TEBD

David Aceituno

Contents

iTEBD on C++

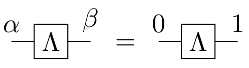
The code follows the steps outlined in the following paper

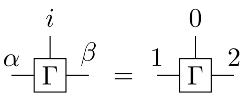
> Kjäll, J. A., Zaletel, M. P., Mong, R. S. K., Bardarson, J. H. & Pollmann, F. Phase diagram of the anisotropic spin-2 XXZ model: Infinite-system density matrix renormalization group study. *Phys. Rev. B* **87,** 235106 (2013).

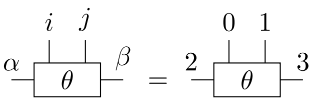
and the file itebd.py.

Notation

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

$\Lambda_\alpha\beta =$  (1)

$\Gamma^i_{\alpha\beta} =$  (2)

$\theta^{ij}_{\alpha\beta} =$  (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

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

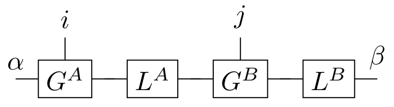
Time Evolution Operator

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. $U = U^{jk}_{j'k'}$, 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 $G^{A,i}_{\alpha\gamma} = G^{B,i}_{\delta\beta}$ 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 $L^A_{\gamma\gamma} = L^B_{\delta\delta}$ 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

```cpp
for(int step = 0; step < N; step++){
  for(long i_bond = 0; i_bond < d; i_bond++) {
    long ia = mod(i_bond-1,d);
    long ib = mod(i_bond  ,d);
    long chia = G[ia].dimension(1); //index alpha (eq. 25-27)
    long chib = G[ib].dimension(2); //index gamma (eq. 25-27)

    // Construct theta matrix and do time evolution
    A        = L[ib].contract(G[ia], idxlist1{idx2(0,1)}).
```

```
                    shuffle(array3{1,0,2}).
                    contract(L[ia],idxlist1{idx2(2,0)});
    B        = G[ib].contract(L[ib], idxlist1{idx2(2,0)});
    theta4  = A.contract(B, idxlist1{idx2(2,1)}).shuffle(array4{0,2,1,3});
    theta2  = U.contract(theta4, idxlist2 {idx2(2,0),idx2(3,1)})
                  .shuffle(array4{0,2,1,3}).reshape(array2{d*chia,d*chib});

    SVD.compute(tensor2_to_matrix(theta2), Eigen::ComputeThinU | Eigen::ComputeThinV);

    chi2 = std::min(SVD.rank(),chi);
    X = matrix_to_tensor3(SVD.matrixU().leftCols(chi2),{d,chia,chi2})
                                              .unaryExpr(&truncate);
    Z = matrix_to_tensor3(SVD.matrixV().leftCols(chi2),{d,chib,chi2})
                        .shuffle(array3{0,2,1}).unaryExpr(&truncate);
    Y = matrix_to_tensor1(SVD.singularValues().head(chi2));
    L[ia] = asDiagonal(Y / SVD.singularValues().head(chi2).norm());

    G[ia] = inverseDiagonal(L[ib]).contract(X, idxlist1{idx2(0,1)})
                                            .shuffle(array3{1,0,2});
    G[ib] = Z.contract(inverseDiagonal(L[ib]), idxlist1{idx2(2,0)} );
  }
}
```