Assignment 2

Entanglement entropy and state compression

Abhiram Cherukupalli[1]
[1] *California Institute of Technology*
(Dated: May 4, 2024)

CONTENTS

All my code and finer details are available at https://github.com/Abhiram2006/ph121c-hwk2.

I. IMAGE COMPRESSION

A. Implementation

```python
def load_image(num):
    # loads image, converts to grayscale, then converts to a matrix
    image = Image.open('images/'+f'image{num}.png')
    gray_image = image.convert('L')
    matrix = np.array(gray_image)
    return matrix

def compute_svd(image_matrix):
    # computes the singular value decomposition of a matrix
    U, s, Vt = np.linalg.svd(image_matrix, full_matrices=False)
    return U, s, Vt

def compress_image(U, s, Vt, k):
    # compresses it to rank k
    S = np.diag(s[:k])
    return np.dot(U[:, :k], np.dot(S, Vt[:k, :]))

def frobenius_percent(original, approx):
    # computes frobenius error
    return 100*(np.linalg.norm(original - approx, 'fro')/np.linalg.norm(original
        , 'fro'))

def memory_saved(m, n, k, s):
    # computes percentage memory saved
    new = (m * k + k + n * k)
    old = m * len(s) + len(s) + n * len(s)
    return 100*(old-new)/old
```

B. Discussion

As you can see in FIG. 1, one can save up to 96.9% memory before the image gets degraded too badly, possibly more with more clever compression techniques. However, as you can see in FIG. 2, figures which have texture and complexity as their main qualities tend to do worse under this compression as most of what distinguished the image is lost as textures take up too much space and when we truncate the singular values we simplify the image a lot. Images that are simple to begin with do better, also faces probably do better in general because our brain would compensate for all the noise and still see a face in there.

FIG. 1: Compression of a figure of Feynman.

II. ENTANGLEMENT ENTROPY IN GROUND STATES

A. Implementation

```python
def entanglement_entropy(l, L, J, h, periodic):

    vec = psi_gs(L, J, h, periodic)
    psi_matrix = vec.reshape((2**l, 2**(L-l))) # does the schmidt cut at l
    u, s, vh = np.linalg.svd(psi_matrix, compute_uv=True)
    lambdas = s**2
    S = -np.sum(lambdas * np.log(lambdas))

    return S
```

B. Discussion

As seen in FIG. 3, the entanglement entropy peaks at $L = l/2$ where it becomes a bipartite system achieving maximal entanglement. With periodic boundary conditions, the subsystem A can interact with its complement through two boundary edges compared to one, so in most cases the entanglement entropy should differ by a factor of 2. So when the correlations are low as in the paramagnetic phase, the thing said above holds. However, when the correlations are maximal like in the ferromagnetic phase, all spins point up anyway so it doesn't matter that the subsystem has double the area of contact with the complement, because entanglement entropy would still be the same and maximal, this is also why it approaches the thermodynamic limit much faster. And at the critical point, this ratio is somewhere in between at about ~ 1.8. It is very interesting to note that for some reason the entanglement entropy with periodic boundary conditions is higher in the critical point than in the ferromagnetic phase. I believe this is because the correlation length goes to infinity near the critical point, this will keep recurring. It is also going to be a recurring theme in this section that there are some weird finite-size effects in the periodic boundary conditions, depending on whether on not you choose a odd or even system size.

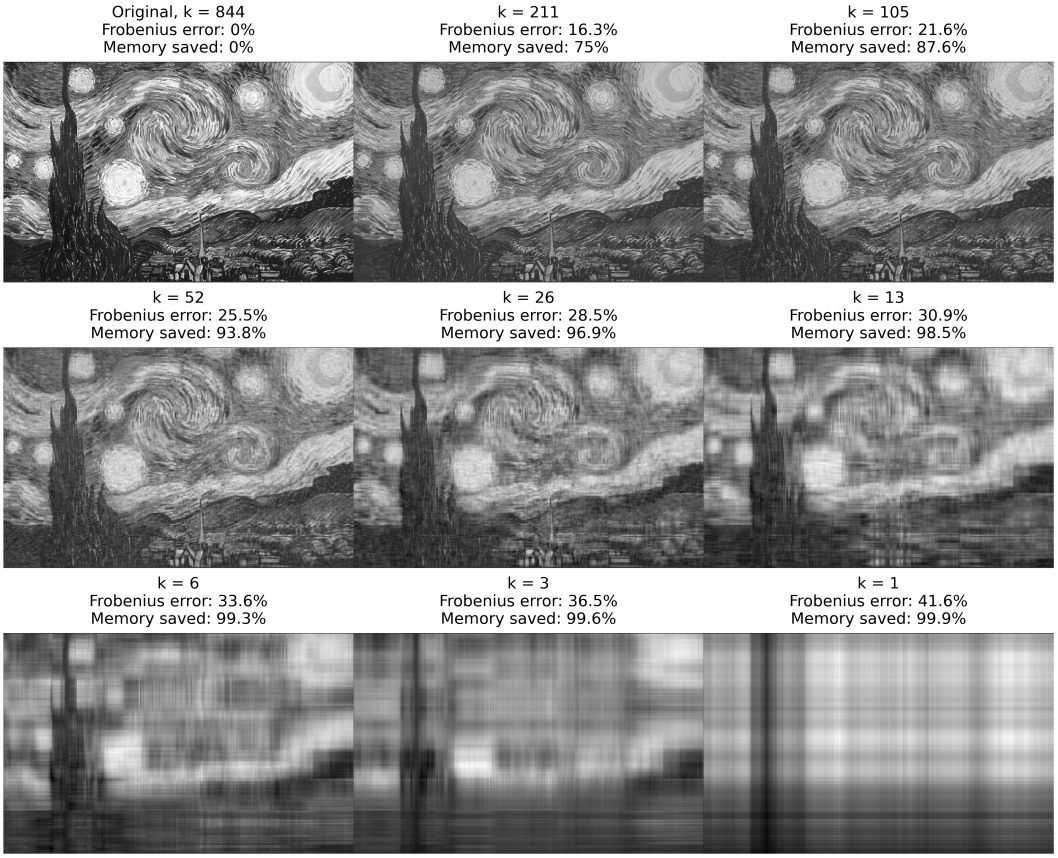


FIG. 2: Compression of a figure of Feynman.

The results mentioned above are concisely described in FIG. 4, in the paramagnetic phase, as expected the periodic B.C. has twice the entanglement entropy, the ferromagnetic phase they approach the same limit and at the critical point they seem to be off by a factor of 1.8 or so.

Next, as shown in FIG. 5, a fit of the entanglement entropy for both boundary conditions at the critical point was done to the area law.

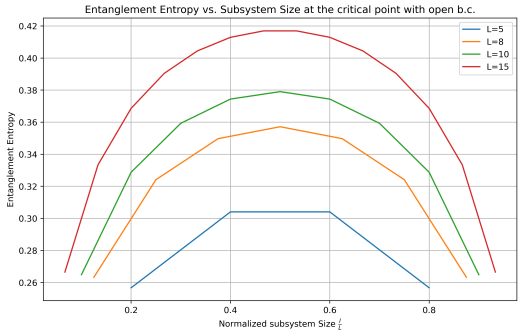
$$\boxed{\text{Area law: } S(l;L) = \frac{c}{3}\log\left(\frac{L}{\pi}\sin\frac{\pi l}{L}\right) + C} \tag{1}$$

The fit for open B.C.

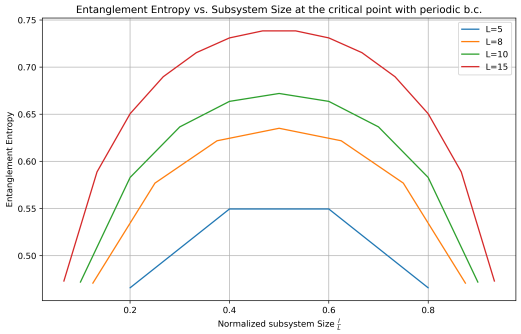
$$S \sim \frac{0.29}{3}\log\left(\frac{L}{\pi}\sin\frac{\pi l}{L}\right) + 0.27 \tag{2}$$

The fit for periodic B.C.

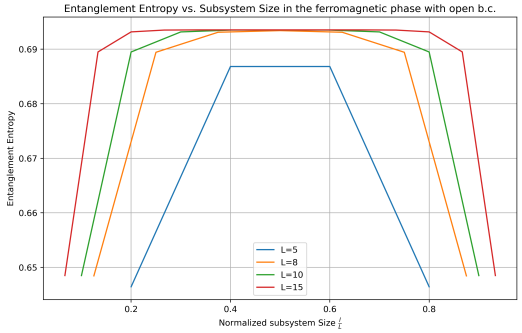
$$S \sim \frac{0.51}{3}\log\left(\frac{L}{\pi}\sin\frac{\pi l}{L}\right) + 0.48 \sim 1.76 \cdot S_{\text{open}} \tag{3}$$



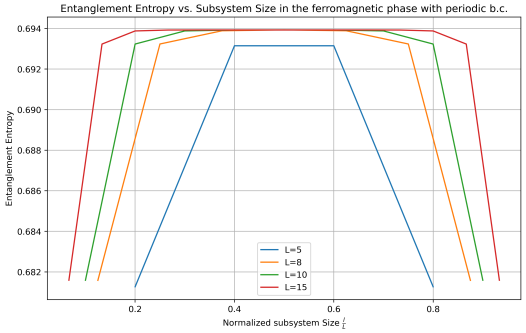
(a) Critical point with open B.C.



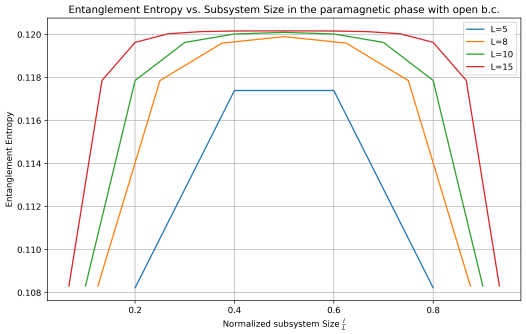
(b) Critical point with periodic B.C.



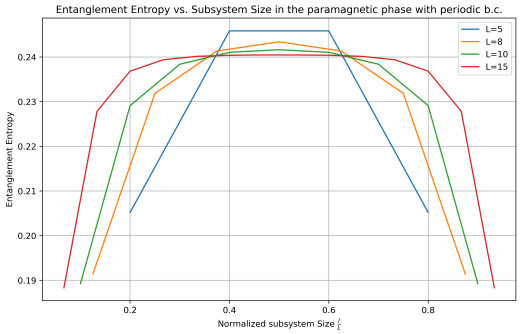
(c) Ferromagnetic phase with open B.C.



(d) Ferromagnetic phase with periodic B.C.



(e) Paramagnetic phase with open B.C.



(f) Paramagnetic phase with periodic B.C.

FIG. 3: Ground state. Comparison of the plots of Entanglement entropy vs. Subsystem size for open and periodic B.C. and at ferromagnetic phase, paramagnetic phase and at the critical point.

Now let us come to the highest-energy state. It should still satisfy the area law because the correlations are still highly localized in the highest energy state, for example, this would correspond to a state where all spins are anti-aligned with their neighbors. Such a case also has mostly local correlations, similarly in the paramagnetic phase instead of being aligned with the magnetic field they become anti-aligned. Since, local correlations would naturally imply a boundary proportionality of entanglement, we would expect it to obey the area law and it does.

As seen in FIG. 6, there are some weird odd-even effects going on for periodic boundary conditions based on the size chosen, clearer in FIG. 7. Infact the entanglement entropy goes above 1, so this is probably an error from my end, but anyway since I was only supposed to do one B.C. and one phase, I did not pay too much attention to this. As seen in FIG. 8 the highest energy state also obeys the area law. So basically, the takeaway is that it behaves very similar to the ground-state as it seems like there is some sort of map between the correlations, if you reverse the correlation of the ground-state from align-¿anti-align you get the highest excited state, so makes sense why they behave the same as the correlations still behave similarly just opposite.

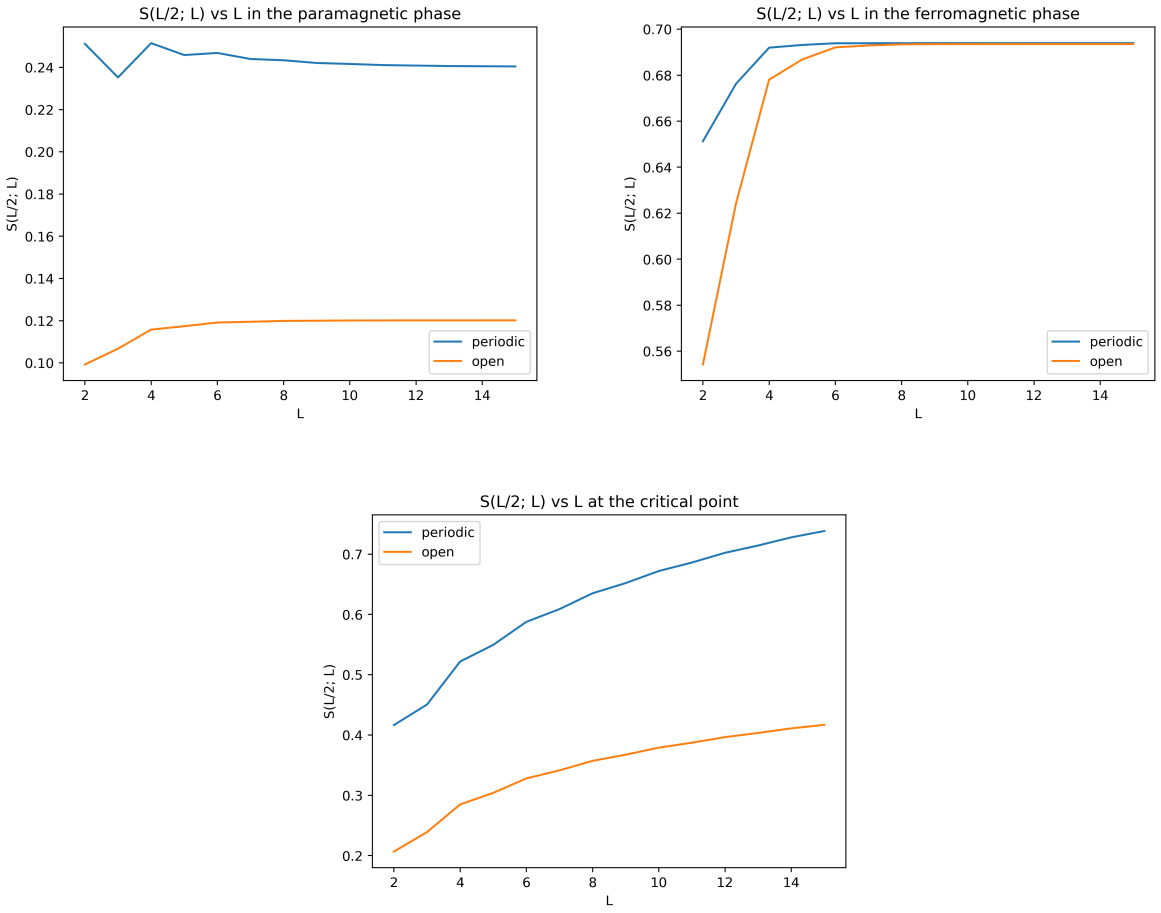
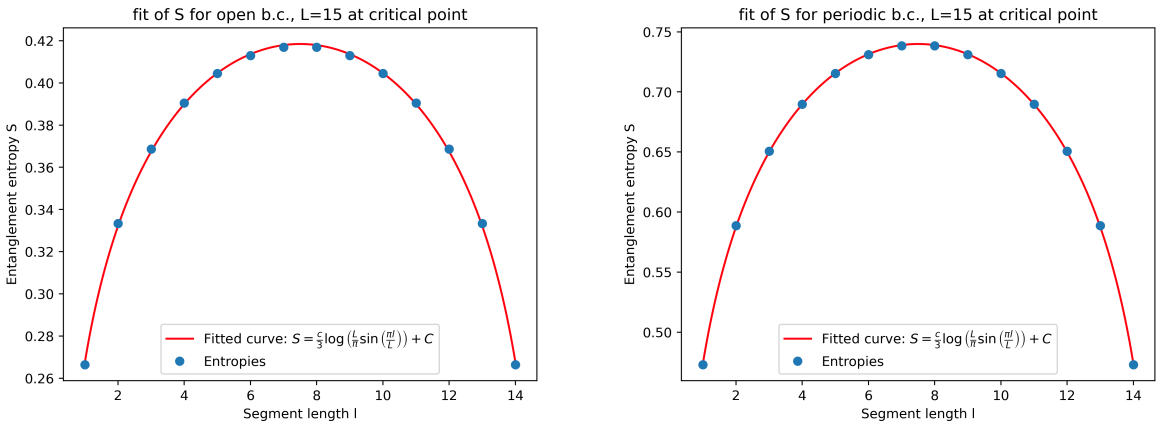


FIG. 4: Ground state. Comparison of the approach to the thermodynamic limit in the ferromagnetic, paramagnetic phase and at the critical point for different boundary conditions of $S(L/2; L)$.



(a) Fit for open B.C.

(b) Fit for periodic B.C.

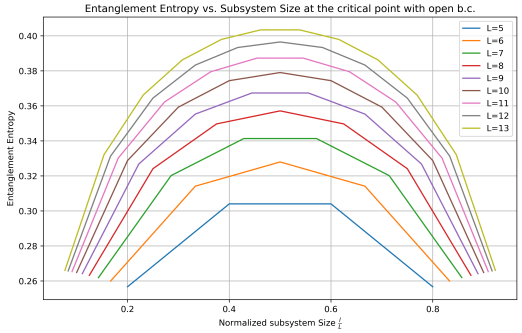
FIG. 5: Ground State. Fits of the Entropy to the area law for both boundary conditions at $L = 15$

Fit for open B.C.

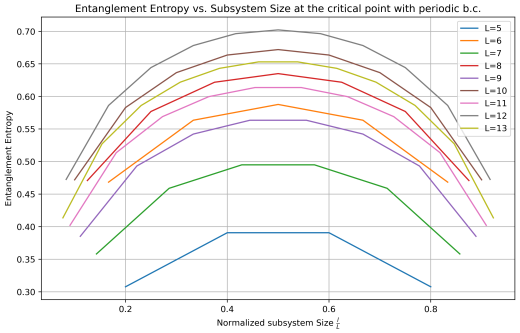
$$S \sim \frac{0.29}{3} \log\left(\frac{L}{\pi} \sin\frac{\pi l}{L}\right) + 0.27 \tag{4}$$

Fit for periodic B.C.

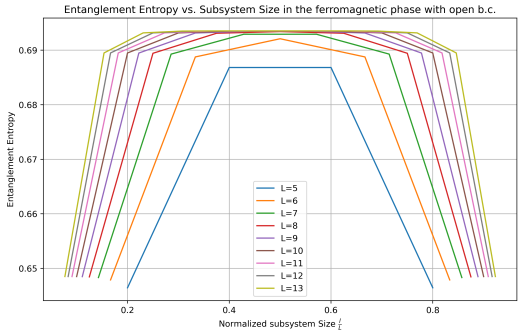
$$S \sim \frac{0.50}{3} \log\left(\frac{L}{\pi} \sin\frac{\pi l}{L}\right) + 0.42 \tag{5}$$



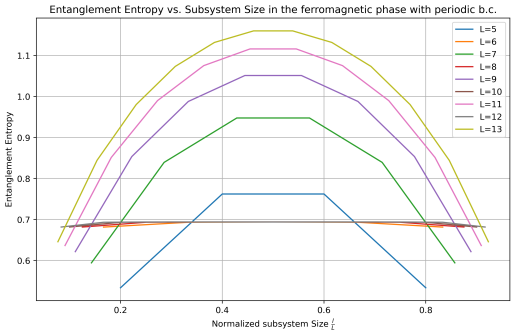
(a) Critical point with open B.C.



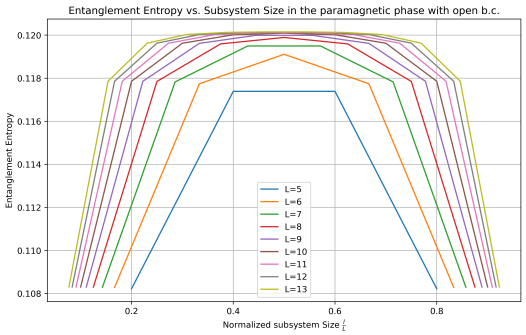
(b) Critical point with periodic B.C.



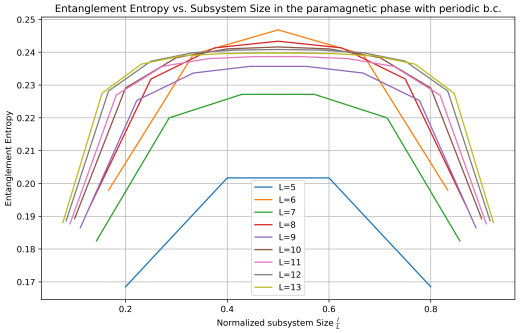
(c) Ferromagnetic phase with open B.C.



(d) Ferromagnetic phase with periodic B.C.



(e) Paramagnetic phase with open B.C.



(f) Paramagnetic phase with periodic B.C.

FIG. 6: Highest energy state. Comparison of the plots of Entanglement entropy vs. Subsystem size for open and periodic B.C. and at ferromagnetic phase, paramagnetic phase and at the critical point.

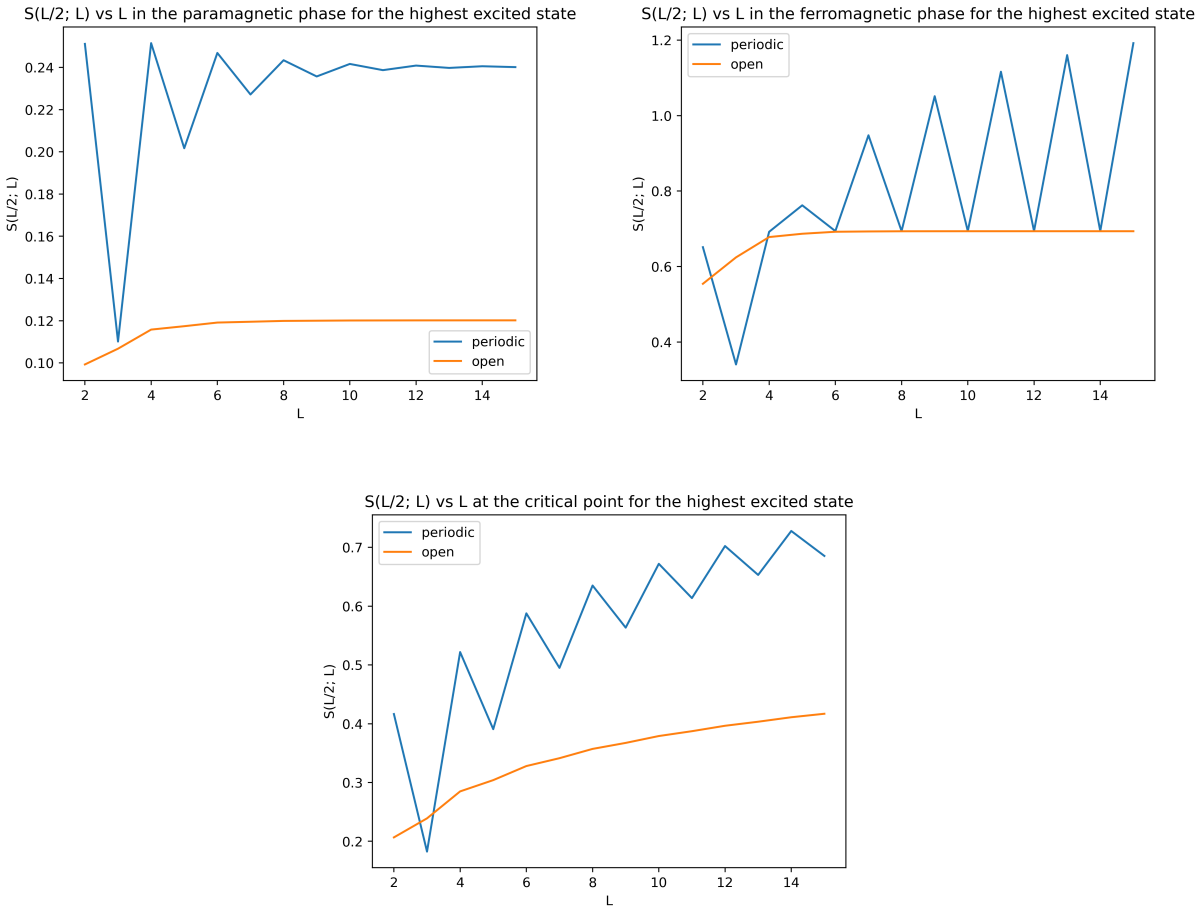


FIG. 7: Highest Energy state. Comparison of the approach to the thermodynamic limit in the ferromagnetic, paramagnetic phase and at the critical point for different boundary conditions of $S(L/2; L)$. As you can see the mentioned, finite size effects based on the parity of size you select is evident, but if you ignore one of them, it seems to be following the same patterns as the ground state

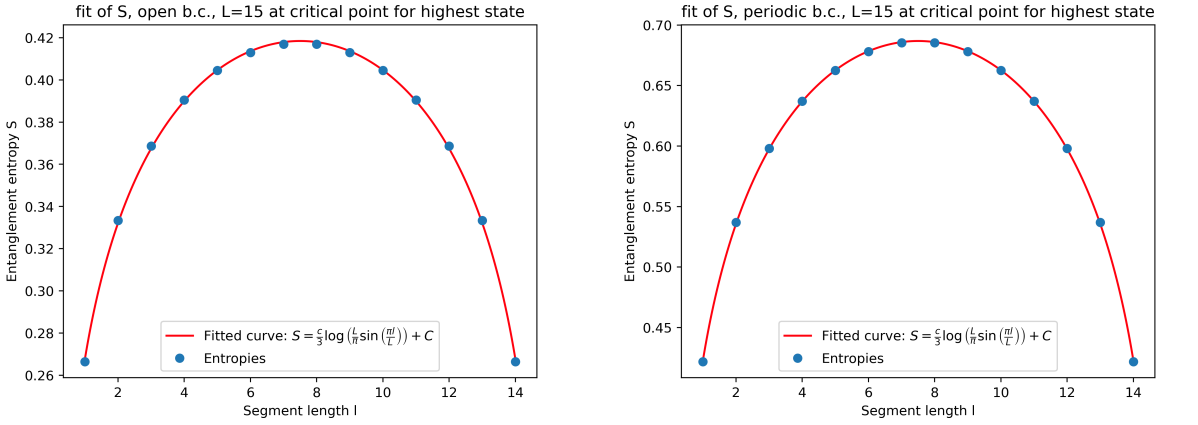


FIG. 8: Highest Energy state. Fits of the Entropy to the area law for both boundary conditions at $L = 15$

III. TRUNCATION ERROR OF SCHMIDT DECOMPOSITION

A. Implementation

I use the function "H_psi" which is a sparse matrix implemented as a pointer that takes in ψ and outputs $H\psi$.

```
def schmidt_psi(k, L, J, h, periodic):

    l = L//2
    eig, vec = psi_gs(L, J, h, periodic)
    vec = vec.flatten()

    psi_matrix = vec.reshape((2**l, 2**(L-l)))
    U, S, Vh = np.linalg.svd(psi_matrix, compute_uv=True)

    S_truncated = S[:k]
    U_truncated = U[:, :k]
    Vh_truncated = Vh[:k, :]

    psi_approx = np.dot(U_truncated, np.dot(np.diag(S_truncated), Vh_truncated))
    psi_approx_flat = psi_approx.flatten()

    Hpsi_approx = H_psi(psi_approx_flat, L, J, h, periodic)
    energy_approx = np.dot(psi_approx_flat.conj().T, Hpsi_approx)
    norm_psi_approx = np.linalg.norm(psi_approx_flat)**2
    normalized_energy_approx = energy_approx / norm_psi_approx
    d_k = np.sqrt(np.sum(S[k:]**2))

    return d_k, normalized_energy_approx-eig[0]
```

B. Discussion

As seen in FIG. 8, the energy difference falls extremely really quickly and so does the Frobenius error as k increases beyond 1. From the fit you can see that for $L = 10$.

$$\Delta E \sim 2.61d^2(k) + 0.04d \tag{6}$$

This just shows us show how much storage you can save by just doing a simple Schmidt truncation.

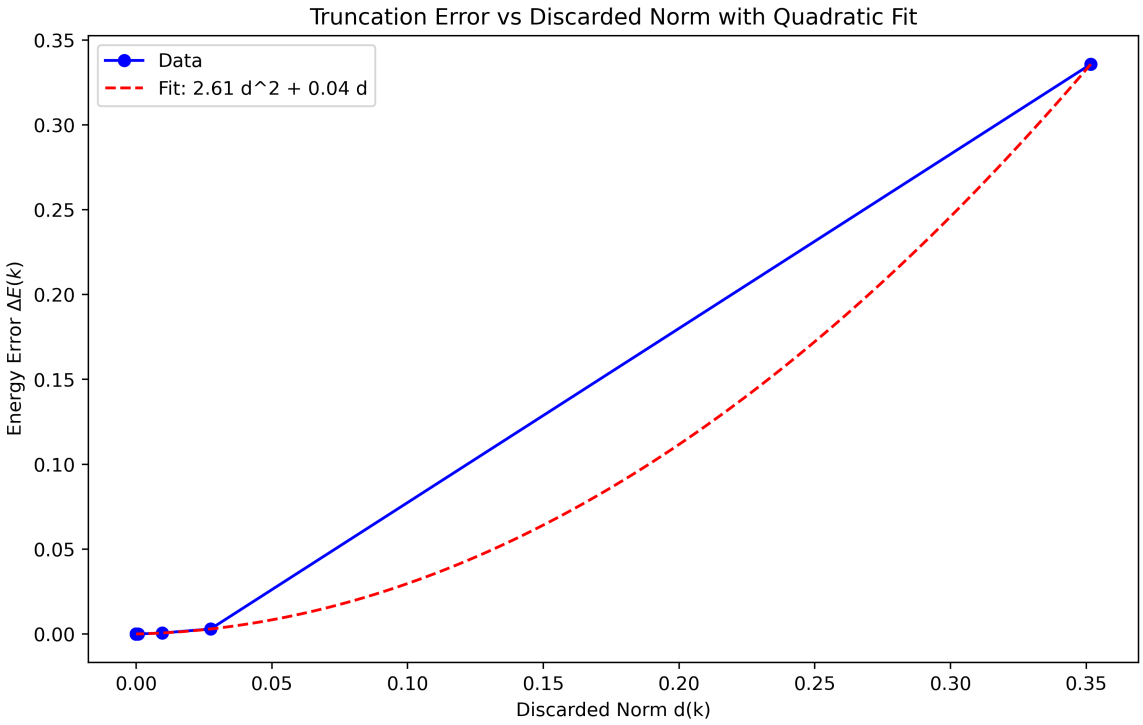


FIG. 9: A quadratic fit of the error ΔE to the Frobenius error.

IV. ENTANGLEMENT ENTROPY OF HIGHLY EXCITED STATES

A. Implementation

To make the plots look a lot cleaner, I find the non-degenerate wavefunctions near 0, compute each of their entanglement entropies then average them out.

```
def compute_middle_spectrum_entropies(L_values, J, h, periodic, num_states):
    results = {}
    for L in L_values:
        H = sparseH(L, J, h, periodic).toarray()
        eigs, vecs = np.linalg.eigh(H)
        middle_indices = np.argsort(np.abs(eigs))[:num_states]
        non_deg_indices = np.unique(middle_indices)

        Entropies = []
        for l in tqdm(range(1, L), desc=f'Computing for L={L}'):
            avg_entropy = np.mean([entanglement_entropy(l, L, vecs[:, idx]) for
                idx in non_deg_indices])
            Entropies.append(avg_entropy)
        results[L] = Entropies
    return results
```

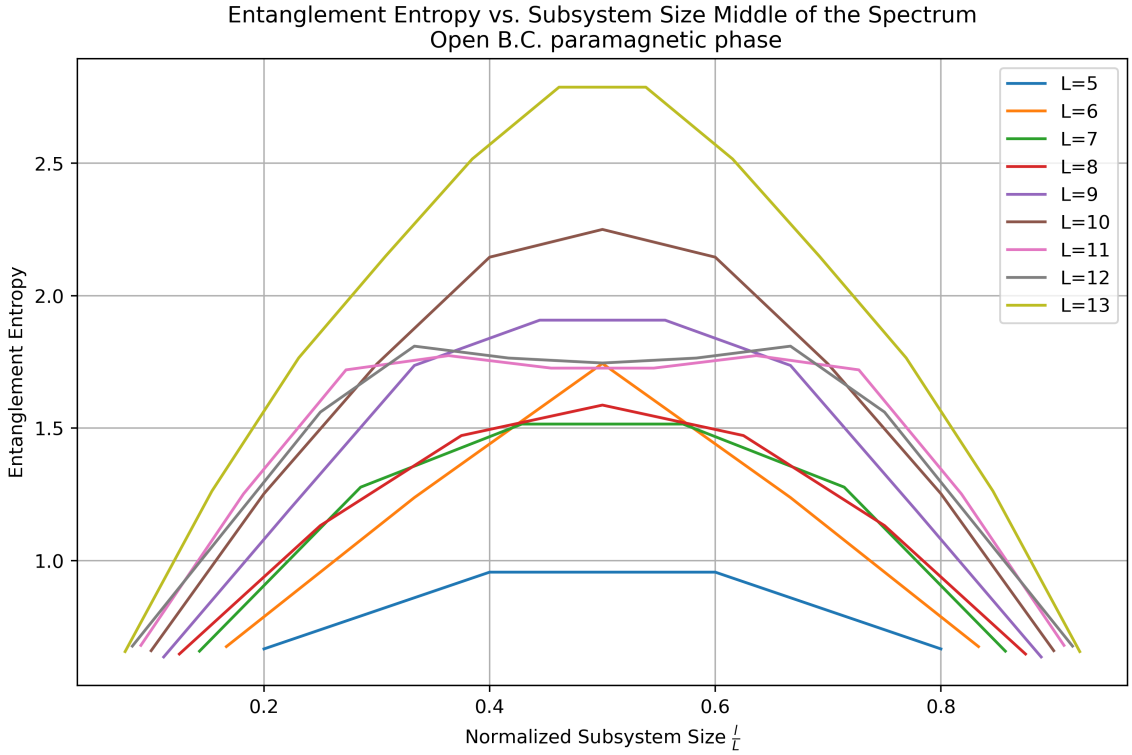


FIG. 10: States in the middle. Plot of averaged out Entanglement entropy for various states in the middle of the spectrum vs. Subsystem size for open B.C. Much higher entropy than the ground state and highest excited state.

B. Discussion

From figures 10 and 11, we can clearly see that states in the middle obey volume law of entanglement. As seen in figure 12, a random vector's entanglement entropy obeys volume law of entanglement. So, clearly the logic for why states near zero energy obey volume law of entanglement is because near zero, the correlations are neither positive nor negative and the vector you obtain is basically random. I think this comes from the Eigenstate Thermalization Hypothesis. And since random matrices would be mixed over all basis states uniformly when you trace out one of the systems the result would be maximally mixed giving rise to maximal entropy. Page's theorem (entropy of a random state is close to maximal), so from that it follows

$$S_A \approx \log(d_A) = l \cdot \log(2) \tag{7}$$

This is also why I suspect the growth toward the peak is also almost linear.

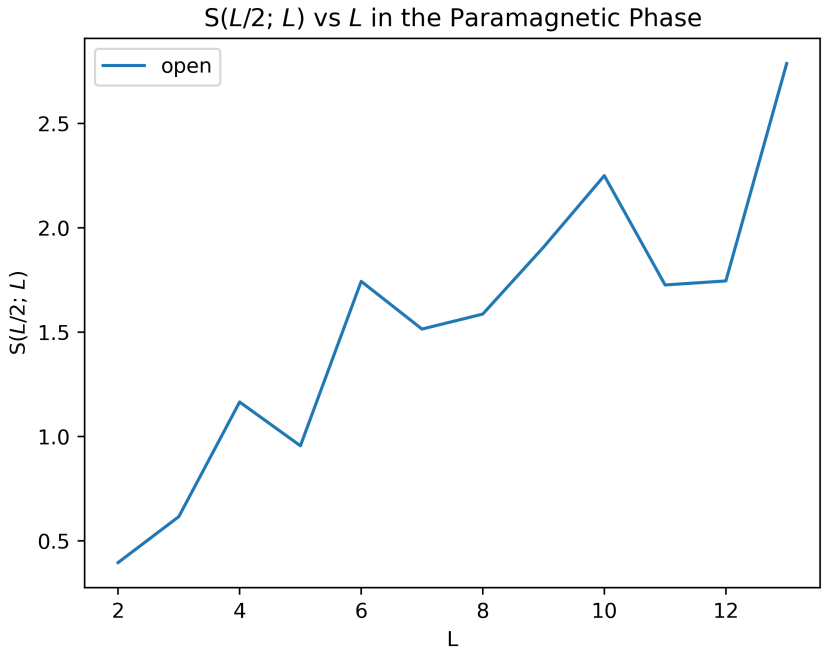


FIG. 11: States in the middle. Plot of averaged out entanglement entropy $S(L/2; L)$ vs L, clearly showing volume law of entanglement.

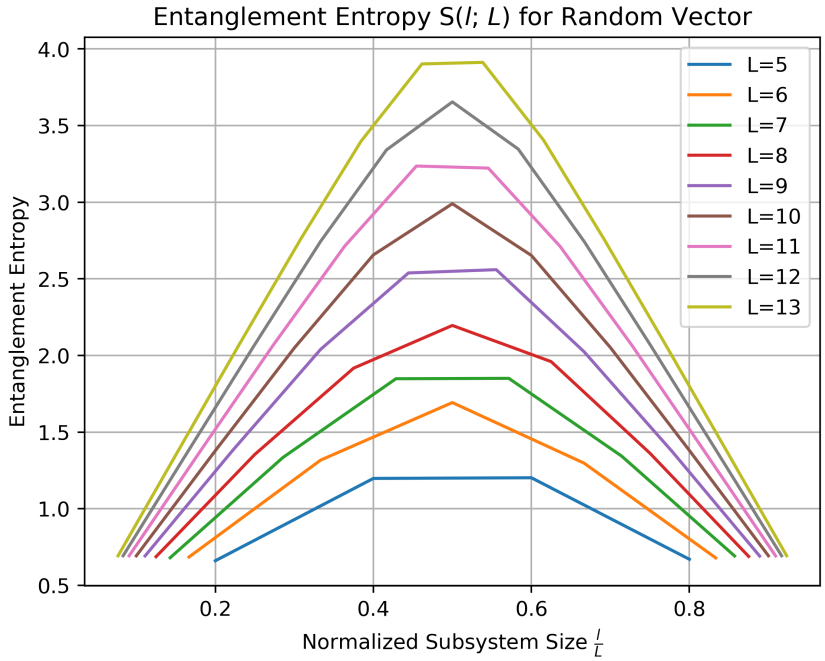


FIG. 12: Random states. Plot of Entanglement entropy for various states in the middle of the spectrum vs. Subsystem size. Clear volume law of entanglement observed.

V. MATRIX PRODUCT STATE REPRESENTATION FOR GROUND STATE

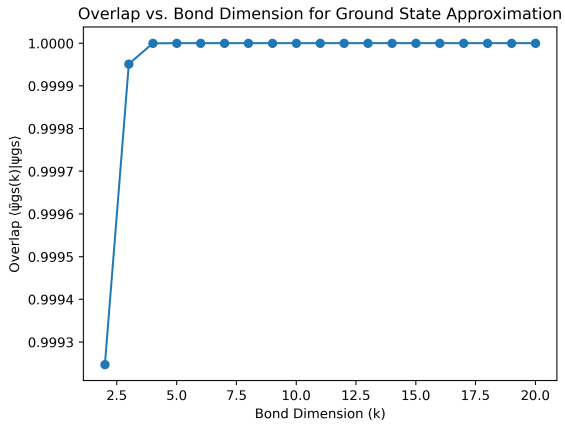
A. Implementation

Vector to MPS tensors

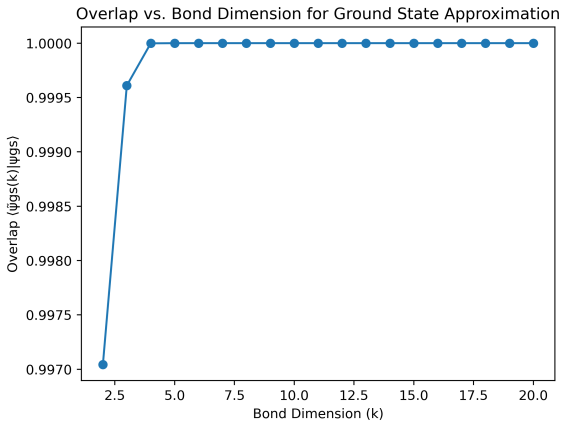
```python
def mps(psi, k):

    mps_tensors = []
    L = int(np.log2(len(psi)))  # Number of spins/sites
    psi_matrix = psi.reshape(2, 2**(L-1))

    for i in range(L-1):

        U, S, Vh = np.linalg.svd(psi_matrix, full_matrices=False)

        chi = min(k, len(S))

        S_truncated = S[: chi]
        U_truncated = U[:, :chi]
        Vh_truncated = Vh[:chi, :]

        if i == 0:
            mps_tensors.append(U_truncated)

        else:
            mps_tensor.append(U_truncated.reshape((-1, 2, chi)))
            mps_tensors.append(mps_tensor)

        if i < L - 2:
            psi_matrix = (np.diag(S_truncated) @ Vh_truncated).reshape(2*chi,
                -1)

        else:
            last_tensor = (np.diag(S_truncated) @ Vh_truncated)
            mps_tensors.append(last_tensor)

    return mps_tensors
```

Contracting MPS tensors to get back the vector

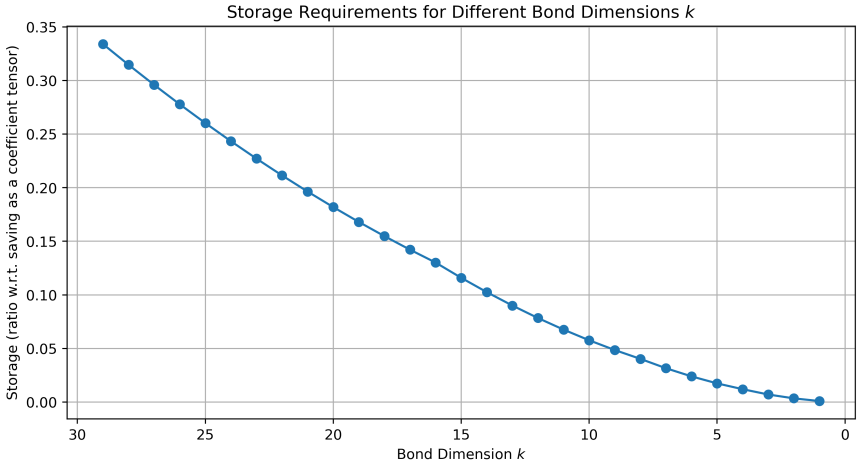
```python
def contract_mps_tensors(mps_tensors):
    state = mps_tensors[0]
    for i, tensor in enumerate(mps_tensors[1:], 1):
        if i == len(mps_tensors) - 1:
            state = np.einsum('...j,jk->...k', state, tensor)
        else:
            state = np.einsum('...j,jkl->...kl', state, tensor)

    return state.flatten()
```



(a) Overlap vs bond-dimension for $(h/J) = 1.25$



(b) Overlap vs bond-dimension for $(h/J) = 1$



(c) Storage savings as a function of k

FIG. 13: Storage savings and overlap as a function of bond-dimension.

B. Discussion

As you can see in FIG. 13, if you examine the scales near the critical point, the wave function converges much slower. I believe this is because since the correlation length goes to infinity at the critical point and the Schmidt coefficients have much slower decay due to the increased correlations. Since, the entanglement is so high and is encapsulated in the Schmidt coefficients, so when you truncate near the critical point you lose a lot more information about the wave function.

As you can see for $k >= 5$ the approximate wave function seems to basically have a full overlap with the actual wave function and $k = 5$, we have a storage saving of about 99.97% as is clear from FIG. 13, so that is about the storage we expect to save probably more in most cases.

VI. EFFICIENT CALCULATIONS WITH MPS

A. Implementation

Inner product of two MPS tensors

```
def inner_product(bra, ket):
    result = np.einsum('ij,im->jm', bra[0].conj(), ket[0])

    for i in range(1, len(ket) - 1):

        A = np.einsum('jkl,mkn->jmln', bra[i].conj(), ket[i])
        result = np.einsum('jm,jmln->ln', result, A)

    final_result = np.einsum('jm,jm', result, np.einsum('ji,mi->jm', bra[-1].
       conj(), ket[-1]))

    return final_result
```

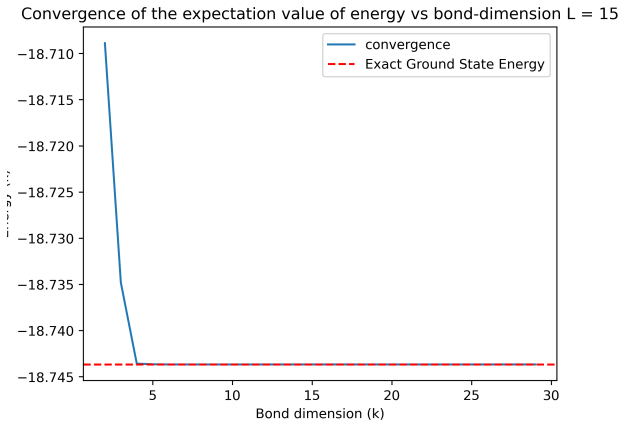
Implementation of expectation value of energy

```
def magnetic_term(ket):

    X = np.array([[0, 1], [1, 0]])

    L = len(ket)
    results = np.zeros(L)

    for i in range(L):

        Hket = [np.copy(k) for k in ket]

        if i == 0:
            Hket[i] = np.einsum('ij,ik->jk', X, ket[i])

        else:
            Hket[i] = np.einsum('ij,ki...->kj...', X, ket[i])

        results[i] = inner_product(ket, Hket)

    return np.sum(results)


def ising_term(ket):

    Z = np.array([[1,0],[0,-1]])
    L = len(ket)
    results = np.zeros(L)

    for i in range(0, L-1):

```

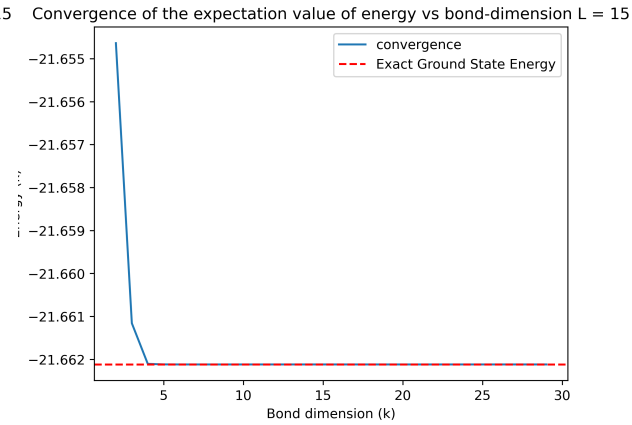
```
        Hket = [np.copy(k) for k in ket]

        if i == 0:
            Hket[i] = np.einsum('ij,ik->jk', Z, ket[i])
        else:
            Hket[i] = np.einsum('ij,ki...->kj...', Z, ket[i])

        Hket[i+1] = np.einsum('ij,ki...->kj...', Z, ket[i+1])

        results[i] = inner_product(ket, Hket)

    return np.sum(results)

def expectation_value_energy(L, J, h, periodic, k):

    ket = mps(psi_gs(L, J, h, periodic), k)

    energy = -h*magnetic_term(ket)-J*ising_term(ket)
    norm = inner_product(ket, ket)

    return energy/norm
```

Implementation of expectation value of correlation functions

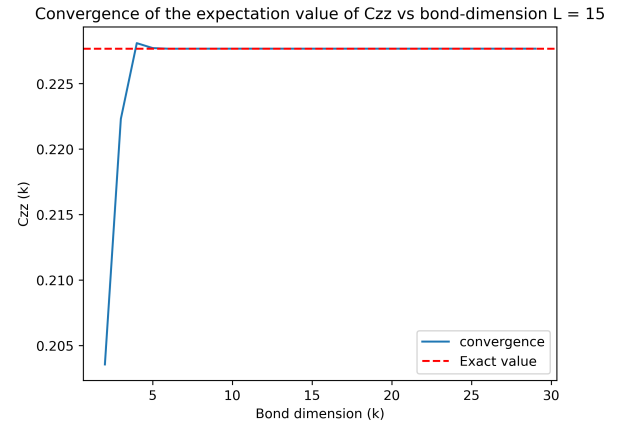
```
def expectation_value_Czz(L, J, h, periodic, k, r):

    Z = np.array([[1,0],[0,-1]])
    ket = mps(psi_gs(L, J, h, periodic), k)

    Czzket = [np.copy(k) for k in ket]

    Czzket[0] = np.einsum('ij,ik->jk', Z, ket[0])
    Czzket[r] =np.einsum('ij,ki...->kj...', Z, ket[r])

    return inner_product(ket, Czzket)

def expectation_value_Cxx(L, J, h, periodic, k, r):

    X = np.array([[0,1],[1,0]])
    ket = mps(psi_gs(L, J, h, periodic), k)

    Cxxket = [np.copy(k) for k in ket]

    Cxxket[0] = np.einsum('ij,ik->jk', X, ket[0])
    Cxxket[r] = np.einsum('ij,ki...->kj...', X, ket[r])

    return inner_product(ket, Cxxket)
```



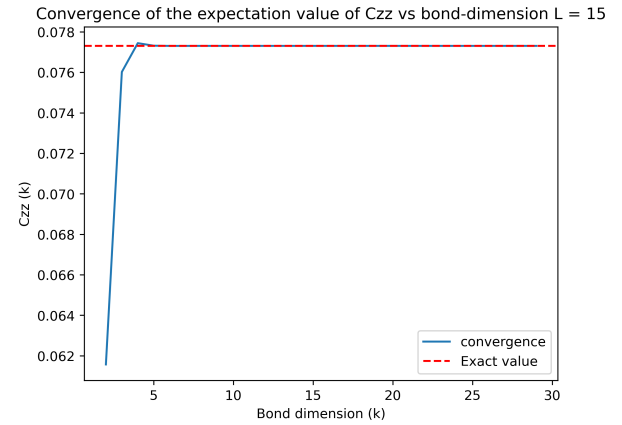
(a) Energy convergence for $(h/J) = 1$



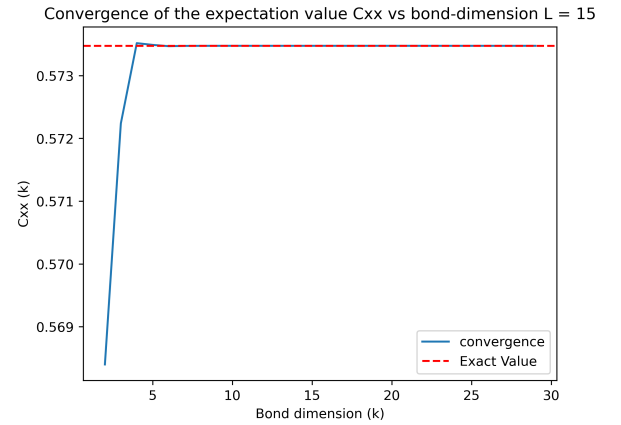
(b) Energy convergence for $(h/J) = 1.25$



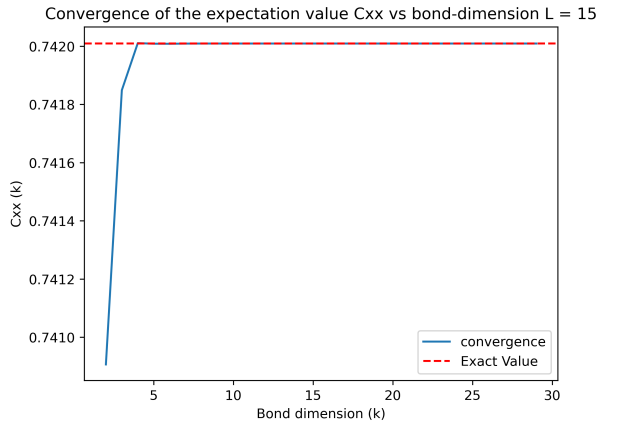
(c) Czz convergence for $(h/J) = 1$



(d) Czz convergence for $(h/J) = 1.25$



(e) Cxx convergence for $(h/J) = 1$



(f) Cxx convergence for $(h/J) = 1.25$

FIG. 14: Energy and Correlation function convergence as a function of bond-dimension..

B. Discussion

Again the same motif continues in FIG. 14, convergence is much much quicker slightly far away from the critical point. I was not sure what else correlation functions I had to plot that could convey any better information about this convergence, the half-correlation function is the same and $\langle (M/L)^2 \rangle$ is just the sum of many $\langle \sigma_z^{(1)} \sigma_z^{(1+r)} \rangle$.

VII. BRINGING AN MPS TO A CANONICAL FORM WITH AN ORTHOGONALITY CENTER

A. First state

Clearly, since the matrices are both left and right normalized, I can put in the same S matrix at every cut, the S-matrix has to be the identity and the highest two Schmidt values across any cut should be $(1, 1)$. I verify this in my code (full thing on github)

Implementation:

```
def compute_vector_optimized(state):
    binary_state = [int(x) for x in format(state, '0{}b'.format(L))]
    result_matrix = A1[binary_state[0]]
    for i in range(1, L):
        if i != L - 1:
            result_matrix = result_matrix @ Aell[binary_state[i]]
        else:
            result_matrix = result_matrix @ AL[binary_state[i]]
    return result_matrix[0]
def analyze_mps_schmidt_values(wavefunction, L):
    schmidt_values = {}
    for l in range(L-1, 0, -1):
        psi_matrix = wavefunction.reshape((2**l, 2**(L-l)))
        U, S, Vh = np.linalg.svd(psi_matrix, compute_uv=True)
        schmidt_values[l] = S[:2]  # Assuming bond dimension 2
    return schmidt_values
```

B. Second state

This state is not in canonical form, so I have to convert it to a canonical form by doing SVDs from the left to right and then come back to read the Schmidt values off any cut. At the same time I do a brute force implementation like previously to check my Schmidt values. They do end up matching, so I have successfully implemented this back and forth procedure. Things to note for next time, I keep saving it in the second notation described in the notes where I absorb the S matrix into one of the As, need to make sure I remember that constantly, something weird is happening with normalization and I used to save my first and last tensors as 2×2 matrices but saving them with a dummy index $2 \times 2 \times 1$ matrix style makes life so much more easier, I have to go back to my mps functions and edit them all.

```
def canonical_converter(mps_tensors, k, center):
    new_mps_tensors = mps_tensors.copy()
    L = len(new_mps_tensors)
    for i in range(L-1):

        W = np.einsum('ijk,klm->ijlm', new_mps_tensors[i], new_mps_tensors[i+1])

        Wmatrix = W.reshape((W.shape[0]*2, -1))

        U, sv, Vt = np.linalg.svd(Wmatrix, full_matrices=False)
        chi = min(k, len(sv))

        U = U[:, :chi]
        new_mps_tensors[i] = U.reshape(-1, 2, chi)

        S = np.diag(sv[:chi])
        Vt = Vt[:chi, :]


        new_mps_tensors[i+1] = (S @ Vt).reshape(chi, 2, -1)

    schmidt_tracker=[]


    for i in range(L-1, 0, -1):


        W = np.einsum('ijk,klm->ijlm', new_mps_tensors[i-1], new_mps_tensors[i])

        Wmatrix = W.reshape((W.shape[0]*2, -1))

        U, sv, Vt = np.linalg.svd(Wmatrix, full_matrices=False)
        chi = min(k, len(sv))

        U = U[:, :chi]
        S = np.diag(sv[:chi])
        Vt = Vt[:chi, :]

        new_mps_tensors[i] = Vt.reshape(chi, 2, -1)

        schmidt_tracker.append(sv[:chi])

        new_mps_tensors[i-1] = (U@S).reshape(-1, 2, chi)

        if i == center:
            break

    return np.array(schmidt_tracker), new_mps_tensors, S
```