Final Documentation:
Matrix Product States for Probability Dynamics

Peter Gjeltema (gjeltema@princeton.edu)
Peiqi Wang (peiqiw@princeton.edu)
Jun Xiong (xiong@princeton.edu)
Bin Xu (binx@princeton.edu)
Liangsheng Zhang(liangshe@princeton.edu)

January 15, 2015

1 Introduction

The matrix product state (MPS) is a novel numerical algorithm that is widely used in quantum many-body physics. For a problem of quantum spin chains, the states live in a Hilbert space expanded by the states of each spin (referred to as "orbitals" hereafter), and the dimension of that Hilbert space is m^N for a chain of length N where each orbital has m choices. The dimension of the state space grows exponentially with N, making it extremely hard to simulate a system consisting of a few dozen sites.

In 1995, White [1] proposed the density matrix renormalization group (DMRG), which proves to be an extremely efficient algorithm for one dimensional systems. Researchers of the next decade were aware of the ubiquitous relation between entanglement and dimensionality allowing the state to be approximated as the product of matrices [2]. Some recent developments include extension of the algorithm to higher dimensions[3] and to critical systems[4].

This inter-departmental collaborative work applies this algorithm to other interesting problems. The algorithm's inherent probabilistic nature motivates its application to other difficult problems in probability theory and stochastic processes. An especially interesting example is the Markov process consisting of many degrees of freedom, which also forms an exponentially growing state space.

We have developed a generic MPS solver and compare results from some simple applications with that of exact transition matrix solutions. We start with a simple model of the dynamics of human relations ("Angry Boys" model), and move on to include other more sophisticated models. Due to the time limitation, this project focuses on one dimensional models for which the algorithm is well-established in physics. Higher dimensionality or highly correlated relationships offering the most realistic relevance will be left to future work.

2 Background

2.1 The difficult many body problem

When we have a state as a vector in the Hilbert space, it is expressed as

$$|\psi\rangle = \sum_{\{\sigma_i\}} A_{\{\sigma_i\}} |\sigma_1\sigma_2\sigma_3\cdots\sigma_N\rangle$$

where $|\sigma_1\sigma_2\sigma_3\cdots\sigma_N\rangle$ is a product basis of the local Hilbert space on different sites.

Applying a linear time-evolution operator $\hat{O}$ on this state, the general form of $\hat{O}$ is

$$\hat{O} = \sum_{\{\sigma\},\{\sigma'\}} |\sigma'_1\sigma'_2\sigma'_3\cdots\sigma'_N\rangle\langle\sigma_1\sigma_2\sigma_3\cdots\sigma_N| O_{\{\sigma'\}\{\sigma\}}$$

where $\langle\sigma_1\sigma_2\sigma_3\cdots\sigma_N|$ is a vector in the dual Hilbert space and $O_{\{\sigma'\}\{\sigma\}}$ is a complex number that represents the amplitude of transition. The application of this operator on the state is naturally written as

$$\hat{O}|\psi\rangle = \sum_{\{\sigma\},\{\sigma'\}} |\sigma'_1\sigma'_2\sigma'_3\cdots\sigma'_N\rangle\langle\sigma_1\sigma_2\sigma_3\cdots\sigma_N|\ O_{\{\sigma'\}\{\sigma\}} \sum_{\{\sigma''_i\}} A_{\{\sigma''_i\}}|\sigma''_1\sigma''_2\sigma''_3\cdots\sigma''_N\rangle$$

Using the orthonormal condition of basis, it can be simplified to

$$\hat{O}|\psi\rangle = \sum_{\{\sigma_i\}\{\sigma'_i\}} A_{\{\sigma_i\}} O_{\{\sigma'\}\{\sigma\}}|\sigma'_1\sigma'_2\sigma'_3\cdots\sigma'_N\rangle$$

Despite its simple form, the actual computation is very difficult since the summation is conducted over N indices of σ_i simultaneously. Therefore, m^N terms are computed. For the simplest case when $m = 2$ (only two elementary states for one site), $m^N \approx 10^9$ for $N = 30$, taking more than 10 seconds for one single contraction. The chain of 40 sites is obviously intractable.

2.2 Matrix Product States (MPS)

The usual way to specify a general state vector is by enumerating its amplitudes for each individual product basis state vector; for example, representing $|\psi\rangle$ as the set $\{A_{\{\sigma_i\}}\}$ for all possible configurations of $\{\sigma_i\}$. Particularly, if each σ_i takes values from the same set $\mathcal{S}$, which is usually the case considered, we can think of each state vector as a function

$$|\psi\rangle : \mathcal{S}^{\otimes N} \to \mathbb{K}$$

where $\mathbb{K}$ is the range of all amplitudes. In this situation, the function is represented as a table of values. On the other hand, we can also represent a function as an analytic expression like Taylor expansion or by some algebraic relations. Such a representation is exactly what matrix product state exploits. In this case, (as the name suggests) the function is represented by a product of matrices. Specifically, we have

$$\begin{aligned} |\psi\rangle &: \mathcal{S}^{\otimes N} \to \mathbb{K} \\ s\sigma_1\sigma_2\cdots\sigma_N &\to M_1^{\sigma_1} M_2^{\sigma_2}\cdots M_N^{\sigma_N} \end{aligned}$$

where $M_i^{\sigma_i}$ is the matrix at site i with state indexed by σ_i. It means, for any given values of σ_1, $\sigma_2\cdots\sigma_N$,

$$A_{\{\sigma_i\}} = M_1^{\sigma_1} M_2^{\sigma_2}\cdots M_N^{\sigma_N}$$

Thus, N matrices are used instead of specifying $|\mathcal{S}|^N$ values $\{A_{\{\sigma_i\}}\}$ for a state vector that may be of different ranks at different sites.

To go from the tabular specification of a state vector to its matrix product state representation, the technique of Singular Value Decomposition (SVD) can be exploited. Let's focus on the coefficient $A_{\{\sigma_i\}} = A_{\sigma_1\sigma_2\sigma_3\cdots\sigma_N}$. From a direct point of view, it is a rank-N tensor which has N indices. We can group some indices so that it becomes $A_{\sigma_1(\sigma_2\sigma_3\cdots\sigma_N)}$ with only 2 indices, where the first has $m(= |\mathcal{S}|)$ possible values and the second has m^{N-1}. Grouping $N-1$ indices means making a bijection map from $N-1$ numbers, each varying between 1 and m to a single integer varying from 1 to m^{N-1}. This re-indicing trick is crucial to MPS algorithms.

We then treat this bi-index object $A_{\sigma_1(\sigma_2\sigma_3\cdots\sigma_N)}$ as a matrix with σ_1 (row index) and $(\sigma_2\sigma_3\cdots\sigma_N)$ (column index), thus forming a $m \times m^{N-1}$ matrix. Applying SVD on this matrix gives

$$A_{\sigma_1(\sigma_2\sigma_3\cdots\sigma_N)} = U^{\sigma_1}_{m_1} S_{m_1m_2} V_{m_2(\sigma_2\sigma_3\cdots\sigma_N)} = U^{\sigma_1}_{m_1} \tilde{V}_{m_1(\sigma_2\sigma_3\cdots\sigma_N)}$$

where U is a $m \times m$ unitary matrix and S is a $m \times m$ diagonal matrix with non-negative diagonal elements arranged in descending order. We apply a similar procedure on $\tilde{V}_{m_1(\sigma_2\sigma_3\cdots\sigma_N)}$ by re-arranging indices as $\tilde{V}_{m_1(\sigma_2\sigma_3\cdots\sigma_N)} = \tilde{V}_{(m_1\sigma_2)(\sigma_3\cdots\sigma_N)}$ and applying a similar SVD:

$$\tilde{V}_{(m_1\sigma_2)(\sigma_3\cdots\sigma_N)} = U_{(m_1\sigma_2)m_2} S_{m_2m'_2} V_{m'_2(\sigma_3\cdots\sigma_N)} = U^{\sigma_2}_{m_1m_2} \tilde{V}_{m_2(\sigma_3\cdots\sigma_N)}$$

where the second "=" reshapes the U matrix and multiplies the S and V matrices.

Noticing that $\tilde{V}_{m_2(\sigma_3\cdots\sigma_N)}$ is almost the same as $\tilde{V}_{m_1(\sigma_2\sigma_3\cdots\sigma_N)}$, with the exception of having one fewer σ_i, we apply this procedure inductively to write the original coefficient tensor as

$$A_{\{\sigma_i\}} = U^{\sigma_1}_{m_1} U^{\sigma_2}_{m_1 m_2} U^{\sigma_3}_{m_2 m_3} \cdots U^{\sigma_N}_{m_{N-1} m_N}$$

This resulting form provides the matrix product state. The σ_i's are called "physical indices', and m_i's are called "auxiliary indices". It is worth noting that in general, operators can also be decomposed in this way - although it is usually more convenient to directly construct operators in the so-called matrix product form which can then directly act on matrix product states. This representation is extremely convenient because each matrix U contains only one physical index, while the application of a matrix product linear operator on a matrix product state only involves the contraction of a few indices.

To illustrate, if we have a linear operator $O_{\{\sigma'\}\{\sigma\}} = O^{\sigma'_1\sigma_1}_{n_1} O^{\sigma'_2\sigma_2}_{n_1 n_2} O^{\sigma'_3\sigma_3}_{n_2 n_3} \cdots O^{\sigma'_N\sigma_N}_{n_{N-1} n_N}$ then applying this operator on the MPS is actually the contraction on m, n, and σ_i space. The traditional methodology corresponds to first contracting m and n spaces, and then σ. However, this novel method contracts the σ space first and then m and n. The specific order of contraction is:

- Compute $\sum_{\sigma_i} O^{\sigma'_i\sigma_i}_{n_{i-1} n_i} A^{\sigma_i}_{m_{i-1} m_i} = M^{\sigma'_i}_{n_{i-1} n_i m_{i-1} m_i}$ for all i's
- $\sum_{n_1 m_1} M^{\sigma'_1}_{n_1 m_1} M^{\sigma'_2}_{n_1 n_2 m_1 m_2} = \tilde{M}^{\sigma'_1\sigma'_2}_{n_2 m_2}$
- $\sum_{n_2 m_2} \tilde{M}^{\sigma'_1\sigma'_2}_{n_2 m_2} M^{\sigma'_3}_{n_2 n_3 m_2 m_3} = \tilde{M}^{\sigma'_1\sigma'_2\sigma'_3}_{n_3 m_3}$
- we keep doing this, until we finish the multiplication of all these objects and get $\tilde{M}^{\sigma'_1\sigma'_2\cdots\sigma'_N} = A_{\sigma'_1\sigma'_2\cdots\sigma'_N}$ This is the final result - coefficients of the new state $|\psi'\rangle = \sum_{\{\sigma'_i\}} A_{\sigma'_1\sigma'_2\cdots\sigma'_N} |\sigma_1\sigma_2\cdots\sigma_N\rangle$.

2.3 Canonization

Canonization is an important operation on the matrix product state. As we shall see in the following, the operations of compression heavily relies on the canonical representation of a MPS. Consider a MPS in the following form

$$|\psi\rangle = \sum_{\sigma} A^{\sigma_1} A^{\sigma_2} \ldots A^{\sigma_{i-1}} \Psi B^{\sigma_{i+1}} \ldots B^{\sigma_{L-1}} B^{\sigma_L} |\sigma\rangle$$

We call that ψ a mixed-canonized MPS if the matices A^{σ_i} are left-normalized and the matices B^{σ_i} are right-normalized, i.e.

$$\sum_{\sigma_i} A^{\sigma_i,*} A^{\sigma_i} = I, \quad \sum_{\sigma_i} B^{\sigma_i} B^{\sigma_i,*} = I$$

where M^* denotes the conjugate transpose of matrix M. Given any MPS $|\psi\rangle$ with matrices M^{σ_i}, the canonization of $|\psi\rangle$ can be done iteratively. The key tool of this process relies on the singular value decomposition. Starting from the first the first physical site σ_1, we can reshape the matrices $M^{\sigma_1}_{a_0,a_1}$ into one matrix $M_{(\sigma_1,a_0),a_1}$. Apply singular value decomposition on this reshaped matrix $M_{(\sigma_1,a_0),a_1} = \sigma_{s_1} U_{(\sigma_1,a_0),s_1} S_{s_1,s_1} V^*_{s_1,a_1}$. where U is a matrix composed of orthogonal column vectors. Now if we reshape $U_{(\sigma_1,a_0),s_1}$ back into a set of matrices $A^{\sigma_1}_{a_0,s_1}$. Then A^{σ_1} will be left-normalized. We then multiply SV to the matrix representing the next physical state M^{σ_2} and repeat the same procedure as before. Obviously, this procedure can be carried out from right to left and that will give the right-normalized matrices. Carrying out the procedure from to directions in the same time will provide us a mix-canonical representation of the MPS. Notice here that the operations mentioned here will not modify the state. It only provides a convenient representation of the state.

2.4 Approximation

It may seem unclear why our algorithm seems to transform a problem scaling exponentially to polynomial time; indeed, it should be. We did not talk about the dimension of auxiliary space m_i and n_i, which should grow exponentially. The method is useful only when the interaction is local or short-range, meaning the degree of freedom on one site effectively interacts only with some of its neighbors. It is proven that in this

case, the diagonal terms in the S matrix decay exponentially, effectively enabling a cut-off in auxiliary space keeping only a finite number of χ dimensions. Thus, all U's are at most $\chi \times \chi$ dimensional matrices. χ is in this case called the bound dimension. The effectiveness of this algorithm relies deeply on the fact that a large χ is unnecessary to achieving accurate results. It is proven that in all 1-dimensional problems away from critical points, χ does not grow with system size (the length); in critical 1-dimensional models, $\chi \propto N^{\lambda}$ (polynomial) and in 2D, $\chi \propto \exp W$ which is the exponential of short dimension.

2.4.1 Compression by SVD

The idea of compression by SVD comes from the procedure of the canonization. When performing the left-canonization on the matrix of a certain physical site, upon obtaining the singular value decomposition, we may simply truncate the diagonal matrix S by throwing away the small eigen values. By truncating the diagonal matrix S, we also truncate accordingly U and V, therefore we can reduce the size of the MPS. Note that this procedure can be carried out from two different directions and yields different compression results. In addition, the error of compression will accumulate as we go on with the iterations. Because of this, the results might be far from optimal.

2.4.2 Compression by variation

Denote $|\psi\rangle = \sum_{\sigma} M^{\sigma_1} \dots M^{\sigma_L}$ the MPS to be compressed. The method of compression by variation consists of finding a new MPS $|\bar{\psi}\rangle = \sum_{\sigma} \bar{M}^{\sigma_1} \dots \bar{M}^{\sigma_L}$ where each $\bar{M}^{\sigma_i}$ of desired size, such that the L^2- distance between $|\bar{\psi}\rangle$ and $|\psi\rangle$ is minimized. Formally:

$$|\bar{\psi}\rangle = \arg \min_{|\phi\rangle = \sum_{\sigma} \bar{M}^{\sigma_1} \dots \bar{M}^{\sigma_L}, M^{\sigma_L} \in \mathbb{R}^{d \times d}} \| |\phi\rangle - |\psi\rangle \|^2$$

This is a rather complicated optimization problem. One heuristic to tackle it involves iterative optimization. We start with an initial guess of the minimizer. We keep all other matrices fixed and minimize $\| |\phi\rangle - |\bar{\phi}\rangle \|^2$ with respect to the matrix of the first physical site $\bar{M}^{\sigma_1}$. Upon obtaining the minimizer we update $|\bar{\phi}\rangle$, move to the next site and repeat this procedure. We do this until we reach the last physical site. This is called a sweep. We then repeat the sweep through the chain and check the L^2- distance between original and compressed MPS after each sweep. We stop the process when L^2- distance stops to decrease with respect to a user-defined level of tolerance.

The key in this algorithm is to minimize $\| |\phi\rangle - |\bar{\phi}\rangle \|^2$ with respect to the matrix of a given physical site. This can be done by exploiting the first-order condition. Let us denote M^* as the conjugate transpose of matrix M. Then at the optimum, for each σ_i (physical states), a_i and a_{i-1} (auxiliary states), we must have

$$0 = \frac{\partial}{\partial M^{\sigma_i,*}_{a_i,a_{i-1}}} \| |\phi\rangle - |\bar{\phi}\rangle \|^2 = \frac{\partial}{\partial M^{\sigma_i,*}_{a_i,a_{i-1}}} \left(-\langle \bar{\psi} | \psi \rangle + \langle \bar{\psi} | \bar{\psi} \rangle \right)$$

The last equality is due to the fact that $M^{\sigma_i,*}_{a_i,a_{i-1}}$ only shows up in $\langle \bar{\psi}|$. Further expansion of the equation provides that

$$\begin{aligned} 0 = \quad & \frac{\partial}{\partial M^{\sigma_i,*}_{a_i,a_{i-1}}} \left(-\langle \bar{\psi} | \psi \rangle + \langle \bar{\psi} | \bar{\psi} \rangle \right) \\ = \quad & \sum_{\sigma \setminus \sigma_i} (\bar{M}^{\sigma_1,*} \dots \bar{M}^{\sigma_{i-1},*})_{1,a_{i-1}} (\bar{M}^{\sigma_{i+1},*} \dots \bar{M}^{\sigma_L,*})_{a_i,1} \bar{M}^{\sigma_1} \dots \bar{M}^{\sigma_i} \dots \bar{M}^{\sigma_L} \\ & - \sum_{\sigma \setminus \sigma_i} (\bar{M}^{\sigma_1,*} \dots \bar{M}^{\sigma_{i-1},*})_{1,a_{i-1}} (\bar{M}^{\sigma_{i+1},*} \dots \bar{M}^{\sigma_L,*})_{a_i,1} M^{\sigma_1} \dots M^{\sigma_i} \dots M^{\sigma_L} \end{aligned}$$

Here $\sigma \setminus \sigma_i$ means the sum takes over all the physical sites except the $i-$th physical site. The above can be further simplified if we assume that $\bar{M}$ is under mixed-canonical form, i.e. $\sum_{\sigma_i} M^{\sigma_j,*} M^{\sigma_i} = Id, \forall j = 1 \dots i-1$ and $\sum_{\sigma_i} M^{\sigma_j} M^{\sigma_i,*} = Id, \forall j = i+1 \dots L$. Under this assumption, the first term on the right hand

side simplifies to $\bar{M}^{\sigma_i}_{a_{i-1},a_1}$, which is exactly the coefficient we are looking for. Then the equation can be written as

$$\bar{M}^{\sigma_i}_{a_{i-1},a_1} = \sum_{a'_{i-1}a'_i} L_{a_{i-1},a'_{i-1}} M^{\sigma_i}_{a'_{i-1},a'_i} R_{a_i,a'_i}$$

where L is the overlap of M and $\bar{M}$ from left to the $(i-1)$th state, and R is the overlap of M and $\bar{M}$ from right to the $(i+1)$th state. We call objects like L and R respectively the left partial overlap and the right partial overlap. Obviously from what has been described about the algorithm, the partial overlaps need to be used and updated during the sweeps. When implementing the algorithm, we store them in the lists `self.partial_overlap_lr` and `self.partial_overlap_rl`. Note that the last element of `self.partial_overlap_lr` and the first element of `self.partial_overlap_lr` are actually the full overlap of the compressed state and the original state. This is useful in computing the L^2-distance and in deciding when to stop the sweep.

Attention must also be paid to the normalization. Indeed, all MPS studied here represent a probability distribution. Thus, summing over all the physical states must yield unity. However, this can no longer be guaranteed after the procedure of compression . Therefore, the result of compression must be normalized by its L^1-norm.

2.5 Matrix Product Operators (MPO)

The Hamiltonian in quantum mechanics, or equivalently the transitional matrix in stochastic processes, can be written as a series of matrix product operators where each matrix only applies on one site. We considered four different models to implement our MPS algorithm.

2.5.1 AngryBoys Model

Consider boys waiting in a line. Each boy's state can be defined as either an angry one, or a happy one. If we desire to model the individual states of the entire line of boys over time, we can implement a model dependent upon a transitional matrix method given by:

$$H = pI + (1-p)\sum_{i=1}^{n-1} \frac{1}{n-1} \sigma_i^x \otimes \sigma_{i+1}^x,$$

where I is a $2^n \times 2^n$ identity matrix and

$$\sigma_i^x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

as described in Model. If he is angry, he has a probability p of remaining angry, but also a probability p of becoming happy. This model also intends a boy to be influenced with equal probability by its left and right neighbors in larger chains. Two emotional boys have probability p that they will change their state together - otherwise, they remain in their respective states. In the case of N emotional boys, a pair of nearest neighbors will change states together with probability p, and otherwise remain in their respective states.

The MPO of this model is

$$O = W^{[1]}W^{[2]}...W^{[L]}$$

And can be encoded by the operator valued matrices below:

$$W^i = \begin{pmatrix} I & 0 & 0 \\ S_x & 0 & 0 \\ 0 & qS_x & I \end{pmatrix}$$

where $W^1 = \begin{pmatrix} pI & qS_x & I \end{pmatrix}$ on the first site of the chain, and

$$W^L = \begin{pmatrix} I \\ S_x \\ 0 \end{pmatrix}$$

on the last site.

2.5.2 RadiatingBoys Model

Consider now a situation where the boys are impacted by more than just their nearest neighbor. Again returning to a transitional matrix method, we can implement another model given by:

$$H = p_0 I + p_1 \sum_{i=1}^{n-1} \frac{1}{n-1} \sigma_i^x \otimes \sigma_{i+1}^x + p_2 \sum_{i=1}^{n-2} \frac{1}{n-2} \sigma_i^x \otimes \sigma_{i+2}^x$$

This model is similar to the AngryBoys model, but now depends additionally on a second-nearest neighbor interaction. In the case of N emotional boys, a pair of nearest neighbors and a pair of second-nearest neighbors influence each other, but at different rates. Sites adjacent to each other will change together with probability p_1, and otherwise remain in their respective states. Sites separated by a single site can also now change together with probability p_2, and otherwise remain in their respective states.

The MPO can be decomposed into the following matrices:

$$W^i = \begin{pmatrix} I & 0 & 0 & 0 \\ S_x & 0 & 0 & 0 \\ 0 & I & 0 & 0 \\ 0 & p_1 S_x & p_2 S_x & I \end{pmatrix}$$

$$W^1 = \begin{pmatrix} p_0 I & p_1 S_x & p_2 S_x & I \end{pmatrix},$$

and

$$W^L = \begin{pmatrix} I \\ S_x \\ 0 \\ 0 \end{pmatrix}$$

2.5.3 Exponentialboys Model

Consider again a line of boys; however, this time the state of each boy is a result of interactions with every single other boy in line. Such a model can be represented with a transition matrix as:

$$H = P(I + J \sum_{i=1}^{n-1} \sum_{j=i+1}^{n-1} K^{j-i} \sigma_i^x \otimes \sigma_j^x)$$

Each site in this model is in turn influenced by an interaction with the remaining sites in the chain. A single site can change its state along with any other site, but this occurrence has a probability p which decays exponentially as the two sites increase in distance from one-another.

The MPO of this model comprises of matrices:

$$W^i = \begin{pmatrix} I & 0 & 0 \\ S_x & KI & 0 \\ 0 & JKS_x & I \end{pmatrix},$$

$$W^1 = \begin{pmatrix} p_0 I & JKS_x & I \end{pmatrix},$$

and

$$W^L = \begin{pmatrix} I \\ S_x \\ 0 \end{pmatrix}$$

.

2.5.4 Projectionboys Model

Finally, consider again the first AngryBoys model, but subject to random events. These random events constitute an automatic transition of a sites state to either happy or angry, independent of its previous state. Modifying the original transition matrix, we create a new model described by:

$$H = p_0 I + \sum_{i=1}^{n-1} \left(\frac{p_1}{n-1} \sigma_i^x \otimes \sigma_{i+1}^x + \frac{q_1}{n-1} \pi_i^+ \otimes \pi_{i+1}^- + \frac{q_2}{n-1} \pi_i^+ \otimes \pi_{i+1}^- \right)$$

As before, a pair of sites in a chain of size N can change state with probability p. However, π matrices given by:

$$\pi^+ = \begin{pmatrix} 1 & 1 \\ 0 & 0 \end{pmatrix}$$

and:

$$\pi^- = \begin{pmatrix} 0 & 0 \\ 1 & 1 \end{pmatrix}$$

represent events occurring with probability q_1 and q_2 that the state of two partner sites will be flipped to either happy or angry respectively, regardless of any previous states.

The components of the MPO are:

$$W^i = \begin{pmatrix} I & 0 & 0 & 0 & 0 \\ S_x & 0 & 0 & 0 & 0 \\ \pi_i^+ & 0 & 0 & 0 & 0 \\ \pi_i^- & 0 & 0 & 0 & 0 \\ 0 & p_1 S_x & q_1 \pi_i^+ & q_2 \pi_i^- & I \end{pmatrix},$$

$$W^1 = \begin{pmatrix} p_0 I & p_1 S_x & q_1 \pi_i^+ & q_2 \pi_i^- & I \end{pmatrix}$$

and

$$W^L = \begin{pmatrix} I \\ S_x \\ \pi_i^+ \\ \pi_i^- \\ 0 \end{pmatrix}$$

.

3 Program Overview

In this project, we extend the usage of MPS and MPO to problems concerning stochastic processes. Particularly, we consider the time evolution of a probability distribution expressed in a vector form, where the dynamics are specified by a probability transition matrix.

The implementation is summarized in Fig. 1. The model is implemented in multiple ways. The classes used are described in Fig. 2.

3.1 `Model` Class

We defined an abstract model class in model.py which contains several frequently used components in the MPO. Each specific model is a derived class from Model class and they contain a string description of the model, model parameters, and a callable method to build the MPO/MPS representation and the transitional matrix for the user.

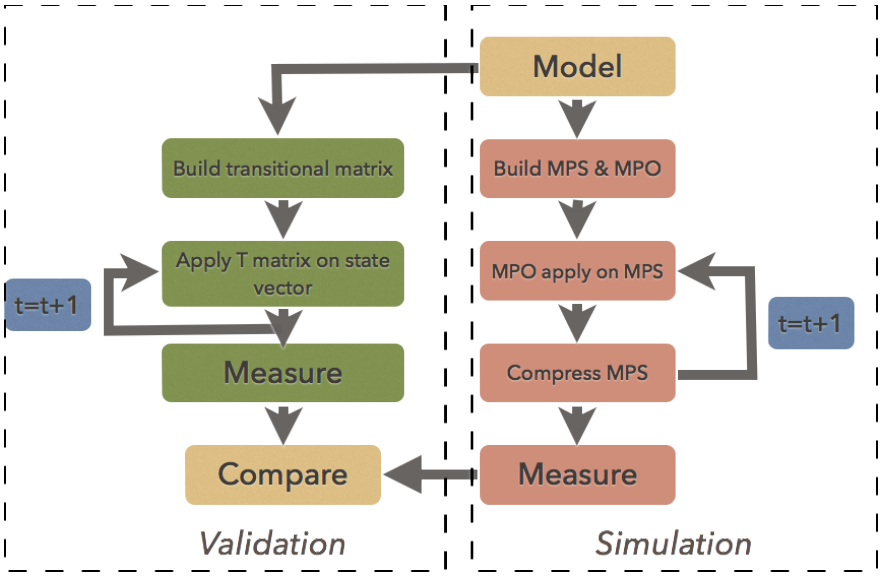


Figure 1: The flow chart of our program.

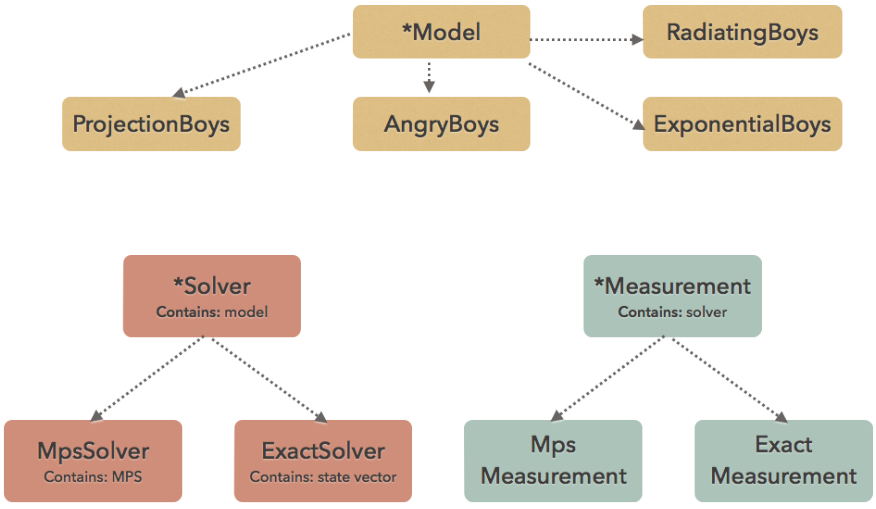


Figure 2: A bird view of our classes.

3.2 `ExactSolver` Class

3.2.1 Solving the Problem in Theory

The many body problem is treated one dimensionally, with n agents on a line at positions $i = 1, 2, \cdots, n$. Each body can occupy one of two states $(1, 0)^T$ and $(0, 1)^T$, corresponding to a state space of size 2^n. The dynamics of the system at each time step can be represented by the following probability transition matrix:

$$H = pI + (1-p)\sum_{i=1}^{n-1} \frac{1}{n-1}\sigma_i^x \otimes \sigma_{i+1}^x,$$

where I is a $2^n \times 2^n$ identity matrix and

$$\sigma_i^x = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

operates on the state space of position i. Qualitatively, the transition matrix models the system by capturing interactions between neighboring bodies. A body's state remains constant with probability p, and flips its state with probability $(1-p)/(N-1)$. Consider the *AngryBoys* model as an example: the system is an abstraction of boys standing in a line, and their mood is in a state of either "anger" $(1, 0)^T$ or "tranquility" $(0, 1)^T$. Their respective states are in turn influenced by their left and right neighbors.

3.2.2 Solving the Problem Computationally

Simulating a solution to the many body problem is computationally intense. Construction of the transition matrix at each time step requires a summation over N indices of σ_i simultaneously, meaning m^N terms are examined. For a simply model where a body occupies one of two elemetary states, $m^N \approx 10^9$ for $N = 30$. A simulation of 40 sites is impossible. Nonetheless, an exact solution class `ExactSolver` bounded by reasonably-sized state spaces has been constructed as a corroboration of results obtained through the MPS algorithm.

The `ExactSolver` class takes the `Solver` class as its parent. Its dependencies include the `numpy` library, the deepcopy function of the `copy` library, and a `Boys` model from its respective parent class. Three functions defined in the class - `interpreter`, `step`, and `evolve` - conduct simulation. Data from the simulation is collected in the form of lists. One list saves the state of the model at each time step, and one list saves the sum of probabilities at each step to ensure proper normalization of the model (probability cannot exceed 1).

The `interpreter` function accepts the imported model, reads its information, and initiates the simulation. The model's information consists of the beginning transition matrix and the initial state. Once the transition matrix is set, and the initial state is set and saved, simulation can commence. This represents a change from the alpha version; orginally, `interpreter` itself was used to initiate the transition matrix and initial state. The beta version now offers functions to do this so that models with more diverse properties can interact with the solver. For instance, models reflecting influence from only a left or right neighbor can direct their unique transition matrix to `ExactSolver`. Additional control over initial state conditions is also given.

The `step` function is very simple. It first increments simulation time, and then produces a new state by calculating the tensor dot product of the transition matrix and previous state.

The `evolve` function controls the flow of the simulation by taking as its argument the total number of time steps desired for the simulation. `evolve` calls `step` to produce a new state, and then appends this state and its norm to its respective data list. This process is done iteratively for the number of time steps, producing a time evolution of the model.

3.3 `MpsSolver` Class

Parallel to the class `ExactSolver`, we construct a class named `MpsSolver` to store the MPS representation of the current state, to hold the MPO representation of the dynamics of the system and to comprise several functions necessary to updating the current state.

3.3.1 Member variables

We have the current state saved under a variable named `mps`, which is a 3-d array storing $M^{\sigma_i}_{a_{i-1},a_i}$. The operator $O^{\sigma_i,\sigma_i'}_{n_{i-1},n_i}$ is stored in a variable named `mpo`, which is a 4-d array. After applying the MPO on the MPS, we do the compression and the result is stored in `mpsc`. In addition, we store the time of the system in the variable `t`. These are variables defining the current state of the system. In addition, `MpsSolver` also hosts parameters and auxiliary variables used by overlap, compression and other routines in the class. For example, `bound_dimension` defines the dimension of the compressed MPS.

3.3.2 Methods

The key method in the `MpsSolver` class is `step()` which updates the system from t to $t+1$. The updating process involves three steps: (i) applying MPO on MPS (ii) compressing MPS to the desired size. (iii) normalizing the compressed state. We implement `contraction` to realize step 1, which simply involves some basic operations of multiplication of arrays and summing over a certain number of indices. For the compression step, we implement two methods: `compressionSVD`, and `compressionVariational` from which the user can choose which to use. `normalizeProba` is implemented to normalize the compressed MPS so that it becomes a probability distribution. Finally, we also implement `evolve()` to allow the user to update the system a certain number of times.

3.3.3 Compression by variation

The main routine for compression by variation is `compressionVariational`. It does the following operations:

- `initializeMpscVar`: Form a initial guess of the compressed MPS of desired size. It is then left or right canonized, depending on the direction of the first sweep. L^2-distance with the original MPS is calculated.
- `initializePartialOvl`: Initialize the partial overlap lists. Note that if the first sweep is from left to right, then we only have to initialize the right partial overlaps.
- `compressionSweepLeftRight`: Perform a sweep from left to right. For each site, we calculate the matrix using left and right partial overlap. Before moving to the next site on the right, we have to left-normalize the current site (to preserve the mixed-canonical form) and update the left partial overlap list. At the end of the sweep, we calculate the L^2-distance with the original MPS and store the value in `cpr_err`.
- `compressionSweepRightLeft`: Perform a sweep from right to left. The idea is the same as described above.
- We start the sweep with direction specified by the user and alternate the left and right sweeps until the L^2-distance recorded by `cpr_err` converges.

Throughout the sweeps, we also keep track of the L^1-norm of the MPS. The User can choose whether to normalize the MPS after each sweep or after the completin of all sweeps.

3.4 `Measurement` Class

The `Measurement` class includes functions for calculating joint probability, correlations among multiple sites and mean as well as variance for a particular site at any time point. For the last two calculations, user can define specific values for being angry (up) or calm (down), and by default, they are 1 and -1 respectively. Both time and site number counts from 0, and a python-way of specifying the number from the end of list is also accepted. Each measurement task must specified in the particular way, though measurement time can be omitted, in which case it is taken to be the most recent time computed. Note in each task tuple, the integer following the task name will always be taken as the time step for all measurement. For more details, please consult the user manual or the sample program.

3.4.1 `ExactMeasurement` Class

The measurement in exact calculations is straightforward. Since each site has only 2 distinct states, any definite state of the system can be encoded as a binary number, i.e., in the program, the down state is encoded as 0 and up state is encoded as 1, and the random state of the system at any time is fully specified by giving the probability of all definite states. From these full joint (in terms of each site) distributions, all quantities at a particular time point can be computed from marginalization. Moreover, the average value of given site is a special case of general correlation function, which is an average of a sequence of sites, and from the average value the variance of a given site can be easily computed by using appropriate up and down values.

3.4.2 `MpsMeasurement` Class

The measurement in the MPS case is a little bit more complicated since we are not directly specifying full joint probability distributions as in the exact calculation. However, from the definition of MPS, we can see that for a particular definite state of the system which is encoded as $s_0 s_1 \cdots s_{L-1}$, where $s_i = 0$ or 1 for any i and L is the length of the chain, its probability can be computed from a matrix product

$$P(s_1 s_2 \dots s_{L-1}) = M_0^{s_0} M_1^{s_1} \cdots M_{L-1}^{s_{L-1}}$$

where $M_i^{s_i}$ is the matrix representation of site i being in the state s_i. Since the matrices at first and last site are row and column vectors respectively, in the end we will obtain a number as expected. From these full joint probability distributions, we then can obtain all measurement quantities at a particular time as in the exact calculation. There is also a little trick to speed up the marginalization by using the distributive law in matrix multiplication. For example, if we want to compute the joint distribution of sites $i_1, i_2, \cdots, i_n$ in states $s_1, s_2, \cdots, s_n$, where, for simplicity, assuming $0 < i_1 < i_2 < \cdots < i_n < L-1$, we then have the desired probability to be

$$P(s_{i_1} s_{i_2} \dots s_{i_n}) = (M_0^0 + M_0^1) \cdots M_{i_1}^{s_{i_1}} \cdots M_{i_2}^{s_{i_2}} \cdots M_{i_n}^{s_{i_n}} \cdots (M_{L-1}^0 + M_{L-1}^1)$$

i.e, we add up the matrices representing up and down for all sites except sites $i_1, i_2, \cdots, i_n$, for which a particular matrix is chosen, and then multiply all matrices. Similarly, for a correlation as $\langle S_{i_1} S_{i_2} \cdots S_{i_n} \rangle$, where S_i is the random variable representing the two possible states at site i, we can obtain it from

$$\langle S_{i_1} S_{i_2} \cdots S_{i_n} \rangle = (\sigma_0^0 M_0^0 + \sigma_0^1 M_0^1)(\sigma_1^0 M_1^0 + \sigma_1^1 M_1^1) \cdots (\sigma_{L-1}^0 M_{L-1}^0 + \sigma_{L-1}^1 M_{L-1}^1)$$

where $\sigma_i^0 = \sigma_i^1 = 1$ for all sites except $i_1, i_2, \cdots, i_n$, where $\sigma_i^0 =$ specified value for down state and $\sigma_i^1 =$ specified value for up state.

4 Results

In this section we look at some results obtained from the program. In the following, we will consider the Angry Boys Model with probability $p = 0.1$. The Radiating Boys Model will have $p_0 = 0.1$, $p_1 = 0.4$ and $p_2 = 0.5$. The Exponential Boys Model is equipped with $J = K = 0.5$, and in Projection Boys Model we study the case where $p_0 = 1$ and $p_2 = q_1 = q_2 = 2$. These parameters are chosen so that the models are not so close to the trivial identity time evolution, while no particular non-trivial term is significantly weighed.

The results are shown in Fig. 3, Fig. 4, Fig. 5 and Fig. 6 for each model respectively. The results are in general satisfactory. The subplots (a) from Fig. 3 to Fig. 6 compare the time evolution from the MPS method to the exact method for a small chain of size 10 for all models. The bond dimension is chosen to be 10. The joint probability is between site 6 and the last site, 10, and the mean as well as variance concern the site 6 with the default up and down values. They clearly show that the results from MPS method almost exactly agree with those obtained from exact method, validating our MPS algorithm at least for short chains. Subplots (b) from Fig. 3 to Fig. 6 give a closer look at these short chain results. Plotted in them are the sum of squared errors of the joint probability against the exact results accumulated in 100 time steps. We also experimented with different choices of bound dimensions. It can be easily seen that roughly, the error decays

exponentially with increasing bound dimensions for all models, though some model-dependent fluctuations do exist. This suggests that to get reasonably good results, a relative small bound dimension already suffice, which makes computation highly tractable.

Subplots (c) in all figures then demonstrate a time evolution for a long chain of size 100 with 200 time steps, which by no means can be handled by the exact method given current technology, where again we look at the joint probability between the 6th site and the last site, and the mean as well as variance of the 6th site, and a bound dimension of 10 is chosen. This indicates the true power of MPS method, that it can approximately solve large-scale models in a reasonable amount time when exact method is of no use. Though in this case no comparison with exact method is available to check our results, at least for Angry Boys Model, Exponential Boys Model and Projection Boys Model, the results look reasonable since they are smooth and follow similar trends as in subplots (a). On the other hand, the results from Projection Boys Model show certain strange kink for the joint probability, while the mean and variance seem smooth. The poor results from the Projection Boys Model may come from its complexity as an eigenvalue problem, which determines the equilibrium probability distribution.

In all subplots (d) we study the choice bound dimensions for relative large system of size 80, where the joint probability between the 6th and last sites is analyzed. It can be seen that, though a difference in time evolution can be spotted, for Angry Boys Model, Exponential Boys Model and Projection Boys Model, the results are reasonably close, which suggests that even for large system, we can still survive with relative small bound dimension, and that as the dimension of state space increases exponentially, the bound dimension which is required for a good result increases much slower. For the Projection Boys Model, however, the results seem to be very sensitive to the choice of bound dimensions. Therefore it seems that, at least for joint probability, the MPS may not be a good method to tackle the Projection Boys Model of large system size.

5 Profiling

To have an idea of how time is consumed in our MPS solver, we pick an angry model boy of length 100 and make it evole for $t = 5$. We pick 10 the bound dimension for compression. Fig.7 shows the output of calling cProfile. It takes 40.049 seconds in total to finish the computation. We observe that 33.197 seconds are consumed by the function `tensordot()`, which basically handles all the n-dimensional array multiplication. Indeed `tensordot()` comes with the `numpy` package which has already been implemented by some lower-level languange and optimized for intensive computation. Therefore we believe that implementing our project in C++ will not significantly accelerate the computation.

6 Conclusion

From Newton's famous invention of calculus, to differential equations in the 19th century, fiber bundles in modern geometry, and tensor category in modern algebra inspired by particle physics; there can be no doubt physics research has stimulated many ideas in mathematics for centuries. We believe that, as physics has adopted computer technology over the past several decades, it should also inspire more ideas in computer science. This project is an attempt at attaining this goal.

The structures in quantum many body physicsgeatly resembles classical probability theory, with the exception of interpreting the 2-norm rather than 1-norm as probability. Physicists developed many methods to compute the time evolution of states (or "wave functions") in the context of quantum mechanics, and we naturally extend this MPS approach to classical probability theory.

It is proved in the physics context that the bound dimension of the MPS does not grow with the chain length for a 1-D problem, which is called the area law. An intuitive way to understand is that, if the states in a finite range connect only to each other, a cut in the chain will break some "information sharing". Nonetheless, the amount of the broken information does not grow with the chain length. While it is unclear whether we have a similar "area law" in this probability context, it was well worth it to give it a try.

As our results have shown, the algorithm works very well for short and intermediate length chains, while we need a larger bound dimension to get a good result for the joint probability. The proper bound dimension

is still an open question, but in most cases, a chain of length 200 can be solved with a bound dimension of 20.

What did we learn?

Developing code using group collaboration is not an easy process, but we managed it quite well. We held several design meetings in the beginning and discussed the algorithm quite thoroughly. The task assignment was very clear and everyone worked on different files to resolve any ensuing collisions easily. The process we could improve in the future is managin writing of the final report. We did not split sections in different files wisely, which caused many collision problems.

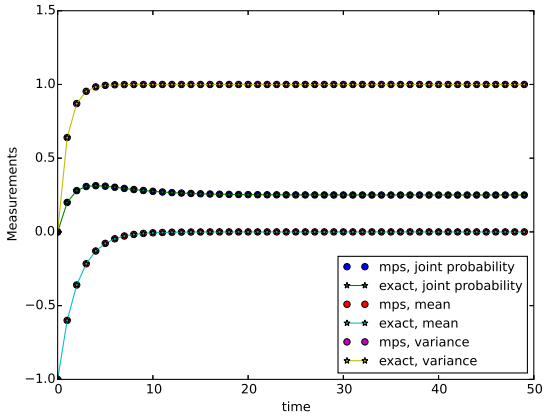
References

[1] S. R. White, *Physical Review Letters* **69**, 2863 (1992)

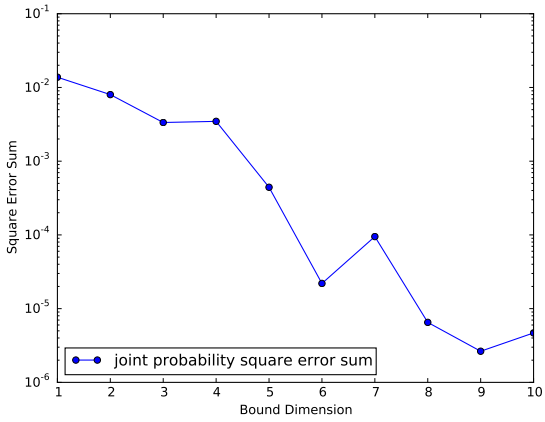
[2] Ulrich Schollwoeck, *Annals of Physics* **326**, 96 (2011)

[3] Verstraete, F., and J. I. Cirac, (2004), arXiv:cond-mat/0407066v1

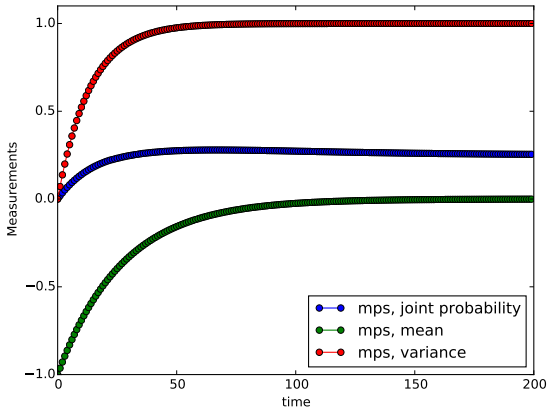
[4] G. Evenbly, G. Vidal, *J Stat Phys* (2011) **145:** 891-918



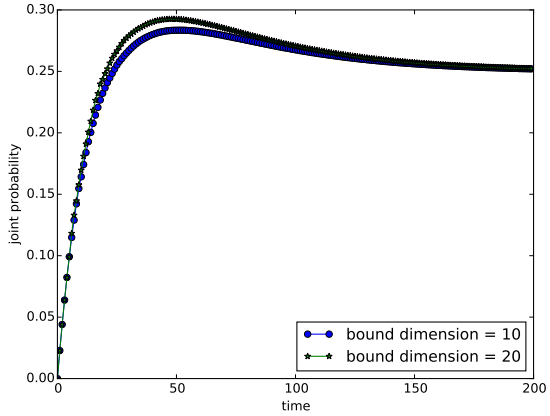
(a) Exact Model(Short Chain)



(b) Error with Different Bound Dimension

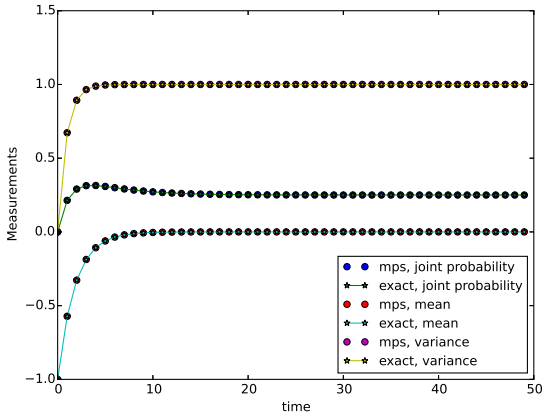


(c) MPS long chain

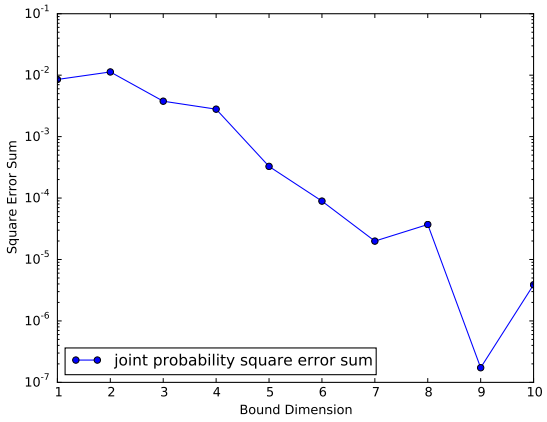


(d) MPS with Different Bound Dimension

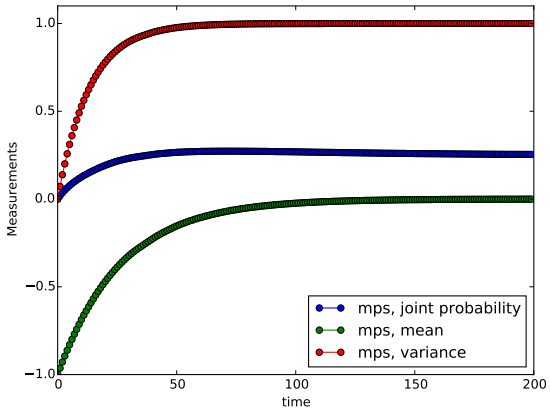
Figure 3: Results of Angry Boys Model. (a) gives the comparison of the joint probability between the 6th and last sites, and the mean value as well as the variance of the 6th site between the MPS method and exact method. The chain size is 10, and the bound dimension χ is chosen to be 10 in the MPS method. (b) shows the decay of square error sum of the joint probability in (a) accumulated in 100 time steps, with increasing bound dimensions for a chain of length 10. The error is computed between the approximation and exact method. (c) is the result of MPS method on a long chain with size $L = 100$ for 200 time steps. The joint probability is again between site 6 and the last site, and both the mean and variance concern site 6. (d) compares the joint probability between site 6 and the last site of the MPS method with bound dimensions 10 and 20. The chain is of size 80.



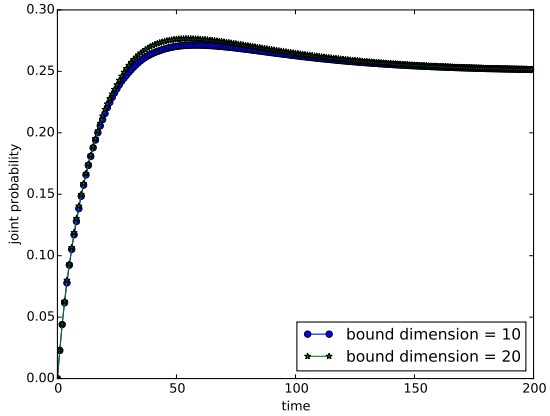
(a) Exact Model(Short Chain)



(b) Error with Different Bound Dimension

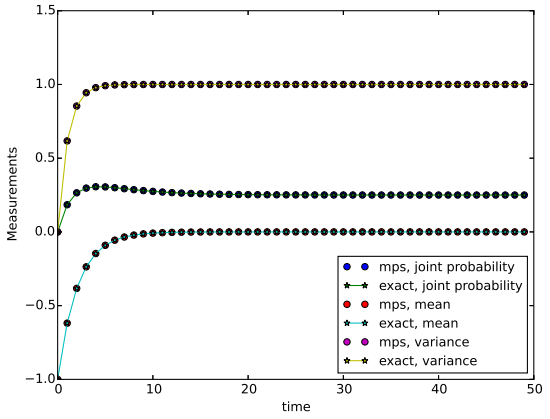


(c) MPS long chain

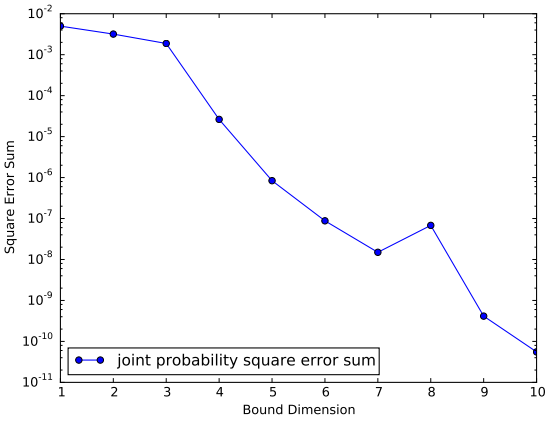


(d) MPS with Different Bound Dimension

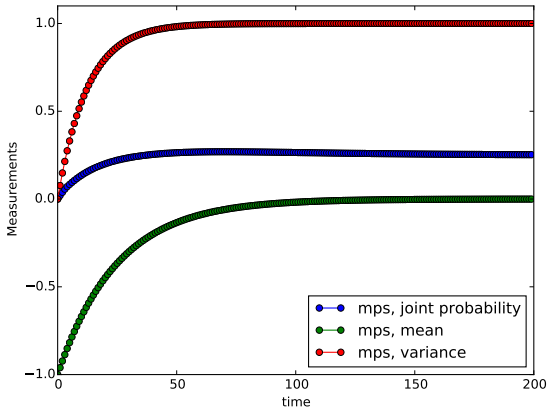
Figure 4: Results of Radiating Boys Model. (a) gives the comparison of the joint probability between the 6th and last sites, and the mean value as well as the variance of the 6th site between the MPS method and exact method. The chain size is 10, and the bound dimension χ is chosen to be 10 in the MPS method. (b) shows the decay of square error sum of the joint probability in (a) accumulated in 100 time steps, with increasing bound dimensions for a chain of length 10. The error is computed between the approximation and exact method. (c) is the result of MPS method on a long chain with size $L = 100$ for 200 time steps. The joint probability is again between site 6 and the last site, and both the mean and variance concern site 6. (d) compares the joint probability between site 6 and the last site of the MPS method with bound dimensions 10 and 20. The chain is of size 80.



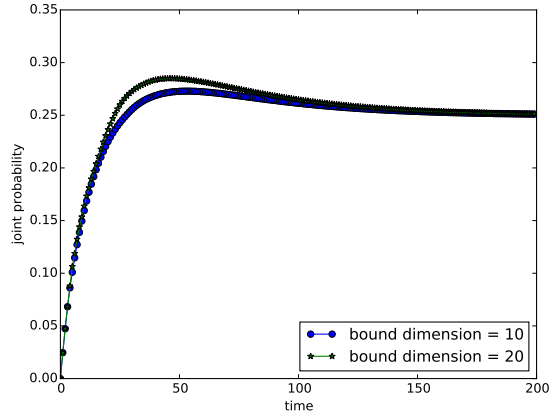
(a) Exact Model(Short Chain)



(b) Error with Different Bound Dimension

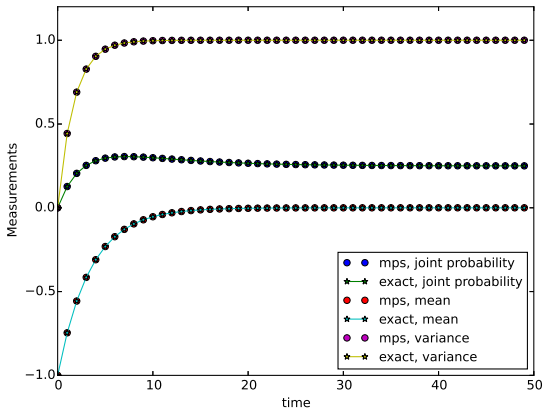


(c) MPS long chain

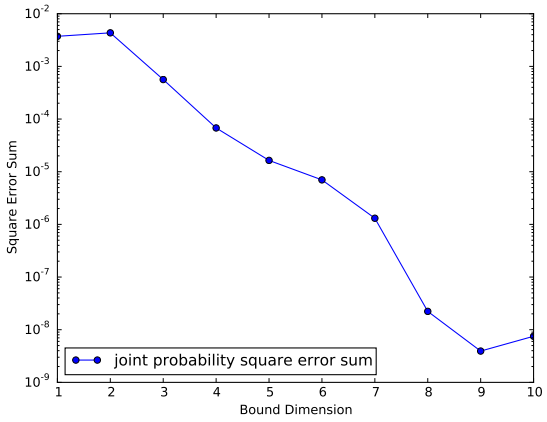


(d) MPS with Different Bound Dimension

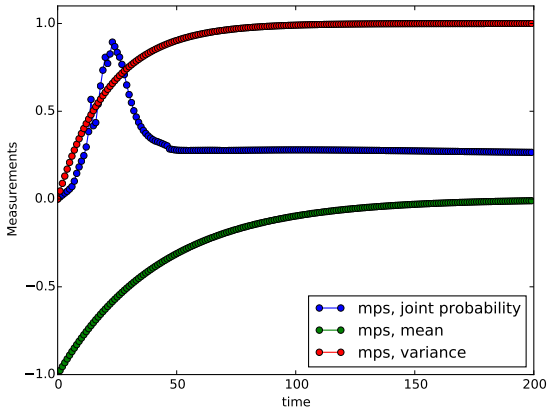
Figure 5: Results of Exponential Boys Model. (a) gives the comparison of the joint probability between the 6th and last sites, and the mean value as well as the variance of the 6th site between the MPS method and exact method. The chain size is 10, and the bound dimension χ is chosen to be 10 in the MPS method. (b) shows the decay of square error sum of the joint probability in (a) accumulated in 100 time steps, with increasing bound dimensions for a chain of length 10. The error is computed between the approximation and exact method. (c) is the result of MPS method on a long chain with size $L = 100$ for 200 time steps. The joint probability is again between site 6 and the last site, and both the mean and variance concern site 6. (d) compares the joint probability between site 6 and the last site of the MPS method with bound dimensions 10 and 20. The chain is of size 80.



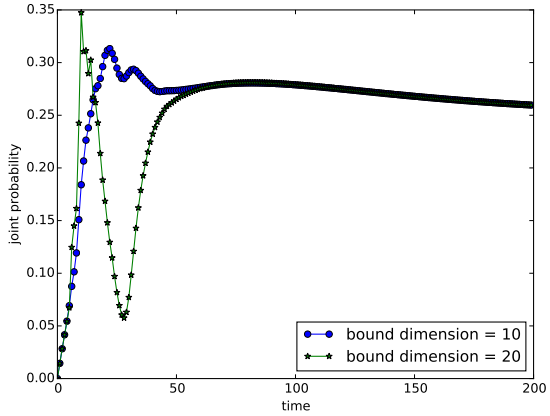
(a) Exact Model(Short Chain)



(b) Error with Different Bound Dimension



(c) MPS long chain



(d) MPS with Different Bound Dimension

Figure 6: Results of Projection Boys Model. (a) gives the comparison of the joint probability between the 6th and last sites, and the mean value as well as the variance of the 6th site between the MPS method and exact method. The chain size is 10, and the bound dimension χ is chosen to be 10 in the MPS method. (b) shows the decay of square error sum of the joint probability in (a) accumulated in 100 time steps, with increasing bound dimensions for a chain of length 10. The error is computed between the approximation and exact method. (c) is the result of MPS method on a long chain with size $L = 100$ for 200 time steps. The joint probability is again between site 6 and the last site, and both the mean and variance concern site 6. (d) compares the joint probability between site 6 and the last site of the MPS method with bound dimensions 10 and 20. The chain is of size 80.

```
nat-oitwireless-inside-vapornet100-a-13115:src theo$ python profiling.py
         227054 function calls in 40.049 seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000   40.049   40.049 <string>:1(<module>)
     1000    0.005    0.000    0.049    0.000 _methods.py:23(_sum)
     3470    0.006    0.000    0.026    0.000 fromnumeric.py:122(reshape)
     1000    0.008    0.000    0.062    0.000 fromnumeric.py:1621(sum)
     1985    0.002    0.000    0.006    0.000 fromnumeric.py:444(swapaxes)
     1485    0.004    0.000    0.004    0.000 linalg.py:101(get_linalg_error_extobj)
     1485    0.004    0.000    0.009    0.000 linalg.py:106(_makearray)
     2970    0.002    0.000    0.005    0.000 linalg.py:111(isComplexType)
     1485    1.026    0.001    1.089    0.001 linalg.py:1225(svd)
     2970    0.003    0.000    0.004    0.000 linalg.py:124(_realType)
     1485    0.011    0.000    0.018    0.000 linalg.py:139(_commonType)
     1485    0.002    0.000    0.002    0.000 linalg.py:198(_assertRankAtLeast2)
     1485    0.003    0.000    0.003    0.000 linalg.py:219(_assertNoEmpty2d)
        5    0.006    0.001    0.208    0.042 mpssolver.py:137(contraction)
        5    0.035    0.007    1.875    0.375 mpssolver.py:189(initializeMpscVar)
        5    1.840    0.368   11.949    2.390 mpssolver.py:232(initializePartialOvl)
        5    1.850    0.370   12.504    2.501 mpssolver.py:252(compressionSweepLeftRight)
        5    1.894    0.379   13.504    2.701 mpssolver.py:308(compressionSweepRightLeft)
        5    0.000    0.000   39.841    7.968 mpssolver.py:370(compressionVariational)
        5    0.008    0.002    0.008    0.002 mpssolver.py:424(normalizeProba)
        1    0.000    0.000   40.049   40.049 mpssolver.py:76(evolve)
        5    0.000    0.000   40.049    8.010 mpssolver.py:84(step)
     8945    0.312    0.000   33.197    0.004 numeric.py:1058(tensordot)
      500    0.002    0.000    0.015    0.000 numeric.py:136(ones)
    19870    0.033    0.000    0.073    0.000 numeric.py:392(asarray)
      495    0.005    0.000    0.007    0.000 twodim_base.py:221(diag)
      505    0.000    0.000    0.000    0.000 {abs}
     1485    0.001    0.000    0.001    0.000 {getattr}
     1000    0.004    0.000    0.004    0.000 {isinstance}
     4455    0.005    0.000    0.005    0.000 {issubclass}
     8945    0.012    0.000    0.012    0.000 {iter}
    37760    0.009    0.000    0.009    0.000 {len}
     2970    0.001    0.000    0.001    0.000 {method '__array_prepare__' of 'numpy.ndarray' objects}
      505    0.001    0.000    0.001    0.000 {method 'append' of 'list' objects}
     4455    0.021    0.000    0.021    0.000 {method 'astype' of 'numpy.ndarray' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
     2970    0.001    0.000    0.001    0.000 {method 'get' of 'dict' objects}
     1000    0.045    0.000    0.045    0.000 {method 'reduce' of 'numpy.ufunc' objects}
    30305   10.884    0.000   10.884    0.000 {method 'reshape' of 'numpy.ndarray' objects}
     1985    0.003    0.000    0.003    0.000 {method 'swapaxes' of 'numpy.ndarray' objects}
    17890    0.029    0.000    0.029    0.000 {method 'transpose' of 'numpy.ndarray' objects}
    10430   21.892    0.002   21.892    0.002 {numpy.core._dotblas.dot}
    19870    0.041    0.000    0.041    0.000 {numpy.core.multiarray.array}
      500    0.011    0.000    0.011    0.000 {numpy.core.multiarray.copyto}
      500    0.002    0.000    0.002    0.000 {numpy.core.multiarray.empty}
```

Figure 7: Output of Cprofiling on executing `evolve(5)` for an angry boy model. The highlighting line shows the time consumed by the function `tensordot`.