Spin Liquid

A M.SC Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Masters of Physics

by

Rajesh Mishra
(182121034)

under the guidance of

Charudatt Kadolkar

to the

DEPARTMENT OF PHYSICS

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

CERTIFICATE

*This is to certify that the work contained in this thesis entitled "**Spin Liquid** " is a bonafide work of **Rajesh Mishra** (**Roll No. 182121034**), carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Charudatt Kadolkar**

Assistant/Associate Professor,

May, 2020 Department of Physics

Guwahati. Indian Institute of Technology Guwahati, Assam.

Declaration

This is to declare that the project report titled **"Spin Liquid "**, submitted by me to the department of Physics, Indian Institute of Technology Guwahati, for the partial fulfilment of the requirement for the degree of bachelor of technology is a bonafide work carried out by me under the supervision of **"Charudatt Kadolkar "**. The content of this work, in full or in parts, has not been submitted elsewhere for the award of any other degree or diploma. I also declare that this report is based on my personal study and/or research and I have acknowledged all materials and resources used in its preparation.

Signature of student:

Name :Rajesh Mishra

Roll Number:182121034

Department: Phyiscs

Date:

Acknowledgements

Pretty much always science progresses through collaborative efforts.I have also benefited immensely form the help of my teacher, project instructor Charudatt Y. Kadolkar , my senior Kallol Da and Durgesh bhaiya

Contents

List of Figures

List of Tables

Abstract

In this report, I am going to exactly exactly diagonalise Heisenberg interaction matrix $H = \sum_{(i,j)} S_i.S_j$, where (i, j) is set of all interacting pairs.We shall compute correlations on triangular lattice.

Overview

At low temperature, To describe phase change we use the concept of symmetry breaking. It is best explained by an example,Consider the Heisenberg antiferromagnet Hamiltonian $H = \sum_{(i,j)} S_i.S_j$ .If system chooses the phase transition by symmetry breaking, We can always get an order parameter,In this case it is $< S_i >= \vec{m}_i$. We can write

$$H = -J \sum_{(i,j)} S_i.S_j$$

$$H = -J \sum_{(i,j)} (S_i - \vec{m}_i + \vec{m}_i).(S_j - \vec{m}_j + \vec{m}_j)$$

$$H = -J \sum (i,j) \vec{m}_i.\vec{m}_j + (S_i - \vec{m}_i).\vec{m}_j + (S_j - \vec{m}_j).\vec{m}_i + (S_i - \vec{m}_i).(S_j - \vec{m}_j)$$

neglecting the fluctuation term and constant term , we get

$$H = -J \sum (i,j) \vec{m}_i.(S_j - \vec{m}_j)$$

This is system with only one spin and can be solved exactly and we get get two different mean field solutions , either all spin are up or down , and other solution is consecutive up down states In case of anti-ferromagnetic latter is the case.

Now, We have certain systems , in which either Hamiltonian does not possess any symmetry,For such systems mean field method does not work as there is no order parameter,QSL is part of these type of system. Consider we have degenerate ground state of

some Hamiltonian, If we can go adiabatically without any level crossing , we can say that all such collection of ground states are in one phase. These phases differ in the topology of ground state.There are different ways in which these system can be studied by parton mean field theory , symmetry and projection groups and through exactly solvable models.

Aim

To exactly diagonalise the Heisenberg Interaction Hamiltonian $H = \sum_{(i,j)} S_i.S_j$ [(i,j) is set of all possible pairs in spin-paired basis/rumer basis And to calculate correlation function efficiently. Also to study properties and implications for correlation function.

Reason For Selecting the problem

In physics basic principles that govern interaction of two atoms are well understood but they can't be used to describe the properties of bulk matter directly.In fact many problems in physics boils down to finding good approximate solution of many body interaction.Through quantum spin liquid we can understand basic theoretical technique , use of computer in solving such problems.properties of quantum spin liquid can help us to get better understanding of superconductivity and other exotic low temperature phenomenon.

Description of problem

Basic building block of that interact in quantum spin liquid is (SU_2) and rationally invariant singlets/valence bond Ground state is product of such singlets,We write down the interaction Hamiltonian and diagonalise it , we calculate the interaction energy of various small collection of spin eg.spin forming triangle , quadrilateral , hexagon We also find the correlation between two spin with varying distance and other observable

Future Outlook

In this project, we are studying the properties of QSL on 2D triangular lattice with nearest neighbour interaction .This problem have many symmetries which simplifies the calculation a lot . Our immediate future goal is to study is in which we also allow next nearest neighbour interactions, or coupling constant is tensor instead of constant for various other 2d and 3d lattice.One more goal is to check if system have ground state in which we have loop of entangled spin and these spin loops can't be unlooped without breaking the loop.Such arrangement of spin can sustain long range entanglement at the same time it will have decreasing correlation

Chapter 1

Review of Prior Works/Literally survey

The term 'quantum spin liquid' (QSL) originates from Anderson's 1973 paper describing a 'quantum liquid' ground state of an anti-ferromagnetic material [And73] and [And87]. Initial idea was to write the ground state of anti-ferromagnet ground state in terms of products of singlet it was not well received initially but in 1987 when Anderson[And87],Bhaskarn,Zou, Anderson [PA87] showed that superconductivity might emerge due to doping in QSL,and also identified the underlying 'pseudo-fermions', now usually called 'spinons', as possible quasi particles of the spin- 1/2 QSL. In parallel, Kalmeyer and Laughlin [Vl87] proposed a 'chiral spin liquid' state, which connected QSLs to the fractional quantum Hall effect (FQHE), and thereby to topology. In the same year, Kivelson Rokhsar and Sethna [DSS] also suggested that topology plays a role in the structure of RVB states, and in 1988 Rokhsar and Kivelson [DS] introduced the quantum dimer model on the square lattice which made a topological structure transparent. Unfortunately, the QSL phase in that model soon proved to be unstable. Though there were many theoretical ideas yet large community was skeptic of RVB paradigm if they can occur in physical systems.

Around 2000, Subject received another break through Senthil and Fisher [TMa],[TMb]

connected the existence of spinons with fractional quantum hall effect and topological order in certain QSL state to Wigner's Ising Lattice gauge theory [FJ],[JB], Whose deconfined phase were known experimentally .In 2001, Moessnor and Shondhi [MS] proposed Quantum dimer model on triangular lattice and showed that for such model stable QSL phase exist. In 2003 ,Kitaev showed that topological phases of QSL can be used in quantum computation through his toric model [AYa]. In 2006, Kitaev [AYb] introduced his honeycomb model and also found exact solution which explains gap less QSL phase. After this paper there is no doubt that QSL phases exists only challenge is to realize that experimentally.

Chapter 2

Representing many spin state

We are trying to study the system in which there are many spins interacting. It is necessary for us to first get some basics about how we can represent such many spin states.For n spins we have 2^n ways to choose set of basis. we will discuss only the most useful one's

2.1 S^2 Basis

Consider Hilbert space defined by $\mathtt{H} = S_1 \bigotimes S_2 \bigotimes ... \bigotimes S_n$ If we have single spin we can two states α and β .To form basis set for N spins we need to take tensor product of N such states. If sayα as 1 and β as 2. Sequence 12221 represent tensor product of $\alpha(1)$,$\beta(2)$ ... $\alpha(5)$.We can see there are 2^N such vectors.Now these vector are eigenvectors of S_z but not S^2.To get the eigenvector of S^2 we need diagonals S^2 matrix.Given

$$\sigma_x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \sigma_y = \begin{pmatrix} 0 & -i \\ i & 0 \end{pmatrix}, \sigma_z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

These operator are in basis of single spin Hilbert space.Let's first represent spins in Hilbert space of n spin

$$S_x(i) = ... \bigotimes S_x(i^{th} position) .. \bigotimes$$

$$S_x = \sum iS_x(i)$$

$$S_y = \sum iS_y(i)$$

$$S_z = \sum iS_z(i)$$

similarly,

$$S_- = \sum iS_-(i)$$

$$S_+ = \sum iS_+(i)$$

$$S^2 = S_+S_- + {S_z}^2 - S_z = S_-S_+ + {S_z}^2 + S_z$$

We also have one extremely important relation called Dirac identity, **P** is permutation operator for spin i and j

$$S(i)S(j) = 1/2(\mathbf{P}_{ij} - 1/2\mathbf{I})$$

2.1.1 $\mathbf{S}_z$*basis*

We define primitive functions as $\theta_i = \theta(1)...\theta(N)$ where $\theta(j)$is either α or β

$S_z = \theta_i(\mu, \nu) = \frac{1}{2}(\mu - \nu)\theta_i(\mu, \nu)$

where μ is number of α and ν is number of β

$[\alpha^\mu \beta^\nu] = Sum of all primitive functions$

Properties of Bracket:

$$S_+ = [\alpha^\mu \beta^\nu] = (\mu + 1)[\alpha^{\mu+1}\beta^{\nu-1}]$$

$$S_- = [\alpha^\mu \beta^\nu] = (\nu + 1)[\alpha^{\mu-1}\beta^{\nu+1}]$$

$$S_+{}^k = [\alpha^\mu \beta^\nu] = \frac{(\mu+k)!}{\mu!}[\alpha^{\mu+k}\beta^{\nu-k}]$$

$$S_-{}^k = [\alpha^\mu \beta^\nu] = \frac{(\nu+k)!}{\nu!}[\alpha^{\mu-k}\beta^{\nu+k}]$$

$$S^2 = [\alpha^\mu \beta^\nu] = \frac{N}{2}(\frac{N}{2}+1)[\alpha^{\mu+1}\beta^{\nu-1}]$$

$$S_z[\alpha^\mu \beta^\nu] = \frac{1}{2}(\mu-\nu)[\alpha^\mu \beta^\nu]$$

2.1.2 S^2 eigenfunction by direct diagonalisation

This is the most straight forward and tedious way to find the eigenvalues and eigenvector of S^2 matrix . see Appendix 1 for program

2.1.3 Using Orthogonalisation Procedure

S^2 is hermitian matrix so all eigenvectors are orthogonal for $S = \frac{N}{2}$ there is just one vector i.e. $\alpha\alpha..ntimes$ for $S = \frac{N}{2} - 1$, If you choose

$$X_1 = \frac{1}{\sqrt{N}}(\theta_1 + \theta_2 + .. + \theta_n)$$

we can choose other N-1 vectors as

$$X_i = \frac{1}{\sqrt{(N-i+1)(N-i+2)}}[(N-i+1)\theta_{i-1} - (\theta_i + .. + \theta_n)]$$

Problem with this method is that it can only be used when eigenvalues of S^2 are non-degenerate and procedure is based on proper setting of X_1 vector so it may be difficult to

get this for arbitrary S^2 matrix

2.1.4 Dimension of spin degeneracy

Consider a primitive function with $M = \frac{1}{2}(\mu - \nu)$ we have $\begin{pmatrix} N \\ (N/2) - S - 1 \end{pmatrix}$ primitive functions similarly we take primitive function of eigenvalue $M + 1$ these two subspace orthogonal S^2 subspace with $S^2 = M^2$ hence number of degenerate state given $f(N, S) = \begin{pmatrix} N \\ (N/2) - S \end{pmatrix}$ - $\begin{pmatrix} N \\ (N/2) - S - 1 \end{pmatrix}$

$$\sum S(2S + 1)f(N, S) = 2^N$$

2.1.5 Construction using genealogical method

In this method we start with eigenstate of single spin and systematically add and subtract using addition and subtraction formula , this way we can write branching symbol as linear combination of path symbols.program for finding is in appendix one

Addition formula: $X(N, S, M; B_i) = C(1, 1, S, M)X(N - 1, S - 1/2, M - 1/2, B_j)\alpha(N) + C(1, 2, S, M)X(N - 1, S - 1/2, M + 1/2, B_j)\beta(N)$ where,

$$C(1, 1, S, M) = \sqrt{\frac{S + M}{2S}}$$

and

$$C(1, 2, S, M) = \sqrt{\frac{S - M}{2S}}$$

Subtraction formula : $X(N, S, M; B_i) = C(2, 1, S, M)X(N - 1, S + 1/2, M - 1/2, B_j)\alpha(N) +$

$C(2, 2, S, M)X(N - 1, S + 1/2, M + 1/2, B_j)\beta(N)$

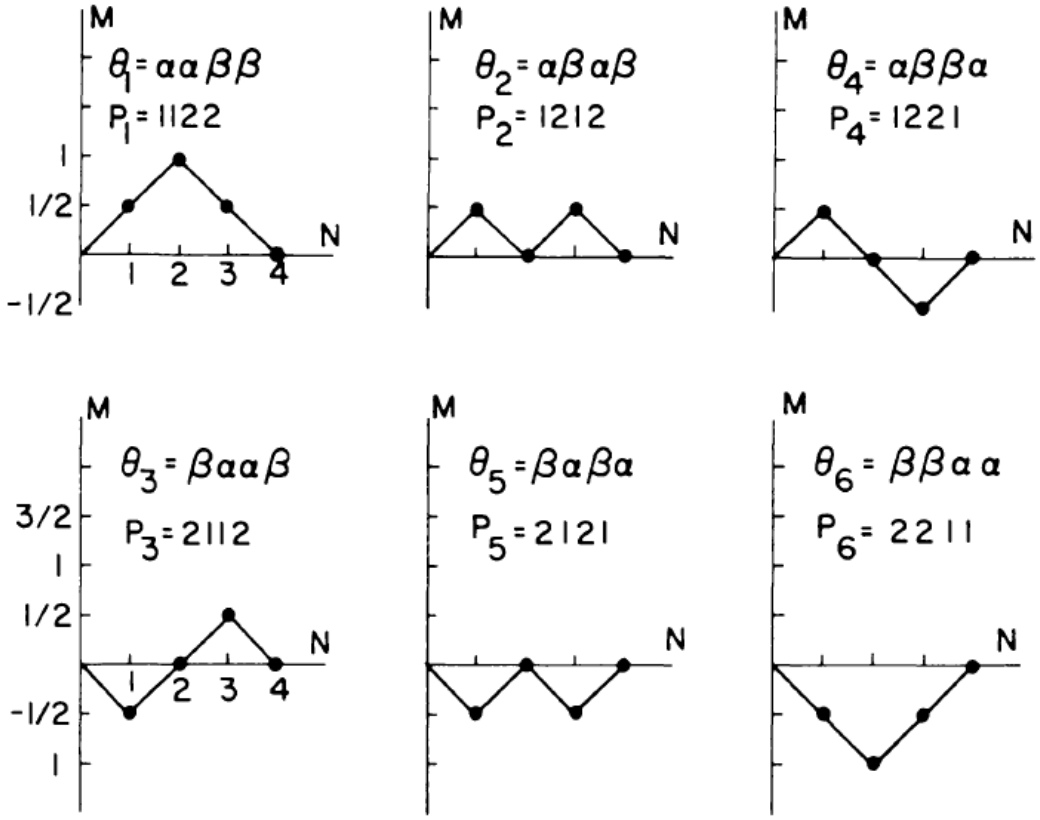
where,

$$C(2,1,S,M) = -\sqrt{\frac{S-M+1}{2S+2}}$$

and

$$C(2,2,S,M) = \sqrt{\frac{S+M+1}{2S+2}}$$

Path Diagrams:In this method we start with state and add or subtract α and β we can represent them pictorially by taking $+45^0$ and -45^0 for up and down spin respectively these diagrams are called path diagram if we take up spin as 1 as down spin as 2 ,we get sequence of 1,2 for each path diagram , they are path symbols , they are arranged from right to left and 2 is considered greater than 1 for eg sequence 12 is greater than 11 this arrangement is called last latter sequence



Branch Diagrams: Branching diagrams are similar to path diagrams except that S can't be negative ,it represent sequence of steps to reach a point N,S in N-S plane they are writ-

ten as $X(N, S, M, b_i)$ where

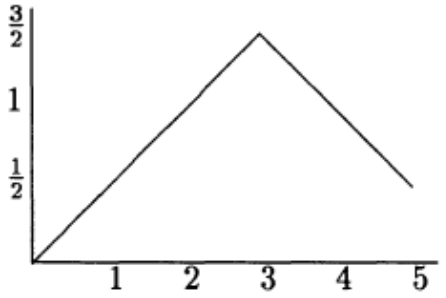
N- total number of spin

S- total spin

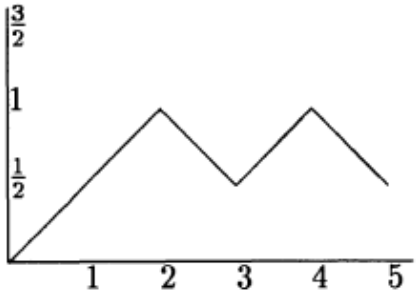
M - S_z of total spin

b_i- sequence of steps to reach point N,S in N-S plane

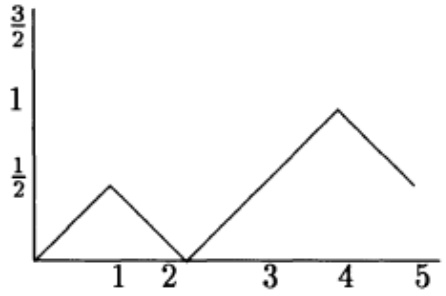
Number of branching symbol is given $f(N, S)$. Set of eigenvector given by branching symbols / branching diagrams for orthonormal basis. Each branching symbol corresponds to one rumer diagram this property we will exploit in complete set of rumer basis.For eg



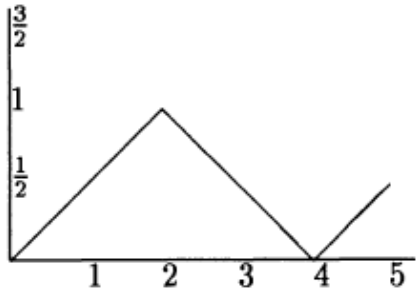
[11122]



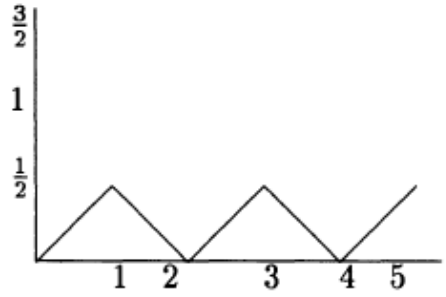
[11212]



[12112]



[11221]



[12121]

2.2 Serber Basis

Serber basis N spin eigenfunction is constructed by combining N-2 and 2 spin eigenfunction either by adding or subtracting singlet or triplet.Construction of Serber basis follows the same line of arguments as genealogical construction hence we need to define similar quantities.Like the genealogical construction we can write S_z's eigenfunction as product of singlets and triplets (also called geminal products) these are called serber path symbols and S^2's eigenfunction by serber branch diagrams
Let's first define geminal product for two spins

$$g_0(N-1,N) = \frac{1}{\sqrt{2}}[\alpha(N-1)\beta(N) - \beta(N-1)\alpha(N)]$$

$$g_1(N-1,N) = \alpha(N-1)\alpha(N)$$

$$g_2(N-1,N) = \frac{1}{\sqrt{2}}[\alpha(N-1)\beta(N) + \beta(N-1)\alpha(N)]$$

$$g_3(N-1,N) = \beta(N-1)\beta(N)$$

2.2.1 Serber path symbols

The geminal product functions can represented by their Serber path diagrams the horizontal axis is the number of pairs, the vertical axis is the S_z quantum number. The geminal function g_0 is represented by a horizontal dotted line, g_2 by a horizontal full line, g_1 by an arrow in the direction of $+45$ degree, and g_3 by an arrow in the direction of -45 degree.This is best explained by eg. (figure2.1)

2.2.2 Serber Branching diagram

In Serber Branching diagram,Horizontal axis is number of pair and vertical axis is total spin quantum number S.We are adding N-2 eigenfunction with 2 eigenfunction there are four ways in which we can add this

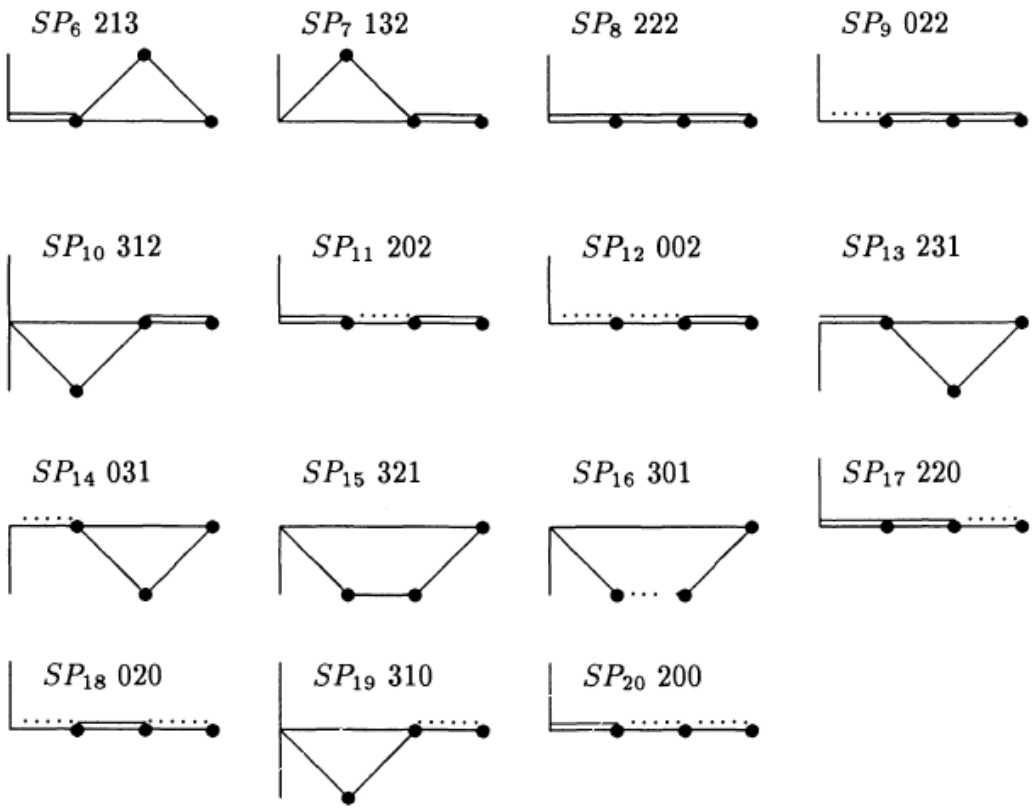


Figure 2.1 Serber path diagrams and their symbols for $N = 6, S = 0$.

1. $S \hookrightarrow S$ we are adding a singlet g_0. this is represent by dotted horizontal line line
2. $S \hookrightarrow S$ we are adding triplet g_2. this is represented by solid horizontal line .
3. $S \hookrightarrow S + 1$ we are adding triplet g_1. this is represented by solid line with $+45$ degrees
4. $S \hookrightarrow S - 1$ we are adding triplet g_3. this is represented by solid line with -45 degrees

This is best explained by eg (figure 2.2)

2.2.3 Addition and Subtraction formula

We can write serber branch diagrams as linear combination of serber path diagrams and to find the coefficients we have the addition and subtraction formula like we had for

genealogical functions

1. $S \hookrightarrow S$

$$Z(N,S,M,k) = Z(N-2,S,M,k')g_0(N-1,N)$$

2. $S \hookrightarrow S+1$

$Z(N,S+1,M,k) = \frac{1}{\sqrt{2(S+1)(2S+1)}}[\sqrt{(S+M)(S+M+1)}Y_1 + \sqrt{2(S-M+1)(S+M+1)}Y_2 + \sqrt{(S-M)(S-M+1)}Y_3]$

3. $S \hookrightarrow S$

$Z(N,S,M,k) = \frac{1}{\sqrt{2S(S+1))}}[-\sqrt{(S+M)(S-M+1)}Y_1 + \sqrt{2}MY_2 + \sqrt{(S-M)(S+M+1)}Y_3]$

4. $S \hookrightarrow S-1$

$Z(N,S-1,M,k) = \frac{1}{\sqrt{2S(2S+1)}}[\sqrt{(S-M)(S-M+1)}Y_1 + \sqrt{2(S-M)(S+M)}Y_2 + \sqrt{(S+M)(S+M+1)}Y_3]$

where

$$Y_1 = Z(N-2,S,M-1)g_1(N-1,N)$$

$$Y_2 = Z(N-2,S,M)g_2(N-1,N)$$

$$Y_3 = Z(N-2,S,M+1)g_3(N-1,N)$$

k is serber branch symbol and k' is also serber branch symbol with last string removed from right .The program for finding coefficient of serber path symbol using addition and subtraction formulae is in appendix

2.3 Rumer basis/spin pairs eigenfunction

Writing spin function for N spins in S^2 is expensive in terms of memory as well as in processing power To avoid this problem. We can write spin function as product of inter-

acting spin pairs.These functions are identical to branching symbol 12121212.. or serber symbol 00000... .Hence are eigenfunction of S^2. Pictorially, It is set of directed arrow for N points such diagrams are called rumer diagrams.No of rumer diagrams are identical to number of ways of connecting directed arrow given N points.Number of rumer diagram $V(N,S) = \frac{N!}{2^g(N-2g)!g!}$.Clearly this number is greater than $f(N,S)$ because many of the rumer diagram are just linear superposition of other diagrams.One way to find set of all rumer diagrams is to take superposition of different rumer diagram and compare them but it is somewhat computationally expensive. There is none to correspondence between branching symbol and rumer diagrams,Also branching diagram are easy to produce and easy to order.We can calculate $< R_i|R_j >$ by counting number of islands, O chains and E chains.

islands : Islands are closed cycles formed by even number of arrows(i) and exactly half number of arrows are in clockwise and anti-clockwise if we change the direction of arrow we include one extra minus sign.It contributes factor of 2^i eg, closed cycles figure 2

O chains : These are open chains with even number of arrows and exactly half number of arrows are in right and left direction.If we change direction of arrow then minus sign in included.It contribute factor of $+1$ or -1 eg. if figure 2 part 2

E chains : These are open chains formed by odd number of arrows.It contributes the factor of 0. hence two rumer diagrams are orthogonal

Hence, dot product of two rumer diagram are given by

$$< R_i|R_j >= \delta_{ss'}\delta_E 2^{i-p}(-1)^r$$

where S and S' are total spin, E in number of chains ,i is number of islands , r is number of reversal of arrow and 2^{-p} is normalisation factor.All rumer diagrams for $S = 0, N = 6$ and corresponding islands and chains are give in given fig 4.1 and 4.2

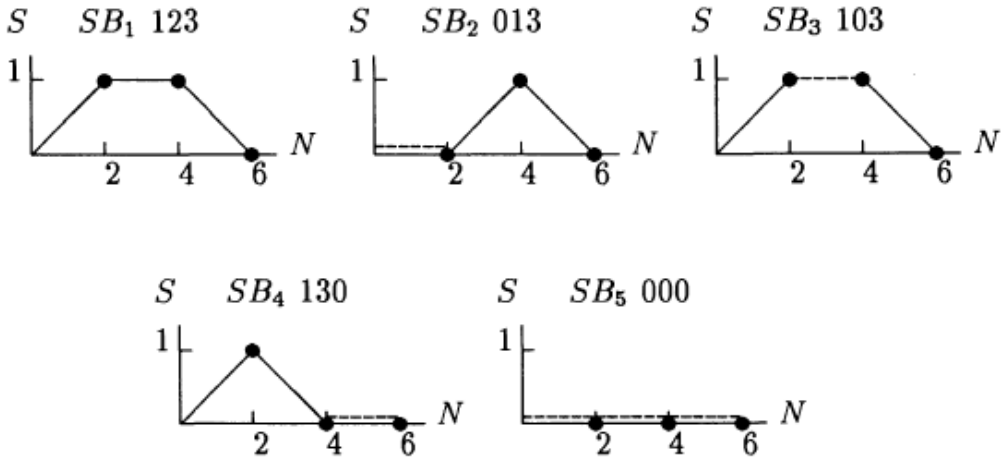
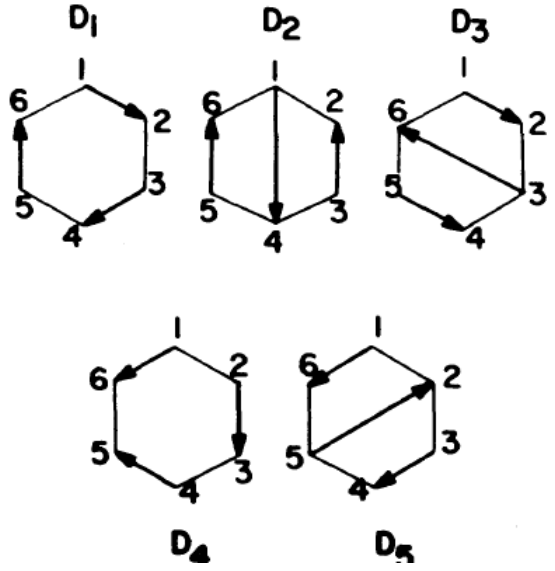
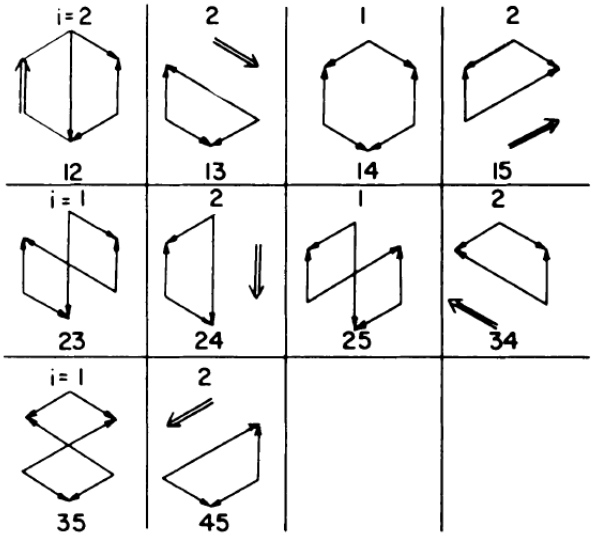


Figure 2.2 Serber branching diagrams and their symbols for $N = 6$, $S = 0$.




i=2
2
1
2
12
13
14
15
i=1
2
1
2
23
24
25
34
i=1
2
35
45

Chapter 3

Spin Liquid

3.1 Overview of work

Everything in this report can be described in four sections

1. Understanding the various ways in which many spin states can be represented efficiently.These include S_z basis ,S^3 basis ,serber type basis and rumer basis discussed in previous chapter
2. We write Heisenberg interaction matrix $H = -J \sum_{(i,j)} S_i.S_j$ on triangular lattice in rumer basis and diagonalise it
3. From the diagonalisation we get ground state energy of system
4. Using eigenvectors of interaction matrix we find correlation function between spins as function of distance
5. We assume that if ground state of this system can be thought as collection of clusters (by cluster we mean , small subsystem of spin that interaction only among themselves) shapes of clusters can be pair, triangle,quadrilateral, hexagon and so on . Using the eigenvectors we find energy of clusters and interpret the result.

3.2 Short Explanation of program

$H_{ij} =< R_i \mathbf{H} R_j >$ this is what we need to calculate. where R_i, R_j are rumer vectors, Using the dirac's identity we can further write $H_{ij} = \frac{1}{2} < R_i(\mathbf{P}_{ij} - 1/2\mathbf{I})R_j >$ so only quantity that we need to calculate is $< R_i R_j >$ and $< R_i \mathbf{P}_{ij} R_j >$ .From the previous chapter we know that $< R_i | R_j >= \delta_{ss'} \delta_E 2^{i-p}(-1)^r$ where

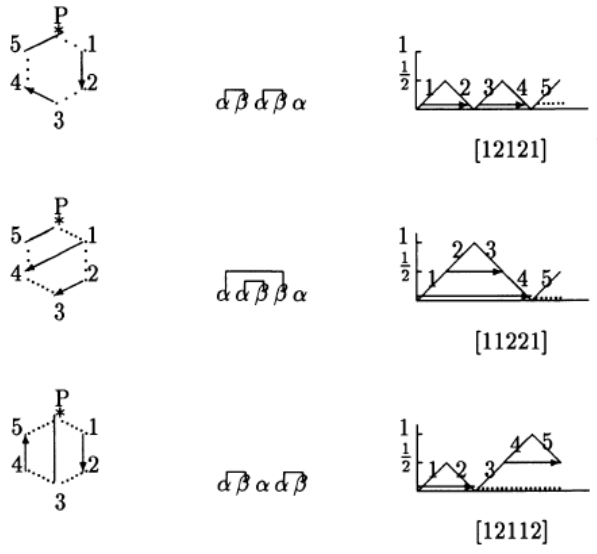
E- number of chains

2^p- is normalisation factor

r- number of arrow reversal

S and S' are total spins which is to say that rumer vectors R_i, R_j are from the same subspace defined by S or S

To simplify our calculation we note the connection that for every rumer diagram there is a branching diagram and they are related as shown in figure below this way we can

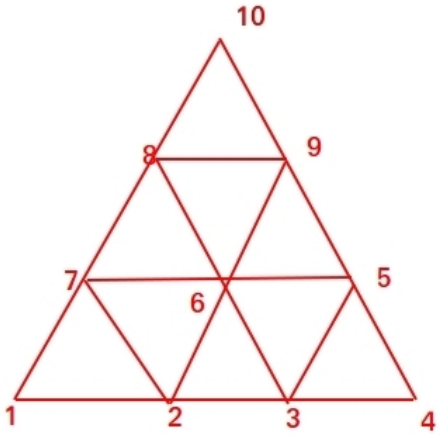


get rumer diagram for each branching symbol.To calculate $< R_i R_j >$ we need to superimpose the two diagrams and find number of **O** chains , **E** chains and islands, We use python library called **NetworkX** .To reduce the size of graph formed by superposition we fist remove all the common pairs and call the list as reduced list.Now our problem is to find number of disjoint graphs from this list we use depth first search of **NetworkX** to

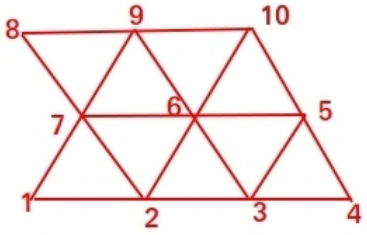
find such graphs. Now we have two problem associated with result obtained this way one result obtained is in form of list of vertices so we covert that into cycle by function $list_{to_{cycle()}}$ other problem is that edges of cycle obtained this way have no direction so as default i add a set of direction so that head of one arrow is connected to tail of exactly one other arrow so that they form a complete cycle Now we compare the arrows to arrows of reduced list for a given cycle and count the reversals using function $sign_{of_{islands()}}$ Once we have number and sign of each island we can find $< R_i R_j >$ using the formula mentioned above To calculate $\mathbf{P}_{ij}$ we define function $permute(i, j)$ operate it on R_j and follow the same procedure. all the programs and result obtained are in appendix

3.3 result's obtained by computation

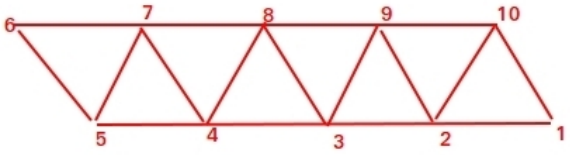
we consider triangular lattice as shown in figure below



pramind arrangement of 10 atoms



two stack arrangement of 10 atoms



Single Stack arranement of 10 atoms

3.3.1 All pairs interact

If we consider all pairs of triangular lattice interact with each other we get the following results

1. Ground state energy per spin $= -4.5742163918$
2. correlation $< S_i.S_j > (d) = -4.1999$ and $standarddeviation = 4.76E-8$
3. correlation $< S_i.S_j > (2d) = -4.1999$ and $standarddeviation = 4.76E-8$ correlation $< S_i.S_j > (3d) = -4.1999$ and $standarddeviation = 4.76E-8$
4. result are same for each arrangement

3.3.2 Nearest neighbour interaction

1. Ground state energy per spin$= -1.8298$
2. correlation $< S_i.S_j > (d) = -4.1999$ and $standarddeviation = 4.76E-8$
3. correlation $< S_i.S_j > (2d) = -4.1999$ and $standarddeviation = 4.76E-8$ correlation $< S_i.S_j > (3d) = -4.1999$ and $standarddeviation = 4.76E-8$
4. result are same for each arrangement
5. interaction energy for pairs= -4.199
6. interaction energy for triangles=41.99
7. interaction energy for quadrilateral=41.99
8. interaction energy for hexagon=25.199

3.4 Conclusion and future work

Result of the following analysis is as follows.

1. Ground state energy per spin depends of atoms and shape of lattice
2. Correlation between two atoms does not depend on shape or numbering of spin/atoms .Also correlation remain unchanged with distance which suggests spins have long range entanglement
3. If we allow that all pairs and interact regardless of their distance , we get constant correlation between to spins regardless of shape arrangement of spin , but if we let only nearest spin pairs to interact we get decreasing coefficient with distance.
4. If we find energy of various small structures such as triangles , quadrilaterals , hexagon we see that , pair have minimum interaction energy. Ground state is sum of such pairs.
5. We also note that after the pair's interaction energy , energy of hexagonal structure is smallest ,if by changing coupling constants we can make hexagon's energy lowest then we will have 1/2 spin excitation which is precisely the property of quantum spin liquids
6. Next goal is to allow nearest as well as next nearest neighbour interaction and check if the above results hold
7. we can also extend nearest neighbour interaction model to 3d triangular lattice (Pyrochlore lattice) , In this type of structure there is an immediate extra feature , loop formed by spins can mess to gather assuming the spin loops have less interaction energy , Under this condition we can have long range entanglement and low correlation. We can also think phases as how these loops are messed to gather this way we can classify QSL phases .My future goal will be to verify if these ideas are correct or not

Chapter 4

Appendix ForS^2 basis

[0]:
```
import numpy as np
I=np.eye(2)
sx=[[0,1],[1,0]]
sy=[[0, -1j],[1j, 0]]
sz=[[1,0],[0,-1]]
# defines the x operater in hilbert space of n spin
def snx(i,n):
  final=[]
  if i==1:
    final=sx
    for k in range(2,n+1):
      final=np.kron(final,I)
  else :
    final=I
    for k in range(2,n+1):
      if k==i:
```

```
        final=np.kron(final,sx)
      else:
        final=np.kron(final,I)
  return final


def snz(i,n):
  final=[]
  if i==1:
    final=sz
    for k in range(2,n+1):
      final=np.kron(final,I)
  else :
    final=I
    for k in range(2,n+1):
      if k==i:
        final=np.kron(final,sz)
      else:
        final=np.kron(final,I)
  return final


def sny(i,n):
  final=[]
  if i==1:
    final=sy
    for k in range(2,n+1):
      final=np.kron(final,I)
```

```
  else :
    final=I
    for k in range(2,n+1):
      if k==i:
        final=np.kron(final,sy)
      else:
        final=np.kron(final,I)
  return final


def S_bar(i,n):
  return snx(i,n)+sny(i,n)+snz(i,n)


def S_bar_total(n):
  final=np.zeros((2**n,2**n),dtype=float)
  for i in range(1,n+1):
    final=final+S_bar(i,n)
  return final


def S_square(n):
  return np.matmul(S_bar_total(n),S_bar_total(n))
```

[0]:
```
import math
def length(a):
    return len(list(a))                          # a is  string  here that␣
 ↪
```

```
def nthletter(a,i):
    k=list(a)
    return k[i]
def coefficient(B,b,s,m):
    c=[]
    for i in range(0,len(b)):
        if nthletter(B,len(b)-1-i)=='1' and nthletter(b,len(b)-1-i)=='1':

            c.append(math.sqrt((s+m)/(2*s)))
            s=s-.5
            m=m-.5

        elif nthletter(B,len(b)-1-i)=='0' and␣
 ↪nthletter(b,len(b)-1-i)=='1':

            c.append((-1)*math.sqrt((s-m+1)/(2*s+2)))
            s=s+.5
            m=m-.5

        elif nthletter(B,len(b)-1-i)=='1' and␣
 ↪nthletter(b,len(b)-1-i)=='0':

            c.append(math.sqrt((s-m)/(2*s)))
            s=s-.5
            m=m+.5
```

```
        else :

            c.append(math.sqrt((s+m+1)/(2*s+2)))
            s=s+.5
            m=m+.5


    x=1
    for i in range(0,len(c)):
        x=x*c[i]
    return x

import itertools as it
import math
def length(a):                      # a is interger here that
    return int(math.log10(a))+1
def nthdigit(a,i):
    return (a%10**i - a%10**(i-1))/10**(i-1)



#max_string(n,k )gives list haveing k times 1 and remaining 0
def max_string(n,k):
    a=[]
    for i in  range(0,k):
        a.append("1")
    for j in range(k,n):
```

```
        a.append("0")
    return a
#this returns word but word are to be taken as strings
def make_word(a):
    word=a[0]
    for i in range(1,len(a)):
        word= word+a[i]
    return word
#this should give nth letter of string
def nthletter(a,i):
    k=list(a)
    return k[i]


# this nck also returns list but element of list are integers



def path_coefficient(n,k):
    a=list(set(list(it.permutations(max_string(n,k))))) # this is list of␣
 ↪sets containing all combinations
    final_list=[]
    path_list=[]
    for i in range(0,len(a)):
        final_list.append(make_word(list(a[i])))
    return final_list


# gives branch coefficient
```

```
def branch_coefficient(n,k):
    a=path_coefficient(n,k)
    l=[]
    for i in range(0,len(a)):
        if nthletter(a[i],0)=='1':
            l.append(a[i])
    return l

#coefficient it give that
import math
def length(a):
    return len(list(a))                          # a is  string  here that␣
 ↪


def nthletter(a,i):
    k=list(a)
    return k[i]
```

```
-0.2886751345948129
```

[0]:

[0]:

Chapter 5

Appendix For serber basis

[2]:
```
'''
a=5
c=[]
try:
    for i in range(0,5):
        c.append(a/i)
except ZeroDivisionError:
    c.append("not defined ")
print(c)
'''
## a is string


def nthletter(a,i):
    k=list(a)
    return a[i]
def length(a):
```

```
    return len(list(a))
import math

'''SB is serber branching coefficient and sb is serber path coeffcient
and s and m are quantum number '''
def serber_coefficient(SB,sb,s,m):
    c=[]
    try:
        for i in range(length(sb)):
            if nthletter(SB,length(sb)-1-i)=='0' and
 ↪nthletter(sb,length(sb)-1-i)=='0':
                c.append(1)
                s=s
                m=m
            elif nthletter(SB,length(sb)-1-i)=='0' and
 ↪nthletter(sb,length(sb)-1-i)=='1':
                c.append(0)
                s=s
                m=m
            elif nthletter(SB,length(sb)-1-i)=='0' and
 ↪nthletter(sb,length(sb)-1-i)=='2':
                c.append(0)
                s=s
                m=m
            elif nthletter(SB,length(sb)-1-i)=='0' and
 ↪nthletter(sb,length(sb)-1-i)=='3':
```

```
                c.append(0)
                s=s
                m=m
            elif nthletter(SB,length(sb)-1-i)=='1' and ␣
↪nthletter(sb,length(sb)-1-i)=='0':
                c.append(0)
                s=s
                m=m
            elif nthletter(SB,length(sb)-1-i)=='1' and ␣
↪nthletter(sb,length(sb)-1-i)=='1':
                c.append(math.sqrt(((s+m-1)*(s+m))/((2*s)*(2*s-1))))
                s=s-1
                m=m-1
            elif nthletter(SB,length(sb)-1-i)=='1' and ␣
↪nthletter(sb,length(sb)-1-i)=='2':
                c.append(math.sqrt((2*(s-m)*(s+m))/((2*s)*(2*s-1))))
                s =s-1
                m=m
            elif nthletter(SB,length(sb)-1-i)=='1' and ␣
↪nthletter(sb,length(sb)-1-i)=='3':
                c.append(math.sqrt(((s-m-1)*(s-m))/((2*s)*(2*s-1))))
                s=s-1
                m=m+1
            elif nthletter(SB,length(sb)-1-i)=='2' and ␣
↪nthletter(sb,length(sb)-1-i)=='0':
                c.append(0)
```

```
                s=s
                m=m
            elif nthletter(SB,length(sb)-1-i)=='2' and␣
↪nthletter(sb,length(sb)-1-i)=='1':
                c.append((-1)*math.sqrt(((s-m+1)*(s+m))/((2*s)*(s+1))))
                s=s
                m=m-1
            elif nthletter(SB,length(sb)-1-i)=='2' and␣
↪nthletter(sb,length(sb)-1-i)=='2':
                c.append(m*math.sqrt(((2))/((2*s)*(s+1))))
                s=s
                m=m
            elif nthletter(SB,length(sb)-1-i)=='2' and␣
↪nthletter(sb,length(sb)-1-i)=='3': # done upto here
                c.append(math.sqrt(((s+m+1)*(s-m))/((2*s)*(s+1))))
                s=s
                m=m+1
            elif nthletter(SB,length(sb)-1-i)=='3' and␣
↪nthletter(sb,length(sb)-1-i)=='0':
                c.append(0)
                s=s
                m=m
            elif nthletter(SB,length(sb)-1-i)=='3' and␣
↪nthletter(sb,length(sb)-1-i)=='1':
                c.append(math.sqrt(((s-m+1)*(s-m+2))/((2*s+2)*(2*s+3))))
                s=s+1
```

```
                m=m-1


            elif nthletter(SB,length(sb)-1-i)=='3' and␣
 ↪nthletter(sb,length(sb)-1-i)=='2':
                c.append((-1)*math.sqrt(((s+m+1)*2*(s-m+1))/
 ↪((2*s+2)*(2*s+3))))
                s=s+1
                m=m
            else :
                c.append(math.sqrt(((s+m+1)*(s+m+2))/((2*s+2)*(2*s+3))))
                s=s+1
                m=m+1
        x=1
        for i in range(0,len(c)):
            x=x*c[i]
        return x
    except :
        c.append(0)
        x=1
        for i in range(0,len(c)):
            x=x*c[i]
        return x
##3
## this for serber_path_diagram_symbols
## givn numeber of electrons and s  vlaue it should give all serber␣
 ↪branching coefficient
```

```
def all_combinations(i):
    k=['0','1','2','3']
    if i==1:
        return k
    else :
        x=[]
        for l in k:
            for m in all_combinations(i-1):
                x.append(l+m)
    return x
### returns s quantum number of string that may be branch or path␣
 ↪coefficient


def value(a):
    dic={'0':0,'1':1,'2':0,'3':-1}
    k=list(a)
    s = 0
    for i in range(0,len(k)):
        s=s+dic[k[i]]


    return s



## takes n of electons and spin quatum nu
def serber_path_diagram_symbols(n,m):
    k=all_combinations(n/2)
```

```
    k1=[]
    for i in range(0,len(k)):
        if value(k[i])==m:
            k1.append(k[i])
    return k1

### it will write down the expansion for a given serber branch diagram␣
 ↪symbol in terms of
##
def serber_expansion(n,SB,s,m):
    print("this is expansion for "+SB)
    sb=serber_path_diagram_symbols(n,m)

    for i in range(0,len(sb)):
        print(str(serber_coefficient(SB,sb[i],s,m))+'*['+sb[i]+']')
#### for value of
def value(a):
    dic={'0':0,'1':1,'2':0,'3':-1}
    k=list(a)
    s=0
    for i in range(0,len(k)):
        s=s+dic[k[i]]
    return s
print(value('1231'))
## return last string of a
def last_string(a):
```

```
    m=len(list(a))-1
    return list(a)[m]




## in this i have found serber branching symbols
def serber_branch_symbols(n,s):
    k=['0','1','2','3']

    l=['0','1']
    m=[]
    sbs=k
    final_list=[]
    if n==1:
        return k
    else:
        for y in range(0,int(n/2)-1):
            for i in range(0,len(sbs)):
                if last_string(sbs[i])=='0':
                    for le in l:
                        if value(sbs[i])<=s:
                            m.append(sbs[i]+le)
                            #m.remove(sbs[i])
```

```
                else:
                    for ke in k:
                        if value(sbs[i])<=s:
                            m.append(sbs[i]+ke)
                            #m.remove(sbs[i])

            sbs=m
        for z in range(0,len(m)):
            if value(m[z])==s and length(m[z])==int(n/2):
                final_list.append(m[z])

    return final_list
```

```
1
```

[0]:

Chapter 6

Appendix For Rumer basis and computational results

[0]:
```
""" this function takes no of up spin and down spin and index of branch␣
 ↪symbol
output  is corresponding branch symbol in last letter sequence """
import networkx as nx
import numpy as np
from itertools import combinations


def bs(up,down,index):
    ul=[]
    for i in range(up):
        ul.append(1)


    dl=[]
    for i in range(down):
```

```
        dl.append(0)


    k=index-1
    q=int(k/(up-1))
    r=k%(up-1)
    if k==0:
        return ul+dl
    if r==0:
        ul[-(up-1)],dl[q-1]=dl[q-1],ul[-(up-1)]
        return ul + dl
    else:
        ul[-r],dl[q]=dl[q],ul[-r]
        return ul + dl
"""return corresponding rumer vector to branch symbol"""
def rumer(bs):
    rm=[]


    for j in range(1,len(bs)):
        for i in range(0,len(bs)-j):
            if bs[i]==1 and bs[i+j]==0:
                bs[i]="x"
                bs[i+j]='x'


                rm.append([i+1,i+j+1])


    for k in range(0,len(bs)):
```

```
        if bs[k]== 1:
            rm.append([k+1])
    return rm

def interchange(a):
    a[0],a[1]=a[1],a[0]



""" return numer of two edge islands to simplify """
def no_of_two_edge_islands(a,b):
    m=0
    for x in a[:]:
        for y in b:
            if x==y:
                b.remove(x)
                a.remove(x)
                m=m+1
    return  m

def reduced_list(a,b):
    for x in a[:]:
        for y in b:
            if x==y:
                b.remove(x)
                a.remove(x)
    return(a+b)
```

```
""" Y has to reduced list(2 islands are removed form combined list )
and Z has to cycle in list form """
def sign_of_island(Y,Z):
    p=0
    for x in Y:
        for y in Z:
            if x==interchange(y):
                p=p+1
    return (-1)**(p%2)


""" find the cycles in given graph """
def find_cycles(x):

    g=nx.Graph()

    g.add_edges_from(x)
    l=nx.algorithms.cycles.find_cycle(g)
    m=[list(ele) for ele in l]
    return m


""" applies the permutation operator on rumer basis vector """

def permute(i,j,b):
```

```
    a=b[:]
    k=0
    l=0
    m=0
    n=0
    for x in range(len(a)) :
        for y in range(len(a[x])):
            if a[x][y]==i:
                k,l=x,y

    for p in range(len(a)) :
        for q in range(len(a[p])) :
            if a[p][q]==j :
                m,n=p,q

    a[k][l],a[m][n]=a[m][n],a[k][l]
    return a



""" take two rumer vectors  and produces the dot product """

def rumer_dot(a,b,n):
    island_2=no_of_two_edge_islands(a,b)
    red_list=reduced_list(a,b)
    cycles=list_of_cycles(red_list)
    island_3=len(cycles)
```

```
    islands=island_2+island_3
    finalsign=1
    for i in range(len(cycles)):
        i_sign=sign_of_island(red_list,cycles[i])
        finalsign=finalsign*i_sign
    rirj=(2**(islands-(n/2)))*finalsign
    return rirj


def si_dot_sj_in_rumer(i,j,n):
    a=rumer(bs(int(n/2),int(n/2),i))
    b=rumer(bs(int(n/2),int(n/2),j))
    print(a,b)
    rirj=rumer_dot(a,b,n)
    p_rirj=rumer_dot(a,permute(i,j,b),n)

    print(rirj,p_rirj)
    return .5*p_rirj-(.25*rirj)


"""  takes number of spins  and total spin as inpur and
     returns number of independent basis vector for that subspace   """
def f(n,s):
    if n==2 and s==0:
        return 1
    if n==2 and s==1:
```

```
        return 1


    if s>(n/2):
        return 0
    if s==(n/2):
        return 1
    if 0<=s<(n/2):
        return f(n-1,s+0.5)+f(n-1,s-.5)
    if s<0:
        return 0


"""takes number of spins  and total spin as input and
     returns   i th basis vector in string of 0 and 1   for that subspace
    where is 0 is -1/2 state and 1 is for 1/2 state of single spin """
def branch(n,s,i):
    k=[]
    if i<=f(n-1,s+.5):
        if f(n-1,s+.5)>1:
            k.append(0)
            return branch(n-1,s+.5,i)+[0]
        if f(n-1,s+.5)==1:
            k.append(0)
            m=[]
            for p in range(n-1):
                m.append(1)
            return m+k
```

```
    if i> f(n-1,s+.5):
        k.append(1)
        return branch(n-1,s-.5,i-f(n-1,s+.5))+[1]




""" to convert cycles obtained list of sequence to list of pairs """
def form_basis_list_cycles(a):
    k=[]
    for i in range(len(a)):
        x=a[i:i+2]
        k.append(x)
    k.pop(-1)
    y=[a[-1],a[0]]
    k.append(y)
    return k



""" Y has to reduced list  and Z has to cycle in list form """
def sign_of_island(Y,Z):
    p=0
    for x in Y:
        for y in Z:
```

```
            if x==y:
                p=p+1
    return -1**((len(Y)-p)%2)

""" it will take reduced list and produce
the list in which each element is a cycle """
def list_of_cycles(red_list):
    g=nx.Graph()
    g.add_edges_from(red_list)
    l=nx.algorithms.cycles.cycle_basis(g)
    final_list=[]
    for i in range(len(l)):
        final_list.append(form_basis_list_cycles(l[i]))

    return final_list

"""   this gives number of independent basis vectors when total spin is s␣
 ↪"""
def f(n,s):
    if n==2 and s==0:
        return 1
    if n==2 and s==1:
        return 1

    if s>(n/2):
        return 0
```

```
    if s==(n/2):
        return 1
    if 0<=s<(n/2):
        return f(n-1,s+0.5)+f(n-1,s-.5)
    if s<0:
        return 0



#this gives basis vector as string of 0 and 1 where
# 0 is lower state and 1 is upper state
def branch(n,s,i):
    k=[]
    if i<=f(n-1,s+.5):
        if f(n-1,s+.5)>1:
            k.append(0)
            return branch(n-1,s+.5,i)+[0]
        if f(n-1,s+.5)==1:
            k.append(0)
            m=[]
            for p in range(n-1):
                m.append(1)
            return m+k
    if i> f(n-1,s+.5):
        k.append(1)
        return branch(n-1,s-.5,i-f(n-1,s+.5))+[1]
```

```
# this give dot product of two rumer vectors
def ridotrj(a,b,n):
    island2=no_of_two_edge_islands(a,b)

    reducedlist=reduced_list(a,b)
    list_of_other_cycles=list_of_cycles(reducedlist)
    island3=len(list_of_other_cycles)

    totalislands=island2+island3

    finalsign=1

    for i in range(len(list_of_other_cycles)):
        island_sign=sign_of_island(reducedlist,list_of_other_cycles[i])
        finalsign=finalsign*island_sign

    rirj=(2**(-n/2))*(2**( totalislands))*finalsign

    return rirj

""" this program takes n  no of spin
  total spin s  and index i  and returns rumer basis vector """
def ru(n,s,i):
    return rumer(branch(n,s,i))
```

```
""" this is for matrix element k l for spin pair  ij given n and s """
def xyzsisj(i,j,k,l,n,s):
    ri=ru(n,s,i)
    rj=ru(n,s,j)
    return .5*ridotrj(ri,permute(k,l,rj),n)- .25*ridotrj(ri,rj,n)


""" this give interaction matrix for spin pair i j  total number of
and total spin s  """
def mat_sisj(i,j,n,s):
    arr = np.zeros((f(n,s),f(n,s)), dtype=float)
    for i in range(0,f(n,s)):
        for j in range(0,(f(n,s))):
            if i==j:
            #print(1)
                arr[i,j]=1
            if i!=j:
                #print(xyzsisj(i,j,1,2, 10,0))
                arr[i,j]=xyzsisj(i+1,j+1,i,j, n,s)
    return arr


""" it gives expectation values of sisj with eigenvectors of H= sum sisj␣
 ↪"""
def correlation(i,j,eigenvectors):
  temp=0
  for  k in range(len(eigenvectors)):
```

```
    temp=temp+ eigenvectors[k]@np.
  ↪matmul(mat_sisj(x[0],x[1],n,s),eigenvectors[k])


  return temp


""" this gives mean and standerd deviation of list """
def mean(x):
  temp=0
  for i in range(len(x)):
    temp=temp+x[i]
  return temp/len(x)



""" takes list of numbers to produce standerd deviation """
def std(x):
  temp=0
  for i in range(len(x)):
    temp=temp+(x[i]*x[i])
  return np.sqrt((temp/len(x))-mean(x)**2)



""" it takes list of pairs and produces the interaction matrix """
def imt(n,s,p):
  k=f(n,s)
  final=np.zeros((k,k),dtype=float)
  for x in p:
```

```
    final=final+mat_sisj(x[0],x[1],n,s)
  return final


  """ expectaion values of matrics """
def expec(p,eigenvectors):
  temp=0
  for  k in range(len(eigenvectors)):
    temp=temp+ eigenvectors[k]@np.matmul(p,eigenvectors[k])


  return temp
```

[4]:
```
""" Diagonaliseing interaction matrix on trianular lattice with only␣
 ↪nearest
neighbour interaction see figure above"""
p=[[1,2],[1,7],[2,3],[2,6],[2,7],[3,6],[3,4],[3,5],[4,5]
    ,[5,6],[5,9],[6,7],[6,8],[6,9],[7,8],[8,9],[8,10],[9,10]]
n=10
s=0
final_vectors=np.linalg.eigh(imt(n,s,p))
eigenvalue=final_vectors[0]
eigenvector=final_vectors[1]


print("ground state energy per spin ",min(eigenvalue)/n)
```

```
ground state energy per spin  -1.829686556719621
```

[7]:
```
""" correlation when distance is one unit """
correlation1=[]
```

```
p1=[[1,2],[1,7],[2,3],[2,6],[2,7],[3,6],[3,4],[3,5],[4,5]
    ,[5,6],[5,9],[6,7],[6,8],[6,9],[7,8],[8,9],[8,10],[9,10]]


for x in p1:
  correlation1.append(correlation(x[0],x[1],eigenvector))


print("correlation",mean(correlation1)/n)
```

```
correlation 4.199999999999999
```

[8]:
```
""" correlation when distace is 2 units """
correlation2=[]
p2=[[1,3],[1,7],[2,3],[2,8],[3,9],[4,8],[5,7],[7,10]]


for x in p2:
  correlation2.append(correlation(x[0],x[1],eigenvector))


print("correlation",mean(correlation2)/n)
```

```
correlation 4.199999999999999
```

[11]:
```
""" correlation when distace is 3 units """
correlation3=[]
p3=[[1,10],[1,4],[4,10]]


for x in p3:
  correlation3.append(correlation(x[0],x[1],eigenvector))


print("correlation",mean(correlation3)/n,"standerd deviation")
```

```
correlation 4.199999999999998 standerd deviation
```

[15]:
```
"""interaction energy for triangles made of 3 spins """
ptri=[[[1,2],[2,9],[9,1]],[[2,3],[3,6],[6,2]],[[8,9],[9,10],[10,8]]]
ptri1=imt(n,s,ptri[0])
ptri2=imt(n,s,ptri[1])
ptri3=imt(n,s,ptri[2])
print("interaction evergy for triangle  ",expec(ptri1,eigenvector)/3)
print("interaction evergy for triangle  ",expec(ptri2,eigenvector)/3)
print("interaction evergy for triangle  ",expec(ptri3,eigenvector)/3)
```

```
interaction evergy for triangle   41.999999999999986
interaction evergy for triangle   41.999999999999986
interaction evergy for triangle   41.999999999999986
```

[13]:
```
""" interaction energy of  spin forming hexagon hexagon """
phex=[[2,7],[7,9],[9,8],[8,5],[5,3],[3,2]]
mhex=imt(n,s,phex)


print("interaction evergy for hexagon ",expec(mhex,eigenvector)/6)
```

```
interaction evergy for hexagon  25.199999999999992
```

[16]:
```
""" interaction energy of quadrilatral  """
pquad=[[[1,2],[2,6],[6,7],[7,1]],[[3,5],[5,8],[8,6],[6,5]]
       ,[[6,7],[7,9],[9,8],[8,6]]]
pquad1=imt(n,s,pquad[0])
print("interaction evergy for quadilatral ",expec(pquad1,eigenvector)/4)
pquad2=imt(n,s,pquad[1])
```

```
print("interaction evergy for quadilatral ",expec(pquad2,eigenvector)/4)
pquad3=imt(n,s,pquad[2])
print("interaction evergy for quadilatral ",expec(pquad3,eigenvector)/4)
```

```
interaction evergy for quadilatral  41.999999999999986
interaction evergy for quadilatral  41.999999999999986
interaction evergy for quadilatral  41.999999999999986
```

[0]:

[0]:

[9]:
```
""" for 10 atoms with single pyramid triangular arrangement """


""" this is complete interaction matrix H= sum si sj
 j=1 in this and all subsequent calculation  """
n=10
s=0


p=all_pairs(n)
k=f(n,s)
final=np.zeros((k,k),dtype=float)
for x in p:
  final=final+mat_sisj(x[0],x[1],n,s)


final_vectors=np.linalg.eigh(final)
eigenvalues=final_vectors[0]
eigenvectors=final_vectors[1]
```

```
print("ground state energy per spin ",min(eigenvalues)/n)


""" p1 is pairs when then the gap is unit atomic distance"""
p1=[[1,2],[1,7],[2,3],[2,6],[2,7],[3,6],[3,4],[3,5],[4,5]
    ,[5,6],[5,9],[6,7],[6,8],[6,9],[7,8],[8,9],[8,10],[9,10]]
correlation1=[]
for x in p1:
  correlation1.append(correlation(x[0],x[1],eigenvectors))
print(" mean correlation per atoms for 1unit distance␣
 ↪",mean(correlation1)/n,
      "\n standered deviation ",std(correlation1))


""" p2 is pairs when then the gap is  2 unit atomic distance"""
p2=[[1,3],[1,9],[2,4],[2,8],[3,9],[4,8],[5,7],[5,10],[7,10]]


correlation2=[]
for x in p2:
  correlation2.append(correlation(x[0],x[1],eigenvectors))
print(" mean correlation per atom for 2 unit distance␣
 ↪",mean(correlation2)/n,
      "\n standered deviation ",std(correlation2)/n)


""" p3 is pairs when then the gap is  3 unit atomic distance"""
p3=[[1,10],[1,4],[4,10]]


correlation3=[]
```

```
for x in p3:
  correlation3.append(correlation(x[0],x[1],eigenvectors))
print(" mean correation per atom for 3 unit distance ",mean(correlation3)/
 ↪n,
      "\n standered deviation ",std(correlation3)/n)
```

```
ground state energy per spin  -4.574216391799053
 mean correlation per atoms for 1unit distance  4.199999999999999
 standered deviation  6.743495761743046e-07
 mean correlation per atom for 2 unit distance  4.199999999999999
 standered deviation  4.76837158203125e-08
 mean correation per atom for 3 unit distance  4.199999999999999
 standered deviation  0.0
```

[10]:
```
""" for 10 atoms double stack arrangement """


n=10
s=0
p=all_pairs(n)
k=f(n,s)
final=np.zeros((k,k),dtype=float)
for x in p:
  final=final+mat_sisj(x[0],x[1],n,s)


final_vectors=np.linalg.eigh(final)
eigenvalues=final_vectors[0]
eigenvectors=final_vectors[1]
```

```
print("ground state energy per spin ",min(eigenvalues)/n)



""" p1 is pairs when then the gap is unit atomic distance"""
p1=[[1,2],[1,7],[2,3],[2,6],[2,7],[3,4],[3,5],[3,6],[5,6],[5,10]
    ,[6,7],[6,9],[6,10],[7,8],[7,9],[8,9],[9,10]]
correlation1=[]
for x in p1:
  correlation1.append(correlation(x[0],x[1],eigenvectors))
print(" mean correlation per atom for 1 unit distance␣
 ↪",mean(correlation1)/n,
      "\n standered deviation ",std(correlation1)/n)



""" p2 is pairs when then the gap is 2 atomic distance"""
p2=[[1,3],[1,9],[2,4],[2,8],[2,10],[3,9],[4,10],[5,7]]
correlation2=[]
for x in p2:
  correlation2.append(correlation(x[0],x[1],eigenvectors))
print(" mean correlation per atom for 2 unit distance␣
 ↪",mean(correlation2)/n,
      "\n standered deviation ",std(correlation2)/n)
```

```
ground state energy per spin  -4.574216391799053
 mean correlation per atom for 1 unit distance  4.199999999999999
```

```
standered deviation  6.743495761743045e-08
mean correlation per atom for 2 unit distance  4.199999999999999
standered deviation  4.76837158203125e-08
```

[11]:
```
""" for single stack arrangement """
n=10
s=0

""" p1 is pairs when then the gap is unit atomic distance"""
p1=[[1,2],[1,10],[2,3],[2,9],[3,4],[3,8],[4,5],[4,7],
    [5,6],[6,7],[7,8],[8,9],[9,10]]

correlation1=[]
for x in p1:
  correlation1.append(correlation(x[0],x[1],eigenvectors))
print(" mean correlation per atom for 1 unit distance␣
 ↪",mean(correlation1)/n,
      "\n standered deviation ",std(correlation1)/n)
```

```
mean correlation per atom for 1 unit distance  4.199999999999999
standered deviation  6.743495761743045e-08
```

[0]:
```

```

References

[And73] P. Anderson. *Matter.Res.bull*, 8:158, 1973.

[And87] P. Anderson. *Science*, 235:1198, 1987.

[AYa] Kitaev AY.

[AYb] Kitaev AY.

[DS] Kivelsion DS and Rokhsar SA.

[DSS] Kivelsion DS, Rokhsar SA, and Sethna.

[FJ] Wegner FJ.

[JB] Kogut JB.

[MS] Moessnor and Shondhi.

[PA87] Bhaskaran P. Anderson, Zau. *Solid State commun.*, 35, pages =, 1987.

[TMa] Senthil T and Fisher MP.

[TMb] Senthil T and Fisher MP.

[Vl87] Kalmeyer V and laughlin. *Phys.Rev.Lett.*, 59:2095, 1987.