

multk(0,0)=multk(0,0)+double(sign1)*delsx;
multk(1,0)=multk(1,0)+double(sign1)*delsy;
multk(2,0)=multk(2,0)+double(sign1)*delsz;

multk(0,1)=multk(0,1)+double(sign2)*delsx;
multk(1,1)=multk(1,1)+double(sign2)*delsy;
multk(2,1)=multk(2,1)+double(sign2)*delsz;

multk(0,2)=multk(0,2)+double(sign3)*delsx;
multk(1,2)=multk(1,2)+double(sign3)*delsy;
multk(2,2)=multk(2,2)+double(sign3)*delsz;

multk(0,3)=multk(0,3)+double(sign4)*delsx;
multk(1,3)=multk(1,3)+double(sign4)*delsy;
multk(2,3)=multk(2,3)+double(sign4)*delsz;

                                                                    //
cout <<"The updated energy is "<< energy<<endl;


                                                                    //
cout<<"spin is flipped"<<endl;
```

```
      n=n+1;
                                                                  }
                                                                  else
                                                                  {
                                                                  //generate a
random number
                                                                  ran2 =
xorshift();
                                                                  r2 =
ran2*2.3283064e-10;
                                                                  if (r2 <
exp(-invtemp*efff))
                                                                  {
      //update the spin, magnetization, and energy
      energy = energy + efff;

      m(0)=m(0)+delsx;
      m(1)=m(1)+delsy;
      m(2)=m(2)+delsz;
      // cout<<"The effective field is "<< efff<<endl;

//sign has to be implemented for order parameter calculation

      sign1=(h%2)*2-1;
      sign2=(k%2)*2-1;
      sign3=(l%2)*2-1;
      sign4=((h+k+l)%2)*2-1;

      multk(0,0)=multk(0,0)+double(sign1)*delsx;
      multk(1,0)=multk(1,0)+double(sign1)*delsy;
      multk(2,0)=multk(2,0)+double(sign1)*delsz;
```

```
      multk(0,1)=multk(0,1)+double(sign2)*delsx;
      multk(1,1)=multk(1,1)+double(sign2)*delsy;
      multk(2,1)=multk(2,1)+double(sign2)*delsz;

      multk(0,2)=multk(0,2)+double(sign3)*delsx;
      multk(1,2)=multk(1,2)+double(sign3)*delsy;
      multk(2,2)=multk(2,2)+double(sign3)*delsz;

      multk(0,3)=multk(0,3)+double(sign4)*delsx;
      multk(1,3)=multk(1,3)+double(sign4)*delsy;
      multk(2,3)=multk(2,3)+double(sign4)*delsz;

      //cout <<"The updated energy is "<<energy<<endl;

      spinx(h,k,l) = sxnew;
      spiny(h,k,l) = synew;
      spinz(h,k,l) = sznew;

      //cout <<"spin is flipped"<<endl;
      n=n+1;
                                                              }//if
loop
                                                               else
nn=nn+1; //the spin is not flipped;
                                                    }
                                                          //else
loop
                              }//if loop (for skipping)
                              else  s=s+1;


                                        //recalculate the energy
refenergy = energycal_scale(spinx, spiny, spinz,j1scale, j2scale,bextx,
m, multk);
```

```
                                                        // cout<< "The
current energy is " << energy << endl;
                                                              //
cout<<"The reference energy is "<< refenergy<<endl;

      //cout<<"The reference magnetic moment is " << m(0) << "," <<
m(1) << "," << m(2) << endl;


                     }//ii-for loop

                        } //i-for loop

                             cout << "Spin was flipped for" << n << "
times." << endl;
                             cout << "Spin was not flipped for
"<<nn<<" times."<< endl;
                             cout << "Empty spin was skipped for "<< s
<< " times." << endl;
                             cout << "After the Monte Carlo run, the
energy is " << energy << endl;


                             //recalculate energy

                             refenergy = energycal_scale(spinx, spiny,
spinz,j1scale, j2scale,bextx, m, multk);
                             cout<<"The reference energy is "<<
refenergy<<endl;
                             cout<<"The reference magnetic moment is "
<< m(0) << "," << m(1) << "," << m(2) << endl;

                             cout << "Data Collection." << endl;

                              for(i=0;i<MC2;i++)
                                { //repeat this for the number of
preruns*number of spins

                                for(ii=0;ii<numspin;ii++)
                             {

                                                        ran1 =
xorshift();

r1=ran1*2.3283064e-10;
                                                        pick =
r1*numspin;
                                                        h=pick%nx;

k=(pick%numperlayer)/nx;
                                                        l=
pick/numperlayer;
```

```
                                                                //random spin
generation

if(sqrt(spinx(h,k,l)*spinx(h,k,l)+spiny(h,k,l)*spiny(h,k,l)+spinz(h,k,l
)*spinz(h,k,l))> 0.99)
                                                                {


randomspin_xorshift(sxnew,synew,sznew);


delsx=sxnew-spinx(h,k,l);

delsy=synew-spiny(h,k,l);

delsz=sznew-spinz(h,k,l);


hp = (h + 1)%nx;

hm = (h + nx-1)%nx;

kp = (k + 1)%ny;

km = (k + ny-1)%ny;

lp = (l + 1)%nz;

lm = (l + nz-1)%nz;


eff(0)=0.0;

eff(1)=0.0;

eff(2)=0.0;



                                                        eff(0) = eff(0)+
j1scale(1,l)*(spinx(hp,k,l)+spinx(h,kp,l)+spinx(hm,kp,l)+spinx(hm,k,l)

             +spinx(hp,km,l)+spinx(h,km,l)); //current layer neighbors

eff(1) = eff(1)+
j1scale(1,l)*(spiny(hp,k,l)+spiny(h,kp,l)+spiny(hm,kp,l)+spiny(hm,k,l)

                           +spiny(hp,km,l)+spiny(h,km,l));

eff(2) = eff(2)+
j1scale(1,l)*(spinz(hp,k,l)+spinz(h,kp,l)+spinz(hm,kp,l)+spinz(hm,k,l)

                           +spinz(hp,km,l)+spinz(h,km,l));
```

```
//upper layer


eff(0) = eff(0) +
j1scale(2,l)*(spinx(h,k,lp)+spinx(h,km,lp)+spinx(hm,k,lp));//upper
layer neighbors

eff(1) = eff(1) +
j1scale(2,l)*(spiny(h,k,lp)+spiny(h,km,lp)+spiny(hm,k,lp));

eff(2) = eff(2) +
j1scale(2,l)*(spinz(h,k,lp)+spinz(h,km,lp)+spinz(hm,k,lp));


eff(0) = eff(0) +
j2scale(2,l)*(spinx(hp,km,lp)+spinx(hm,kp,lp)+spinx(hm,km,lp));//upper
neighbor neighbors

eff(1) = eff(1) +
j2scale(2,l)*(spiny(hp,km,lp)+spiny(hm,kp,lp)+spiny(hm,km,lp));

eff(2) = eff(2) +
j2scale(2,l)*(spinz(hp,km,lp)+spinz(hm,kp,lp)+spinz(hm,km,lp));



//lower layer



eff(0) =
eff(0)+j1scale(0,l)*(spinx(h,k,lm)+spinx(hp,k,lm)+spinx(h,kp,lm));//low
er layer neighbors

eff(1) =
eff(1)+j1scale(0,l)*(spiny(h,k,lm)+spiny(hp,k,lm)+spiny(h,kp,lm));

eff(2) =
eff(2)+j1scale(0,l)*(spinz(h,k,lm)+spinz(hp,k,lm)+spinz(h,kp,lm));


eff(0) =
eff(0)+j2scale(0,l)*(spinx(hp,km,lm)+spinx(hm,kp,lm)+spinx(hp,kp,lm));/
/lower layer neighbors

eff(1) =
eff(1)+j2scale(0,l)*(spiny(hp,km,lm)+spiny(hm,kp,lm)+spiny(hp,kp,lm));

eff(2) =
eff(2)+j2scale(0,l)*(spinz(hp,km,lm)+spinz(hm,kp,lm)+spinz(hp,kp,lm));
```

```
b_sum=delsx*bextx;



efff= delsx*eff(0)+delsy*eff(1)+delsz*eff(2)+b_sum; //to remove the
double-count



//efff is the energy difference


if(efff < 0.0)

{

        //update the spin, magnetization, and energy

        //it is possible to update the magnetization too


                energy = energy + efff;


                m(0)=m(0)+delsx;

                m(1)=m(1)+delsy;

                m(2)=m(2)+delsz;


                spinx(h,k,l) = sxnew;

                spiny(h,k,l) = synew;

                spinz(h,k,l) = sznew;



//sign has to be implemented for order parameter calculation


              sign1=(h%2)*2-1;

              sign2=(k%2)*2-1;

              sign3=(l%2)*2-1;

              sign4=((h+k+l)%2)*2-1;


              multk(0,0)=multk(0,0)+double(sign1)*delsx;
```

```
                multk(1,0)=multk(1,0)+double(sign1)*delsy;
                multk(2,0)=multk(2,0)+double(sign1)*delsz;

                multk(0,1)=multk(0,1)+double(sign2)*delsx;
                multk(1,1)=multk(1,1)+double(sign2)*delsy;
                multk(2,1)=multk(2,1)+double(sign2)*delsz;

                multk(0,2)=multk(0,2)+double(sign3)*delsx;
                multk(1,2)=multk(1,2)+double(sign3)*delsy;
                multk(2,2)=multk(2,2)+double(sign3)*delsz;

                multk(0,3)=multk(0,3)+double(sign4)*delsx;
                multk(1,3)=multk(1,3)+double(sign4)*delsy;
                multk(2,3)=multk(2,3)+double(sign4)*delsz;

            n1=n1+1;
      }

else
{
//generate a random number

        ran2 = xorshift();
        r2=ran2*2.3283064e-10;


        if (r2 < exp(-invtemp*efff))

            {
                  //update the spin, magnetization, and energy
                  energy = energy + efff;
```

```
m(0)=m(0)+delsx;
m(1)=m(1)+delsy;
m(2)=m(2)+delsz;
// cout<<"The effective field is "<< efff<<endl;

 //sign has to be implemented for order parameter
calculation

sign1=(h%2)*2-1;
sign2=(k%2)*2-1;
sign3=(l%2)*2-1;
sign4=((h+k+l)%2)*2-1;

multk(0,0)=multk(0,0)+double(sign1)*delsx;
multk(1,0)=multk(1,0)+double(sign1)*delsy;
multk(2,0)=multk(2,0)+double(sign1)*delsz;

multk(0,1)=multk(0,1)+double(sign2)*delsx;
multk(1,1)=multk(1,1)+double(sign2)*delsy;
multk(2,1)=multk(2,1)+double(sign2)*delsz;

multk(0,2)=multk(0,2)+double(sign3)*delsx;
multk(1,2)=multk(1,2)+double(sign3)*delsy;
multk(2,2)=multk(2,2)+double(sign3)*delsz;

multk(0,3)=multk(0,3)+double(sign4)*delsx;
multk(1,3)=multk(1,3)+double(sign4)*delsy;
multk(2,3)=multk(2,3)+double(sign4)*delsz;

//cout <<"The updated energy is "<<energy<<endl;
```

```
                    spinx(h,k,l) = sxnew;

                    spiny(h,k,l) = synew;

                    spinz(h,k,l) = sznew;


                    //cout <<"spin is flipped"<<endl;

                    n1=n1+1;


                }//if loop

                 else nn1=nn1+1; //the spin is not flipped;

            }

            //else loop
                                                          } //if loop
                                                          else  s1=s1+1;


//recalculate energy here


                                                                        //
check for energy and magnetization and etc

                                                           }//ii-for loop


                                                  // statistical calculation

                                                    ensum=ensum+energy;

ensqsum=ensqsum+energy*energy;


magsq=m(0)*m(0)+m(1)*m(1)+m(2)*m(2);

magsqsum=magsqsum+magsq;


kmags0 = multk(0,0)* multk(0,0)+ multk(1,0)*multk(1,0) +
multk(2,0)*multk(2,0);

kmags1 = multk(0,1)* multk(0,1)+ multk(1,1)*multk(1,1) +
multk(2,1)*multk(2,1);

kmags2 = multk(0,2)* multk(0,2)+ multk(1,2)*multk(1,2) +
multk(2,2)*multk(2,2);
```

```
kmags3 = multk(0,3)* multk(0,3)+ multk(1,3)*multk(1,3) +
multk(2,3)*multk(2,3);


ksqsum0 = ksqsum0 + kmags0;

ksqsum1 = ksqsum1 + kmags1;

ksqsum2 = ksqsum2 + kmags2;

ksqsum3 = ksqsum3 + kmags3;

                                                  //magnetization


msumx=msumx+m(0);

msumy=msumy+m(1);

msumz=msumz+m(2);

//reduced magnetization


k0sumx=k0sumx+ multk(0,0);

k0sumy=k0sumy+ multk(1,0);

k0sumz=k0sumz+ multk(2,0);


k1sumx=k1sumx+ multk(0,1);

k1sumy=k1sumy+ multk(1,1);

k1sumz=k1sumz+ multk(2,1);


k2sumx=k2sumx+ multk(0,2);

k2sumy=k2sumy+ multk(1,2);

k2sumz=k2sumz+ multk(2,2);


k3sumx=k3sumx+ multk(0,3);

k3sumy=k3sumy+ multk(1,3);

k3sumz=k3sumz+ multk(2,3);
```

```
                                } //i-for loop

                                cout << "In the data run, the spins were
flipped for "<<n1<<" times."<<endl;
                                cout << "The spins were not flipped for
" << nn1 << " times." << endl;
                           cout << "Empty spins were encounterd for "<<s1<<
" times." << endl;
                               //statistical variable calculations

                                ofstream debug;
                                debug.open("statcheck.dat");
                                debug.close();
                                envar=ensqsum/double(MC2)-
ensum*ensum/double(MC2*MC2);
                           mmags=msumx*msumx+msumy*msumy+msumz*msumz;
                           mvar=magsqsum/double(MC2)-(mmags)/double(MC2);
                                k0var=ksqsum0/double(MC2)-
(k0sumx*k0sumx+k0sumy*k0sumy+k0sumz*k0sumz)/double(MC2*MC2);
                                k1var=ksqsum1/double(MC2)-
(k1sumx*k1sumx+k1sumy*k1sumy+k1sumz*k1sumz)/double(MC2*MC2);
                                k2var=ksqsum2/double(MC2)-
(k2sumx*k2sumx+k2sumy*k2sumy+k2sumz*k2sumz)/double(MC2*MC2);
                                k3var=ksqsum3/double(MC2)-
(k3sumx*k3sumx+k3sumy*k3sumy+k3sumz*k3sumz)/double(MC2*MC2);;


                                ofstream st;
                                st.open("stddev.dat",ios::app);



                                //cout << "The temperature is " <<
temperature << endl;
                                //cout << "The energy varienace is "<<
envar<<endl;
                                //cout <<"The magnetization variance is
" << magvar<<endl

                               st << temperature << "  "<<bextx<<"
"<<envar <<"  "<< mvar << "  " << k0var << "  " << k1var << "  " <<
k2var << "  " << k3var << endl;

                                   st.close();

                              return;
            }

            void optest_l(Matrix<double,2>& multk,Matrix<double,3>&
spinx, Matrix<double,3>& spiny, Matrix<double, 3>& spinz,double
temperature, double bextx,
                              Matrix<double,3>& multkl)
            {
```

```
                Matrix<double, 1> kmag(4);
                Matrix<double,2> kmagl(4,nz);

                kmag=0.0;
                kmagl=0.0;

                multk=0.0; //order parameter
                multkl=0.0;//layer order parameter
            int count=numspin;
                cout <<"optest: the number of spins is "<<count<<endl;
                int iii,ii;
                int h,k,l;
                int numl;
                double sign1,sign2,sign3,sign4;

                ofstream opout;
                ofstream opout2;
                opout.open("optest.dat",ios::app);
                opout2.open("optest_l.dat",ios::app);

                 //magnetization and reduced magnetization can be
calculated from m and op

                for(l=0;l<nz;l++)
                  {
                    for(k=0;k<ny;k++)
                    {
                      for(h=0;h<nx;h++)
                        {
                            //implement the periodic boundary condition



                            //order parameter

                            sign1=double(h%2)*2.0-1.0;
                            sign2=double(k%2)*2.0-1.0;
                            sign3=double(l%2)*2.0-1.0;
                            sign4=double((h+k+l)%2)*2.0-1.0;

                           if(sign1!=1.0 && sign1!=-1.0)cout<<"wrong
sign!"<<endl;

                            //regular order parameter calculation

                            multk(0,0)=multk(0,0)+sign1*spinx(h,k,l);//x-
comp
                            multk(1,0)=multk(1,0)+sign1*spiny(h,k,l);//y-
comp
                            multk(2,0)=multk(2,0)+sign1*spinz(h,k,l);//z-
comp

                            multk(0,1)=multk(0,1)+sign2*spinx(h,k,l);
```

```
                              multk(1,1)=multk(1,1)+sign2*spiny(h,k,l);
                             multk(2,1)=multk(2,1)+sign2*spinz(h,k,l);

                             multk(0,2)=multk(0,2)+sign3*spinx(h,k,l);
                             multk(1,2)=multk(1,2)+sign3*spiny(h,k,l);
                             multk(2,2)=multk(2,2)+sign3*spinz(h,k,l);

                             multk(0,3)=multk(0,3)+sign4*spinx(h,k,l);
                             multk(1,3)=multk(1,3)+sign4*spiny(h,k,l);
                             multk(2,3)=multk(2,3)+sign4*spinz(h,k,l);

                           //layer order parameter calculation


multkl(0,0,l)=multkl(0,0,l)+sign1*spinx(h,k,l);

multkl(1,0,l)=multkl(1,0,l)+sign1*spiny(h,k,l);

multkl(2,0,l)=multkl(2,0,l)+sign1*spinz(h,k,l);


multkl(0,1,l)=multkl(0,1,l)+sign2*spinx(h,k,l);

multkl(1,1,l)=multkl(1,1,l)+sign2*spiny(h,k,l);

multkl(2,1,l)=multkl(2,1,l)+sign2*spinz(h,k,l);


multkl(0,2,l)=multkl(0,2,l)+sign3*spinx(h,k,l);

multkl(1,2,l)=multkl(1,2,l)+sign3*spiny(h,k,l);

multkl(2,2,l)=multkl(2,2,l)+sign3*spinz(h,k,l);


multkl(0,3,l)=multkl(0,3,l)+sign4*spinx(h,k,l);

multkl(1,3,l)=multkl(1,3,l)+sign4*spiny(h,k,l);

multkl(2,3,l)=multkl(2,3,l)+sign4*spinz(h,k,l);

                        }
                    }
                }
                     for(iii=0;iii<2;iii++)
                     {
                       //regular order parameter
                       kmag(0)=kmag(0)+multk(iii,0)*multk(iii,0);
                       kmag(1)=kmag(1)+multk(iii,1)*multk(iii,1);
                       kmag(2)=kmag(2)+multk(iii,2)*multk(iii,2);
                       kmag(3)=kmag(3)+multk(iii,3)*multk(iii,3);

                           for(ii=0;ii<nz;ii++)
                             {
```

```
kmagl(0,ii)=kmagl(0,ii)+multkl(iii,0,ii)*multkl(iii,0,ii);

kmagl(1,ii)=kmagl(1,ii)+multkl(iii,1,ii)*multkl(iii,1,ii);

kmagl(2,ii)=kmagl(2,ii)+multkl(iii,2,ii)*multkl(iii,2,ii);

kmagl(3,ii)=kmagl(3,ii)+multkl(iii,3,ii)*multkl(iii,3,ii);
                           }


                     }


                     //normalization

                    //total order parameter

                    cout<<"Before the normalizaion, the k vectors
are:"<<endl;
                      cout << "(k0):"<<sqrt(kmag(0))<<endl;
                      cout << "(k1):"<<sqrt(kmag(1))<<endl;
                      cout << "(k2):"<<sqrt(kmag(2))<<endl;
                      cout << "(k3):"<<sqrt(kmag(3))<<endl;

                         kmag(0)=sqrt(kmag(0))/double(count);
                         kmag(1)=sqrt(kmag(1))/double(count);
                         kmag(2)=sqrt(kmag(2))/double(count);
                         kmag(3)=sqrt(kmag(3))/double(count);

                        numl=nx*ny; //number of spins per layer;
                    //layer order parameter
                        for(ii=0;ii<nz;ii++)
                          {
                              kmagl(0,ii)=sqrt(kmagl(0,ii))/double(numl);
                              kmagl(1,ii)=sqrt(kmagl(1,ii))/double(numl);
                              kmagl(2,ii)=sqrt(kmagl(2,ii))/double(numl);
                              kmagl(3,ii)=sqrt(kmagl(3,ii))/double(numl);
                            }


                   opout <<  temperature << "  " << kmag(0)<<"
"<<kmag(1)<<"  "<<kmag(2)<<"  "<<kmag(3)<<endl;
                   for(ii=0;ii<nz;ii++)
                     {
                     //opout2<<"The layer number is "<<ii<<endl;
                       opout2  << ii << "  " << temperature << "  " <<
kmagl(0,ii)<<"  "<<kmagl(1,ii)<<"  "<<kmagl(2,ii)<<"  "<<kmagl(3,ii) <<
endl;
                     }
                   cout  << temperature <<"  "<<kmag(0)<<"  "<<kmag(1)<<"
"<<kmag(2)<<"  "<<kmag(3)<<endl;
                   opout.close();
                   opout2.close();
```

```
            return;
        }
```

J2. Structure factor program for MnSe/ZnTe superlattice model

```
#include <math.h>
#include <iostream>
#include <fstream>
#include "Matrix.h"
#include "MatrixIO.h"

/*
 * There are two versions of this program
 * 3d spin array
 *
 *
 * Created Date: February 19, 2009 (Thursday)
 * Author: Seongweon Park
 *
 * Program description:
 *
 *  This program gets two input files.
 *  One is the spin input file and the other is the scan parameter
 *  file. Once it gets the input files, it calculates the structure
 *  factor of the spin system.
 *
 *  Algorithm description:
 *
 *  The spin input files has the spin vectors (sx,sy,sz) and if it has
 *  N nonzero spins (each spin file starts with 5 lines of zero spins.),
 *  the program converts N into the spin location coordinate (u,v,w).
 *
 *  The structure factor is written as the following:
 *  F=sum(Sj*exp(2*pi*i*(huj+kvj+lwj))),
 *  where the q=(h,k,l) and r=(uj,vj,wj) and pi=3.141592... and i is
the
 *  imaginary part in this case.

 *  ************************************************************
 *  The q vector parameters are input manually.
 *
 *  ************************************************************
 *  To figure out the spin projected onto the plane
 *  perpendicular to the q vector:
 */
```

```
using namespace std;
using namespace Numeric_lib;

//function declaration

void print_qvector(Matrix<double, 3> & qvx, Matrix<double, 3> & qvy,
Matrix<double, 3> & qvz);
void coordinateconverter(int flag);
void structure_factor(Matrix<double, 3> & spinx, Matrix<double, 3> &
spiny, Matrix<double, 3> & spinz, Matrix<double, 3> & qvx,
                          Matrix<double, 3> & qvy, Matrix<double, 3> &
qvz, double px, double py, double pz);
void print_spin(Matrix<double, 3> & spinx, Matrix<double, 3> & spiny,
Matrix<double, 3> & spinz);
double read_spin(char filename[], Matrix<double, 3> & spinx,
Matrix<double, 3> & spiny, Matrix<double, 3> & spinz);
int file_counter(char filename[]);

int numqx, numqy, numqz;
int numnzspin;
int nx,ny,nz;
int hmin, hmax, kmin, kmax, lmin, lmax;

int main() {

      double qxi, qyi, qzi; //initial q values
      double delqx, delqy, delqz; //q step sizes
      double qx, qy, qz;
    double qbx,qby,qbz; //cross product
    double spinv, spinh;

      // Coordinate dimension of the spin system

      int n;
      int numperlayer;
      int flag = 9;

      int count1;
      int h, k, l;
      int i, ii, iii;

      char filename[15], spinfile[25]; //file names of the spin file

      cout << "Getting the spin input file." << endl;

      cout << "Enter the input spin file name: ";
      cin >> filename;

      count1 = file_counter(filename);

      cout << "There are " << count1 << " nonzero spins." << endl;

      ifstream qs;
```

```
      qs.open("qspec3dt.dat");

      qs >> qxi >> qyi >> qzi;
      qs >> delqx >> delqy >> delqz;
      qs >> numqx >> numqy >> numqz;

      qs >>nx >> ny >> nz;
      qs.close();

      cout << "qxi: "<< qxi <<" qyi: "<< qyi <<" qzi: "<< qzi<<endl;
      cout << " delqs: "<< delqx << " delqy: "<< delqy << " delqz: " <<
delqz << endl;
      cout << "numqx: "<< numqx <<" numqy: "<< numqy <<" numqz: "<<
numqz << endl;
    cout << "nx: "<< nx <<" ny: "<< ny <<" nz:  "<< nz<<endl;

     numnzspin=nx*ny*nz;

     cout << "The total number of spins is " << numnzspin << endl;

    Matrix <double, 3> spinx(nx, ny, nz);
    Matrix <double, 3> spiny(nx, ny, nz);
    Matrix <double, 3> spinz(nx, ny, nz);

      Matrix<double, 3> qvx(numqx, numqy, numqz);
      Matrix<double, 3> qvy(numqx, numqy, numqz);
      Matrix<double, 3> qvz(numqx, numqy, numqz);

      qvx = 0.0;
      qvy = 0.0;
      qvz = 0.0;

      // load the qvector array with values

      for (i = 0; i < numqz; i++) {
            for (ii = 0; ii < numqy; ii++) {
                  for (iii = 0; iii < numqx; iii++) {

                        qvx(iii, ii, i) = qxi + double(iii) * delqx;
                        qvy(iii, ii, i) = qyi + double(ii) * delqy;
                        qvz(iii, ii, i) = qzi + double(i) * delqz;
                  }
            }
      }

   print_qvector(qvx, qvy, qvz);

   count1 = read_spin(filename, spinx, spiny, spinz);

   if (numnzspin != count1) cout <<"Something is wrong."<<endl;
      cout << "There are " << count1 << " spins loaded." << endl;

      print_spin(spinx,spiny,spinz);
```

```
      coordinateconverter(flag);
      cout<<"Coordinates are all printed out"<<endl;
      structure_factor(spinx, spiny, spinz, qvx, qvy, qvz,px,py,pz);

      return 0;

}

int file_counter(char filename[])
{
      int num = 0;
      int i, j;
      double sx, sy, sz;

      ifstream in_file1;
      in_file1.open(filename);

      while (in_file1 >> sx >> sy >> sz) {
            num = num + 1;
      }

      in_file1.close();

      cout << "The file " << filename << "  has " << num << " lines.
\n";
      return num;
}

double read_spin(char filename[], Matrix<double, 3> & spinx,
Matrix<double, 3> & spiny, Matrix <double,3> & spinz) {

      int spincount = 0;
      //number of nonzero spins

      int h,k,l;
      double sx, sy, sz;
    int nxny=nx*ny;

      ifstream inspin;
      inspin.open(filename);

      while (inspin >> sx >> sy >> sz) {
            h= spincount%nx;
            k= (spincount%nxny)/nx;
            l= spincount /nxny;

            spinx(h,k,l) = sx;
            spiny(h,k,l) = sy;
            spinz(h,k,l) = sz;

            spincount = spincount + 1;
      }
```

```
      cout << "There are " << spincount << " nonzero spins in the
file." << endl;
      cout << "Spins are loaded." << endl;
      inspin.close();

      return spincount;

}

void print_spin(Matrix<double, 3> & spinx,Matrix <double, 3> & spiny,
Matrix <double, 3> & spinz ) {

            int i;
      int flag;
      int h, k, l;
      int numperlayer = nx*ny;

      ofstream spinp;
      spinp.open("pspin.out");

      spinp << "The stored values of the spins are the following:" <<
endl
                  << endl;

      for (i = 0; i < numnzspin; i++)
      {

            h=i%nx;
            k=(i%numperlayer)/nx;
            l=i/numperlayer;

            spinp << i << "  " << h <<"  " << k << "  " << l << "    "
<< spinx(h,k,l) << "   " << spiny(h,k,l) << "    " << spinz(h,k,l) <<
endl;
      }
      // return to the main function after printing out the spins

      spinp << "Printing spins ended." << endl;
      spinp.close();

      return;

}

void structure_factor(Matrix<double, 3> & spinx,Matrix <double,3> &
spiny, Matrix<double, 3> & spinz, Matrix<double, 3> & qvx
            , Matrix<double, 3> & qvy, Matrix<double, 3> & qvz,double
px, double py, double pz) {

      /*
       * The structure factor is given by
F=sumj(Pj*exp{2pi*i*(qx*x+qy*y+qz*z)}
       * Fxi,Fyi,Fzi, Fxr,Fyr,Fzr (real part vector and imaginary
vector)
```

```
       */

      //spin coordinates
      int h, k, l;
      double u, v, w;
      double Arg, si, co;
      double qmag; // to normalize q vectors

    double sdotq, spprx, sppry, spprz;
    double c0, s0, c1, s1, c2, s2;
      /*
       * Arg is the power in the exponential
       * and the expr and expi are the real part and the imaginary part
of the exponential respectively
       */

      double sumfxi, sumfyi, sumfzi; //imaginary parts vector
      double sumfxr, sumfyr, sumfzr; //real part vector
      double fxi, fyi,fzi;
      double fxr, fyr,fzr;
      double pi = 3.14159265358979;
      double Fabsx, Fabsy, Fabsz;
      int h,k,l;

      int j, jj, jjj;
      int i;
      int br; //break point
      int scount = 0;
      double qunx, quny, qunz;
      double sdotq, spprx, sppry, spprz;

      double numperregion = (hmax-hmin)*(kmax- kmin)*(lmax-lmin);
      //number of nonzero spins per layer
      numperrow = nx; //number of nonzero spins per row

      ofstream strf3;

      strf3.open("strab.dat");

   cout << "The Structure factor function."<<endl;
   cout << "The number of spins is " << numnzspin << endl;

                    pmag=sqrt(px*px+py*py+pz*pz);
                    px=px/pmag;
                    py=py/pmag;
                    pz=pz/pmag;

      for (j = 0; j < numqz; j++) {

            for (jj = 0; jj < numqy; jj++) {

                  for (jjj = 0; jjj < numqx; jjj++) {
```

```
qmag=sqrt(qvx(jjj,jj,j)*qvx(jjj,jj,j)+qvy(jjj,jj,j)*qvy(jjj,jj,j)+qvz(j
jj,jj,j)*qvz(jjj,jj,j));
                                        qunx=qvx(jjj,jj,j)/qmag;

      quny=qvy(jjj,jj,j)/qmag;

      qunz=qvz(jjj,jj,j)/qmag;


                                  sumfxi = 0.0;
                                      sumfyi = 0.0;
                                      sumfzi = 0.0;

                                      sumfxr = 0.0;
                                      sumfyr = 0.0;
                                      sumfzr = 0.0;

                                for (l = lmin; l < lmax; l++)
{ //calculate the coordinate of the spin

                                  for(k = kmin; k < kmax; k++)
                                  {
                                        for(h=hmin;h<hmax; h++)
                                        {

                                            l = i / (numperlayer);
                                              k = (i % numperlayer) / nx;
                                              h = i % numperrow;

                                  u = (double(k-h))*0.5;
                                  v = (double(k+h+l))*0.5;
                                  w = double(l)*0.5;

                                  sdotq = spinx(h,k,l)*qunx +
spiny(h,k,l)*quny + spinz(h,k,l)*qunz;
                                  spprx = sdotq*spinx(h,k,l)-spinx(h,k,l);
                                  sppry = sdotq*spiny(h,k,l)-spiny(h,k,l);
                                spprz = sdotq*spinz(h,k,l)-spinz(h,k,l);


                                  Arg = 2.0 * pi * (u * qvx(jjj, jj, j) + v
* qvy(jjj, jj, j) + w * qvz(jjj, jj, j));
                                  si = sin(Arg); //imaginary part
                                  co = cos(Arg); //real part

                                  fxi = spprx * si; // imaginary part
                                  fyi = sppry * si; // imaginary part
                        fzi = spprz * si; // imaginary part
                                  fxr = spprx * co; // real part
                                  fyr = sppry * co; // real part
                        fzr = spprz * co; // real part
                                  sumfxi = sumfxi + fxi;
                                  sumfyi = sumfyi + fyi;
```

```
                              sumfzi = sumfzi + fzi;


                              sumfxr = sumfxr + fxr;
                              sumfyr = sumfyr + fyr;
                            sumfzr = sumfzr + fzr;

                          }

                          Fabsx = sumfxi * sumfxi + sumfxr * sumfxr;

                          Fabsy =sumfyi * sumfyi + sumfyr * sumfyr;

                          Fabsz = sumfzi*sumfzi + sumfzr * sumfzr;


                         strf3 << qvx(jjj, jj, j) << "  " << qvy(jjj, jj,
j) << "  " << qvz(jjj, jj, j) << "  " << Fabsx << "  "<<Fabsy<<"  "<<
((Fabsx+Fabsy+Fabsz)*100.0)/double(numperregion*numperregion) << endl;
                          cout << qvx(jjj, jj, j) << "  " << qvy(jjj, jj,
j) << "  " << qvz(jjj, jj, j) << "  " << Fabsx << "  "<<Fabsy<<"  "<<
((Fabsx+Fabsy+Fabsz)*100.0)/double(numperregion*numperregion) << endl;


                                              }
               }
      }

      strf3.close();

      return;
}

void coordinateconverter(int flag) {

      double u, v, w; //the spin coordinates
      int h, k, l;
      //the spin loading coordinates as in my Monte Carlo Program
      int numperlayer; //number of nonzero spins per row and per layer
      int i;


      //num1=number of nonzero spins per layer
      //num2=number of nonzero spins per line

      numperlayer = nx*ny;
      cout << "The number of nonzero spins per layer is " <<
numperlayer << endl;
    ofstream coord;
    coord.open("ccheck.dat");


      for(i=0;i<numnzspin;i++) {
```

```
            coord << "The current array index number is " << i << endl;

            l =  i / numperlayer;
            k =  (i % numperlayer) / nx;
            h =  (i % nx);

          coord << "(h,k,l)=(" << h << ", " << k << ", " << l << ")" <<
endl;
            u = (double(k-h))*0.5;
            v = (double(k+h+l))*0.5;
            w = double(l)*0.5;

            coord << "The corresponding spin coordinate (u,v,w)=(" << u
<< ", " << v << " ," << w << " )" << endl;

      }
     coord.close();
      return;
}

void print_qvector(Matrix<double, 3> & qvx, Matrix<double, 3> & qvy
            , Matrix<double, 3> & qvz)
            {

      int j, jj, jjj;
      cout << "Printing out the q vector set" << endl;

      ofstream qbug;
      qbug.open("qprint.dat");

      for (j = 0; j < numqz; j++) {
            for (jj = 0; jj < numqy; jj++) {
                  for (jjj = 0; jjj < numqx; jjj++) {
                        qbug << "Index: " << jjj << "," << jj << "," <<
j << endl;
                        qbug << "(qx,qy,qz)=(" << qvx(jjj, jj, j) <<
","
                                    << qvy(jjj, jj, j) << "," <<
qvz(jjj, jj, j) << ",)"
                                    << endl;
                  }
            }
      }

      qbug.close();
      return;
}
```