MQST Spring Lab Report
Matrix Product State (MPS) by Payal Kaushik

Matrix product state, also referred to as Tensor Train in the field of physics, offers a method to represent an N-dimensional tensor by decomposing it into a product of smaller dimensional tensors. This representation involves factorizing a tensor with N indices into a series of interconnected three-index tensors, forming a chain-like structure.
In tensor diagram notation (also known as Penrose notation), a Ndimensional tensor T with indices s_1, s_2, ..., s_N can be decomposed into an MPS of N small tensors-

$$T^{s_1 \ldots s_N} = \sum_{\{\alpha\}} A^{s_1}_{\alpha_1} A^{s_2}_{\alpha_1 \alpha_2} A^{s_3}_{\alpha_2 \alpha_3} \cdots A^{s_{N-1}}_{\alpha_{N-2}\, \alpha_{N-1}} A^{s_N}_{\alpha_{N-1}}$$

Figure 1: MPS in Traditional tensor notation [1]

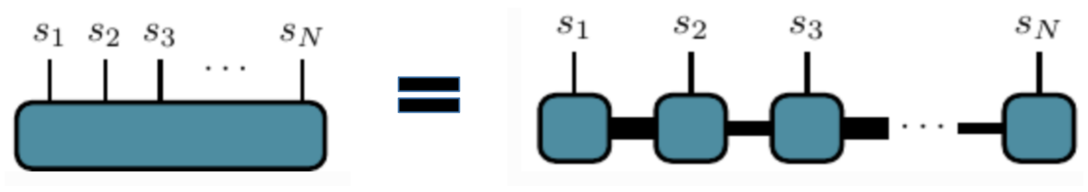


Figure 2: Graphical representation of MPS [1]

In MPS, the uncontracted indices are referred to as the physical or visible indices (s_1, s_2, ..., s_N), whilst the others (α_1, α_2, ..., α_{N-1}), connecting two tensors, are called virtual, bond, or matrix indices.
In the realm of quantum computing, Matrix Product States (MPSs) serve as a representation for quantum states by organizing them into a linear chain or ring structure composed of tensors. Remarkably, any quantum state can be precisely depicted using this MPS format, allowing for an exact representation. Moreover, this representation is particularly efficient when it comes to approximating a specific class of one-dimensional gapped systems. These systems, characterized by a significant energy gap between the ground state and excited states, can be effectively approximated using MPSs, making them a valuable tool for studying and simulating such systems in a computationally efficient manner.
For an n-qubit quantum state,

$$|\psi\rangle = \sum_{ij..k} \psi_{ij..k} |ij..k\rangle$$

the number of independent coefficients $\psi_{ij..k}$ scales exponentially (2^n) with n, which will be computationally unmanageable as n exceeds a certain limit. Using the MPS format, the quantum state can be alternatively represented with less amount of data as-

$$|\psi\rangle = \sum_{ij..k} Tr(A_i^{[1]} A_j^{[2]} .. A_k^{[n]}) |ij..k\rangle$$

where $A_i^{[1]}$, $A_j^{[2]}$,.. $A_k^{[n]}$ are indexed sets of matrices. Now, the terms of $|\psi\rangle$ can be easily calculated as a product of matrices [2].

The Need for a Matrix Product State (MPS)

Now, the question comes, why do we need an MPS in the first place? Why can't we work with the big N-dimensional tensor? To answer this question, let's analyze the Fig. 1 once again. Say, If the dimensions of each of the indices (s_1, s_2, ..., s_N) of N-dimensional tensor is d, then the number of parameters for this tensor will be d^N. Whereas, if this tensor is decomposed into an MPS of N small tensors with uniform bond dimension, say χ, the number of total parameters will be only $Nd\chi^2$ as shown in Fig.3.

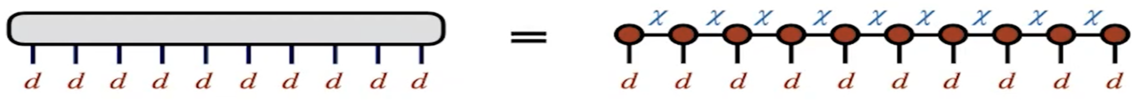


Figure 3: MPS with bond dimension χ

Here, χ is the dimension of the bond indices, hence known as **Bond dimension**. It can vary from bond to bond, but for the sake of simplicity of this example, we are taking it to be uniform and equal to χ. The bond dimension can be understood as a parameter that governs the level of flexibility or complexity achievable in an MPS network. It acts as a control mechanism, influencing the network's capacity to represent and capture information. If there is no drop in values during decomposition, the MPS will give the exact representation, however, there will be no speedup and no parameter reduction. But we can drop some singular values, without loss of information and hence, reduce the number of parameters [3].

How to prepare Matrix Product State ?

There are various ways to decompose a big dimensional tensor into an MPS like Singular Value Decomposition (SVD), Euler Decomposition, and others. The most commonly used is **Singular Value Decomposition (SVD)**. The goal of SVD is to decompose a matrix M into three matrices U, S, and V with the diagonal matrix S containing the singular values of M, unitaries U and V containing the left singular vectors and right singular vectors of matrix M respectively, such that-

$$M = USV^\dagger$$

To get two tensors A and B as shown in Fig.4, the singular value matrix S can be absorbed on either side or on both sides equally depending on the need of the situation. The tensor U and V act as just basis matrices. The information of the tensor is contained in its singular values and wherever the singular values are accumulated, that point is known as **Orthogonality center**. For more details about the orthogonality center, refer to Ryan's section. The method used to decompose high order tensors using SVD is known as **Higher Order SVD (HOSVD)** which works on Tucker decomposition. To know more about Tucker decomposition, refer to [4].

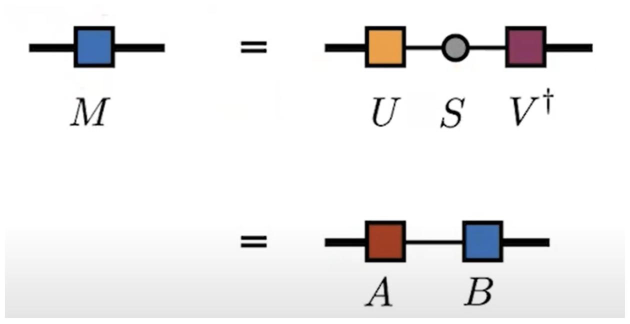


Figure 4: Singular Value Decomposition

Decomposing a 4 order tensor into MPS form

A 4-order tensor can be decomposed into an MPS as explained in Fig.5. The decomposition can start from either of the sides, it starts from the leftmost index in this example. After the first svd, we get U_1, S_1, and V_1^T. The S_1 is absorbed with V_1^T in the next step, to keep the orthogonality center at the rightmost tensor. The same process continues for the next 6 steps, and we obtain the MPS form of the 4-order tensor with orthogonality center in A_4. The MPS from contains 4 tensors A_1, A_2, A_3, and A_4 of small dimensions.

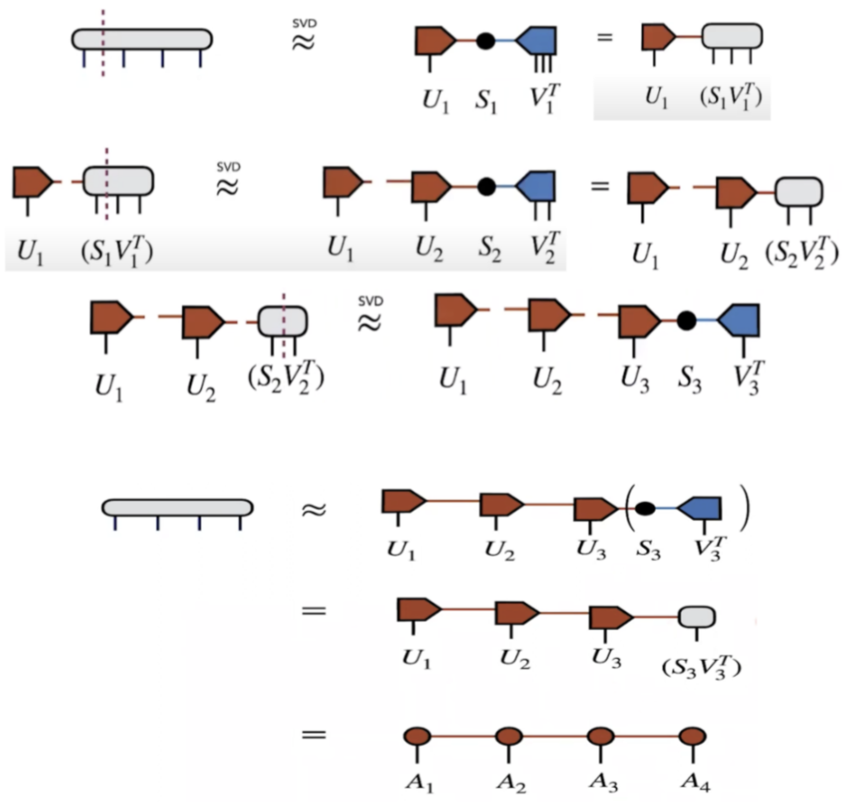
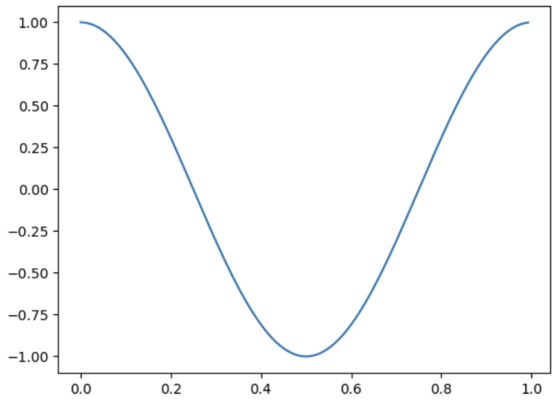


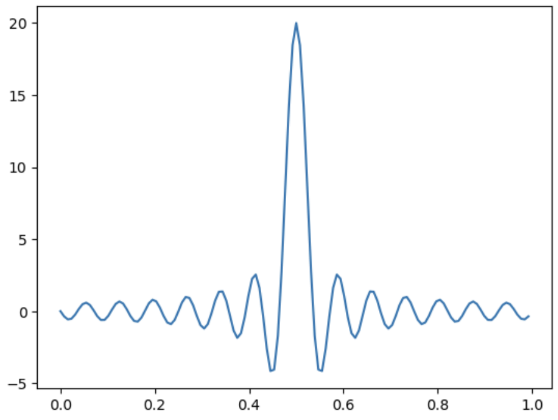
Figure 5: MPS from a 4-order tensor

Encoding of Functions as MPS

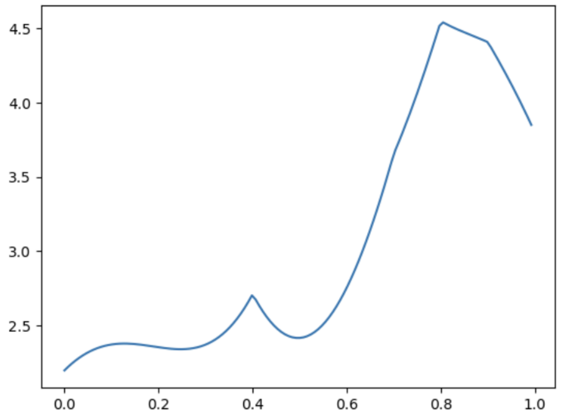
For the purpose of this lab, we tried to reproduce the results from the paper [5]. The authors have performed QFT using tensor networks for three different functions namely *cosine function*, *20 cosine function*, and *cosine + cusp function*. The *cosine function* is the trigonometric cosine function with frequency 2π. The *20 cosine function* is a sum of 20 cosine functions with different frequencies. The *cosine + cusp function* is a sum of the cosine function and 4 exponential terms which forms a *cusp*-like shape, hence the name *cosine + cusp function*.



(a) Cosine function



(b) 20 Cosine function



(c) Cosine + cusp function

Figure 6: Three functions used for encoding the data

For the data to be given to MPS, the data is first **bitwise encoded** which is done by creating a finely spaced grid of spacing $1/2^n$ where n is the number of qubits. Then, these data points are passed to the corresponding function (*cosine function*, *20 cosine function*, and *cosine + cusp function*), and the final tensor is obtained which serves as an input to the **Discrete Fourier Transform (DFT)**. Now, iterative svd or higher-order singular value decomposition is performed on the n-dimensional tensor to obtain the final MPS to be applied to QFT-MPO. Refer to Ryan's section to find QFT-MPO (Quantum Fourier Transform- Matrix Product Operator). In quantum physics and quantum computing, this is essentially equivalent to **Amplitude encoding**. This is an efficient way of representing quantum states when the entanglement is moderate or low [6].

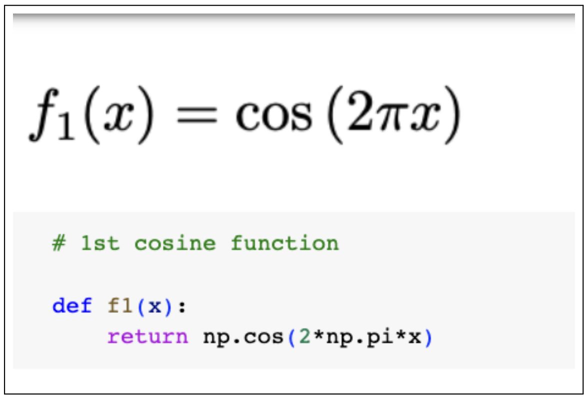
Code and Results

The code for the MPS can be found on my GitHub `https://github.com/PAYAL980/Matrix-Product-States-Preparation-`. All of the code is written in **Python** and **Quimb** [7] **'quantum information many-body'** which is a fast Python library mainly for tensor networks. The repository contains the code for preparing the Matrix Product State and benchmarking it with classical state prep.

The MPSs have been prepared for all three functions (*cosine function*, *20 cosine function*, and *cosine + cusp function*). The **cutoff** of 10^{-12} is used for svd truncation which is same as the paper and the maximum bond dimensions have been also matched with the paper, $\chi = 2$ for *cosine function* and $\chi = 10$ for *20 cosine function* and *cosine + cusp function* where χ is the maximum bond dimension.

The paper uses the process of `rsvd` (Randomized Singular Value Decomposition) for decomposition. However, we get better results for `svd` which calls the **LAPACK** routine **`_gesdd`**. The MPS is created using the Quimb library's `MPS` function which performs iterative svd on the given tensor and outputs the final matrix product state after specifying the correct inputs. The number of SVDs that need to be performed to obtain the MPS of a N-dimensional tensor is equal to N.

The correctness of the MPSs has been checked by contracting the MPS and obtaining the initial high-dimensional tensor.

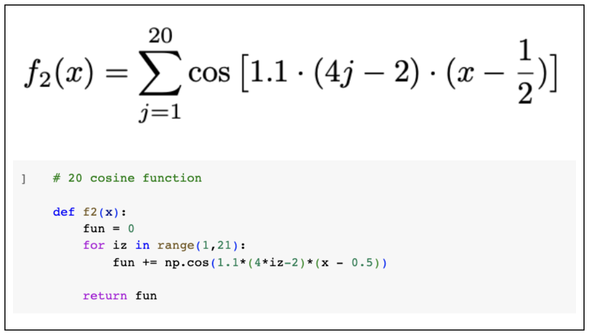


$$f_1(x) = \cos(2\pi x)$$

```
# 1st cosine function

def f1(x):
    return np.cos(2*np.pi*x)
```

(a) Cosine function



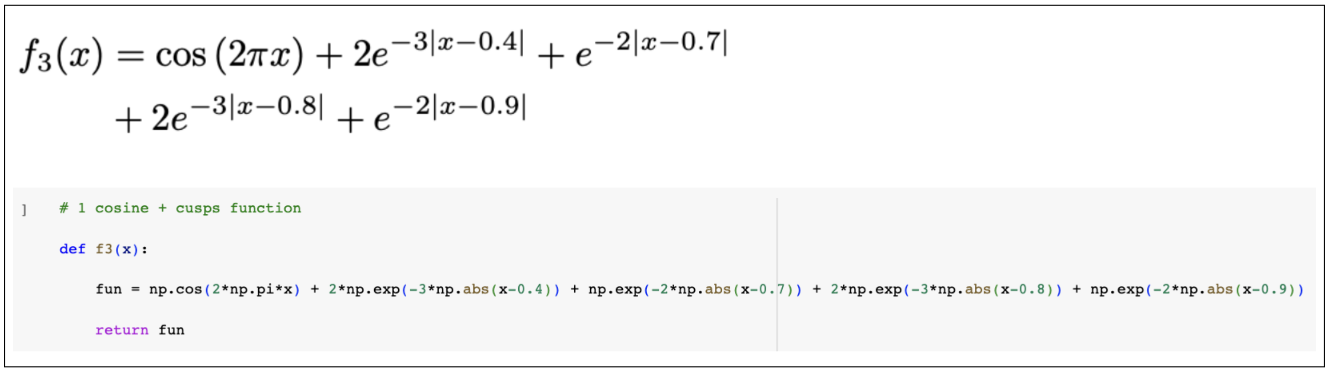
$$f_2(x) = \sum_{j=1}^{20} \cos\left[1.1 \cdot (4j-2) \cdot (x - \frac{1}{2})\right]$$

```
]   # 20 cosine function

    def f2(x):
        fun = 0
        for iz in range(1,21):
            fun += np.cos(1.1*(4*iz-2)*(x - 0.5))

        return fun
```

(b) 20 Cosine function



$$f_3(x) = \cos(2\pi x) + 2e^{-3|x-0.4|} + e^{-2|x-0.7|} + 2e^{-3|x-0.8|} + e^{-2|x-0.9|}$$

```
]   # 1 cosine + cusps function

    def f3(x):

        fun = np.cos(2*np.pi*x) + 2*np.exp(-3*np.abs(x-0.4)) + np.exp(-2*np.abs(x-0.7)) + 2*np.exp(-3*np.abs(x-0.8)) + np.exp(-2*np.abs(x-0.9))

        return fun
```

(c) Cosine + cusp function

Figure 7: Code snippets for the three functions

Fig.8 shows a Matrix Product State for 6 qubits prepared using Quimb (Quantum Information library.

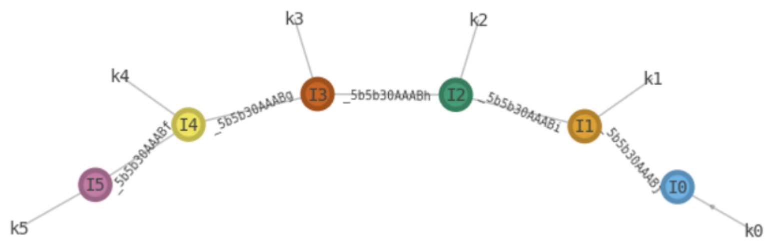


Figure 8: MPS of 6 qubits

The time taken for classical state preparation is shown in Fig.9. As evident from the graph, for 26 qubits, it took almost 140 seconds for state preparation for classical fft whereas as visible in Fig.10, time for creating an MPS of 26 qubits for cosine functions just took 5 seconds.

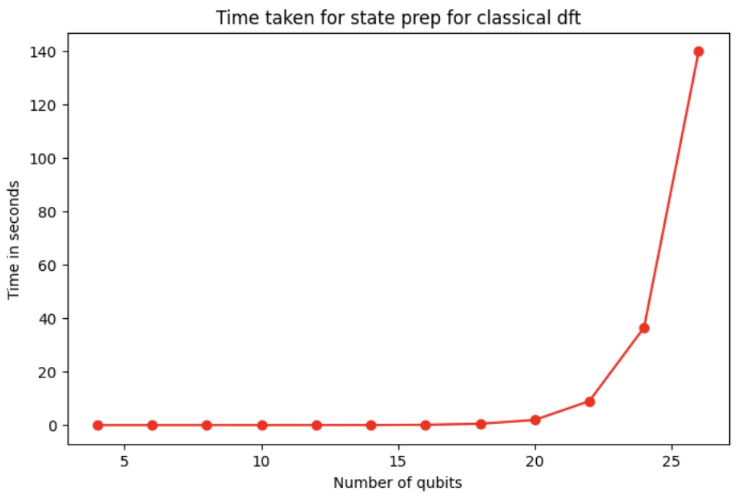
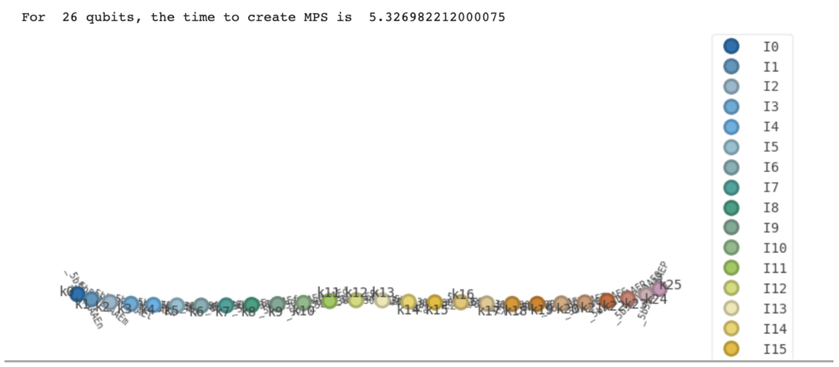


Figure 9: Classical state preparation time vs number of qubits



```
For  26 qubits, the time to create MPS is  5.326982212000075
```

Figure 10: MPS of 26 qubits for cosine function

Fig.11 and Fig.12 contain the Time vs Number of qubits graphs for using **rsvd** and **svd** as the method of truncation. As clearly evident from the graphs, svd gave better results than rsvd. The crossover point for the rsvd approach is at 14 qubits whereas for svd it comes as quickly as at 10. The time for preparing MPSs also varies with the functions and totally dependent on the type of function. Nonetheless, the time for MPSs for all functions still remains less than the non-MPS preparation.

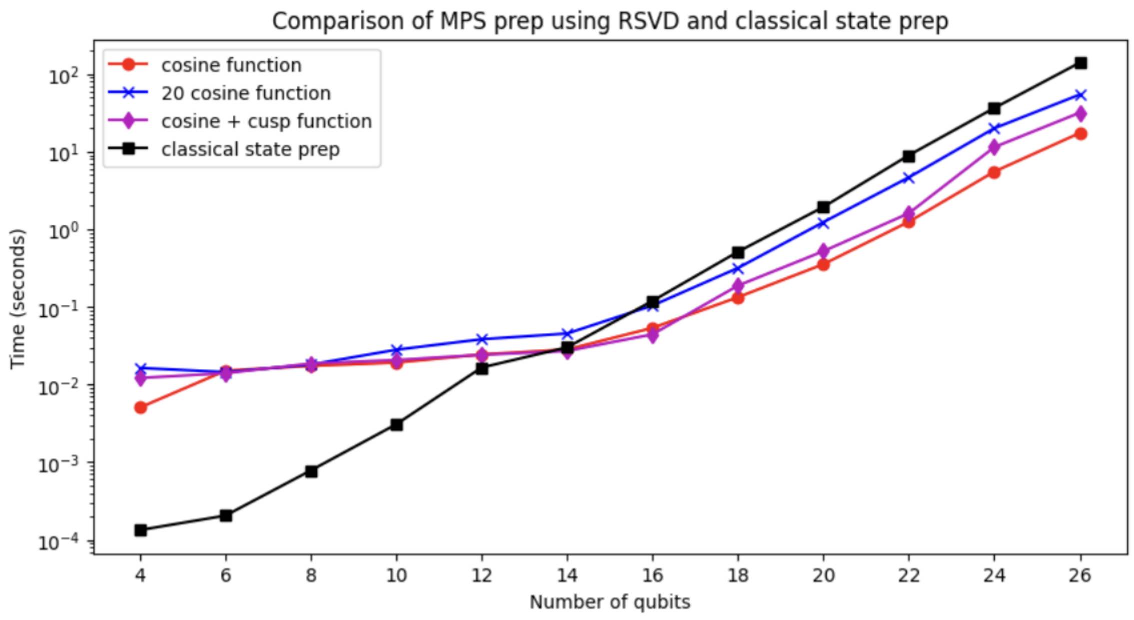


Figure 11: Comparison of MPS preparation using **RSVD** and classical state preparation

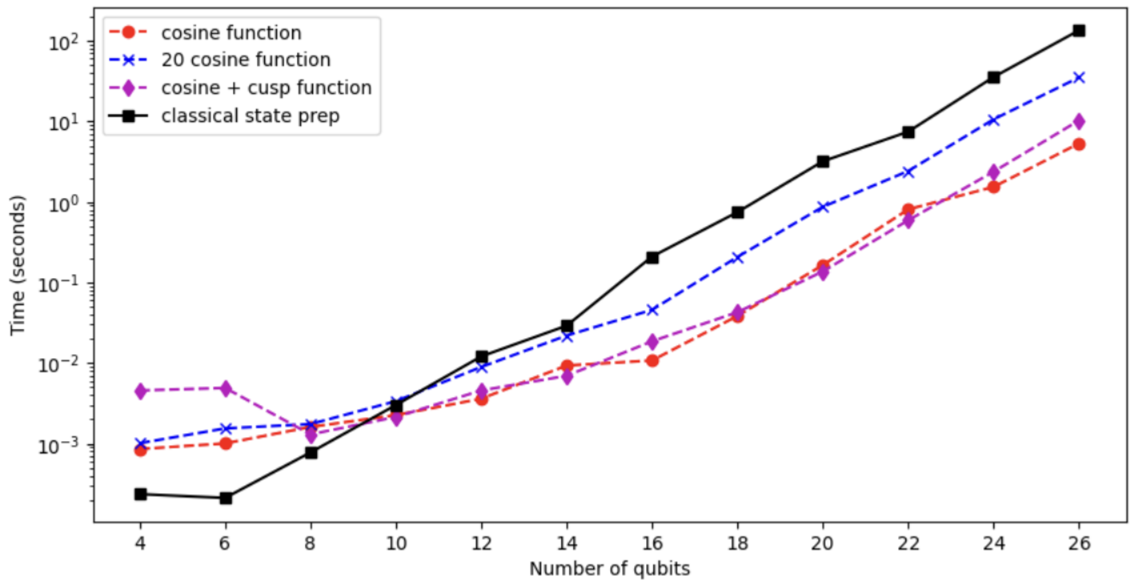


Figure 12: Comparison of MPS preparation using **SVD** and classical state preparation

Conclusion

We were able to create the correct Matrix Product States and generalize it to N qubits. We have tested the correctness of the Matrix Product States up to $N = 26$. After trying various methods of decomposition, we found that 'svd' gives the best results. The prepared Matrix Product States have been successfully applied to the QFT-MPO and obtained correct results. We also observed speedup from the classical state preparation method, around 10 qubits. We gained a strong understanding of Tensor Networks and got the ability to code them for Quantum circuits. We also learned different tensor networks algorithms like DMRG and new libraries like Quimb, Tensor Network, and iTensor.

Future

We have successfully reproduced the results from the paper, however, we still do not see the speedup as expected. In the future, we can try other different ways of decomposition to get more speedup. We are also not sure about the time checkpoints that authors have considered to plot their graphs, we can try asking the authors regarding this. Another thing to try out is to use the DMRG algorithm to apply MPS to QFT-MPO. The tensor networks in general have many interesting applications in Quantum chemistry, quantum machine learning which are worth giving a try.

References

[1] Basic introduction to matrix product states. `https://tensornetwork.readthedocs.io/en/latest/basic_mps.html`.

[2] Jacob Biamonte and Ville Bergholm. Tensor networks in a nutshell. *arXiv preprint arXiv:1708.00006*, 2017.

[3] Matrix product state / tensor train. `https://tensornetwork.org/mps/#Oseledets:2011_5`.

[4] Frank L Hitchcock. The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189, 1927.

[5] Jielun Chen, E. M. Stoudenmire, and Steven R. White. The quantum fourier transform has small entanglement, 2022.

[6] Functions of continuous variables (quantum inspired). `https://tensornetwork.org/functions/`.

[7] Johnnie Gray. quimb: a python library for quantum information and many-body calculations. *Journal of Open Source Software*, 3(29):819, 2018.