Design Document:
Matrix Product State 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)

November 22, 2014

1 Introduction

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. As the dimension of the state space grows exponentially with N, it is extremely hard to be simulated when we deal with a few dozens of 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 in the next decade were aware of the ubiquitous relation between entanglement and dimensionality which allows the state to be approximated as the product of matrices [2]. Some recent developments include extending the algorithm to higher dimensions[3] and critical systems[4].

This inter-departmental collaborative work seeks to apply this algorithm to other interesting problems. The probabilistic nature of quantum mechanics stimulates our study of applying this powerful algorithm in quantum mechanics to difficult problems in probability theory and stochastic processes. An especially interesting example is the Markov process with many degrees of freedom, which also forms an exponentially growing state space.

We will develop a generic MPS solver and compare our results of some simple applications with that of Monte Carlo and exact transition matrix solutions. We will start with a simple model of the dynamics of human relations ("Angry Boys" model) and, if time permits, we may work on an interesting application pertaining to a financial market crisis. This project is research-based and we hope to start with simple models. Due to the time limitation, this project is focusing on one dimensional models of which the algorithm in physics is well-established, and higher dimensional or highly correlated relationships which are of more interest and 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 we need to perform the summation over N indices of σ_i simultaneously, thus gives m^N terms. 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)

To deal with the large number of terms in the contraction, we want to effectively split the space. 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, but 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 possible values and the second has m^{N-1}. Grouping $N-1$ indices means making a bijection map from $N-1$ numbers varying between 1 and m to a single integer varying from 1 to m^{N-1}. This re-indicing trick is crucial in MPS algorithms.

We can then treat this 2 indices object $A_{\sigma_1(\sigma_2\sigma_3\cdots\sigma_N)}$ as a matrix with σ_1 the row index and $(\sigma_2\sigma_3\cdots\sigma_N)$ the column index, thus forming a $m \times m^{N-1}$ matrix. Applying singular value decomposition (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 can 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 multiplied the S and V matrices.

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

$$A_{\{\sigma_i\}} = U^{\sigma_1}_{m_1}U^{\sigma_2}_{m_1m_2}U^{\sigma_3}_{m_2m_3}\cdots U^{\sigma_N}_{m_{N-1}m_N}$$

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

For example, if we have a linear operator $O_{\{\sigma'\}\{\sigma\}} = O^{\sigma_1'\sigma_1}_{n_1}O^{\sigma_2'\sigma_2}_{n_1n_2}O^{\sigma_3'\sigma_3}_{n_2n_3}\cdots O^{\sigma_N'\sigma_N}_{n_{N-1}n_N}$ then applying this operator on the MPS is actually the contraction on m, n and σ_i space. The traditional way corresponds to first contracting m and n spaces and then σ but this novel method contracts the σ space first and then m and n. The specific order of contraction is:

- Compute $\sum_{\sigma_i} O^{\sigma_i'\sigma_i}_{n_{i-1}n_i}A^{\sigma_i}_{m_{i-1}m_i} = M^{\sigma_i'}_{n_{i-1}n_im_{i-1}m_i}$ for all i's

- $\sum_{n_1 m_1} M^{\sigma'_1}_{n_1 m_1} M^{\sigma'_2}_{n_1 n_2 m_1 m_2} = \tilde{M}^{\sigma'_1 \sigma'_2}_{n_2 m_2}$
- $\sum_{n_2 m_2} \tilde{M}^{\sigma'_1 \sigma'_2}_{n_2 m_2} M^{\sigma'_3}_{n_2 n_3 m_2 m_3} = \tilde{M}^{\sigma'_1 \sigma'_2 \sigma'_3}_{n_3 m_3}$
- keep doing this, until we finish the multiplication of all these objects and get $\tilde{M}^{\sigma'_1 \sigma'_2 \cdots \sigma'_N} = A_{\sigma'_1 \sigma'_2 \cdots \sigma'_N}$ which 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 Approximation

You may wonder why our algorithm seems to transform an exponentially scaling problem to polynomial time; the reason is that we hide something. We did not talk about the dimension of auxiliary space m_i and n_i, and they should grow exponentially. The method is useful only when the interaction is local or short-range, which means the degree of freedom on one site only effectively interacts with some of its neighbors. It is proven that in this case, the diagonal terms in S matrix decay exponentially and we can make an effective cut-off in auxiliary space, keeping only a finite number χ dimensions so that all U's are at most $\chi \times \chi$ dimensional matrices. χ is called the bound dimension. The effectiveness of this algorithm relies deeply on that we do not need a large χ to achieve 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 Matrix Product Operators (MPO)

The Hamiltonian in quantum mechanics, or equivalently the transitional matrix in stochastic process, can be written as a series of matrix product operators where each matrix only applies on one site. A lot of examples of various types of interactions are developed and more details will be shown in our final report.

3 Project Overview

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

3.1 Model

We will be considering a one dimensional problem with n agents on a line at positions $i = 1, 2, \cdots, n$. Each agent has two states $(1,0)^T$ and $(0,1)^T$, so the state space is of dimension 2^n, and the dynamics at each time step is determined by the following probability transition matrix:

$$H = pI + (1-p) \sum_{i=1}^{n-1} \frac{1}{n-1} \sigma^x_i \otimes \sigma^x_{i+1},$$

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

$$\sigma^x_i = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

operating on the 2-state space at position i. Intuitively, we can interpret the two states of each agent as "angry" $(1,0)^T$ and "calm" $(0,1)^T$, and the transition matrix describes the interactions among agents in terms of neighboring pairs. The chance such that the state remains untouched is p and otherwise, with a probability $(1-p)/(N-1)$, one of the pairs of agents will change their states. We will start with the state where every agent is calm, and try to compute various joint probabilities at later times.

3.2 Implementation

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

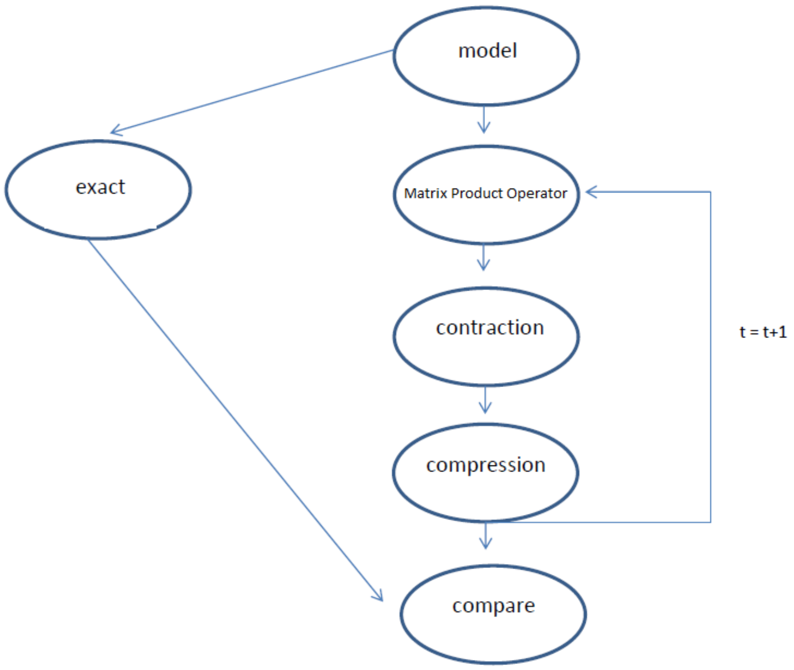

Figure 1: The flow chart of our program.

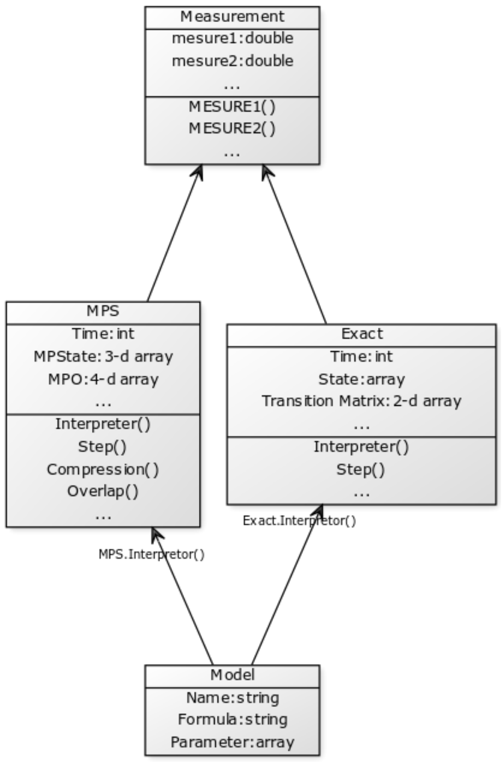

Figure 2: A bird view of our classes.

3.2.1 Exact Solution

An exact solution may be obtained for the proposed system. Because the dimension of state space grows exponentially, implementing such a solution becomes too computationally costly when dealing with a few dozen spins and/or chain sites. Nonetheless, an exact solution bounded by reasonably-sized state spaces will be needed to corroborate results obtained through the MPS algorithm. To generate an exact solution for reasonably-sized state-spaces, a class `Exact` will be implemented. The class will contain two functions. A function `Interpreter` will read data contained in an instance of the class `Model`, including the length of the chain and local dimensions for each chain site. This data will be used to construct a transitional matrix, to be implemented as a sparse matrix, which will be used for simulating the current state of the system. The interpreter will then call on the `Step` function to solve for the state by stepping in discrete time. Additionally, a Monte Carlo simulation can be used to solve this problem. If time permits, a Monte Carlo simulation of the system will be implemented to serve as an addition comparator to the MPS algorithm.

3.2.2 MPS class

Parallel to the class `Exact`, we construct a class named `MPS` 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 which are necessary in updating the current state. The `MPS` class will include the following:

Member data We will have the current state in a variable named `MPState` which is a 3-d array storing $M^{\sigma_i}_{a_{i-1},a_i}$. The operator $O^{\sigma_i,\sigma_i'}_{n_{i-1},n_i}$ will be stored in a variable named `MPO` which is a 4-d array. In addition, we will store the time of the system in the variable `T`.

Constructor The constructor will take a model of type `Model` and an initial state as argument and initilize the members `MPState`, `MPO` and `T`. In the first versions, we only focus on certain well-defined models.

Methods As is shown in the flow chart, to update the state from time t to $t+1$, one must first apply the MPO to the MPS and then compress the state. The reason why the compression step is necessary is that after applying MPO to MPS, the resulting MPS will have a greater auxiliary degrees of freedom. The compression step consists of projecting the resulting MPS (which is of higher dimension) back to the initial space (of a lower dimension). We shall write the routine `Step` responsible for applying MPO and updating the current state `MPState`. This routine will call `Compression` to make sure that the new state has the desired degrees of freedom. Both `Step` and `Compression` will involve some basic operations of multiplication of arrays and summing over a certain number of indices (similar to matrix multiplication). We shall implement these basic operations first, so `Step` and `Compression` can call these routines. Note that if we would like to calculate the expectation of a physical quatity, or the marginal probability of a physical states, we have to do the "overlapping" operation of two MPS's. This again involves a number of basic operations defined in the class `MPS`. So the members in the class `Measurement` which handles the evaluation of expectations and probabilities, should be able to call the routines in the class `MPS`.

The `Compression` function of MPS will work iteratively. To initialize for the compression, we will apply SVD to the MPS representation of the states, and then truncate the decomposed matrices with χ largest single values retained. The following steps of compression will minimize $||\psi^i - \psi^{i-1}||^2$ iteratively, where ψ^{i-1} is the state from the previous step while ψ^i is the state obtained for the new *ith* step. The extrema condition in minimizing $||\psi^i - \psi^{i-1}||^2$ will require solving linear equations to get the new coefficients in the new matrices of the *ith* step. After that the new state will also be normalized.

3.2.3 Measurement Class

The `Measurement` class includes functions for calculating the expectation of a given operator, the conditional probability of certain states, and possibly the correlation functions of different time steps. Both the conditional probability and correlation function may require us to store the MPS at each step. By calling this storage, we don't need to rerun the MPS procedure to calculate new physical quantities. And this is one of the advantages of MPS over Monte Carlo method. In the simplest form, the user can specify a set of states/operators through a defined structure, which are then combined with final state of the system to give (conditional) probabilities. If time permitted, we will try to extend the measurement to include time-dependent information.

3.3 Milestones

Alpha version: To be released on 12/15/2014, written in Python. It will include the following work:

- Model and Interface will include user specified information and determine the model used, number of steps, physical quantities to measure. Headed by Bin Xu.
- The `measurement` of interesting variables, such as the average values and various correlation functions is defined in the measurement class. Headed by Peiqi Wang.
- The `Compression` function uses the building blocks in `MPS` to compress the new MPS matrices in the new step. Headed by Liangsheng Zhang and Jun Xiong.
- The implementation of the `Measurement` will implement different functions to calculate physical properties . Headed by Liangsheng Zhang and Jun Xiong.
- The `Exact` class calculates the exact time evolution of the initial state for smaller sizes. It may use the functions in `Measurement` to calculate physical properties. Headed by Peter Gjeltema.

Beta version: To be released on 01/15/2015. Several interesting work will include:

- Study other interesting models that are more practical, and/or more difficult.
- Implement the algorithm in C++ to gain a speed-up.
- If time permits, we can try some basic ideas of a time evolving PEPS algorithm to study 2-dimensional 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