Exact diagonalization methods for quantum spin system

Tokuro Shimokawa

OIST

OKINAWA INSTITUTE OF SCIENCE AND TECHNOLOGY GRADUATE UNIVERSITY

沖縄科学技術大学院大学

Goals of the tutorial lectures

- To understand the principles of the three Lanczos-based ED methods. (Lanczos, Finite-temperature Lanczos, Continued fraction methods)

- To understand the importance of bit representations/bitwise operations and symmetries to make fast S=1/2 ED codes.

- To read and run the actual python ED sample codes and customize them for better understanding above two.

The lecture notes and Python codes are available from

https://github.com/tshimokaw/PythonEDs

Outline

1. Introduction
2. Lanczos method
3. Continued fraction method for spin dynamics
4. Finite-temperature Lanczos method
5. Binary representations and bitwise operations
6. Symmetries and look-up/look-back procedures

1. Introduction

Introduction

- Exact diagonalization (ED) method is a numerical technique to solve the Schrödinger equation of a quantum many body system.

Schrödinger equation

$$\mathscr{H}\,|\,n\rangle = E_n\,|\,n\rangle$$

- If you can get all eigenvalue/eigenvector of the quantum Hamiltonian, you can calculate the expectation value of an any observable *A* at an any temperature *β*.

Thermal expectation

$$\langle A\rangle = \frac{1}{Z}\sum_n \mathrm{e}^{-\beta E_n}\langle n\,|\,A\,|\,n\rangle$$

- Or you can also compute Green's function to see the dynamics of the many body system.

Green's function

$$G^R(t) = -\,i\theta(t)\langle[A(t), B(t=0)]\rangle$$

Introduction

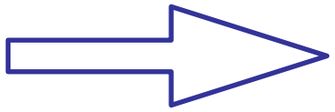
Ex) S=1/2 Heisenberg model, two spin system

$$\mathscr{H} = \mathbf{S}_i \cdot \mathbf{S}_j = S_i^z S_j^z + \frac{1}{2}(S_i^+ S_j^- + S_i^- S_j^+)$$

Hamiltonian matrix

$$H = \begin{pmatrix} \langle\uparrow\uparrow|\mathbf{S}_i\cdot\mathbf{S}_j|\uparrow\uparrow\rangle & \langle\uparrow\uparrow|\mathbf{S}_i\cdot\mathbf{S}_j|\uparrow\downarrow\rangle & \langle\uparrow\uparrow|\mathbf{S}_i\cdot\mathbf{S}_j|\downarrow\uparrow\rangle & \langle\uparrow\uparrow|\mathbf{S}_i\cdot\mathbf{S}_j|\downarrow\downarrow\rangle \\ \langle\uparrow\downarrow|\mathbf{S}_i\cdot\mathbf{S}_j|\uparrow\uparrow\rangle & \langle\uparrow\downarrow|\mathbf{S}_i\cdot\mathbf{S}_j|\uparrow\downarrow\rangle & \langle\uparrow\downarrow|\mathbf{S}_i\cdot\mathbf{S}_j|\downarrow\uparrow\rangle & \langle\uparrow\downarrow|\mathbf{S}_i\cdot\mathbf{S}_j|\downarrow\downarrow\rangle \\ \langle\downarrow\uparrow|\mathbf{S}_i\cdot\mathbf{S}_j|\uparrow\uparrow\rangle & \langle\downarrow\uparrow|\mathbf{S}_i\cdot\mathbf{S}_j|\uparrow\downarrow\rangle & \langle\downarrow\uparrow|\mathbf{S}_i\cdot\mathbf{S}_j|\downarrow\uparrow\rangle & \langle\downarrow\uparrow|\mathbf{S}_i\cdot\mathbf{S}_j|\downarrow\downarrow\rangle \\ \langle\downarrow\downarrow|\mathbf{S}_i\cdot\mathbf{S}_j|\uparrow\uparrow\rangle & \langle\downarrow\downarrow|\mathbf{S}_i\cdot\mathbf{S}_j|\uparrow\downarrow\rangle & \langle\downarrow\downarrow|\mathbf{S}_i\cdot\mathbf{S}_j|\downarrow\uparrow\rangle & \langle\downarrow\downarrow|\mathbf{S}_i\cdot\mathbf{S}_j|\downarrow\downarrow\rangle \end{pmatrix} = \begin{pmatrix} 1/4 & 0 & 0 & 0 \\ 0 & -1/4 & 1/2 & 0 \\ 0 & 1/2 & -1/4 & 0 \\ 0 & 0 & 0 & 1/4 \end{pmatrix}$$

Diagonalizing



Eigenvalues

$$E = \begin{pmatrix} 1/4 & 0 & 0 & 0 \\ 0 & 1/4 & 0 & 0 \\ 0 & 0 & 1/4 & 0 \\ 0 & 0 & 0 & -3/4 \end{pmatrix}$$

Eigenvectors

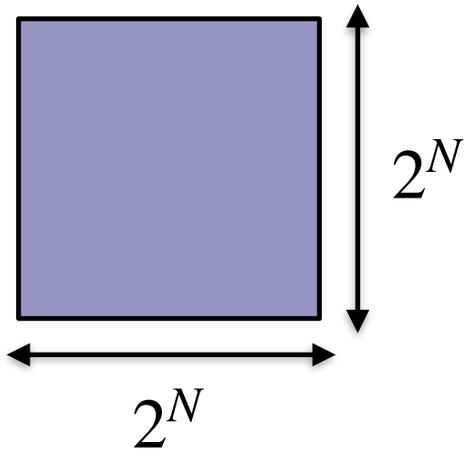
$$|\uparrow\uparrow\rangle$$
$$|\downarrow\downarrow\rangle$$
$$\frac{1}{\sqrt{2}}(|\uparrow\downarrow\rangle + |\downarrow\uparrow\rangle$$
$$\frac{1}{\sqrt{2}}(|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle$$

Introduction

Ex) S=1/2 Heisenberg model, *N* spin chain

$$\mathcal{H} = \sum_{i=0}^{N-1} J\ \mathbf{S}_i \cdot \mathbf{S}_{i+1}$$

Hamiltonian matrix

$H =$ 

Matrix dimension: 2^N

Required computational memory:

$$8 \times 2^N \times 2^N [\text{byte}]$$

(Double precision/real)

cf. # of spin and memory

N=10	8 [MB]
N=16	32 [GB]
N=20	8 [TB]

Matrix dimension/computational cost grow exponentially…

What can we do for reducing the cost?

To access the physics in larger finite-size systems…

- Lanczos type method for $T=0$ physics
- Finite-temperature Lanczos type method for $T\neq 0$ physics
- Continued fraction method for dynamical correlations
- Block diagonalization by symmetries

2. Lanczos method for T=0 physics

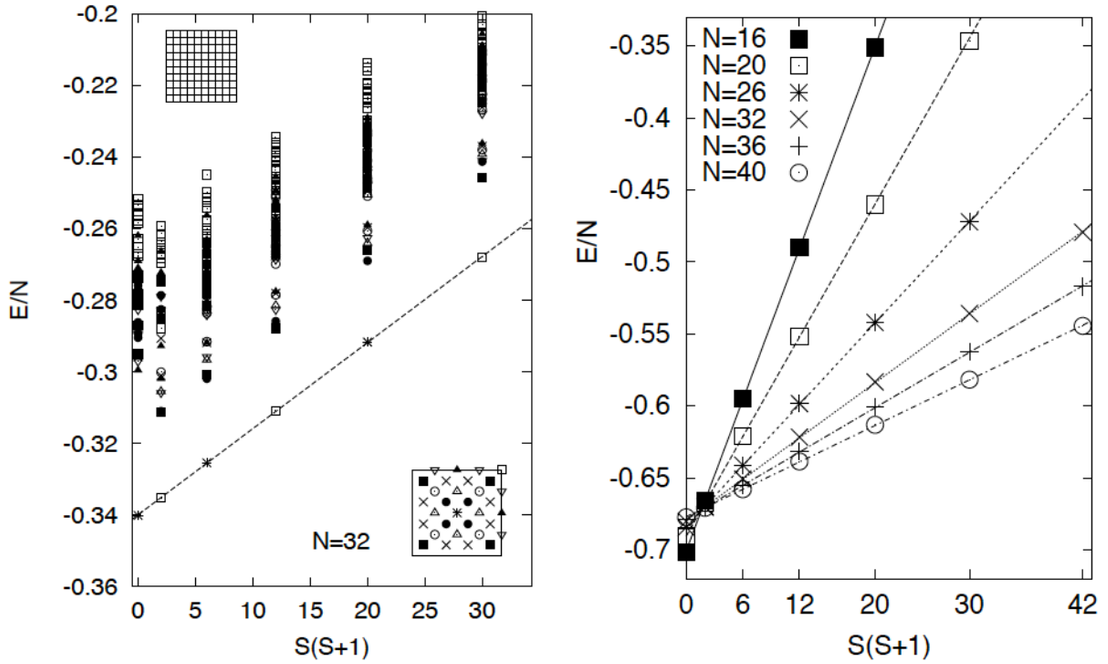
Background

- The role of the quantum fluctuation is still less understood in quantum magnets.
- Especially, S=1/2 low-dimensional frustrated magnets have been main playgrounds for finding non-intuitive exotic quantum ground states (cf. quantum spin liquids).
- Exact diagonalization based on the Lanczos method have been used to investigate if the ground state of the target system may have semi-classical LRO or not from finite-size systems (cf. tower of states analysis).

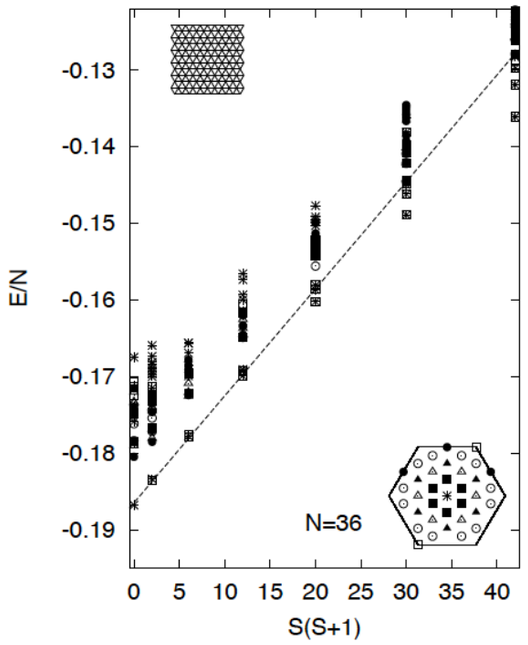
Background

The spontaneous symmetry breaking in semi-classically Néel ordered antiferromagnet can be revealed in the spectrum of a finite system.

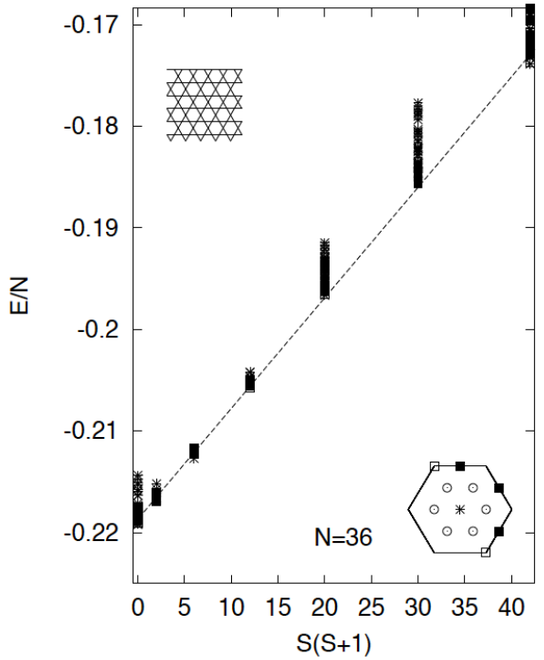
S=1/2 square-lattice AF



S=1/2 triangular-lattice AF



S=1/2 kagome-lattice AF



$$H_{\text{eff}} \simeq E_0 + \frac{\mathbf{S}^2}{2N\chi_0} + H_{\text{magnons}} \;\rightarrow\; E_{\text{min}}(S) \simeq E_0 + \frac{S(S+1)}{2N\chi_0} + E_{\text{magnons}}$$

U. Schollwock et al, “Quantum Magnetism”, chapter 2

The linear relation between $E_{\text{min}}(S)$ and $S(S+1)$ can indicate a Néel order in the thermodynamic limit

The power method

- A simple method to get the eigenvalue of a matrix with the largest absolute value.
- Start from an arbitrary initial vector, $|v_0\rangle = \sum_{n=0}^{D-1} c_n |\psi_n\rangle$, where $E_n, |\psi_n\rangle$ are the eigenvalues and eigenvectors of the D-dimensional matrix $\mathscr{H}$.
- Multiply the matrix repeatedly until the resulting vector converges to the desired one.

$$|v_k\rangle \equiv \mathscr{H}^k |v_0\rangle = \sum_{n=0}^{D-1} c_n E_n^k |\psi_n\rangle = c_{n_{\max}} E_{n_{\max}}^k [\,|\psi_{n_{\max}}\rangle + \sum_{n \neq n_{\max}} \frac{c_n}{c_{n_{\max}}} \left(\frac{E_n}{E_{n_{\max}}}\right)^k |\psi_n\rangle]$$

$\downarrow$

0

We can get the desired eigenvector but with large k…

Lanczos method

C. Lanczos, J. Res. Natl. Bur. Stand. B. **45**, 255 (1950).

- Lanczos method gives us an acceleration of convergence over the simple power method by subtracting the component of previous vectors $|v_{k-1}\rangle, |v_{k-2}\rangle, \ldots$ from $|v_k\rangle$. That means one can eliminate the effect of the arbitrary chosen initial random vector as fast as possible.

$$|v_0\rangle = \sum_{n=0}^{D-1} c_n |\mathrm{S^z\ basis}\rangle_n \qquad \sum_{n=0}^{D-1} |c_n|^2 = 1$$

$$|v_1\rangle = \mathscr{H}|v_0\rangle - \alpha_0 |v_0\rangle \qquad \alpha_0 = \langle v_0|\mathscr{H}|v_0\rangle / \langle v_0|v_0\rangle$$

$$|v_2\rangle = \mathscr{H}|v_1\rangle - \alpha_1 |v_1\rangle - \beta_0 |v_0\rangle \qquad \alpha_1 = \langle v_1|\mathscr{H}|v_1\rangle / \langle v_1|v_1\rangle \qquad \beta_0 = \langle v_1|v_1\rangle / \langle v_0|v_0\rangle$$

$$|v_k\rangle = \mathscr{H}|v_{k-1}\rangle - \alpha_{k-1} |v_{k-1}\rangle - \beta_{k-2} |v_{k-2}\rangle$$

$$\alpha_{k-1} = \langle v_{k-1}|\mathscr{H}|v_{k-1}\rangle / \langle v_{k-1}|v_{k-1}\rangle$$

$$\beta_{k-2} = \langle v_{k-1}|v_{k-1}\rangle / \langle v_{k-2}|v_{k-2}\rangle$$

Lanczos method

- Rewriting the original matrix $\mathscr{H}$ by using normalized Lanczos vectors $\{|v_k\rangle' = |v_k\rangle / \sqrt{\langle v_k | v_k \rangle}\}$ gives us the following tridiagonal matrix $\mathbf{T}$.

$$\mathbf{T} \equiv \mathscr{H} = \begin{pmatrix} \alpha_0 & \sqrt{\beta_0} & 0 & 0 & 0 & 0 & \cdots & 0 \\ \sqrt{\beta_0} & \alpha_1 & \sqrt{\beta_1} & 0 & 0 & 0 & \cdots & 0 \\ 0 & \sqrt{\beta_1} & \alpha_2 & \sqrt{\beta_2} & 0 & 0 & \cdots & 0 \\ 0 & 0 & \sqrt{\beta_2} & \alpha_3 & \sqrt{\beta_3} & 0 & \cdots & 0 \\ 0 & 0 & 0 & \sqrt{\beta_3} & \alpha_4 & & & \vdots \\ \vdots & \vdots & \vdots & \vdots & & & & \vdots \\ \vdots & \vdots & \vdots & \vdots & & & & \vdots \\ \vdots & \vdots & \vdots & \vdots & & & \alpha_{k-2} & \sqrt{\beta_{M-2}} \\ 0 & 0 & 0 & 0 & \cdots & \cdots & \sqrt{\beta_{M-2}} & \alpha_{M-1} \end{pmatrix} \begin{matrix} \cdots |v_0\rangle' \\ \cdots |v_1\rangle' \\ \cdots |v_2\rangle' \\ \cdots |v_3\rangle' \\ \\ \\ \\ \cdots |v_{M-2}\rangle' \\ \cdots |v_{M-1}\rangle' \end{matrix}$$

The eigenvalues and eigenvectors $\{\epsilon_l, |\phi_l\rangle\}$ correspond to the ones of the original matrix.

- The eigenvalues/vectors of the original Hamiltonian are obtained from diagonalizing $\mathbf{T}$. (In python, for example “scipy.linalg.eigh_tridiagonal” is used for this diagonalization.)

- Typically, the needed $k=M$ for the convergence is 100~1000 (<< D).

Lanczos method

- The *M*-dimensional eigenvectors of the tridiagonal matrix $\{\,|\phi_l\rangle = \sum_{k=0}^{M-1} t_{k,l}\,|v_k\rangle',\ l = 0 \sim M-1\}$ correspond to the eigenvectors of the *D*-dimensional original matrix. (M<<D)

- Consider the 0-th eigenvector of $\mathbf{T}$ for the GS wave function, $|\phi_{l=0}\rangle = \sum_{k=0}^{M-1} t_{k,0}\,|v_k\rangle'$ and rewrite this by using original S^z basis state.

$$|\mathrm{GS}\rangle \equiv |\phi_{l=0}\rangle = \sum_{k=0}^{M-1} t_{k,0}\,|v_k\rangle' = \sum_{k=0}^{M-1} t_{k,0}\,[\sum_{n=0}^{D-1} a_n\,|\mathrm{S^z\ basis}\rangle_n]$$

$$= \sum_{n=0}^{D-1} [\sum_{k=0}^{M-1} t_{k,0}\ a_n]\,|\mathrm{S^z\ basis}\rangle_n$$

Python coding

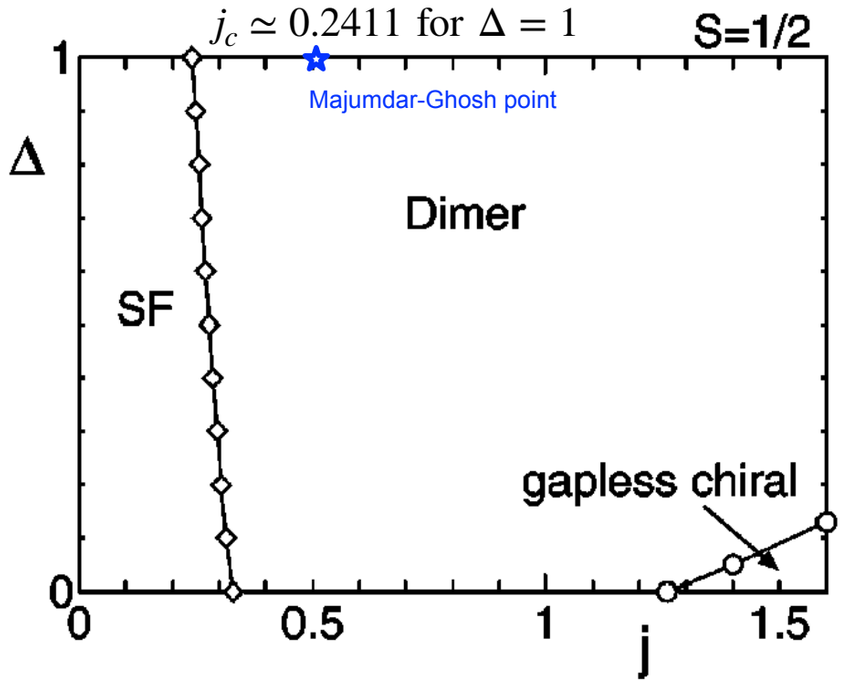
```
101 def simple_lanczos(Jxx,Jzz,list_isite1,list_isite2,N,Nint,Nhilbert,irght,ilft,ihfbit,list_1,list_ja,list_jb,itr_max,eps):
102     alphas = []                          #Diagonal parts of the trigonal matrix
103     betas = [0.]                         #Off-diagonal parts of the trigonal matrix
104     np.random.seed(seed=12345)
105     v1 = np.random.rand(Nhilbert)       #old Lanczos vector (real number vector)
106     v1 /= np.linalg.norm(v1)             #normalization
107     v0 = np.zeros(Nhilbert, dtype=float)  #new Lanczos vector
108     w  = np.zeros(Nhilbert, dtype=float)
109 
110     alpha = 0.
111     beta = 0.
112     pre_energy=0
113 
114     for k in range(0, itr_max):
115         w = ham_to_vec(w,v1,Jxx,Jzz,list_isite1,list_isite2,Nint,Nhilbert,irght,ilft,ihfbit,list_1,list_ja,list_jb)
116         alpha = np.dot(v1,w)
117         w = w -alpha*v1 -beta*v0
118         v0 = np.copy(v1)
119         beta = np.sqrt(np.dot(w,w))
120         v1 = w/beta
121         alphas.append(alpha)
122         betas.append(beta)
123 
124         t_eigs,t_vecs = scipy.linalg.eigh_tridiagonal(alphas,betas[1:-1])
125         print(min(t_eigs)/4)
126 
127         if np.abs(min(t_eigs)-pre_energy) < eps:
128           print("Lanczos converged in", k, "iterations")
129           conv_itr=k #M value in the tutorial slide
130           print("The lowest 5 energy", t_eigs[0:4]/4)
131           break
132 
133         pre_energy = min(t_eigs)
```

```
134 
135 
136     #calcu GS eigenvector
137     np.random.seed(seed=12345)    #set the same seed value we used above
138     v1 = np.random.rand(Nhilbert)
139     v1 /= np.linalg.norm(v1)
140     v0 = np.zeros(Nhilbert, dtype=float)
141     w = np.zeros(Nhilbert, dtype=float)
142     alpha = 0.
143     beta = 0.
144 
145     GS = t_vecs[0,0]*v1  # GS wavefunction from 0-th eigenvector of the tridiagonal matrix
146 
147     for k in range(0,conv_itr-1):
148         w = ham_to_vec(w,v1,Jxx,Jzz,list_isite1,list_isite2,Nint,Nhilbert,irght,ilft,ihfbit,list_1,list_ja,list_jb)
149         alpha = np.dot(v1,w)
150         w = w -alpha*v1 -beta*v0
151         v0 = np.copy(v1)
152         beta = np.sqrt(np.dot(w,w))
153         v1 = w/beta
154 
155         GS = GS + t_vecs[k+1,0]*v1
156         print("eigenvector iteretion", k)
157 
158 
159     return t_eigs, np.array(alphas), np.array(betas[1:]), t_vecs, GS
```

Simple_Lanczos_szconserv.py

Calculation example 1

S=1/2 J_1-J_2 XXZ spin chain

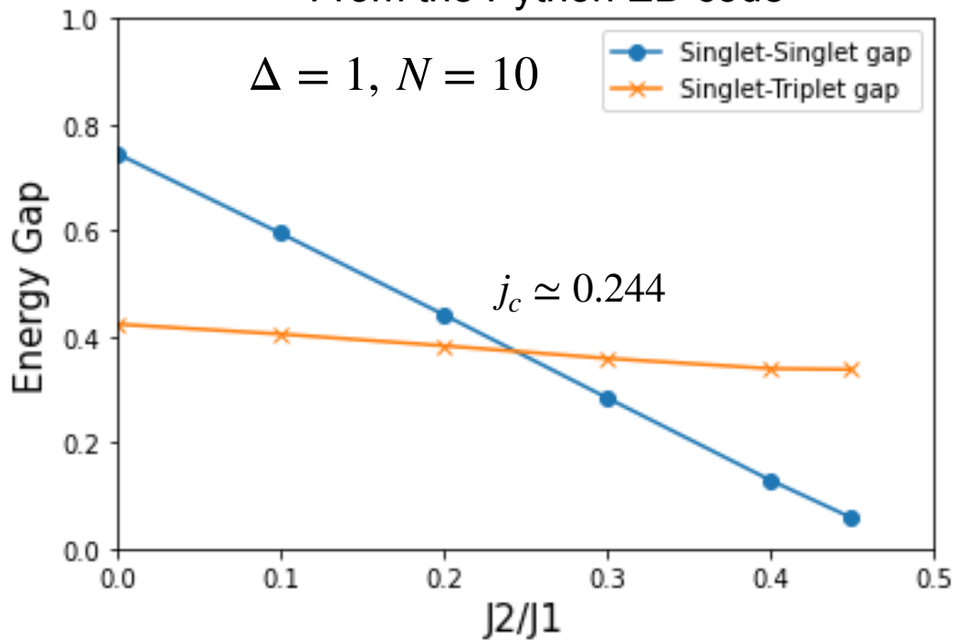


$$\mathcal{H} = \sum_{\rho=1}^{2} \left\{ J_\rho \sum_l \left(S_l^x S_{l+\rho}^x + S_l^y S_{l+\rho}^y + \Delta S_l^z S_{l+\rho}^z \right) \right\}$$

$$j \equiv J_2 / J_1 > 0$$

T. Hikihara et al, Phys. Rev. B. **63**, 174430 (2001).

From the Python ED code



The phase transition between the fluid and dimer states can be obtained by the crossing point in the parameter dependence of the singlet-singlet gap and singlet-triplet gap.

→ K. Okamoto et al, Phys. Lett. A **169**, 433 (1992).

Advantage/disadvantage

- Required computational memory is significantly reduced from that in the full ED method.

Required memory for S=1/2 *N* spin system

Full ED: $8\times 2^N \times 2^N$ [Byte]

N=10	8 [MB]
N=16	32 [GB]
N=20	8 [TB]

Lanczos method : $8\times 2^N \times 3$ [Byte]

N=10	24 [KB]	*N*=24	384 [MB]
N=16	1.5 [MB]	*N*=32	96 [GB]
N=20	24 [MB]	*N*=36	1.5 [TB]

- Lanczos vectors $\{|v_k\rangle'\}$ tend to loose their orthogonality after ground state energy is converged. As a result, we may get “spurious” eigenvalues especially for excited states.
- Weak against degeneracies of eigenvalues. You may need to use other techniques such as thick-restarted Lanczos method and LOBPCG method.

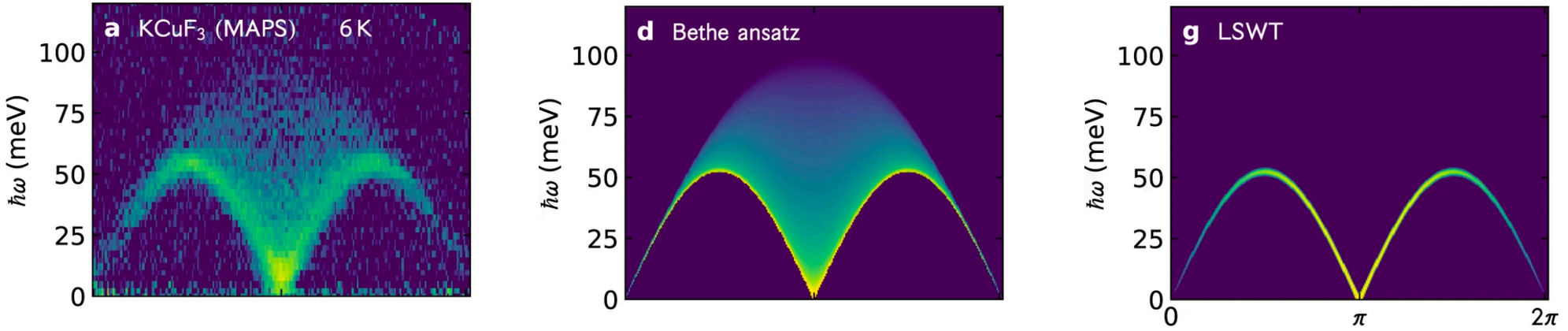
K. Wu et al, J. Comp. Phys. **154**, 156 (1999). A. V. Knyazev et al, SIAM J. Sci. Comp. **23**, 517 (2001).

3. Continued fraction method for spin dynamics

Background

- Spin dynamics is a very useful quantity to know the magnetic order/disorder ground states and the corresponding excitations in quantum magnets.

- Spin dynamics is accessible via several experimental techniques such as inelastic neutron scattering, electron spin resonance, nuclear magnetic resonance.



A. Scheie et al, Nat. Comm. **13**, 5796 (2022).

- Dynamical spin structure factor can be evaluated, based on Lanczos-type procedure.

R. Haydock et al, J. Phys. C:Solid State Phys. **8**, 2591 (1975). E. R. Gagliano et al, Phys. Rev. Lett. **59**, 2999 (1987).

A self-correlation function

- Now we obtain ground state wave function $|\phi_0\rangle$ from the Lanczos method.
  A self-correlation function at zero temperature for an operator A is defined as

$$C_A(t-t') \equiv \langle\phi_0|A^\dagger(t)A(t')|\phi_0\rangle \qquad A(t) = \mathrm{e}^{i\mathscr{H}t/\hbar}A\mathrm{e}^{-i\mathscr{H}t/\hbar}$$

- Consider to use all eigenstates of the *D*-dimensional matrix $\mathscr{H}$ as $1 = \sum_{n=0}^{D-1} |\phi_n\rangle\langle\phi_n|$ and the corresponding eigenvalues $\{E_n\}$ to rewrite the above equation.

$$C_A(t-t') = \sum_{n=0}^{D-1} \langle\phi_0|A^\dagger|\phi_n\rangle\langle\phi_n|A|\phi_0\rangle \mathrm{e}^{(E_n-E_0)(t+t')i/\hbar}$$

The real-frequency correlation

- The Fourier transformation with respect to time (t, t') gives the real-frequency correlation function $C_A(\omega)$.

$$\frac{1}{\pi^2}\int\int C_A(t-t')\mathrm{e}^{-i\omega t}\mathrm{e}^{-i\omega' t'}dtdt' = \sum_{n=0}^{D-1}\langle\phi_0|A^\dagger|\phi_n\rangle\langle\phi_n|A|\phi_0\rangle\frac{1}{\pi}\int \mathrm{e}^{\frac{i}{\hbar}(E_n-E_0-\hbar\omega)t}dt\frac{1}{\pi}\int \mathrm{e}^{\frac{i}{\hbar}(E_n-E_0-\hbar\omega')t'}dt'$$

$$= \sum_{n=0}^{D-1}\langle\phi_0|A^\dagger|\phi_n\rangle\langle\phi_n|A|\phi_0\rangle\delta(E_n-E_0-\omega\hbar)\delta(E_n-E_0-\omega'\hbar)$$

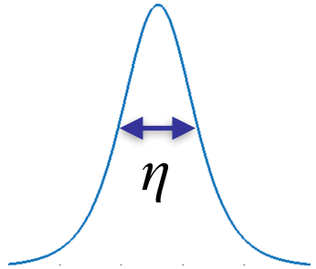
$$= \sum_{n=0}^{D-1}\langle\phi_0|A^\dagger|\phi_n\rangle\langle\phi_n|A|\phi_0\rangle\delta(\omega\hbar+E_0-E_n)$$

$$\equiv C_A(\omega)$$

The real-frequency correlation

- We rewrite $C_A(\omega)$ with the Lorentzian function instead of the delta function as

$$\delta(\omega\hbar + E_0 - E_n) = \lim_{\eta\to 0}\frac{1}{\pi}\frac{\eta}{(\omega\hbar + E_0 - E_n)^2 + \eta^2} \qquad \eta : \text{Lorentzian width}$$



$$= \lim_{\eta\to 0}\frac{1}{\pi}\mathrm{Im}[\frac{-1}{(\omega\hbar + E_0 - E_n) + i\eta}]$$

$$C_A(\omega) = \sum_{n=0}^{D-1}\langle\phi_0|A^\dagger|\phi_n\rangle\langle\phi_n|A|\phi_0\rangle\delta(\omega\hbar + E_0 - E_n)$$

$$= \lim_{\eta\to 0}\{-\frac{1}{\pi}\mathrm{Im}[\langle\phi_0|A^\dagger\frac{1}{(\omega\hbar + E_0 - \mathscr{H} + i\eta)}A|\phi_0\rangle]\}$$

Continued fraction method

- We can deal with the resolvent part in the form of a continued fraction.

$$C_A(\omega) = \lim_{\eta\to 0}\{-\frac{1}{\pi}\mathrm{Im}[\langle\phi_0|A^\dagger\frac{1}{(\omega\hbar + E_0 - \mathscr{H} + i\eta)}A|\phi_0\rangle]\}$$

$$= \lim_{\eta\to 0}\{-\frac{1}{\pi}\mathrm{Im}[\frac{\langle\phi_0|A^\dagger A|\phi_0\rangle}{Z-\alpha_0-\dfrac{\beta_1^2}{Z-\alpha_1-\dfrac{\beta_2^2}{Z-\alpha_2-\dfrac{\beta_3^2}{Z-\cdots}}}}]\}$$

- The coefficients α_k, β_k can be evaluated from the moments $\langle\phi_0|A^\dagger\mathscr{H}^k A|\phi_0\rangle$.

  That means α_k, β_k are equal to the elements of the tridiagonal matrix obtained when we use $A|\phi_0\rangle$ as the initial vector in the Lanczos procedure.

Dynamical spin structure factor (DSSF)

- For example, *z*-component of the DSSF is defined as

$$S^z(\mathbf{q},\omega) = \lim_{\eta\to 0}\{-\frac{1}{\pi}\mathrm{Im}[\langle\phi_0|S^{z\dagger}_{\mathbf{q}}\frac{1}{(\omega\hbar + E_0 - \mathscr{H} + i\eta)}S^z_{\mathbf{q}}|\phi_0\rangle]\}$$

$$= \lim_{\eta\to 0}\{-\frac{1}{\pi}\mathrm{Im}[\frac{\langle\phi_0|S^{z\dagger}_{\mathbf{q}}S^z_{\mathbf{q}}|\phi_0\rangle}{Z-\alpha_0-\frac{\beta_1^2}{Z-\alpha_1-\frac{\beta_2^2}{Z-\alpha_2-\frac{\beta_3^2}{Z-\cdots}}}}]\}$$

$$S^z_{\mathbf{q}} = \frac{1}{\sqrt{N}}\sum_i S^z_i e^{i\mathbf{q}\cdot\mathbf{R}_i} \qquad \mathbf{R}_i \text{ is the position vector for site } i$$

α_k, β_k are obtained from Lanczos procedure up to $k = \mathrm{itr\_dsf}$ iterations

- In the actual calculation, we set small value in η such as $\eta = 0.02$.

Python coding

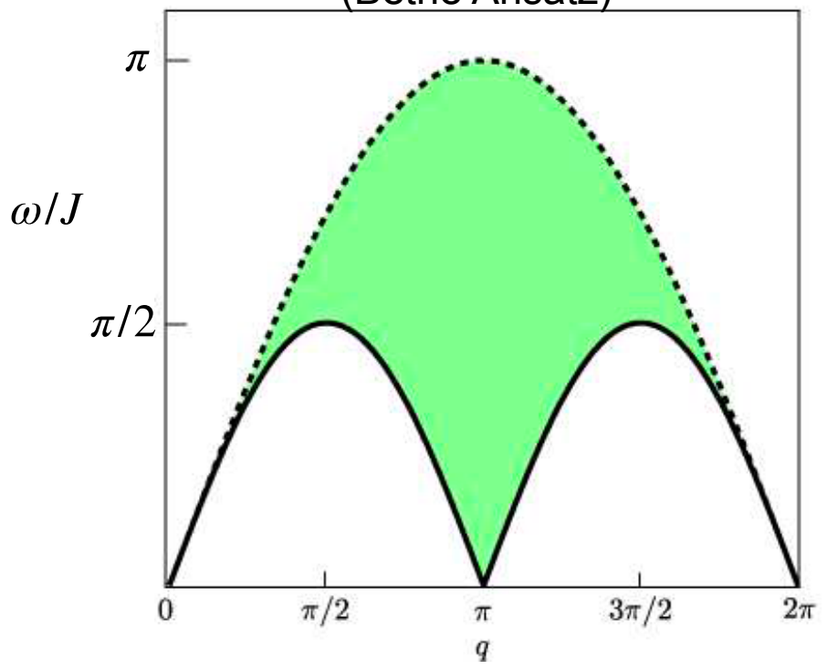
```python
def sz_spin_dynamics(itr_dsf,qx,GS,Jxx,Jzz,list_isite1,list_isite2,N,Nint,Nhilbert,irght,ilft,ihfbit,list_1,list_ja,list_jb)

    salphas = []
    sbetas = [0.]

    #calculate |v1> = Sq^z |GS>
    v1 = np.zeros(Nhilbert, dtype=complex)
    v0 = np.zeros(Nhilbert, dtype=complex)
    w = np.zeros(Nhilbert,dtype=complex)
    tmp = np.zeros(2,dtype=float)
    for n in range(Nhilbert):
         ii = list_1[n]
         tmp[0:2]=0.0

         for i in range(N): #We define R_i=i for 1D chain
           is1 = 1<<i
           ibit = ii & is1

           if(ibit==is1): #site-i has up spin
              tmp[0] += np.cos(qx*i)
              tmp[1] += np.sin(qx*i)
           else:
              tmp[0] -= np.cos(qx*i)
              tmp[1] -= np.sin(qx*i)

         v1[n] = GS[n]*tmp[0]/2.0 + 1.0j*GS[n]*tmp[1]/2.0

    v1 = v1/np.sqrt(N)
    norm = np.vdot(v1,v1)
    v1 /= np.linalg.norm(v1)

    salpha=0.
    sbeta=0.

    for itr in range(0,itr_dsf):
        w = ham_to_vec_complex(w,v1,Jxx,Jzz,list_isite1,list_isite2,Nint,Nhilbert,irght,ilft,ihfbit,list_1,list_ja,list_jb)
        salpha = np.vdot(v1,w)
        w = w - salpha*v1 -sbeta*v0
        v0 = np.copy(v1)
        sbeta = np.sqrt(np.vdot(w,w))
        v1 = w/sbeta
        salphas.append(salpha)
        sbetas.append(sbeta)

    return salphas, sbetas, norm
```

[Simple_spindynamics_szconserv.py]

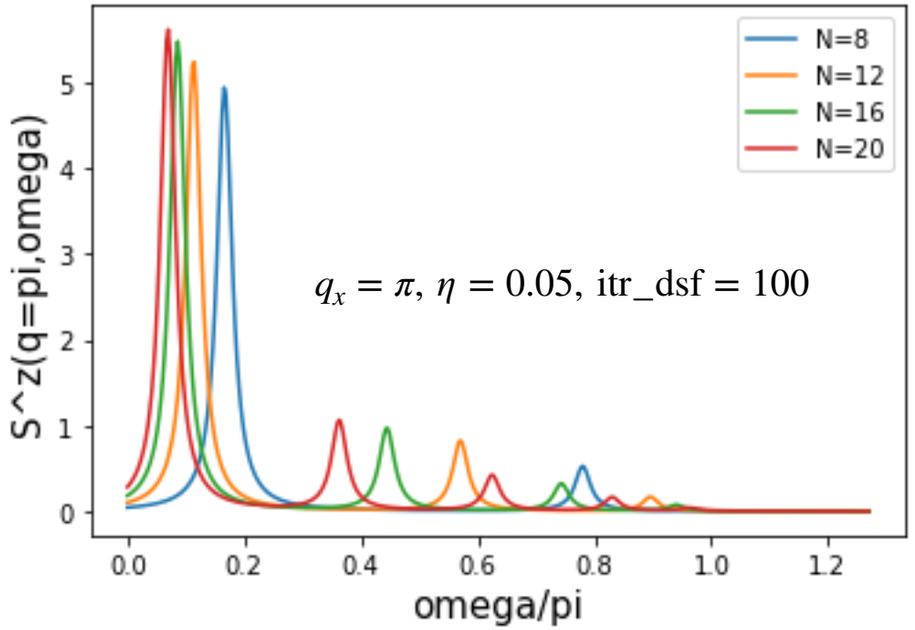
```python
def continued_fraction(norm,salphas,sbetas,itr_dsf,eta,minene):

    print(norm,minene,eta)
    Sqw = np.zeros((1000,2),dtype=float)
    for omega in range(0,1000):
        Z = omega*0.01 + 1.0j*eta + minene
        tmp = 0.0 + 0.0j
        for itr in reversed(range(1,itr_dsf)):
            tmp = Z - salphas[itr] - tmp
            tmp = (sbetas[itr]**2)/tmp
        tmp = Z - salphas[0] - tmp
        Sqw[omega,:]=omega*0.01, -((norm/tmp).imag)/np.pi

    return Sqw
```

Calculation example 2

S=1/2 AF Heisenberg chain
(Bethe Ansatz)



From the Python DSSF code



I. Umegaki et al, Phys. Rev. B **92**,174412 (2015).

M. Takahashi, *Thermodynamics of One-Dimensional Solvable Models*
(Cambridge University Press, 1999).

4. Finite-T Lanczos method for $T \neq 0$ physics

Background

- The effect of the thermal fluctuation on the frustrated systems is interesting problem from the view point of “order by disorder” phenomenon.
- The numerical data of thermodynamic quantities are important also in experiment.
- However, quantum frustrated magnets are less suitable for QMC method, the full ED method has severe limitations in accessible system sizes.
- Complemental methods without ensemble average have been used to investigate thermal nature of quantum frustrated magnets (cf. finite-temperature Lanczos method, transfer Monte Carlo method, Hams-de Raedt method, thermal pure quantum state method…).

Finite-temperature Lanczos method (FTLM)

J. Jaklic and P. Prelovsek, Phys. Rev. B **49**, 5065 (1994).

- A conventional way is to use the canonical ensemble average which requires all eigenvalues E_n and eigenvectors $|\psi_n\rangle$ of the target D-dimensional matrix $\mathscr{H}$.

$$\langle A\rangle = \frac{1}{Z}\sum_{n=0}^{D-1} \mathrm{e}^{-\beta E_n}\langle\psi_n|A|\psi_n\rangle \qquad Z = \sum_{n=0}^{D-1}\mathrm{e}^{-\beta E_n} \qquad \beta = 1/T$$

- This canonical expectation can be expressed by using general orthonormal basis states $|n\rangle$ as the following equation.

$$\langle A\rangle = \frac{1}{Z}\sum_{n=0}^{D-1} \langle n|\mathrm{e}^{-\beta\mathscr{H}}A|n\rangle \qquad Z = \sum_{n=0}^{D-1}\langle n|\mathrm{e}^{-\beta\mathscr{H}}|n\rangle$$

Finite-temperature Lanczos method (FTLM)

- Let's consider to expand $\mathrm{e}^{-\beta\mathscr{H}}$ by β (high-T expansion).

$$\langle A\rangle = \frac{1}{Z}\sum_{n=0}^{D-1}\sum_{l=0}^{\infty}\frac{(-\beta)^l}{l!}\langle n\,|\,\mathscr{H}^l A\,|\,n\rangle \qquad Z = \sum_{n=0}^{D-1}\sum_{l=0}^{\infty}\frac{(-\beta)^l}{l!}\langle n\,|\,\mathscr{H}^l\,|\,n\rangle$$

- We can evaluate each term $\langle n\,|\,\mathscr{H}^l A\,|\,n\rangle$ by using Lanczos method when we set $|\,n\rangle$ as the initial random vector of the method

$$|\,n\rangle \equiv |\,v_0^{(n)}\rangle = \sum_{m=0}^{D-1} c_m^{(n)}\,|\,\mathrm{S^z\ basis}\rangle_m$$

$$\sum_{m=0}^{D-1}|\,c_m^{(n)}\,|^2 = 1$$

Lanczos method

Iteration k=M>l

Lanczos vectors

$$\{\,|\,v_0^{(n)}\rangle, |\,v_1^{(n)}\rangle \ldots |\,v_{M-1}^{(n)}\rangle, |\,v_{k=M}^{(n)}\rangle\}$$

Eigenvalues/eigenvectors of tridiagonal matrix $\mathbf{T}^{(n)}$

$$\{\epsilon_0^{(n)}, \epsilon_1^{(n)} \ldots, \epsilon_{M-1}^{(n)}\}$$

$$\{\,|\,\phi_0^{(n)}\rangle, |\,\phi_1^{(n)}\rangle \ldots, |\,\phi_{M-1}^{(n)}\rangle\}$$

Finite-temperature Lanczos method (FTLM)

- Each term of $\langle n|\mathscr{H}^l A|n\rangle$ is written as $\langle n|\mathscr{H}^l A|n\rangle = \sum_{k=0}^{M-1} \langle v_0^{(n)}|\phi_k^{(n)}\rangle\langle\phi_k^{(n)}|A|v_0^{(n)}\rangle(\epsilon_k^{(n)})^l$

  because $\mathscr{H}$ is diagonal in the basis of $\{|\phi_0^{(n)}\rangle, |\phi_1^{(n)}\rangle \ldots, |\phi_{M-1}^{(n)}\rangle\}$ so that the

  expectation and partition function are written as

$$\langle A\rangle = \frac{1}{Z}\sum_{n=0}^{D-1}\sum_{k=0}^{M-1} \mathrm{e}^{-\beta\epsilon_k^{(n)}}\langle v_0^{(n)}|\phi_k^{(n)}\rangle\langle\phi_k^{(n)}|A|v_0^{(n)}\rangle \qquad Z = \sum_{n=0}^{D-1}\sum_{k=0}^{M-1} \mathrm{e}^{-\beta\epsilon_k^{(n)}}\langle v_0^{(n)}|\phi_k^{(n)}\rangle\langle\phi_k^{(n)}|v_0^{(n)}\rangle$$

Sum over initial random vectors Lanczos part for each different initial random vector $|v_0^{(n)}\rangle$

↳ Approximate this sum by the summation over few $R \ll D$ realizations of initial random vectors.

$$\sum_{n=0}^{D-1} \to \frac{D}{R}\sum_{r=1}^{R\ll D} + (\text{a statistical error})$$

Finite-temperature Lanczos method (FTLM)

- Finally, we can get the following equations.

$$\langle A \rangle \simeq \frac{D}{RZ} \sum_{r=1}^{R} \sum_{k=0}^{M-1} \mathrm{e}^{-\beta \epsilon_k^{(r)}} \langle v_0^{(r)} | \phi_k^{(r)} \rangle \langle \phi_k^{(r)} | A | v_0^{(r)} \rangle \qquad Z \simeq \frac{D}{R} \sum_{r=1}^{R} \sum_{k=0}^{M-1} \mathrm{e}^{-\beta \epsilon_k^{(r)}} | \langle v_0^{(r)} | \phi_k^{(r)} \rangle |^2$$

- The statistical errors scale $\mathcal{O}(1/\sqrt{R})$ for low *T* and $\mathcal{O}(1/\sqrt{RD})$ for high *T*.

- When the observable A is commutable with the matrix $\mathscr{H}$, it is very easy to calculate the expectation values.

Total energy

$$\langle E \rangle \simeq \frac{D}{RZ} \sum_{r=1}^{R} \sum_{k=0}^{M-1} \epsilon_k^{(r)} \mathrm{e}^{-\beta \epsilon_k^{(r)}} | \langle v_0^{(r)} | \phi_k^{(r)} \rangle |^2$$

Specific heat

$$\langle C \rangle \simeq \frac{D}{T^2 RZ} \sum_{r=1}^{R} \sum_{k=0}^{M-1} | \epsilon_k^{(r)} |^2 \mathrm{e}^{-\beta \epsilon_k^{(r)}} | \langle v_0^{(r)} | \phi_k^{(r)} \rangle |^2 - \frac{| \langle E \rangle |^2}{T^2}$$

$\langle v_0^{(r)} | \phi_k^{(r)} \rangle$ corresponds to the 0-th component of the $| \phi_k^{(r)} \rangle$

Python coding

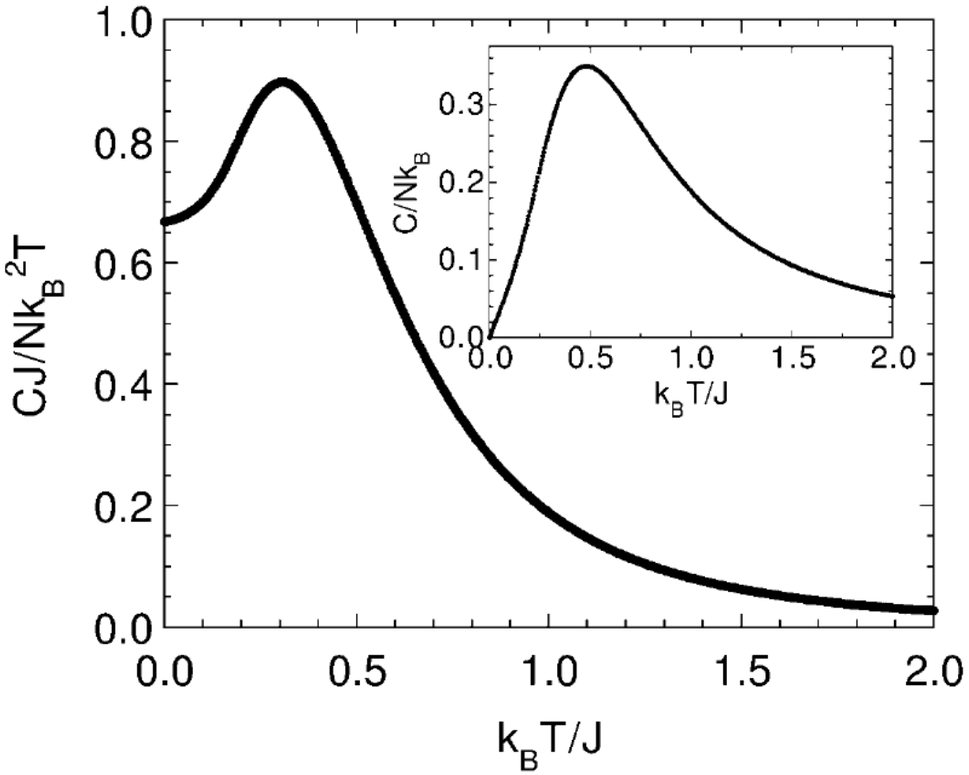
```python
def simple_FTL_nosz_conserv(R,M,Jxx,Jzz,list_isite1,list_isite2,N,Nint,seed0):

    Nhilbert = 2**N
    epsilons = np.zeros((R,M),dtype=float)
    vphi = np.zeros((R,M),dtype=float)
    for r in range(R): # random sampling
        np.random.seed(seed=seed0+r)

        alphas = []                             #Diagonal parts of the trigonal matrix
        betas = [0.]                            #Off-diagonal parts of the trigonal matrix
        v1 = (1+1)*np.random.rand(Nhilbert)-1   #Initial random vector, |phi_0^(r)>
        v1 /= np.linalg.norm(v1)                #normalization
        v0 = np.zeros(Nhilbert, dtype=float)    #Lanczos vector |phi_^(r)>
        w  = np.zeros(Nhilbert, dtype=float)    #Lanczos vector |phi_^(r)>

        alpha = 0.
        beta = 0.
        pre_energy=0

        for k in range(0, M): # Lanczos iteration
            w = ham_to_vec_nosz_conserv(w,v1,Jxx,Jzz,list_isite1,list_isite2,N,Nint,Nhilbert)
            alpha = np.dot(v1,w)
            w = w -alpha*v1 -beta*v0
            v0 = np.copy(v1)
            beta = np.sqrt(np.dot(w,w))
            v1 = w/beta
            alphas.append(alpha)
            betas.append(beta)

        t_eigs,t_vecs = scipy.linalg.eigh_tridiagonal(alphas,betas[1:-1])
        epsilons[r,:]=t_eigs[:]/4
        minene = min(t_eigs)/4
        vphi[r,:]=t_vecs[0,:] # The 0-th component of the eigenvector, <V_0^(r)|phi_k^(r)>

    return epsilons,vphi,minene
```

```python
def calc_Tdep_ene(R,M,epsilons,vphi,minene):

    TdepEne = np.zeros((2000,2),dtype=float)
    epsilons0 = epsilons - minene
    for T0 in range(1,2000):
        T= T0*0.01
        beta = 1.0/T
        PartZ = 0.0
        ene = 0.0

        for r in range(R):
            for k in range(M):
                PartZ += np.exp(-beta*epsilons0[r,k])*abs(vphi[r,k])**2
                ene  +=  epsilons0[r,k]*np.exp(-beta*epsilons0[r,k])*abs(vphi[r,k])**2

        TdepEne[T0,:] =T, ene/PartZ+minene

    return TdepEne

def calc_Tdep_C(R,M,epsilons,vphi,minene,TdepEne):

    TdepC = np.zeros((2000,2),dtype=float)
    epsilons0 = epsilons - minene
    for T0 in range(1,2000):
        T = T0*0.01
        beta = 1.0/T
        PartZ = 0.0
        C = 0.0

        for r in range(R):
            for k in range(M):
                PartZ += np.exp(-beta*epsilons0[r,k])*abs(vphi[r,k])**2
                C  +=  (abs(epsilons[r,k])**2)*np.exp(-beta*epsilons0[r,k])*abs(vphi[r,k])**2

        TdepC[T0,:] = T, C/PartZ/T/T - abs(TdepEne[T0,1])**2/T/T

    return TdepC
```

Simple_FiniteTLanczos.py

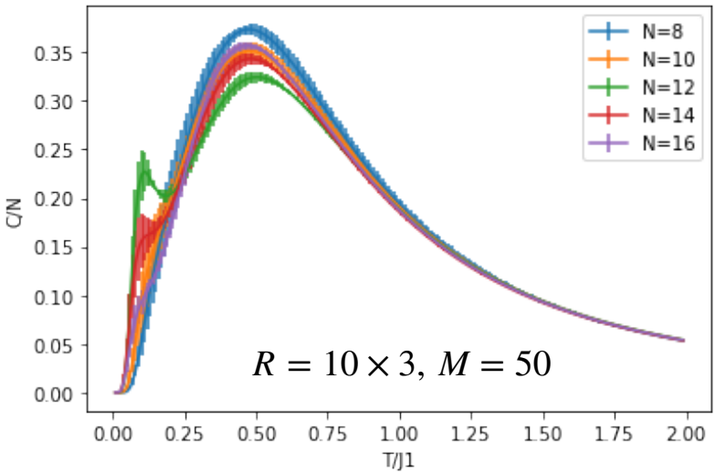
Calculation example 3

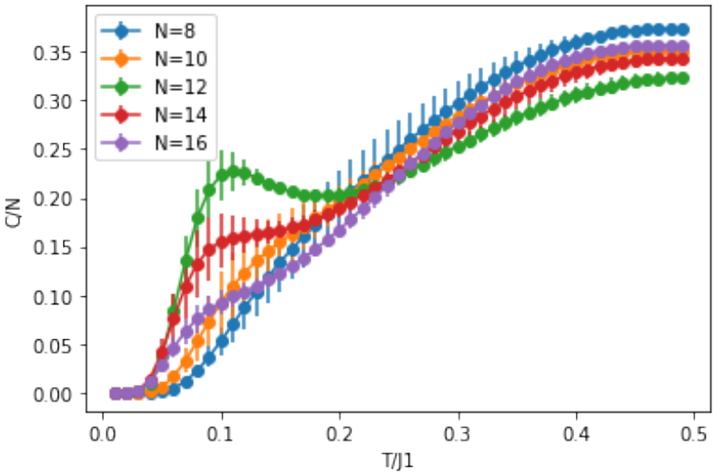
S=1/2 AF Heisenberg chain (Bethe Ansatz)



A. Klümper et al, Phys. Rev. Lett. **84**,4701 (2000).

From the Python FTL code





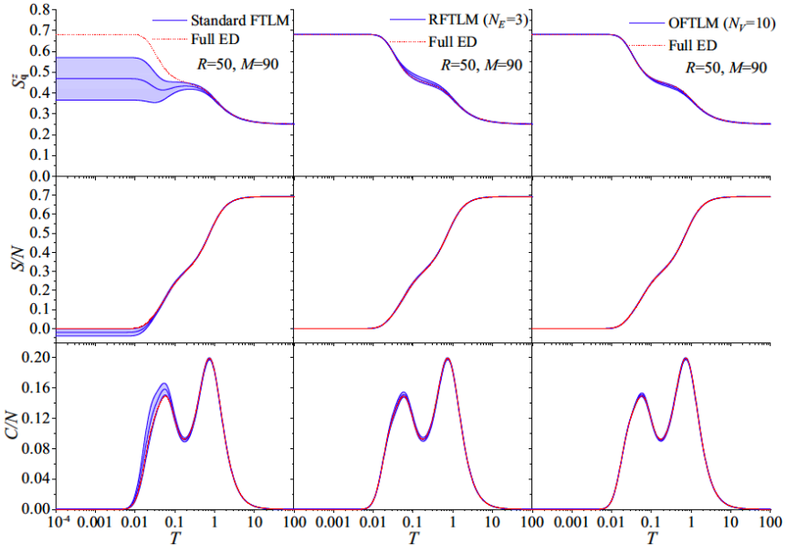
Advantage/disadvantage

- Required computational memory is significantly reduced from that in the full ED method. (same as in the Lanczos method)

- Accuracy at low temperatures becomes worse (big statistical error in $T \to 0$). Some improvement techniques are needed such as replaced FTLM, orthogonalized FTLM…

- Dynamical quantities can also be computed based on FTLM. The details of the method are given by P. Prelovsek and J. Bonca as a book →

K. Morita et al, Phys. Rev. Res. **2**, 013205 (2020).



Part 2

5. Binary representations and bit operations

Background

- An iteration algorithm such as Lanczos method often includes a multiplication of the Hamiltonian matrix to a vector.

$$|v_m\rangle \equiv \mathscr{H}^m |v_0\rangle = (\sum_{i,j} J_{i,j}\, \mathbf{S}_i \cdot \mathbf{S}_j)^m |v_0\rangle$$

- For S=1/2 spin systems, binary representations and bit operations are efficient ways to speed up the above multiplication.

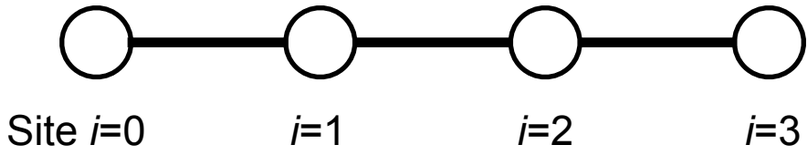
- Look-up/look-back tables between basis states are also important for a fast code.

- Recalculating the Hamiltonian matrix elements “on the fly” in each iteration makes us free to store the matrix elements in our computer.

Binary representations for S=1/2

We usually use binary representation to describe basis states in our computer.

Ex.) S=1/2 Heisenberg chain, *N*=4



$$\mathscr{H} = \sum_{i=0}^{2} J\ \mathbf{S}_i \cdot \mathbf{S}_{i+1}$$

# of basis states

$$2^4 = 16$$

A basis state	Binary representation	Decimal representation
$\lvert \uparrow \uparrow \downarrow \downarrow \rangle$	$\lvert 1100 \rangle$	$2^3 + 2^2 = 12$

Site index → 3 2 1 0

With 8-byte integers (64 bit), the bit representation works up to *N*=64.

Bitwise operations

Multiplication of the Hamiltonian matrix to a vector is decomposed into the iterations of the bit operations for basis states in our ED code.

$$|v_m\rangle \equiv \mathscr{H}^m |v_0\rangle = \Big(\sum_{i,j} J_{i,j}\, \mathbf{S}_i \cdot \mathbf{S}_j\Big)^m |v_0\rangle$$

$$|v_0\rangle = \sum_{n=0}^{2^N-1} c_n |\text{basis}\rangle_n, \quad \sum_n |c_n|^2 = 1$$

The iterations of the following operations

$$(S_i^+ S_j^- + S_i^- S_j^+)\,|\text{basis}\rangle_n$$

$$S_i^z\, S_j^z\,|\text{basis}\rangle_n$$

The off-diagonal terms of the Heisenberg model change the basis state.

Ex.)

$$(S_1^+ S_2^- + S_1^- S_2^+)\,|\uparrow\uparrow\downarrow\downarrow\rangle = |\uparrow\downarrow\uparrow\downarrow\rangle$$

3 2 1 0

Basis state	Binary representation	Decimal representation
$\|\uparrow\uparrow\downarrow\downarrow\rangle$	$\|1100\rangle$	$2^3 + 2^2 = 12$
$\|\uparrow\downarrow\uparrow\downarrow\rangle$	$\|1010\rangle$	$2^3 + 2^1 = 10$

Bitwise operations

$$(S_1^+ S_2^- + S_1^- S_2^+)|\uparrow\uparrow\downarrow\downarrow\rangle = |\uparrow\downarrow\uparrow\downarrow\rangle$$
3 2 1 0

Basis state	Binary representation	Decimal representation
$\|\uparrow\uparrow\downarrow\downarrow\rangle$	$\|1100\rangle$	$2^3 + 2^2 = 12$
$\|\uparrow\downarrow\uparrow\downarrow\rangle$	$\|1010\rangle$	$2^3 + 2^1 = 10$

XOR (exclusive or) operation is used to flip two spins.

Input		Output
A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

For example in python, “^” is the XOR gate.

```
1 x=10
2 y=7
3 x ^ y

13
```

x=10 → 1010 (binary)

y=7 → 0111 (binary)

x ^ y = 1101 (binary)

Bitwise operations

$$(S_1^+ S_2^- + S_1^- S_2^+)|\uparrow\uparrow\downarrow\downarrow\rangle = |\uparrow\downarrow\uparrow\downarrow\rangle$$
3 2 1 0

Basis state	Binary representation	Decimal representation
$\|\uparrow\uparrow\downarrow\downarrow\rangle$	$\|1100\rangle$	$2^3 + 2^2 = 12$
$\|\uparrow\downarrow\uparrow\downarrow\rangle$	$\|1010\rangle$	$2^3 + 2^1 = 10$

When you would like to flip sites 1 and 2 spins with off-diagonal term,

1. Set original spin basis in decimal number (ii)
2. Set up-up spin state only at sites 1 and 2 in decimal number (is12)
3. Do XOR operation for above two integers

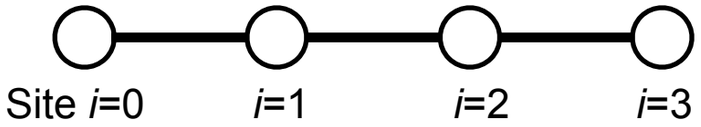
```
1 ii=12
2 is12= 2**1 + 2**2
3 ii ^ is12

10
```

(Binary)

1100
0110
1010

Multiplication of the Hamiltonian matrix to a vector

Ex.) S=1/2 Heisenberg chain, N=4



$$\mathscr{H} = \sum_{i=0}^{2} J\ \mathbf{S}_i \cdot \mathbf{S}_{i+1}$$

# of basis states

$$2^4 = 16$$

Input vector

$$|v_1\rangle = \sum_{n=0}^{2^N-1} c_n |\text{basis}\rangle_n$$

$$= \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{15} \end{pmatrix} \begin{matrix} \cdots|\downarrow\downarrow\downarrow\downarrow\rangle & 0 \\ \cdots|\downarrow\downarrow\downarrow\uparrow\rangle & 1 \\ \cdots|\downarrow\downarrow\uparrow\downarrow\rangle & 2 \\ \cdots|\downarrow\downarrow\uparrow\uparrow\rangle & 3 \\ & \\ \cdots|\uparrow\uparrow\uparrow\uparrow\rangle & 15 \end{matrix}$$

Output vector

$$|w\rangle \equiv \mathscr{H}|v_1\rangle = \sum_{i=0}^{2} J\ \mathbf{S}_i \cdot \mathbf{S}_{i+1} |v_1\rangle$$

$$|w\rangle \equiv \begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_{15} \end{pmatrix} = J\mathbf{S}_0 \cdot \mathbf{S}_1 \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{15} \end{pmatrix} + J\mathbf{S}_1 \cdot \mathbf{S}_2 \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{15} \end{pmatrix} + J\mathbf{S}_2 \cdot \mathbf{S}_3 \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{15} \end{pmatrix}$$

Multiplication of the Hamiltonian matrix to a vector

$$J\mathbf{S}_i \cdot \mathbf{S}_j \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{15} \end{pmatrix} = \frac{J}{2}(S_i^+ S_j^- + S_i^- S_j^+) \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{15} \end{pmatrix} + JS_i^z S_j^z \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_{15} \end{pmatrix}$$

Which basis contributes
to which basis of the output vector?

$$\begin{pmatrix} w_0 \\ w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_{15} \end{pmatrix} \quad \begin{matrix} \cdots|\downarrow\downarrow\downarrow\downarrow\rangle & 0 \\ \cdots|\downarrow\downarrow\downarrow\uparrow\rangle & 1 \\ \cdots|\downarrow\downarrow\uparrow\downarrow\rangle & 2 \\ \cdots|\downarrow\downarrow\uparrow\uparrow\rangle & 3 \\ & \\ \cdots|\uparrow\uparrow\uparrow\uparrow\rangle & 15 \end{matrix}$$

For n-th basis, if i and j spins are $\uparrow\uparrow$ or $\downarrow\downarrow$,

$$\frac{J}{4}c_n \text{ is added in } w_n$$

Otherwise,

$$-\frac{J}{4}c_n + \frac{J}{2}c_{n'} \text{ is added in } w_n$$

Integer n' is obtained with fliping i and j spins.

i and j spin configurations
are obtained by another bitwise gate AND

```
ii=2                    # 2nd basis = |down-down-up-down> state
is12=2**1 + 2**2 # For i=1 and j=2 sites
ibit = ii & is12
if(ibit==0 or ibit==is12):
   print("i and j spins are up-up or down-down")
else:
   print("i and j spins are up-down or down-up")

```

```
i and j spins are up-down or down-up
```

We don't have to store the matrix Hamiltonian elements during the multiplications.

Multiplication of the Hamiltonian matrix to a vector

$$|w\rangle \equiv \mathscr{H}\,|v_1\rangle = \sum_{i,j} J_{i,j}\,\mathbf{S}_i \cdot \mathbf{S}_j\,|v_1\rangle$$

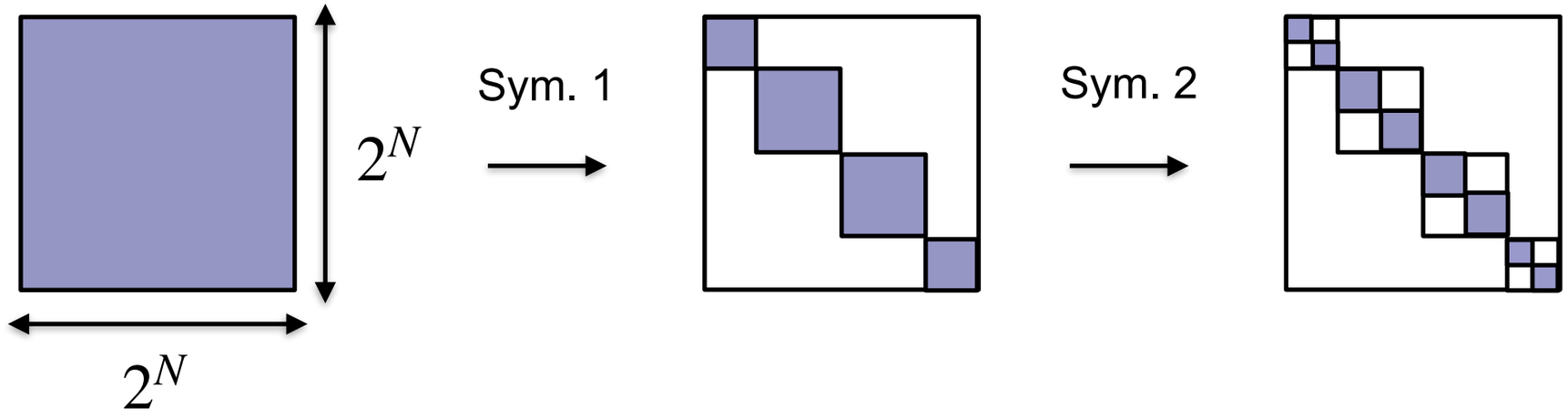
```
11  def ham_to_vec_nosz_conserv(w,v1,Jxx,Jzz,list_isite1,list_isite2,N,Nint,Nhilbert):
12      w = np.zeros(Nhilbert,dtype=float) #output vector
13
14      for n in range(Nhilbert):    #loop for all basis state (from 0 to 2**N-1)
15         for ij in range(Nint):    #loop for all interaction (Nint = # of interaction J_ij)
16          isite1 = list_isite1[ij] # site i for J_ij S_i S_j
17          isite2 = list_isite2[ij] # site j for J_ij S_i S_j
18          is1 = 1<<isite1
19          is2 = 1<<isite2
20          is12 = is1 + is2         #up-up spin state only at i=isite1 and j=isite2 sites
21          wght = 2.0*Jxx[ij]       #The transverse J_ij
22          diag = Jzz[ij]           #The longitudinal J_ij
23          ibit = n & is12          #To check the spin config at i=isite1 and j=isite2 sites
24
25          if (ibit==0 or ibit==is12):  #if i and j sites have up-up or down-down spin configs.
26               w[n] += diag*v1[n]      #For S_i^z S_j^z term
27          else:                        #if i and j sites have up-down or down-up spin configs.
28              w[n] -= diag*v1[n]       #For S_i^z S_j^z term
29              iexchg = n ^ is12        #Flip two spins and get to know the new basis number (decinal number)
30              w[n] += wght*v1[iexchg]  #For (S_i^+ S_j^- + S_i^- S_j^+) term
31      return w
32
```

Simple_FiniteTLanczos.py

6. Block diagonalization and symmetries

Block diagonalization

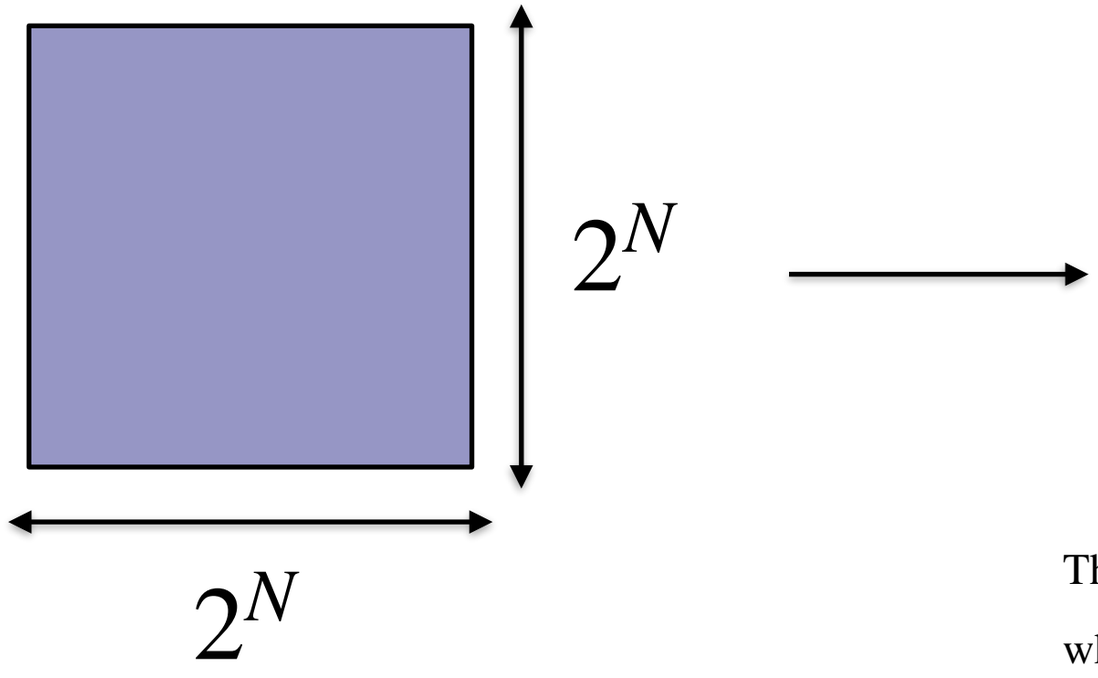
- Basis states labeled with a conserved quantum number can break up the matrix into blocks.
- Conserved quantum numbers are related to some symmetries such as translational symmetry, U(1) symmetry, reflection symmetry…
- Each block can be diagonalized independently (the required memory is significantly reduced).

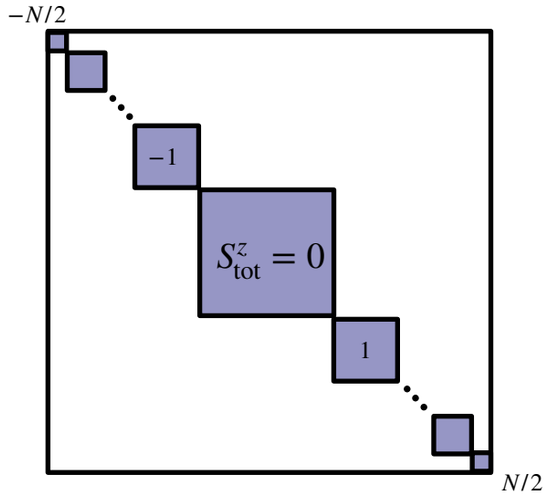


Block diagonalization by magnetization

z-component of the total spin is conserved in a S=1/2 XXZ model on a lattice.

$$\mathscr{H} = \sum_{i,j} \frac{J^{xy}_{i,j}}{2} (S^+_i S^-_j + S^-_i S^+_j) + J^z_{i,j} S^z_i S^z_j \qquad S^z_{\mathrm{tot}} = \sum_i^N S^z_i \qquad [S^z_{\mathrm{tot}}, \mathscr{H}] = 0$$



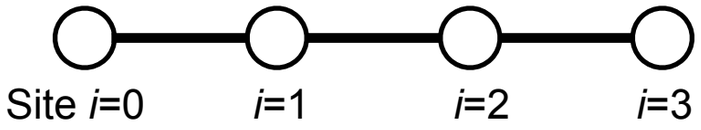


The dimension of each block is $N!/(N-N_\downarrow)!/N_\downarrow!$,

where $N_\downarrow$ is # of down spin. (assume N is even)

Construction of a lookup/back tables labeled by magnetization

Ex.) S=1/2 Heisenberg chain, *N*=4



$$\mathscr{H} = \sum_{i=0}^{2} J\ \mathbf{S}_i \cdot \mathbf{S}_{i+1}$$

# of basis states

$$2^4 = 16$$

Without any symmetries

Serial #	Basis state	Basis state (decimal)
0	$\|\downarrow\downarrow\downarrow\downarrow\rangle$	0
1	$\|\downarrow\downarrow\downarrow\uparrow\rangle$	1
2	$\|\downarrow\downarrow\uparrow\downarrow\rangle$	2
3	$\|\downarrow\downarrow\uparrow\uparrow\rangle$	3
14	$\|\uparrow\uparrow\uparrow\uparrow\rangle$	14
15	$\|\uparrow\uparrow\uparrow\uparrow\rangle$	15

Consider $S^z_{\text{tot}} = 0$ sector

Serial #	Basis state	Basis state (decimal)
0	$\|\downarrow\downarrow\uparrow\uparrow\rangle$	3
1	$\|\downarrow\uparrow\downarrow\uparrow\rangle$	5
2	$\|\downarrow\uparrow\uparrow\downarrow\rangle$	6
3	$\|\uparrow\downarrow\downarrow\uparrow\rangle$	9
4	$\|\uparrow\downarrow\uparrow\downarrow\rangle$	10
$4!/2!/2! - 1 = 5$	$\|\uparrow\uparrow\downarrow\downarrow\rangle$	12

The serial # and basis state (decimal) are stored in a lookup table.

In our ED code, “list_1” is used. [list_1(0)=3, list_1(1)=5,…list_1(5)=12]

Construction of a lookup/back tables labeled by magnetization

Look-up table gives the basis state (decimal) for a given serial #.

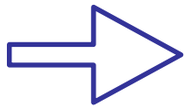
$$\text{list\_1(serial \#)} = \text{basis state [decimal]}$$

While, the look-back table (inverse table) gives the serial # for a given basis state.

$$\text{list\_2(basis state [decimal])} = \text{serial \#}$$

Ex.) S=1/2 Heisenberg chain, N=4 and $S^z_{\text{tot}} = 0$

Serial #	Basis state	Basis state (decimal)
0	$\lvert\downarrow\downarrow\uparrow\uparrow\rangle$	3
1	$\lvert\downarrow\uparrow\downarrow\uparrow\rangle$	5
2	$\lvert\downarrow\uparrow\uparrow\downarrow\rangle$	6
3	$\lvert\uparrow\downarrow\downarrow\uparrow\rangle$	9
4	$\lvert\uparrow\downarrow\uparrow\downarrow\rangle$	10
$4!/2!/2! - 1 = 5$	$\lvert\uparrow\uparrow\downarrow\downarrow\rangle$	12



Look-up table	Look-back table
list_1(0) = 3	list_2(3) = 0
list_1(1) = 5	list_2(5) = 1
list_1(2) = 6	list_2(6) = 2
list_1(3) = 9	list_2(9) = 3
list_1(4) = 10	list_2(10) = 4
list_1(5) = 12	list_2(12) = 5

Problem: The maximum integer argument in list_2 will be too big for a large N.
(Waste of the computational memory)

Construction of a lookup/back tables labeled by magnetization

Masao Ogata and H.Q. Lin solved this problem by splitting the basis state (binary) into two parts.

Rule 1. Split the binary representation of basis states into most significant bit (ib) and the least significant bit (ia).

Rule 2. “ja” starts from 0 and will be increased by 1 as “ia” changes as long as “ib” is unchanged.
When “ib” is changed “ja” starts again from 0.

Rule 3. “jb” starts from 0 and will be changed into previous “ja+jb+1” when “ib” changes.

Serial #	Basis state	Basis state (decimal)	ib	ia	jb	ja	ja+jb
0	$\|0011\rangle$	3	00	11	0	0	0
1	$\|0101\rangle$	5	01	01	1	0	1
2	$\|0110\rangle$	6	01	10	1	1	2
3	$\|1001\rangle$	9	10	01	3	0	3
4	$\|1010\rangle$	10	10	10	3	1	4
5	$\|1100\rangle$	12	11	00	5	0	5

list_ja(ia) and list_jb(ib) are stored for the lookback table in our computer.

The memory reduction from 2^N (list_2) to $2 \times 2^{N/2}$ (list_ja, list_jb)

Python coding

```
39  def make_list(Nup,Nhilbert,ihfbit,irght,ilft,iup):
40      list_1 = np.zeros(Nhilbert,dtype=int)
41      list_ja = np.zeros(ihfbit,dtype=int)
42      list_jb = np.zeros(ihfbit,dtype=int)
43      ii = iup
44      ja = 0
45      jb = 0
46      ia_old = ii & irght
47      ib_old = (ii & ilft) // ihfbit
48      list_1[0] = ii
49      list_ja[ia_old] = ja
50      list_jb[ib_old] = jb
51      ii = snoob(ii)
52      for n in range(1,Nhilbert):
53          ia = ii & irght
54          ib = (ii & ilft) // ihfbit
55          if (ib == ib_old):
56              ja += 1
57          else:
58              jb += ja+1
59              ja = 0
60          list_1[n] = ii
61          list_ja[ia] = ja
62          list_jb[ib] = jb
63          ia_old = ia
64          ib_old = ib
65          ii = snoob(ii)
66      return list_1, list_ja, list_jb
```

```
68  def get_ja_plus_jb(ii,irght,ilft,ihfbit,list_ja,list_jb):
69      ia = ii & irght
70      ib = (ii & ilft) // ihfbit
71      ja = list_ja[ia]
72      jb = list_jb[ib]
73      return ja+jb
```

Simple_Lanczos_szconserv.py

7. Appendix

Quantum spin solver near saturation (QS^3)

- We can deal with the blocks near saturation (the number of down spins is few) even in larger system sizes for the S=1/2 XXZ model in principle.

- Binary representations/bit operations are essential for a fast ED code, but the expression more than 63 site spins are not easy in our 64-bit processor.

- An alternative way to express a spin basis is to construct a 1-dimensional array storing the site position of each down spin in the basis.

Binary representation	1-dim. array representation
$\uparrow\uparrow\downarrow\uparrow\downarrow\uparrow\uparrow\uparrow\downarrow = (110101110)_2$	$\uparrow\uparrow\downarrow\uparrow\downarrow\uparrow\uparrow\uparrow\downarrow$ = {n(0)=0, n(1)=4, n(2)=6}

Quantum spin solver near saturation (QS^3)

Developers: Hiroshi Ueda (Osaka Univ. Japan) and T. S.

Language: Fortran 90

Target model: S=1/2 XXZ spin model

$$\mathscr{H} = \sum_{i,j} \{J^z_{i,j} S^z_i S^z_j + \frac{J^{xy}_{i,j}}{2}(S^+_i S^-_j + S^-_i S^+_j)\} - h^z \sum_i S^z_i$$

https://github.com/QS-Cube/ED

Methods: Lanczos and Thick-restarted Lanczos methods

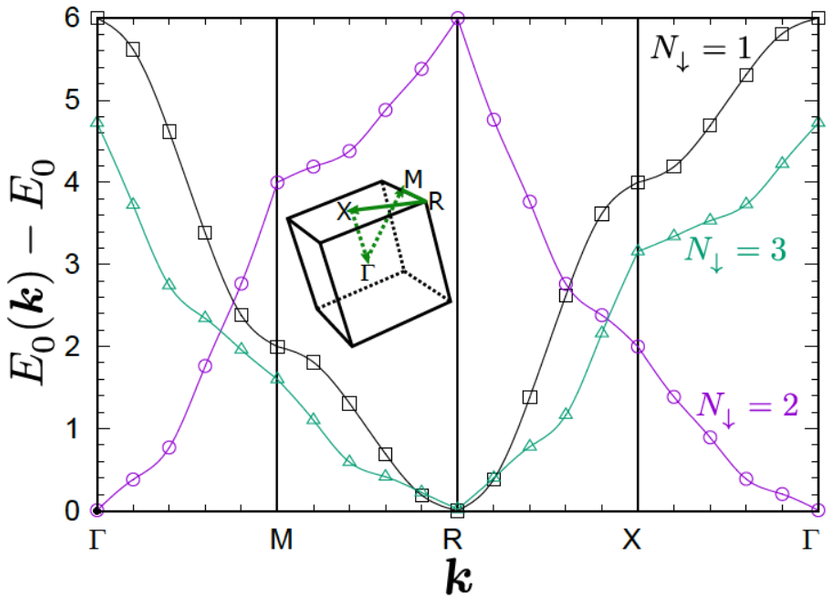
Block diagonalizations: Magnetization, translational symmetry

Observables: Energy, spin-spin correlation, dynamical spin structure factors

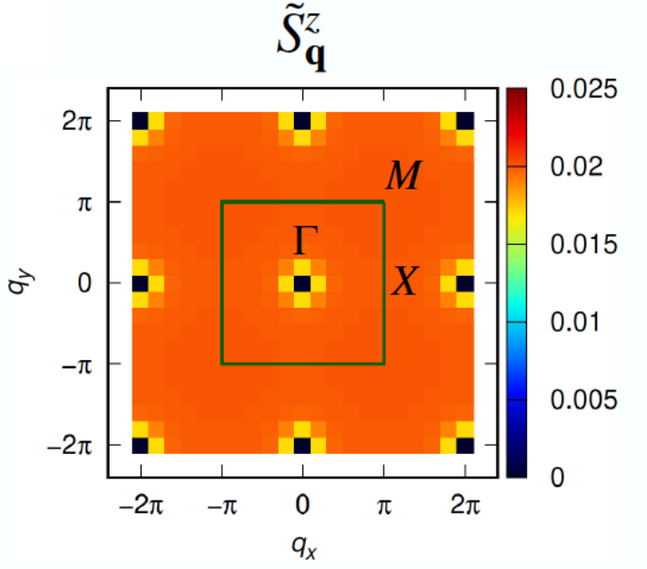
For the details, please see H. Ueda, S. Yunoki and T. S., Comp. Phys. Comm. **277**, 108369 (2022).

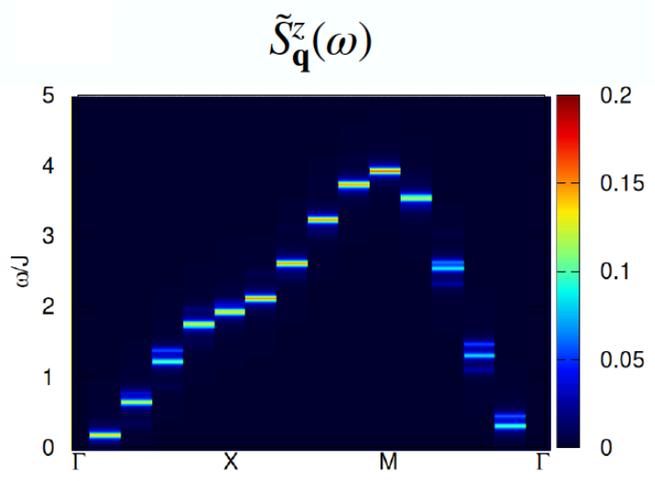
Quantum spin solver near saturation (QS^3)

S=1/2 cubic-lattice Heisenberg model, *N*=1000



S=1/2 square-lattice Heisenberg model, *N*=100



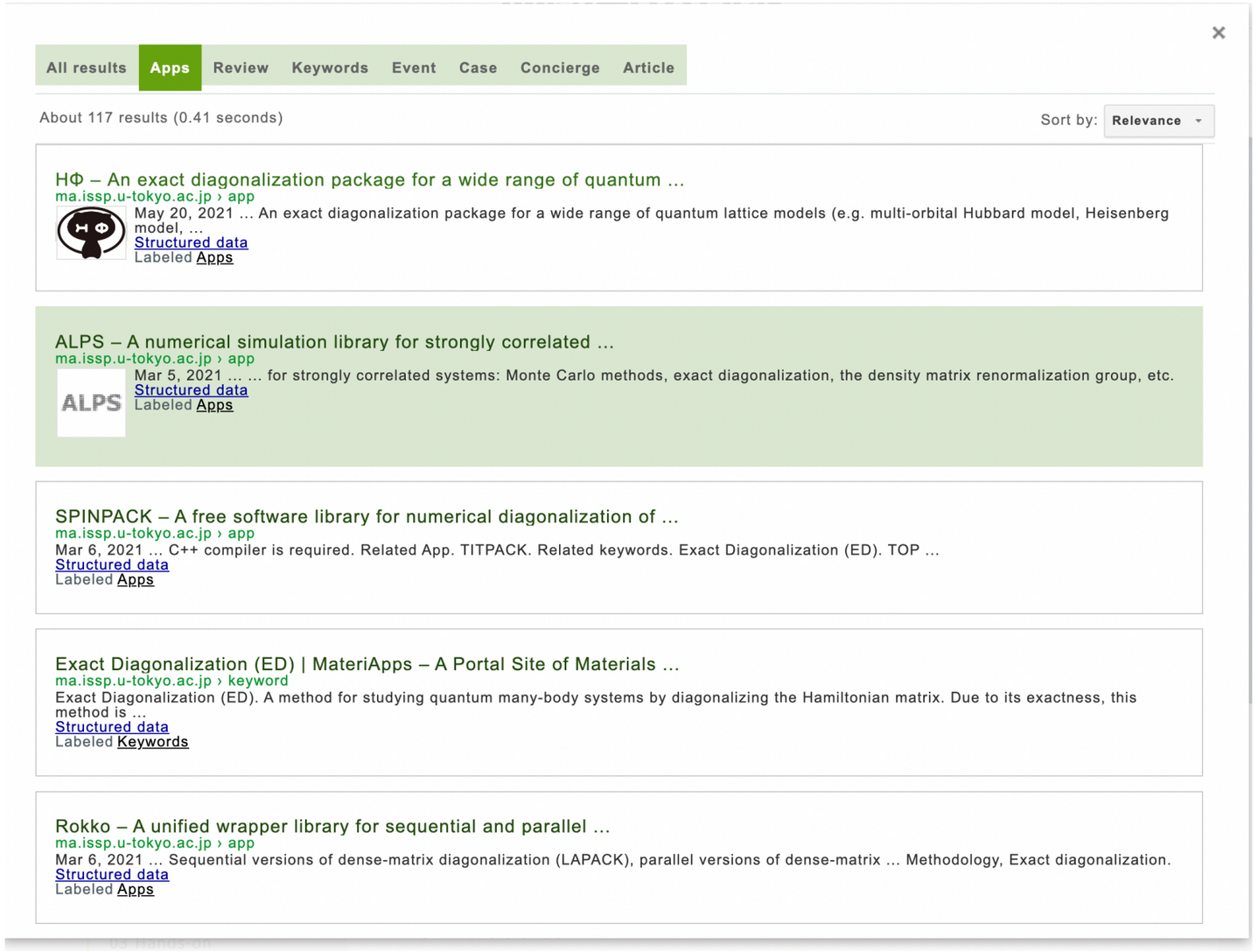


O(100)~O(1000) sites finite-size clusters are accessible near saturated field.

Other ED libraries

MateriApps
A Portal Site of Materials Science Simulation

https://ma.issp.u-tokyo.ac.jp/en



ED literatures

- "**Computational Studies of Quantum Spin Systems**" by Anders W. Sandvik

  arXiv:1101.3281/AIP Conf. Proc. **1297**, 135 (2010).

- "**Ground State and Finite Temperature Lanczos methods**" by P. Prelovšek and J. Bonča

  arXiv:1111.5931/Strongly Correlated Systems. Springer Series in Solid-State Sciences, **176** Springer (2013)

- "**Simulations of pure doped low-dimensional spin-1/2 gapped systems** " by N. Laflorencie and D. Poilblanc

  arXiv:0408363/Lect. Notes Phys. **645**, 227 (2004)

- "**Numerical Simulations of Frustrated Systems** " by AM Läuchli

  Introduction to Frustrated Magnetism. Springer Series in Solid-State Sciences **164** Springer (2011)

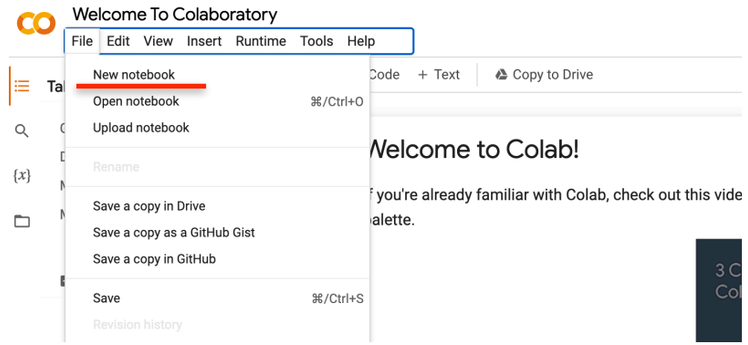
8. For Hands-on Tutorial

How to run tutorial Python codes on Google Colab.

1. Create a Google account https://www.google.com/account/about/

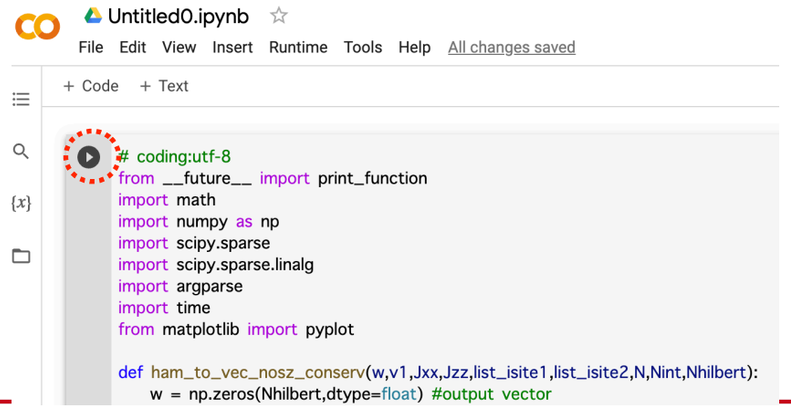
2. Access to https://colab.research.google.com/

3. “File” → “New notebook”→”Sign-in”



4. Get the tutorial Python codes from https://github.com/tshimokaw/PythonEDs, and paste it in a “code cell”.

5. You can run the code with “play icon”.



Tutorial works

1. Play with three Python codes and read them for deep understanding what you learned.

2. Reproduce “Calculation example 1,2,3” graphs by using the Python ED codes.
   (But the largest size data in example 2&3 are time consuming…you don’t have to do…)