DMRG Study of the Kitaev Chain

Sajag Kumar[1,*]

[1] *National Institute of Science Education and Research (NISER) Bhubaneswar.*

In this course project, we study the Kitaev chain using tensor network methods. In particular, we are interested in computing four-point spin correlation functions that are relevant for experiments. In this report, we provide a brief review of the tensor network methods used, an exact solution of the Kitaev chain, a basic introduction to Resonant-Inelastic X-ray scattering (RIXS), and finally, we present our results where we compute the RIXS spectra for the Kitaev chain using four-point and two-point correlation functions.

CONTENTS

[*] sajag.kumar@niser.ac.in

I. INTRODUCTION

Kitaev-type models are rich in physics and applications. They support Majorana modes that can be used for topological quantum computation. They are also examples of exotic phases of matter like quantum spin liquids, and they also have a bulk-edge correspondence, which highlights the applications of topology in condensed matter physics. Experimental verification of such models remains somewhat ambiguous. Few experimental techniques allow us to probe the collective excitations of such systems in a broad energy range. Our motivation in this project is to compute the expected RIXS spectrum of a material described by the Kitaev chain. The actual goal was to study the density matrix renormalization group algorithm. As an application, we chose to study the Kitaev chain. We further computed dynamical correlation functions using time evolution methods beyond DMRG.

In sections II, III, IV, and V, we present a brief and self-contained review of the numerical methods that we use. We begin in section II with an introduction to *matrix product states* (MPS) and their properties, then in section III, we discuss the *density matrix renormalization group* (DMRG) algorithm as a variational algorithm over MPS to find the ground state of one-dimensional Hamiltonians, after this in sections IV and V we discuss *time-evolving block* *decimation* (TEBD) and *time-dependent variational principle* (TDVP) algorithms for time evolution of MPS, respectively. In these sections, we will follow the standard notations and presentations provided in [1], [2], [3], [4], and [5].

In the rest of the report, in section VI we describe the model we study, the Kitaev chain and present an exact solution using Jordan-Wigner transformation. In section 1, we briefly discuss the Resonant Inelastic X-ray scattering experiment and the theory relating the RIXS spectrum to correlation functions. We then present the results of our simulations and their interpretation. In the appendices, we provide all the codes used to do the simulations and discuss the numerical technicalities of our simulation.

II. MATRIX PRODUCT STATES (MPS)

Consider a spin-S model that has N-sites; the local Hilbert space at each site is $S(S+1) \equiv L$ dimensional. Consequently, the resulting Hilbert space for the entire system is L^N dimensional. The time complexity of exact diagonalization of such Hamiltonians is $\mathcal{O}(L^{3N})$ while the space complexity is $\mathcal{O}(L^{2N})$. Consequently, exact diagonalization is simply not possible when N is large. In fact, the best computers can only go as far as $N = 32$ for spin-1/2 systems (where the local Hilbert space is 2-dimensional). The most general state of such a system may be represented as,

$$|\Psi\rangle = \sum_{\sigma_{i_1}\cdots\sigma_{i_N}} c^{\sigma_{i_1}\cdots\sigma_{i_N}} |\sigma_{i_1}\cdots\sigma_{i_N}\rangle\,, \tag{1}$$

here, $|\sigma_{i_1}\cdots\sigma_{i_N}\rangle$ is the direct product of local states at each site and $c^{\sigma_{i_1}\cdots\sigma_{i_N}}$ is an N-rank tensor with L^N scalar components (the coefficient tensor). The approximation schemes for computing the spectrum of many-body Hamiltonians involve reducing to a subspace of the original Hilbert space. Which means we assume a simple form for the coefficient tensor, for example in the mean field approximation we assume that the coefficient tensor factorizes, $c^{\sigma_{i_1}\cdots\sigma_{i_N}} \sim c^{\sigma_{i_1}}\cdots c^{\sigma_{i_N}}$, so the number of unknowns reduce from L^N to LN. This factorization implies that the produced state has no entanglement. When we are interested in states with entanglement, we have to look at MPS here. The idea is to decompose the coefficient tensor in terms of N rank-3 tensors as follows,

$$c^{\sigma_{i_1}\cdots\sigma_{i_N}} \sim \sum_{m_0\cdots m_N} M^{\sigma_{i_1}}_{1,\,m_0 m_1}\cdots M^{\sigma_{i_N}}_{N,\,m_{N-1}m_N}. \tag{2}$$

For a specific set of states, we use a single matrix per site, hence the name *matrix product* states. Its bond dimension is the index m_j for a matrix $M_j^{\sigma_j}$. The usefulness of MPS lies in its ability to represent lowly entangled states with low bond dimensions. With arbitrary high bond dimensions, any state can be written as an MPS.

A. Matrix Product Operators (MPO)

To work with MPS, it is convenient to define analogous operators expressed in terms of matrix products called MPOs. A general operator on the Hilbert space is,

$$\hat{\mathcal{O}} = \sum_{\sigma_{i_1}\cdots\sigma_{i_N},\sigma'_{i_1}\cdots\sigma'_{i_N}} c^{\sigma_{i_1}\cdots\sigma_{i_N},\sigma'_{i_1}\cdots\sigma'_{i_N}} \left|\sigma_{i_1}\cdots\sigma_{i_N}\right\rangle \left\langle\sigma'_{i_1}\cdots\sigma'_{i_N}\right|, \tag{3}$$

where the coefficient tensor can be expressed as a product of N rank-4 tensors as follows,

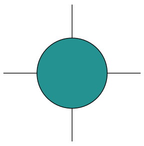
$$c^{\sigma_{i_1}\cdots\sigma_{i_N}} \sim \sum_{m_0\cdots m_N} M_{1,\,m_0 m_1}^{\sigma_{i_1}\sigma'_{i_1}} \cdots M_{N,\,m_{N-1}m_N}^{\sigma_{i_N}\sigma'_{i_N}}. \tag{4}$$

This expression for an operator gives the operator a natural action on an MPS.

B. Tensor Network Notation

Writing all the indices for matrices in MPS and MPO is very cumbersome. To simplify our lives and better understand the equations in the following sections, we will work with Penrose's tensor network representation. This is a pictorial representation of tensor manipulations that is very helpful when working with tensor networks.

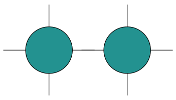
1. A tensor is represented as a figure (in our report, we will use a square, circle, or triangle) with legs (the lines coming out of the circle); the legs represent the tensor indices.

$A^{ijkl} =$  (5)

The above example is of a 4-rank tensor. For a n-rank tensor, n-legs will come out of the figure.

2. Contraction of indices. When two tensors are contracted along an index, the corresponding leg is joined.

$$A^{ijkl}A_{mjop} =$$

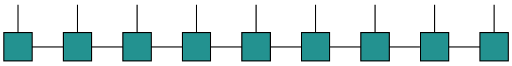


(6)

The j-th index is contracted, which results in a rank-6 tensor. Einstein's summation convention is assumed.

3. MPS

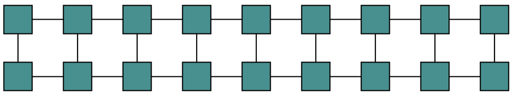
$$|\Psi\rangle =$$



(7)

4. Norm of an MPS

$$\langle\Psi|\Psi\rangle =$$

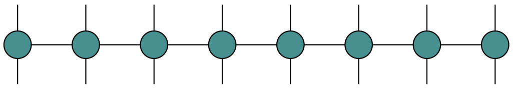


(8)

As this network has no legs, this is a scalar.

5. MPO

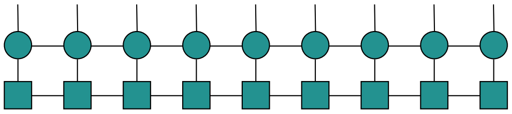
$$\hat{\mathcal{O}} =$$



(9)

6. Action of an MPO on MPS

$$\hat{\mathcal{O}}\,|\Psi\rangle =$$

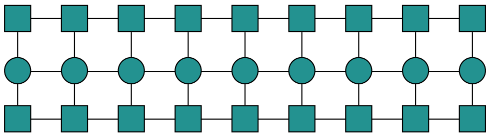


(10)

The action of an operator on a wave function should give a wave function. Here, the action of an MPO on an MPS results in a structure of MPS, so we are consistent.

7. Expectation value of an operator

$$\langle\Psi|\,\hat{\mathcal{O}}\,|\Psi\rangle =$$



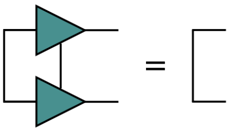
(11)

Once again, there are no free legs, so the above network represents a scalar as it should.

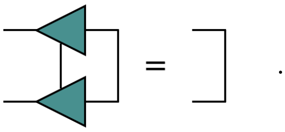
C. MPS Manipulations

MPS are used for numerical simulations because there are efficient numerical techniques for their manipulation, i.e., they can be added, acted upon by operators, or cast into some useful form while maintaining a low enough bond dimension. Some of the important techniques and manipulations are listed below:

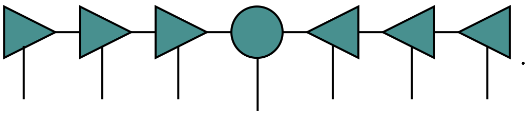
1. *Gauge freedom.* We could insert AA^{-1} between any two matrices in the MPS/MPO form of the coefficient tensor this would change the numerical content of the matrices but leave the tensor invariant, this is a *gauge freedom.* To fix this gauge, we choose the right or left normalized matrices. Right normalization means,

    (12)

   and left normalization is,

    (13)

   An MPS in the canonical form only contains right or left-normalized matrices. However, at the boundary where right and left normalized matrices meet, the matrix cannot be normalized and is called the active site or orthogonality center.

    (14)

2. *Normalization and truncation.* To normalize an MPS, we use QR decomposition and multiply the so-obtained R by the subsequent matrix. Operations on MPS and MPO change their bond dimension, we need an appropriate strategy to keep the bond dimensions in check, we do this by Support Vector Decomposition (SVD) of the factor matrices of the coefficient tensor. This is called truncating the MPS or MPO.

3. *Any wave function can be written as an MPS.* One important property of MPS is that any wave function can be written as an MPS if we allow the bond dimensions to grow exponentially towards the center of the chain.

4. *Area law of entanglement entropy.* MPS are useful ansatz for one-dimensional quantum systems because their ground state (at least in the case of gapped Hamiltonians) follows an area law for entanglement entropy.

III. DENSITY MATRIX RENORMALIZATION GROUP (DMRG)

The DMRG algorithm variationally minimizes an MPS ansatz to reach the ground state of one-dimensional Hamiltonians (typically) with low entanglement. The name of the algorithm is due to historical reasons that we discuss in the following section. The modern viewpoint of a variational algorithm over the MPS class is described in the subsequent section.

A. History

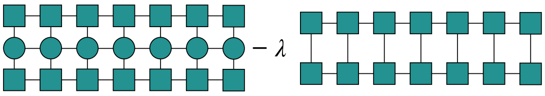
Wilson developed the numerical renormalization group (NRG) in the 1970s [6]. Consider a spin system in one dimension. The idea of NRG is to block a set of spins together, find the eigenvalues of the Hamiltonian reduced to these sites, store the lowest m eigenvalues and their corresponding eigenstates, and repeat the procedure by treating a set of blocks as new sites in the next step. NRG only worked for the Kondo problem. The principle of failure of NRG was elucidated by Steve White in 1992, and the correct way of truncating the Hilbert space is not on an energy basis but on the basis of the density matrix. Steve White developed what is now known as the *density matrix* (because we truncated in the basis of density matrices) *renormalization group* (because we use Wilson's RG idea of using blocks of spins as single sites and iterating over them) [7] [8]. White's DMRG involved choosing a block of spins, finding the reduced density matrix for this subsystem, and truncating the Hilbert space on this basis.

B. *Modern* DMRG

In this section, will describe DMRG in the modern language of MPS and tensor networks. The aim of the algorithm is to find an MPS $|\Psi\rangle$ such that,

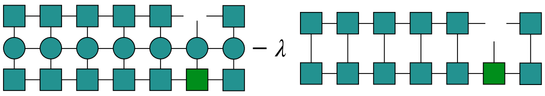
$$E = \frac{\langle\Psi|\,\mathcal{H}\,|\Psi\rangle}{\langle\Psi|\Psi\rangle}, \tag{15}$$

is minimum. To ensure normalization, we introduce a Lagrange multiplier λ and optimize the following tensor network,



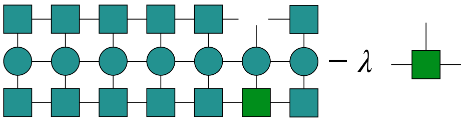
$$= 0. \tag{16}$$

In the DMRG algorithm, the next step is to minimize this network with respect to single tensors at a time. To minimize the network we need to take a derivative with respect to one of the matrices in the MPS, since the MPS in linear, it simply results in removal of that matrix.



$$= 0. \tag{17}$$

The green matrix is the one with respect to which we carry out the minimization in this step. In case the MPS is in mixed canonical form with the active site as the matrix with which we are minimizing, we can contract the second term of the above expression to obtain,



$$= 0. \tag{18}$$

The above equation is an eigenvalue problem, which is solved in the DMRG algorithm using Lanczos methods. The complete DMRG algorithm involves sweeping over the MPS; we start from an MPS in right canonical form and reach a left canonical MPS. The active site is moved from one end to the other.

IV. TIME EVOLVING BLOCK DECIMATION (TEBD)

If the Hamiltonian of the system only involves nearest-neighbor interactions, we can do the time evolution of MPS with this Hamiltonian using TEBD. To time evolve an MPS for time t we need,

$$\hat{U} = e^{-i\mathcal{H}t}. \tag{19}$$

This is difficult because diagonalizing the Hamiltonian is difficult due to its large dimension. So we need to reduce the dimension of the Hamiltonian, i.e., reduce to an appropriate sector of the Hilbert space. In TEBD, we decompose the system's Hamiltonian as follows,

$$\mathcal{H} = \mathcal{H}_{\text{even}} + \mathcal{H}_{\text{odd}}. \tag{20}$$

The TEBD approximation of the time evolution operator is,

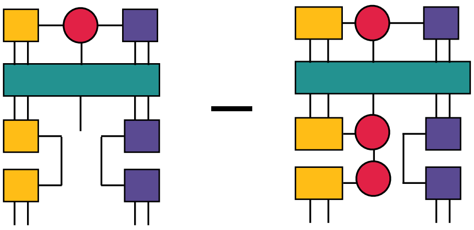
$$\hat{U}_{\text{TEBD}}(t) = e^{-i\mathcal{H}_{\text{even}}t} e^{-i\mathcal{H}_{\text{odd}}t}. \tag{21}$$

V. TIME-DEPENDENT VARIATIONAL PRINCIPLE (TDVP)

Consider the evolution of MPS with the Schrodinger equation,

$$i\hbar\partial_t \left|\Psi\right\rangle = \mathcal{H}\left|\Psi\right\rangle, \tag{22}$$

the solution to this equation may not be an MPS. To maintain the MPS form and low bond dimension, in TDVP, we project the ansatz MPS to its tangent manifold of the same bond dimensions and then solve the time-dependent Schrodinger equation only on this manifold. In this algorithm, we do not calculate the time evolution operator but rather the action of the operator on the MPS. We effectively guess an MPS from the tangent manifold described above and variationally minimize it to obtain the time-evolved state. The expression to be minimized is,



$$= 0 \tag{23}$$

the green block is the Hamiltonian, which acts upon the MPS in canonical form. The solution to this variational problem is an MPS in the tangent space of the initial MPS ansatz. The advantage of this method over TEBD is simply the fact that it does not assume any structure of the Hilbert space. However, solving the variational problem above is not easy and sometimes is not possible.

VI. THE KITAEV CHAIN

In this section, we follow the treatment in [9], [10] and [11]. The Hamiltonian of the system is,

$$\mathcal{H} = J_x \sum_{n=0}^{L/2} \sigma^x_{2n}\sigma^x_{2n+1} + J_y \sum_{n=0}^{(L-1)/2-1} \sigma^y_{2n+1}\sigma^y_{2n+2} + g\sum_{n}^{L}\sigma^z \tag{24}$$

where J_x and J_y are the x-type and y-type interaction strengths respectively. This model can be exactly diagonalized using the Jordan-Wigner transformation given by,

$$\sigma_i^z = 2n_i - 1 \tag{25}$$

$$\sigma_i^+ = e^{i\pi \sum_j^{i-1} n_j} c_i^\dagger. \tag{26}$$

Going to the momentum space and defining a set of four operators, as follows,

$$F_\pm = \frac{c_{q-\pi} \pm c_{-q}^\dagger}{\sqrt{2}}, \tag{27}$$

$$G_\pm = \frac{c_q \pm c_{\pi-p}^\dagger}{\sqrt{2}}, \tag{28}$$

and writing the Hamiltonian in this basis we obtain,

$$H = \begin{bmatrix} -2\varepsilon_{1q} & -2i\varepsilon_{2q} & g & 0 \\ 2i\varepsilon_{2q} & 2\varepsilon_{1q} & 0 & g \\ g & 0 & 0 & 0 \\ 0 & g & 0 & 0 \end{bmatrix} \tag{29}$$

where $\varepsilon_k = J_x e^{-ik} + J_y e^{ik}$. The eigenvalues are,

$$\lambda_{n_1 n_2} = n_1 |\varepsilon_q| + n_2 \sqrt{|\varepsilon_q|^2 + g^2}. \tag{30}$$

In terms of the diagonal modes $\eta_{n_1 n_2}$ the Hamiltonian is,

$$\mathcal{H} = \sum_{n_1=\pm, n_2=\pm} \lambda_{n_1 n_2} \eta_{n_1 n_2}^\dagger \eta_{n_1 n_2}. \tag{31}$$

The GS energy is given by,

$$E = -4 \sum_{0<q<\pi/2} \sqrt{|\varepsilon_q|^2 + g^2}. \tag{32}$$

The GS is constructed by the action of negative-energy modes,

$$|GS\rangle = \Pi_{0<q<\pi/2} \eta_{--}^\dagger \eta_{+-}^\dagger \left|\Omega\right\rangle. \tag{33}$$

It is non-degenerate in the presence of a magnetic field. In the absence of a magnetic field, the GS is degenerate, for each mode is four-fold degenerate, so the total degeneracy is $4^{L/4}$. However, we either work in even or odd sectors (in our case,e even), so the effective GS degeneracy is given by

$$D_{GS} = 2^{L/2-1}. \tag{34}$$

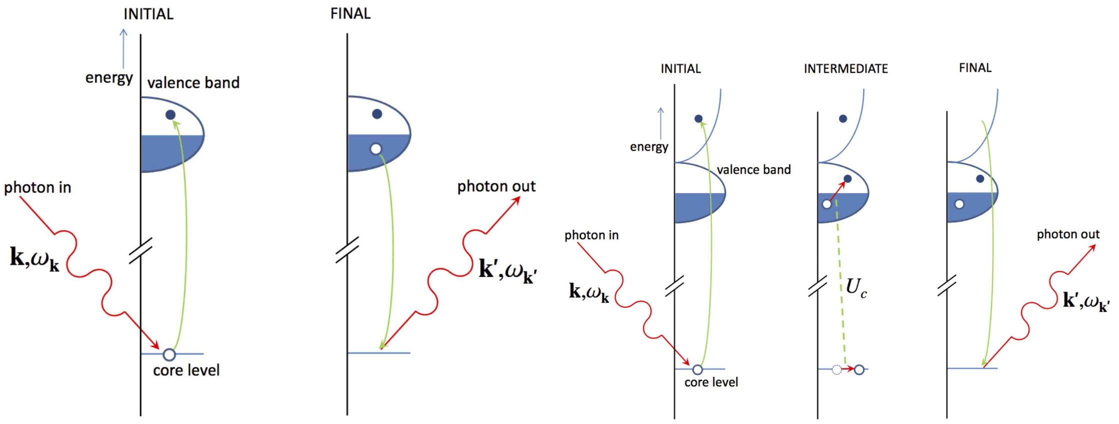


FIG. 1: Schematic of the direct and indirect RIXS process.

VII. RESONANT-INELASTIC X-RAY SCATTERING (RIXS)

RIXS is a spectroscopic method where photons are shined on materials, and the spectrum of emitted photons from this process is recorded. Two processes contribute to the RIXS spectrum. In the direct process, the core electron is excited to the valence shell, and one of the electrons in the filled shell de-excites to the core hole, resulting in an X-ray emission. In the indirect process, the incoming photon excites the core electron far beyond the chemical potential of the material. The Coulomb interaction between the core hole and valence electrons produces the excitations. The indirect RIXS spectrum is not very prominent. By looking at the cross-section of the scattered photons, we can gather a wealth of information about the excitations in the system. RIXS is especially very good at capturing the low energy excitations of the system. Our motivation is to capture this spectrum theoretically. We do this by following the formalism in, [12], [4] and [13]. We do not perform all the calculations in great detail but mention the key ideas at each step.

1. The photon cross-section is obtained in the Kramer-Heisenberg formalism using second-order Fermi's golden rule.

2. The cross-section is obtained as a series in Kramer-Heisenberg amplitudes. The amplitudes are obtained by ultra-short core-hole lifetime (UCL) expansion. The amplitude is expanded in the powers of $1/\Gamma$, and Γ is the lifetime of the core-hole excitation.

$$A_{fg} = \omega_{\text{res}} \frac{1}{i\Gamma} \sum_{l=0}^{\infty} \langle f | D \frac{(H_0 + H')^l}{(i\Gamma)^l} D | g \rangle. \tag{35}$$

D is the dipole operator given by,

$$D = \frac{1}{im\omega_k} \sum_{i=1}^{N} e^{i\mathbf{k}\cdot\mathbf{r}_i}\, \varepsilon \cdot \mathbf{p}_i. \tag{36}$$

H_0 is the Kitaev model's Hamiltonian and,

$$H' = \sum_{i \in odd} h_i h_i^\dagger J_1 S_i^x S_{i+1}^x + \sum_{i \in odd} h_i h_i^\dagger J_1 S_i^y S_{i+1}^y. \tag{37}$$

3. We get an expression for the cross-section upon substituting these approximations in the Kramer-Heisenberg formula.

4. The first and second order terms in the cross-section are,

$$\chi_1(q,\omega) = \left| \sum_{i,j} \langle g| S_i^\alpha S_j^\beta |g\rangle\, e^{-iq(r_i - r_j)} e^{-iwt} \right|^2, \tag{38}$$

$$\chi_2(q,\omega) = \left| \frac{1}{N} \sum_{i,j,t} \langle g| S_i^\alpha(t) S_{i+1}^\beta(t) S_j^{\alpha\dagger}(0) S_{j+1}^{\beta\dagger}(0) |g\rangle\, e^{-iq(r_i - r_j)} e^{-iwt} \right|^2. \tag{39}$$

As we can see, two and four-point dynamical correlation functions are key quantities that we need, and we calculate them using DMRG and TEBD/TDVP in the next section.

VIII. RESULTS

Calculations were performed using the TeNPy Library (version 1.0.4) [14]. We compute the four-point and two-point correlation functions for the Kitaev chain. Here is our strategy for the simulations.

1. We first find the ground state MPS for the Kitaev chain using the DMRG algorithm. Let's say we obtain the state $|\Psi\rangle$.

2. We then act $S_i^\alpha S_{i+a}^\beta$ on $|\Psi\rangle$ and time evolve this state using TEBD and TDVP algorithm.

3. We finally obtain the four-point correlation function by computing the overlap of the time-evolved MPS with the initial MPS,

$$\chi(i,j,t) = \langle \Psi| S_i^\alpha(t) S_{i+a}^\beta(t) S_j^{\alpha\dagger}(0) S_{j+a}^{\beta\dagger}(0) |\Psi\rangle\,. \tag{40}$$

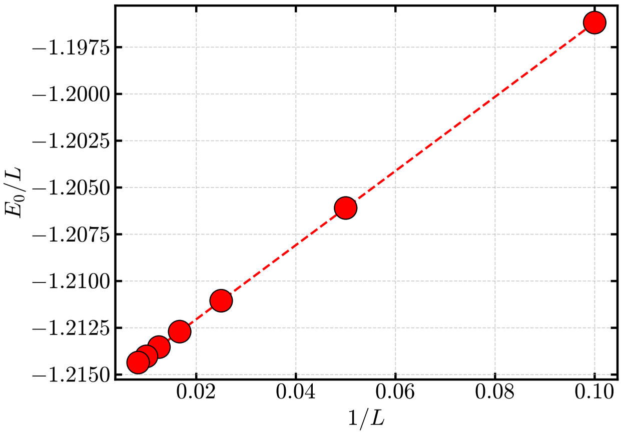


FIG. 2: The energy scaling per unit site obtained from the DMRG algorithm. The linear behavior is expected, and hence we have a sanity check for our DMRG setup.

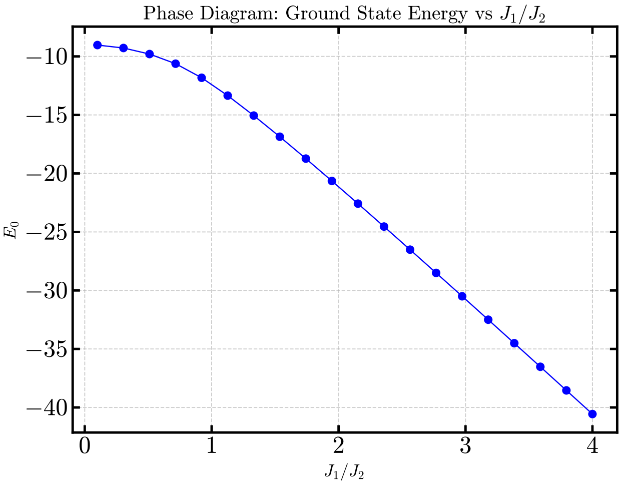


FIG. 3: We plot the ground state energy of the system as a function of the interaction strength. We find the expected non-linear scaling from the exact solution.

4. We then do a space and time Fourier transform to obtain,

$$\chi(k,\omega) = \left| \sum_{i,j} \sum_{t} \langle\Psi| S_i^{\alpha}(t) S_{i+a}^{\beta}(t) S_j^{\alpha\dagger}(0) S_{j+a}^{\beta\dagger}(0) |\Psi\rangle e^{i\omega t} e^{iq(r_i - r_j)} \right|^2, \tag{41}$$

this is what we call RIXS intensity and we plot this for several cases.

For the results that we present in the case of $L = 62$ and 128. We do not perform the sum over all possible combinations of lattice sites. We choose a unit cell and sum over all possible combinations of the two sites of the unit cell with the rest of the sites.

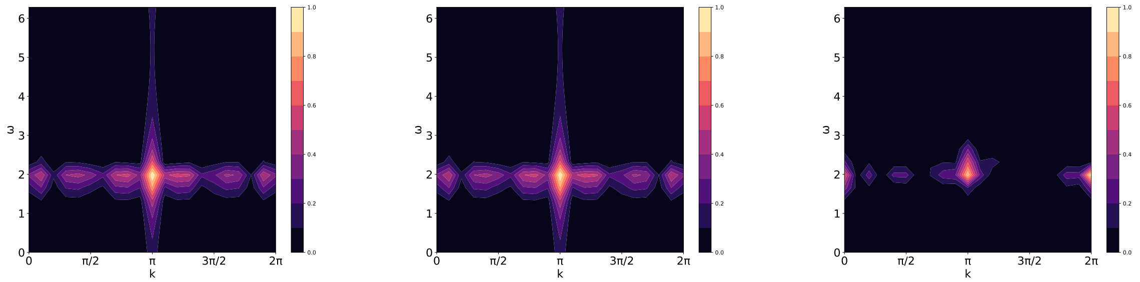


FIG. 4: The RIXS intensity for no magnetic field. These are the results for the four-point correlation functions. (Left) Evolution using TEBD (Center) Evolution using TDVP (Right) Evolution using TEBD, but the correlation function is $S^+S^+S^-S^-$ instead of $S^+S^-S^+S^-$ in the previous two cases.

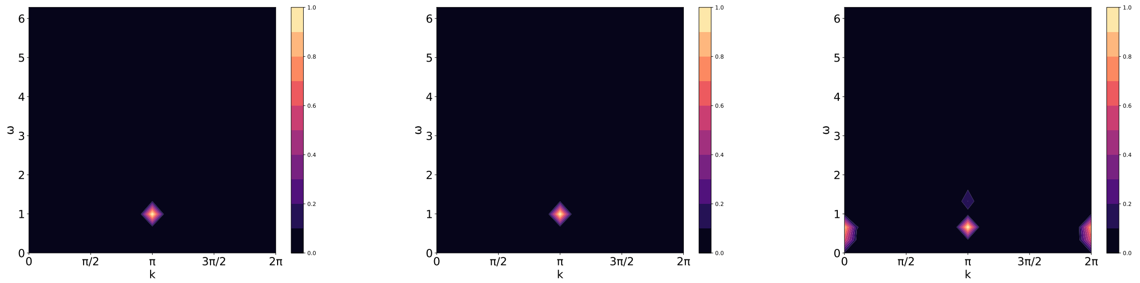


FIG. 5: The RIXS intensity with magnetic field ($g/J_1 = 1$). These are the results for the four-point correlation functions. (Left) Evolution using TEBD (Center) Evolution using TDVP (Right) Evolution using TEBD, but the correlation function is $S^+S^+S^-S^-$ instead of $S^+S^-S^+S^-$ in the previous two cases.

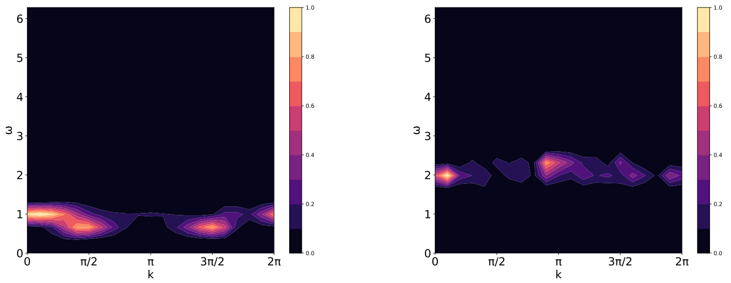


FIG. 6: The RIXS intensity with (Left) magnetic field ($g/J_1 = 1$) and without (Right) magnetic field for two-point correlation functions S^+S^-.

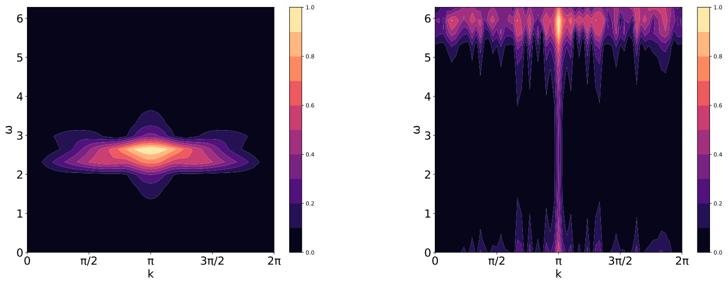


FIG. 7: The RIXS intensity with (Left) magnetic field ($g/J_1 = 1$) and without (Right) magnetic field for four-point correlation functions $S^+S^-S^+S^-$. These are results for a chain of length $L = 62$, which are evolved for 20 seconds.

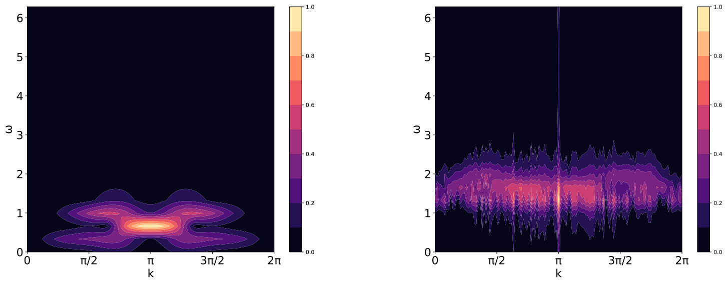


FIG. 8: The RIXS intensity with (Left) magnetic field ($g/J_1 = 1$) and without (Right) magnetic field for four-point correlation functions $S^+S^-S^+S^-$. These are results for a chain of length $L = 128$, which are evolved for 20 seconds.

IX. CONCLUSIONS

We have successfully setup the simulation to compute the RIXS spectrum of the Kitaev chain. Our results are quite preliminary but promising. In particular a few interesting inferences can be made:

1. The two point correlation functions do not have any structure.

2. The second method of choosing just one unit cell and calculating its correlation with other cells do not quite work for the chain length that we have set. So our results do have significant boundary effects.

3. The spectrum shows the best features when the correlation function for $S^+S^-S^+S^-$ is computed other combinations result in almost featureless spectrum.

4. The spectrum in the presence of magnetic field and its absence is significantly different.

5. TEBD and TDVP agree on their results.

Some of the shortcomings of our work are addressed next with reasons for the same:

1. We understood what we actually needed to compute quite late, leaving us with little time to run experiments.

2. The simulations are painstakingly slow since the sum has to be done over the lattice.

3. Going to larger lattice sizes will alleviate the boundary effects but they also require going to larger times which significantly increases the time complexity, so obtaining features of the spectrum takes a lot of time.

However, we have found some empirical evidence for the original goal of finding some features in the four-point spectrum of the Kitaev chain.

Appendix A: Periodic Boundary Conditions (PBCs) and MPS

We use open boundary conditions because MPS does not work well for PBCs. In particular we cannot apply a local operator on an infinite MPS (iMPS). This is necessary in our case to probe the excitations. We cannot do this because the action of a local operator breaks translational invariance and hence the resulting MPS cannot be an iMPS. To ensure that the MPS upon the action of a local operator is still an iMPS, we have to ensure segment BCs during the action of the operator, we did try to implement this but we could not achieve reasonable time complexity, so we abandoned the idea.

Appendix B: Code

In this section we provide the code used to do the simulations, all the parameters of the simulations are explicitly mentioned. The required libraries are:

```python
import numpy as np
import matplotlib.pyplot as plt
from tqdm import tqdm
from pathlib import Path
import os
from matplotlib import colors
from tenpy.models.model import CouplingMPOModel
from tenpy.models.model import CouplingModel
from tenpy.models.model import NearestNeighborModel
from tenpy.models.model import MPOModel
from tenpy.models.lattice import Chain
from tenpy.networks.site import SpinHalfSite
from tenpy.networks.mps import MPS
from tenpy.algorithms import tebd
from tenpy.algorithms import tdvp
from tenpy.algorithms import dmrg
```

1. Ground State Using DMRG

```
    class KitaevChain(CouplingMPOModel, NearestNeighborModel):

        def init_sites(self, model_params):
            conserve = model_params.get('conserve', 'parity')
            assert conserve != 'Sz'
            if conserve == 'best':
                conserve = 'parity'
                self.logger.info("%s: set conserve to %s", self.name, conserve)
            sort_charge = model_params.get('sort_charge', True)
            site = SpinHalfSite(conserve=conserve, sort_charge=sort_charge)
            return site

        def init_terms(self, model_params):
            length = model_params.get('L', 12)
            strx = []
            stry = []
            for i in range(length-1):
                if i%2 == 0:
                    strx.append(1)
                    stry.append(0)
                else:
                    strx.append(0)
                    stry.append(1)
            J_1 = np.asarray(model_params.get('J_1', 1))
            J_2 = np.asarray(model_params.get('J_2', 1))
            g = np.asarray(model_params.get('g', 1))
            for u1, u2, dx in self.lat.pairs['nearest_neighbors']:
                self.add_coupling(strx, u1, 'Sigmax', u2, 'Sigmax', dx)
            for u1, u2, dx in self.lat.pairs['nearest_neighbors']:
                self.add_coupling(stry, u1, 'Sigmay', u2, 'Sigmay', dx)
            for u in range(len(self.lat.unit_cell)):
                self.add_onsite(-g, u, 'Sigmaz')

model_params = {'J_1': 1,
        'J_2': 1,
        'g': 1,
        'L': 22,
        'bc_MPS': 'finite',
        'sort_charge': True}

H = KitaevChain(model_params)
psi = MPS.from_lat_product_state(H.lat, [['up']])

dmrg_params = {
    'trunc_params': {
        'chi_max': 100,
        'svd_min': 1.e-10,
        'trunc_cut': None
    },
    'max_E_err': 0.01,
    'max_S_err': 0.01,
    'max_sweeps': 10,
    'mixer': False
}

eng = dmrg.TwoSiteDMRGEngine(psi, H, dmrg_params)

E0, psi = eng.run()
```

2. Two-Point Correlation Functions Using TEBD

```
steps = 20

tebd_params = {
    'N_steps': 1,
    'dt': 0.1,
    'preserve_norm': True,
    'trunc_params': {'chi_max': 100, 'svd_min': 1.e-12}
}

corrs = np.zeros((model_params['L'], model_params['L'], steps))
corrsi = np.zeros((model_params['L'], model_params['L'], steps))

psi_steps_1 = []
psi_steps_2 = []

for i in range(model_params['L']-1):
    phi_1 = psi.copy()
    phi_1.apply_local_op(i, 'Sp', unitary=True)
    psi_steps_1.append(phi_1)

for i in range(model_params['L']-1):
    phi_2 = psi.copy()
    phi_2.apply_local_op(i, 'Sm', unitary=True)
    psi_steps_2.append(phi_2)

for step in tqdm(range(steps)):
    for i, state in enumerate(psi_steps_2):
        eng = tebd.TEBDEngine(state, H, tebd_params)
        eng.run()

    for i in range(model_params['L']-1):
        for j in range(model_params['L']-1):
            corrs[i, j, step] = psi_steps_2[i].overlap(psi_steps_1[j]).real
            corrsi[i, j, step] = psi_steps_2[i].overlap(psi_steps_1[j]).imag

```

3. Two-Point Correlation Functions Using TDVP

```
steps = 20

tdvp_params = {
    'N_steps': 1,
    'dt': 0.1,
    'preserve_norm': True,
    'trunc_params': {'chi_max': 100, 'svd_min': 1.e-12}
}

corrs = np.zeros((model_params['L'], model_params['L'], steps))
```

```
corrsi = np.zeros((model_params['L'], model_params['L'], steps))

psi_steps_1 = []
psi_steps_2 = []

for i in range(model_params['L']-1):
    phi_1 = psi.copy()
    phi_1.apply_local_op(i, 'Sp', unitary=True)
    psi_steps_1.append(phi_1)

for i in range(model_params['L']-1):
    phi_2 = psi.copy()
    phi_2.apply_local_op(i, 'Sm', unitary=True)
    psi_steps_2.append(phi_2)

for step in tqdm(range(steps)):
    for i, state in enumerate(psi_steps_2):
        eng = tdvp.TwoSiteTDVPEngine(state, H, tebd_params)
        eng.run()

    for i in range(model_params['L']-1):
        for j in range(model_params['L']-1):
            corrs[i, j, step] = psi_steps_2[i].overlap(psi_steps_1[j]).real
            corrsi[i, j, step] = psi_steps_2[i].overlap(psi_steps_1[j]).imag

```

4. Four-Point Correlation Functions Using TEBD

```
steps = 20

tebd_params = {
    'N_steps': 1,
    'dt': 0.1,
    'order': 2,
    'trunc_params': {'chi_max': 100, 'svd_min': 1.e-12}
}

corrs = np.zeros((model_params['L'], model_params['L'], steps))
corrsi = np.zeros((model_params['L'], model_params['L'], steps))

psi_steps_1 = []
psi_steps_2 = []

for i in range(model_params['L']-1):
    phi_1 = psi.copy()
    phi_1.apply_local_op(i, 'Sp', unitary=True)
    phi_1.apply_local_op(i+1, 'Sm', unitary=True)
    psi_steps_1.append(phi_1)

for i in range(model_params['L']-1):
    phi_2 = psi.copy()
    phi_2.apply_local_op(i, 'Sp', unitary=True)
    phi_2.apply_local_op(i+1, 'Sm', unitary=True)
    psi_steps_2.append(phi_2)

for step in tqdm(range(steps)):
    for i, state in enumerate(psi_steps_2):
        eng = tebd.TEBDEngine(state, H, tebd_params)
        eng.run()

    for i in range(model_params['L']-1):
        for j in range(model_params['L']-1):
            corrs[i, j, step] = psi_steps_2[i].overlap(psi_steps_1[j]).real
            corrsi[i, j, step] = psi_steps_2[i].overlap(psi_steps_1[j]).imag

```

5. Four-Point Correlation Functions Using TDVP

```
steps = 20

tdvp_params = {
    'N_steps': 1,
    'dt': 0.1,
    'preserve_norm': True,
    'trunc_params': {'chi_max': 100, 'svd_min': 1.e-12}
}

corrs = np.zeros((model_params['L'], model_params['L'], steps))
corrsi = np.zeros((model_params['L'], model_params['L'], steps))

psi_steps_1 = []
psi_steps_2 = []

for i in range(model_params['L']-1):
    phi_1 = psi.copy()
    phi_1.apply_local_op(i, 'Sp', unitary=True)
    phi_1.apply_local_op(i+1, 'Sm', unitary=True)
    psi_steps_1.append(phi_1)

for i in range(model_params['L']-1):
    phi_2 = psi.copy()
    phi_2.apply_local_op(i, 'Sp', unitary=True)
    phi_2.apply_local_op(i+1, 'Sm', unitary=True)
    psi_steps_2.append(phi_2)
```

```

for step in tqdm(range(steps)):
    for i, state in enumerate(psi_steps_2):
        eng = tdvp.TwoSiteTDVPEngine(state, H, tebd_params)
        eng.run()

    for i in range(model_params['L']-1):
        for j in range(model_params['L']-1):
            corrs[i, j, step] = psi_steps_2[i].overlap(psi_steps_1[j]).real
            corrsi[i, j, step] = psi_steps_2[i].overlap(psi_steps_1[j]).imag
```

Appendix C: Fourier Transform of the Correlation Functions

We save the data for the correlation functions using the following program.

```
L = model_params.get('L')
g = model_params.get('g')
folder = f'tdvp_correlators_L_{L}_g_{g}_steps_{steps}_ppmm2'
if not os.path.exists(folder):
    os.makedirs(folder)

for i in range(model_params['L']-1):
    real_part = corrs[i, :, :]
    imag_part = corrsi[i, :, :]

    complex_matrix = real_part + 1j * imag_part

    filename = os.path.join(folder, f'{i}.txt')
    np.savetxt(filename, complex_matrix, fmt='%.6f', delimiter=' ')

    print(f'Saved complex matrix for i={i} to {filename}')
```

Then we read this data and do a Fourier transform as follows.

```
steps = 20
L = 22
g = 1
folder = f'tdvp_correlators_L_{L}_g_{g}_steps_{steps}_ppmm2'
folder_path = folder
txt_files = [
    f for f in os.listdir(folder_path)
    if f.endswith(".txt")
    and os.path.isfile(os.path.join(folder_path, f))
]

num_files = len(txt_files)

fft_sum = None
for i in range(0, num_files):
    file_path = os.path.join(folder_path, f"{i}.txt")

    with open(file_path, "r") as f:
        lines = f.readlines()[1:]
    complex_matrix = []
    for line in lines:
        row = []
        elements = line.split()
        for element in elements:
            row.append(complex(element.replace("j", "j")))
        complex_matrix.append(row)

    matrix = np.array(complex_matrix)
    fft_matrix = np.fft.fft2(matrix)

```

```python
    if fft_sum is None:
        fft_sum = fft_matrix
    else:
        fft_sum += fft_matrix

p,t = matrix.shape
time = np.linspace(0,t-1,t)
pos = np.linspace(0,p-1,p)
dt = (time[-1] - time[0])/len(time)
dx = (pos[-1] - pos[0])/len(pos)
freq = 1/dt*np.arange(0,len(time))/len(time)*2*np.pi
k = 1/dx*np.arange(0,len(pos))*2*np.pi/len(pos)
fft_sum = np.fft.fftshift(fft_sum)

I = np.imag(fft_sum)
R = np.real(fft_sum)
psd = abs(fft_sum/len(pos))**2
norm = colors.Normalize(vmin=0, vmax=1)
y_ticks = [0, np.pi / 2, np.pi, 3 * np.pi / 2, 2 * np.pi]
y_labels = ['0', '\u03C0/2', '\u03C0', '3\u03C0/2', '2\u03C0']
plt.figure(figsize=(10, 8))
plt.contourf(k,freq,np.log(psd.T+1)/np.max(np.log(psd.T+1)),
levels=np.linspace(0, 1, 11), cmap='plasma', norm=norm)
plt.colorbar(label='')
plt.title('')
plt.xlabel('k', fontsize=20)
plt.ylabel('\u03C9', fontsize=20)
plt.xticks(ticks=y_ticks, labels=y_labels, fontsize=20)
plt.yticks(fontsize=20)
plt.savefig(f'figures/{folder}.pdf')
plt.show()
```

[1] S. Paeckel, T. Köhler, A. Swoboda, S. R. Manmana, U. Schollwöck, and C. Hubig, Annals of Physics **411**, 167998 (2019).

[2] U. Schollwöck, Annals of Physics **326**, 96 (2011).

[3] U. Schollwöck, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences **369**, 2643 (2011).

[4] A. Nocera, U. Kumar, N. Kaushal, G. Alvarez, E. Dagotto, and S. Johnston, Scientific Reports **8**, 11080 (2018).

[5] G. Benenti, G. Casati, D. Rossini, and G. Strini, *Principles of Quantum Computation and Information* (WORLD SCIENTIFIC, 2018).

[6] K. G. Wilson, Rev. Mod. Phys. **47**, 773 (1975).

[7] S. R. White, Phys. Rev. B **48**, 10345 (1993).

[8] S. R. White, Phys. Rev. Lett. **69**, 2863 (1992).

[9] V. K. Vimal, H. Wanare, and V. Subrahmanyam, Phys. Rev. A **106**, 032221 (2022).

[10] V. Subrahmanyam, Phys. Rev. A **88**, 032315 (2013).

[11] V. K. Vimal and V. Subrahmanyam, Phys. Rev. A **98**, 052303 (2018).

[12] Prabhakar, S. Pal, U. Kumar, M. Kumar, and A. Mukherjee, Predicting fractionalized multi-spin excitations in resonant inelastic x-ray spectra of frustrated spin-1/2 trimer chains (2024), arXiv:2410.15082 [cond-mat.str-el].

[13] S. Pal, U. Kumar, Prabhakar, and A. Mukherjee, Phys. Rev. B **108**, 214405 (2023).

[14] J. Hauschild and F. Pollmann, SciPost Phys. Lect. Notes , 5 (2018), code available from `https://github.com/tenpy/tenpy`, arXiv:1805.00055.