Computer session

MPS and DMRG

Practical DMRG calculations using Block/PySCF

Quick tutorials for Ground / Excited states calculations

How to implement MPS type methods?

iTEBD code with Tensor manipulations

Practical DMRG calculation

1. Installation and settings
2. DMRG-CASSCF calculation
3. Using localized molecular orbitals
4. DMRG-NEVPT2 calculation
5. DMRG-CI calculation for Full-AS

Installation of Block

```
$ git clone https://github.com/gkc1000/Block.git
$ cd Block
edit “makefile” suite for your environment…
$ make
```

Add Block path to your .bashrc and source it

```
export LD_LIBRARY_PATH=$HOME/Block:$LD_LIBRARY_PATH
```

Prerequisites:

- GCC-4.8 or later to enable C++11 features
- BLAS/LAPACK or Intel MKL library
- Boost C++ library

Prerequisites

Download BLAS/LAPACK from http://www.netlib.org/lapack/

```
$ tar xvzf lapack-3.6.0.tgz
$ cd lapack-3.6.0/
$ cmake .
$ make                                  build “libblas.a” and “liblapack.a”

$ cat INSTALL/make.inc.gfortran > make.inc
$ cd CBLAS/
$ make                                  build “libcblas.a”

$ cd ../LAPACKE
$ make                                  build “liblapacke.a”
```

Download Boost C++ library from http://www.boost.org/

```
$ tar xvzf boost_1_59_0.tar.gz
$ cd boost_1_59_0/
$ ./bootstrap --prefix=/path/to/install
$ ./b2 install
```

Settings for PySCF

```
$ cd pyscf/future/dmrgscf
$ cp settings.py.example settings.py
```

Edit the following line in “settings.py”

```
BLOCKEXE = '/path/to/your/Block'
```

and just comment the following out

```
#PYCHEMPS2BIN = …
```

Try “test01.py” to see whether it works or not

Note that DMRG with small active space would be much slower than normal CASCI, seems no response but still running

If # of active space > 12, situation will be changed

DMRG-CASSCF calculation

Exircise 01:

Modify “cas.py” (problem 4.2 in the 1st day),

to perform CASSCF(**6o**, 6e) calculation using DMRG

by adding following 3 lines appropriately

```
from pyscf.dmrgscf.dmrgci import *
from pyscf.dmrgscf import settings
```

```
mc.fcisolver = DMRGCI(mol, 20)
```

DMRG with M = 20

and compare the result with that from normal CASSCF

Optionally, add following to see detailed CASSCF output

```
mc.verbose = 4
```

DMRG-CASSCF calculation

Exircise 02:

Try “test02.py” to perform DMRG-CASSCF(8o, 8e) of C_8H_{10}

Then, change M value from 10 to 100, plot the energies against M, and see energy converges when M becomes large

You’ll find 3 files which are automatically generated

```
dmrg.conf      dmrg.out      FCIDUMP
```

dmrg.out also contains useful information

You can find something from Block documentation

DMRG-CASSCF calculation

Exircise 03:

Following lines perform orbital localization for active space in ex. 02

```
from pyscf.tools import localizer

...

act = [26,27,28,29,30,31,32,33] # indices of pi orbitals in SCF
lmo = mf.mo_coeff
loc = localizer.localizer(mol, lmo[:, act], 'boys')
lmo[:, act] = loc.optimize()
```

Modify “test02.py” to pass the localized orbitals as an initial guess for the DMRG-CASSCF calculation (“test03.py” is a possible answer)

DMRG-NEVPT2 calculation

Exircise 04:

Modify “nevpt2.py” @ 1st day, or modify “test01.py”, or run “test04.py” to perform DMRG-NEVPT2 calculation

Then, compare the results with the normal NEVPT2 calculation

DMRG-CI calculation for full-AS

Exircise 05:

Try “test05.py” to perform a single DMRG-CI calculation for full-AS (Note 1s of F atom still being frozen)

Compare the results with “cas.py” and/or “nevpt2.py”

Think what’s the meaning of these energy differences?

How to implement MPS

1. Compilation using BTAS library
2. Tensor manipulations
3. iTEBD source code

Basic Tensor Algebra Subprograms (BTAS)

Description

Basic Tensor Algebra Subroutines (BTAS) is a C++ headers-only library for tensor algebra. BTAS is a reference implementation of Tensor Working Group concept spec.

Visit also

https://github.com/BTAS/BTAS

and/or

http://itensor.org/btas/

Prerequisites

Download BTAS as

```
$ git clone https://github.com/BTAS/BTAS.git
```

- GCC-4.8 or later to enable C++11 features
- BLAS/LAPACK or Intel MKL library
- Boost C++ library

Compilation

With CBLAS/LAPACKE

```
$ g++ -std=c++11 -D_HAS_CBLAS xxx.cpp
  -I/path/to/BTAS -I/path/to/lapack/CBLAS/include
  -I/path/to/lapack/LAPACKE/include -I/path/to/boost/include
  /path/to/lapack/libcblas.a /path/to/lapack/liblapacke.a
  /path/to/lapack/lib/liblapack.a /path/to/lapack/lib/libblas.a
  -lgfortran
```

With Intel MKL

```
$ g++ -std=c++11 -D_HAS_CBLAS -D_HAS_INTEL_MKL xxx.cpp
  -I/path/to/BTAS -I/path/to/MKL/include
  -L/path/to/MKL/lib/intel64 -lmkl_intel_lp64 -lmkl_sequential -lmkl_core
```

Tensor object

Exercise 01:

Construct double-prec. rank-4 tensor with size 4 for each dimension

```
#include <btas/btas.h>

int main ()
{
    btas::Tensor<double> A(4,4,4,4);
    A.fill(1.0);

    return 0;
}
```

Rank-4 tensor having dimension 4 for each index

Fill all elements with 1.0

Permutation

Exercise 02: Carry out permutation of index $B_{kilj} = A_{ijkl}$

```
#include <functional>
#include <random>
#include <btas/btas.h>
using namespace btas;

int main ()
{
    std::mt19937 rgen;
    std::uniform_real_distribution<double> dist(0.0, 1.0);

    enum { i, j, k, l };
    Tensor<double> A(4,2,4,2);
    A.generate(std::bind(dist,rgen));

    Tensor<double> B;
    permute(A,{i,j,k,l},B,{k,i,l,j});

    return 0;
}
```

Fill all elements with random number

$\longleftarrow B_{kilj} = A_{ijkl}$

Contraction

Exercise 03: Compute $C_{ijmn} := 1.0 \times \sum_{kl} A_{ijkl} B_{kmln} + 1.0 \times C_{ijmn}$

```
#include <btas/btas.h>
using namespace btas;

int main ()
{
    enum { i, j, k, l, m, n };
    Tensor<double> A(4,4,4,4);
    A.fill(1.0);
    Tensor<double> B(4,2,4,2);
    B.fill(2.0);

    Tensor<double> C;
    contract(1.0,A,{i,j,k,l},B,{k,m,l,n},1.0,C,{i,j,m,n});

    return 0;
}
```

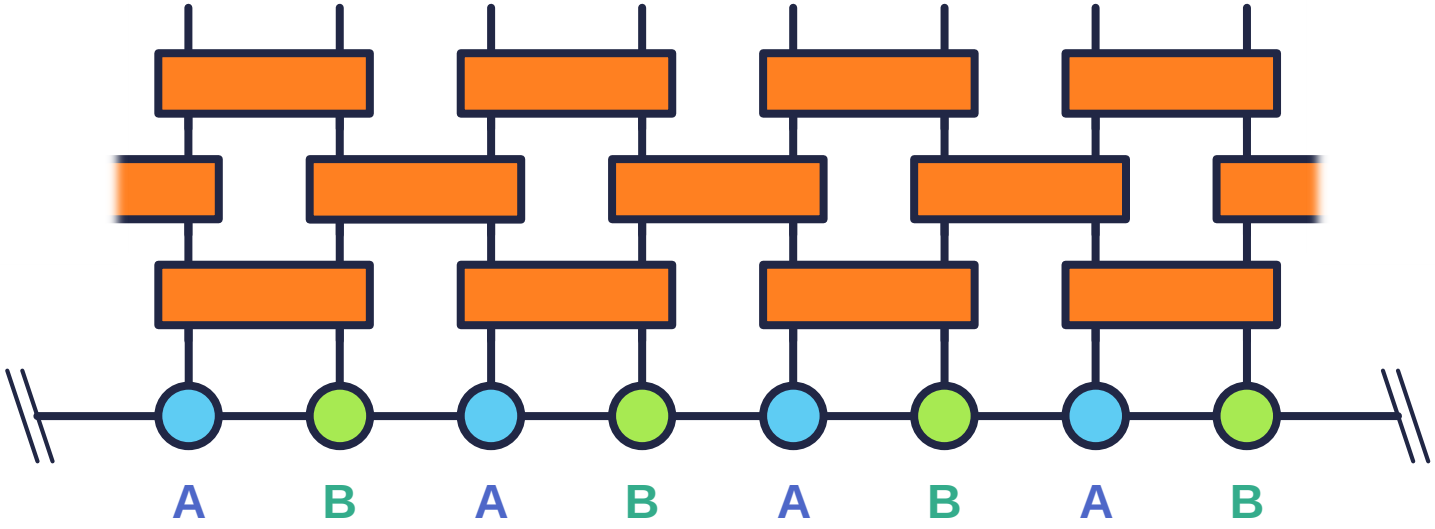
SVD

Exercise 04: Compute $A_{ijkl} = \sum_m U_{ijm} s_m V^{\dagger}_{mkl}$

```
#include <functional>
#include <random>
#include <btas/btas.h>
using namespace btas;

int main ()
{
    std::mt19937 rgen;
    std::uniform_real_distribution<double> dist(0.0, 1.0);
    Tensor<double> A(4,2,4,2);
    A.generate(std::bind(dist,rgen));

    Tensor<double> S(0);
    Tensor<double> U(4,2,0);
    Tensor<double> Vt(0,4,2);
    gesvd('S', 'S', A, S, U, Vt);

    return 0;
}
```

iTEBD algorithm

Infinite version of TEBD

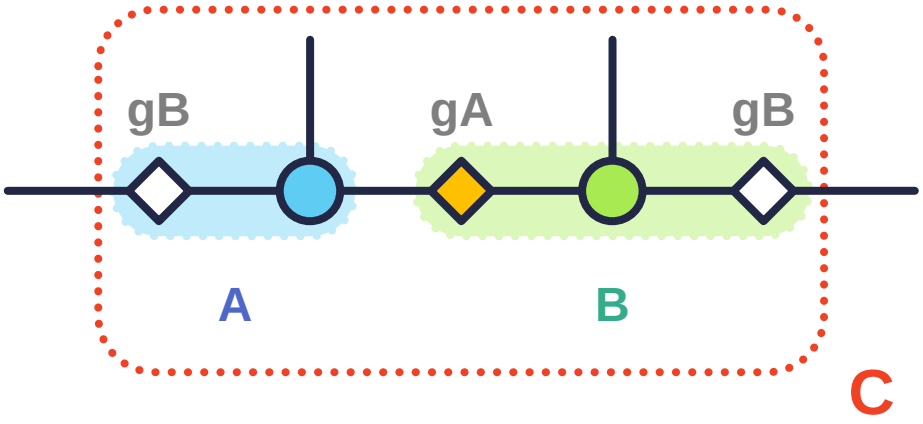


$= e^{-\hat{h}\Delta t}$ where $\hat{h} = J\hat{S}(i)\hat{S}(i+1)$

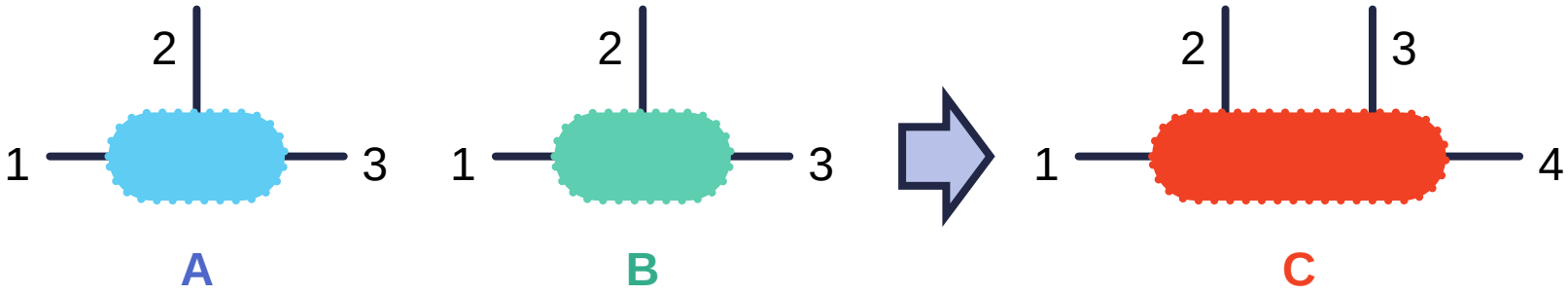
"iTEBD.cpp" contains a source code with 3 exercises

"iTEBD.cpp.sample" gives you a possible solution

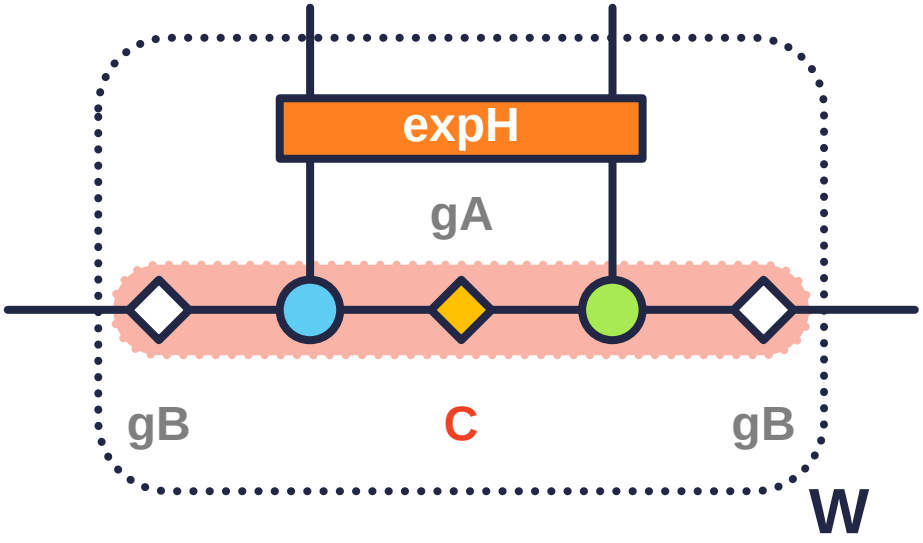
iTEBD algorithm: Step 1



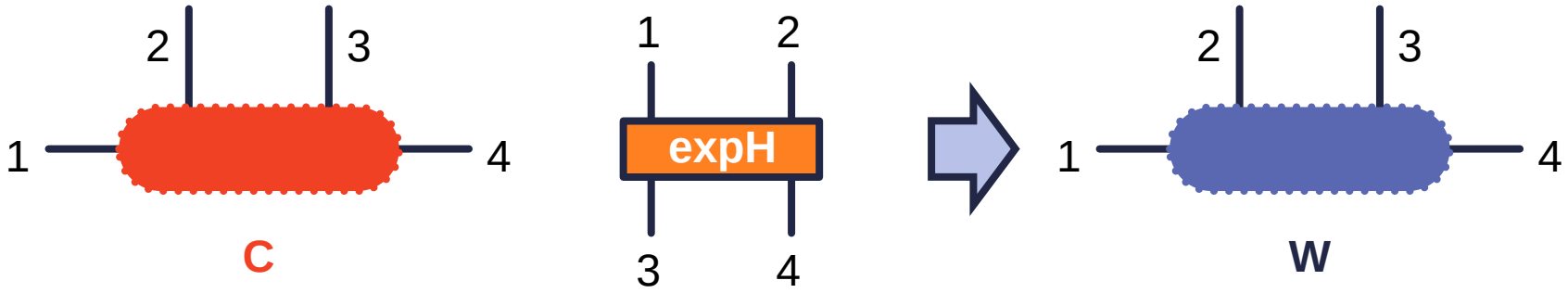
Ex.1) Contract **A** and **B** to get 2-site tensor **C**



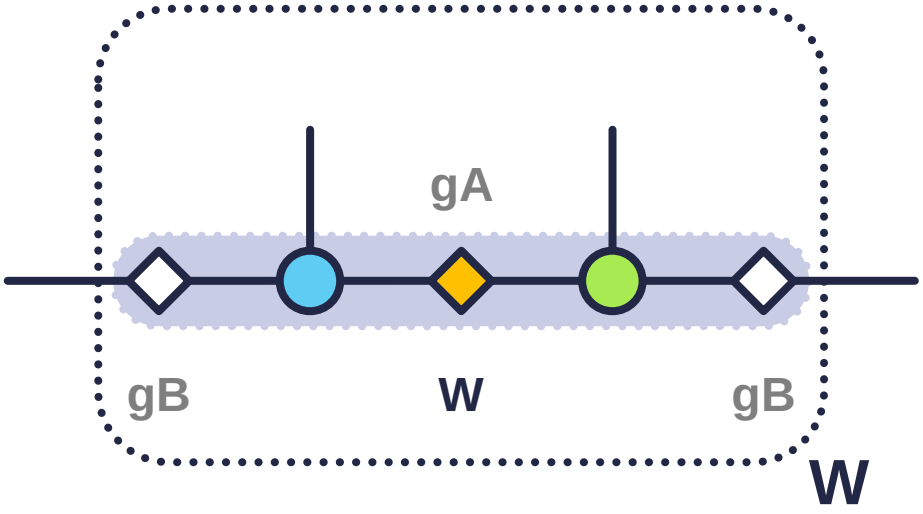
iTEBD algorithm: Step 2



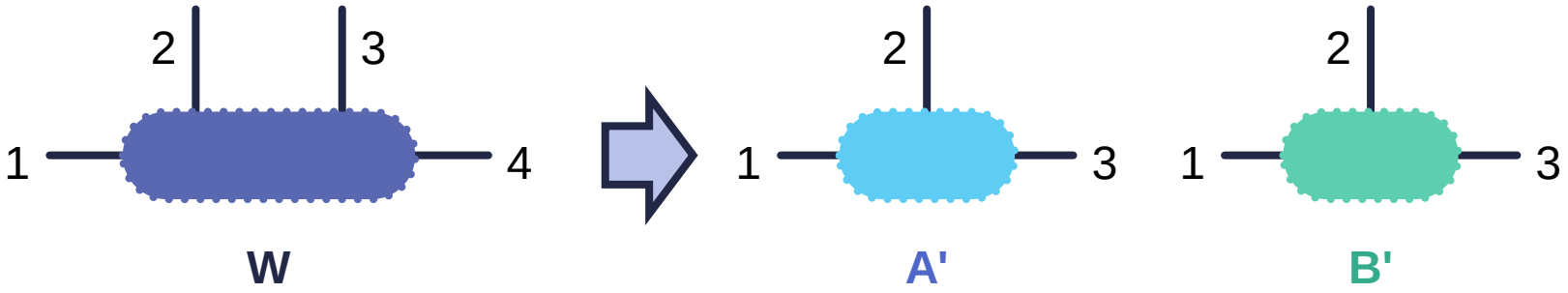
Ex.2) Contract **C** and **expH** to get 2-site tensor **W**



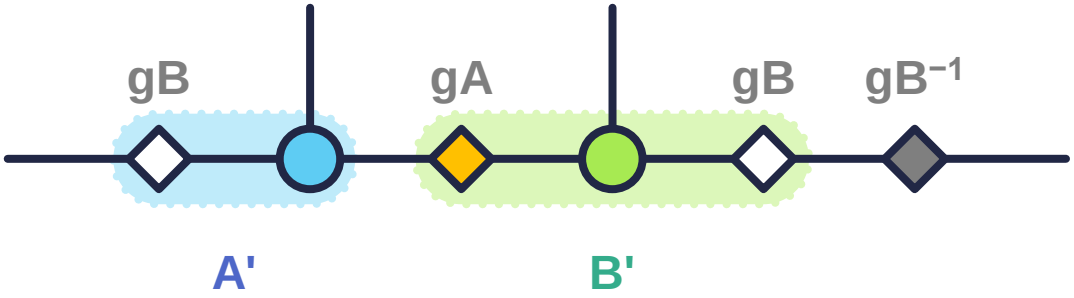
iTEBD algorithm: Step 3



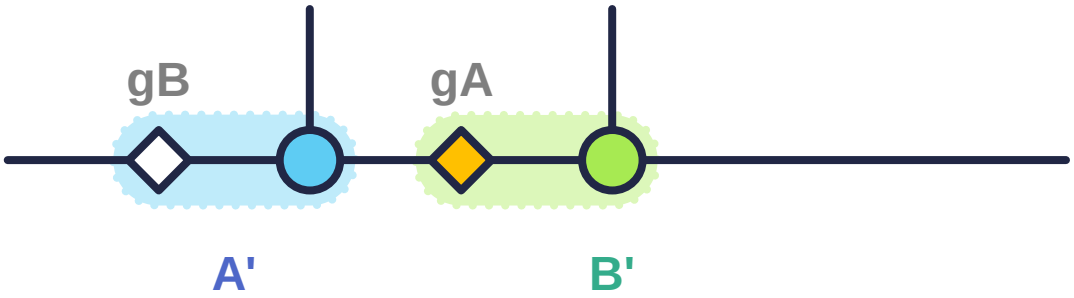
SVD on **W** to get 1-site tensors **A'** and **B'**



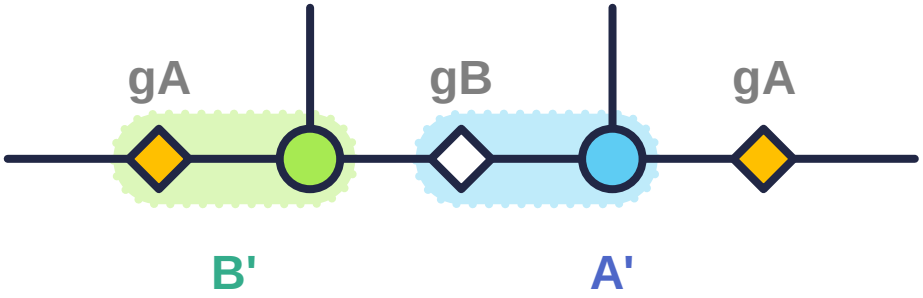
iTEBD algorithm: Step 4



Fix the gauge…



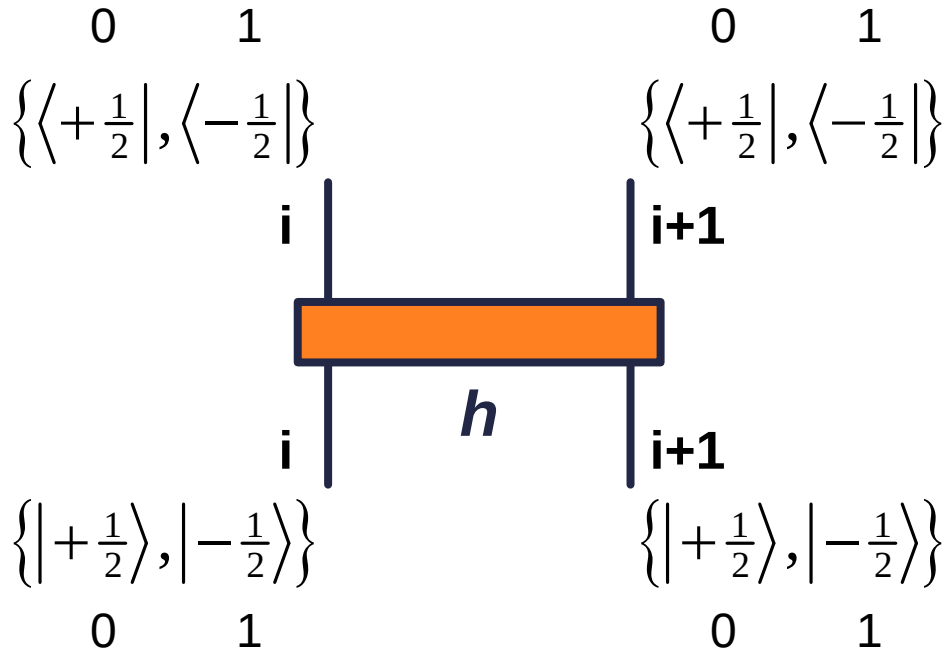
iTEBD algorithm: Step 5



Swap **A** and **B**

then perform the same operation for **BA**

iTEBD algorithm: What's expH?



$$\hat{h} = J\hat{S}(i)\hat{S}(i+1)$$

$$= \frac{J}{2}\left(\hat{S}^{+}(i)\hat{S}^{-}(i+1) + \hat{S}^{-}(i)\hat{S}^{+}(i+1)\right) + J\hat{S}^{Z}(i)\hat{S}^{Z}(i+1)$$

iTEBD algorithm: What's expH?

"2 x 2 x 2 x 2" tensor viewed as "4 x 4" matrix

$$\boldsymbol{h} = \begin{pmatrix} +\frac{J}{4} & 0 & 0 & 0 \\ 0 & -\frac{J}{4} & \frac{J}{2} & 0 \\ 0 & \frac{J}{2} & -\frac{J}{4} & 0 \\ 0 & 0 & 0 & +\frac{J}{4} \end{pmatrix}$$

$\left|+\frac{1}{2}\right\rangle_i \left|+\frac{1}{2}\right\rangle_{i+1}$ $\quad \left|+\frac{1}{2}\right\rangle_i \left|-\frac{1}{2}\right\rangle_{i+1}$ $\quad \left|-\frac{1}{2}\right\rangle_i \left|+\frac{1}{2}\right\rangle_{i+1}$ $\quad \left|-\frac{1}{2}\right\rangle_i \left|-\frac{1}{2}\right\rangle_{i+1}$

iTEBD algorithm: What's expH?

Diagonalize

$$\boldsymbol{h} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} +\frac{J}{4} & 0 & 0 & 0 \\ 0 & -\frac{3J}{4} & 0 & 0 \\ 0 & 0 & +\frac{J}{4} & 0 \\ 0 & 0 & 0 & +\frac{J}{4} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$\left|+\frac{1}{2}\right\rangle_i \left|+\frac{1}{2}\right\rangle_{i+1}$ $\left|+\frac{1}{2}\right\rangle_i \left|-\frac{1}{2}\right\rangle_{i+1}$ $\left|-\frac{1}{2}\right\rangle_i \left|+\frac{1}{2}\right\rangle_{i+1}$ $\left|-\frac{1}{2}\right\rangle_i \left|-\frac{1}{2}\right\rangle_{i+1}$

iTEBD algorithm: What's expH?

Compute expH

$$e^{-h\Delta t} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} e^{+\frac{J}{4}\Delta t} & 0 & 0 & 0 \\ 0 & e^{-\frac{3J}{4}\Delta t} & 0 & 0 \\ 0 & 0 & e^{+\frac{J}{4}\Delta t} & 0 \\ 0 & 0 & 0 & e^{+\frac{J}{4}\Delta t} \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$\left|+\frac{1}{2}\right\rangle_i \left|+\frac{1}{2}\right\rangle_{i+1}$ $\left|+\frac{1}{2}\right\rangle_i \left|-\frac{1}{2}\right\rangle_{i+1}$ $\left|-\frac{1}{2}\right\rangle_i \left|+\frac{1}{2}\right\rangle_{i+1}$ $\left|-\frac{1}{2}\right\rangle_i \left|-\frac{1}{2}\right\rangle_{i+1}$

iTEBD algorithm: What's expH?

Ex.3) Fill the non-zero components

$$e^{-\boldsymbol{h}\Delta t} = \begin{pmatrix} ? & 0 & 0 & 0 \\ 0 & ? & ? & 0 \\ 0 & ? & ? & 0 \\ 0 & 0 & 0 & ? \end{pmatrix}$$

$\left|+\frac{1}{2}\right\rangle_i \left|+\frac{1}{2}\right\rangle_{i+1}$ $\quad \left|+\frac{1}{2}\right\rangle_i \left|-\frac{1}{2}\right\rangle_{i+1}$ $\quad \left|-\frac{1}{2}\right\rangle_i \left|+\frac{1}{2}\right\rangle_{i+1}$ $\quad \left|-\frac{1}{2}\right\rangle_i \left|-\frac{1}{2}\right\rangle_{i+1}$

You may get "−0.8859" if you correctly implemented