

TEBD

David Aceituno

Contents

| | |
|-----------------------------------|----------|
| iTEBD on C++ | 1 |
| Notation | 1 |
| Model | 2 |
| Hamiltonian | 2 |
| Time Evolution Operator | 2 |
| The iTEBD algorithm | 2 |
| Initialization | 2 |
| SVD Truncation | 2 |
| Loop | 2 |

iTEBD on C++

The code follows the steps outlined in the following paper

Kjäll, J. A., Zaletel, M. P., Mong, R. S. K., Bardarson, J. H. & Pollmann, F. Phase diagram of the anisotropic spin-2 XXZ model: Infinite-system density matrix renormalization group study. *Phys. Rev. B* **87**, 235106 (2013).

and the file itebd.py.

Notation

Latin letters are reserved for physical indices, and greek indices for bond indices. The order of indices in all tensors is *physical indices* \rightarrow *bond indices*. In diagrammatic notation we have

$$\Lambda_{\alpha\beta} = \alpha \text{---} \boxed{\Lambda} \text{---} \beta = \begin{matrix} 0 \\ \text{---} \end{matrix} \boxed{\Lambda} \begin{matrix} 1 \\ \text{---} \end{matrix} \quad (1)$$

$$\Gamma_{\alpha\beta}^i = \alpha \text{---} \boxed{\Gamma} \text{---} \beta = \begin{matrix} i \\ | \end{matrix} \boxed{\Gamma} \begin{matrix} 0 \\ | \end{matrix} \quad (2)$$

$$\theta_{\alpha\beta}^{ij} = \alpha \text{---} \boxed{\theta} \text{---} \beta = \begin{matrix} i & j \\ | & | \end{matrix} \boxed{\theta} \begin{matrix} 0 & 1 \\ | & | \end{matrix} \quad (3)$$

In C++ these are explicitly objects of type `Eigen::Tensor<double,rank,Eigen::ColMajor>;`, which are `typedef`'ed into shorthand `TensorR`, where `R` is the rank (0 to 4).

Model

Hamiltonian

We study the 1D Ising model with the following two-site Hamiltonian

$$H = \begin{pmatrix} J & -g/2 & -g/2 & 0 \\ -g/2 & -J & 0 & -g/2 \\ -g/2 & 0 & -J & -g/2 \\ 0 & -g/2 & -g/2 & J \end{pmatrix}.$$

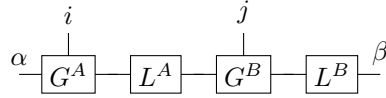
Time Evolution Operator

We now define $U = e^{-H\delta\tau}$, a unitary matrix that performs the time evolution. U is reshaped into a rank 4 tensor with dimensions (2,2,2,2), i.e. $U = U_{j'k'}^{jk}$, where j and k are the original physical indices, and j' and k' are the updated indices.

The iTEBD algorithm

Initialization

We initialize a two-site MPS state in Vidal canonical form



- Define $G_{\alpha\gamma}^{A,i} = G_{\delta\beta}^{B,i}$ with dimensions (2,1,1). Set $G^{A,B}(0,0,0) = 1$ and the rest to zero. In code: `std::array<Tensor3,sites>G;`, where `constexpr int sites =2;`
- Define diagonal matrices $L_{\gamma\gamma}^A = L_{\delta\delta}^B$ with dimensions (1,1). Set $L^{A,B}(0,0) = 1$. In code: `std::array<Tensor2,sites>G;`, where `constexpr int sites =2;`

SVD Truncation

We define χ , the maximum rank of each SVD decomposition simply by `long chi =15;`.

Loop

Next we have a double for-loop.

C++ code

```
1 for(int step = 0; step < N; step++){
2   for(long i_bond = 0; i_bond < d; i_bond++) {
3     long ia = mod(i_bond-1,d);
4     long ib = mod(i_bond ,d);
5     long chia = G[ia].dimension(1); //index alpha (eq. 25-27)
6     long chib = G[ib].dimension(2); //index gamma (eq. 25-27)
7
8     // Construct theta matrix and do time evolution
9     A      = L[ib].contract(G[ia], idxlist1{idx2(0,1)}).
```

```

10         shuffle(array3{1,0,2}).
11         contract(L[ia],idxlist1{idx2(2,0)});
12     B      = G[ib].contract(L[ib], idxlist1{idx2(2,0)});
13     theta4 = A.contract(B, idxlist1{idx2(2,1)}).shuffle(array4{0,2,1,3});
14     theta2 = U.contract(theta4, idxlist2 {idx2(2,0),idx2(3,1)})
15         .shuffle(array4{0,2,1,3}).reshape(array2{d*chia,d*chib});
16
17     SVD.compute(tensor2_to_matrix(theta2), Eigen::ComputeThinU | Eigen::ComputeThinV);
18
19     chi2 = std::min(SVD.rank(),chi);
20     X = matrix_to_tensor3(SVD.matrixU().leftCols(chi2),{d,chia,chi2})
21         .unaryExpr(&truncate);
22     Z = matrix_to_tensor3(SVD.matrixV().leftCols(chi2),{d,chib,chi2})
23         .shuffle(array3{0,2,1}).unaryExpr(&truncate);
24     Y = matrix_to_tensor1(SVD.singularValues().head(chi2));
25     L[ia] = asDiagonal(Y / SVD.singularValues().head(chi2).norm());
26
27     G[ia] = inverseDiagonal(L[ib]).contract(X, idxlist1{idx2(0,1)})
28         .shuffle(array3{1,0,2});
29     G[ib] = Z.contract(inverseDiagonal(L[ib]), idxlist1{idx2(2,0)} );
30 }
31 }

```