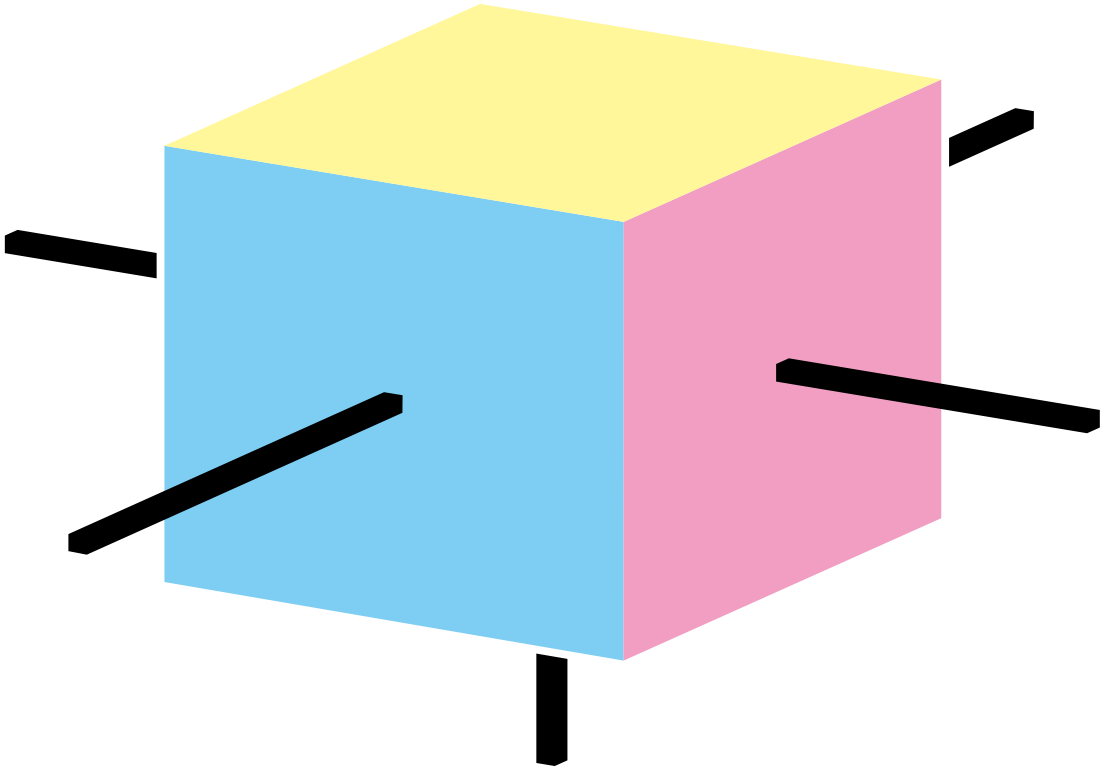


TeNeS

TeNeS Documentation

Release 1.1.0

Institute for Solid State Physics, University of Tokyo

Jul 08, 2020

CONTENTS

CHAPTER

ONE

WHAT IS TENES ?

1.1 Overview

TeNeS (**Te** nsor **Ne** twork **S** olver) is an open-source program package for calculation of two-dimensional many-body quantum states based on the tensor network method. This package calculates ground-state wavefunctions for user-defined Hamiltonian, and evaluates user-defined physical quantities such as magnetization and correlation functions. For predefined models and lattices, there is a tool that makes it easy for users to generate input files. TeNeS uses an OpenMP/MPI hybrid parallelized tensor operation library and thus can deal with large-scale calculation by using massively parallel machines.

1.2 Developers

TeNeS is developed by the following members.

- Tsuyoshi Okubo (Graduate School of Science, Univ. of Tokyo)
- Satoshi Morita (Institute for Solid State Physics, Univ. of Tokyo)
- Yuichi Motoyama (Institute for Solid State Physics, Univ. of Tokyo)
- Kazuyoshi Yoshimi (Institute for Solid State Physics, Univ. of Tokyo)
- Takeo Kato (Institute for Solid State Physics, Univ. of Tokyo)
- Naoki Kawashima (Institute for Solid State Physics, Univ. of Tokyo)

1.3 Version information

- ver. 1.1.0: 2020-07-09.
- ver. 1.0.0: 2020-04-17.
- ver. 1.0-beta: 2020-03-30.
- ver. 0.1: 2019-12-04.

1.4 License

This package is distributed under GNU General Public License version 3 (GPL v3) or later.

1.5 Copyright

© 2019- The University of Tokyo. All rights reserved.

This software was developed with the support of "*Project for advancement of software usability in materials science*" of The Institute for Solid State Physics, The University of Tokyo.

CHAPTER

TWO

INSTALL

2.1 Download

You can download the source code for TeNeS from the GitHub page . If you have git installed on your machine, type the following command to start download:

```
$ git clone https://github.com/issp-center-dev/TeNeS
```

2.2 Prerequisites

The following tools are required for building TeNeS.

1. C++11 compiler
2. CMake (>=3.6.0)

TeNeS depends on the following libraries, but these are downloaded automatically through the build process.

1. mptensor
2. cpptoml
3. sanitizers-cmake

TeNeS can use MPI and ScaLAPACK for parallel operations of tensors. MPI and ScaLAPACK must be installed by yourself. For example, if you use Debian GNU/Linux (or Debian based system such as Ubuntu) and have root priviledges, you can easily install them by the following:

```
sudo apt install openmpi-bin libopenmpi-dev libscalapack-mpi-dev
```

For others, see the official instruction of some MPI implementation and ScaLAPACK.

Python3 is required for the input file generators, `tenes_simple` and `tenes_std` . Additionary, the following python packages are also required.

1. numpy
2. scipy
3. toml

2.3 Install

1. Build TeNeS by typing the following commands:

```
$ mkdir build
$ cd build
$ cmake  -DCMAKE_INSTALL_PREFIX=<path to install to> ..
$ make
```

The default value of the <path to install to> is /usr/local.

(Some environment such as CentOS provides CMake3 as cmake3 .)

The executable file tests will be generated in build/src directory. By typing the following command, tests for tenes can be done.

```
$ make tests
```

2. Install TeNeS by typing the following commands:

```
$ make install
```

In this case, tenes, tenes_std and tenes_simple are installed into the <path to install to>/bin .

Disable MPI/ScaLAPACK parallelization

If you want to disable MPI/ScaLAPACK parallelization, pass -DENABLE_MPI=OFF option to cmake command. On macOS, some functions of ScaLAPACK are incompatible with the system's BLAS and LAPACK, and TeNeS ends in error. It is recommended to disable MPI parallel.

Specify compiler

CMake detects your compiler automatically but sometimes this does not work. In this case, you can specify the compiler by the following way,

```
$ cmake -DCMAKE_CXX_COMPILER=<path to your compiler> ../
```

Use the pre-built mptensor

TeNeS is based on the parallerized tensor library mptensor. The build system of TeNeS installs this automatically, but if you want to use the specific version of the mptensor, please add the following option in cmake.

```
$ cmake -DMPTENSOR_ROOT=<path to mptensor> ../
```

Specify Python interpreter

TeNeS tools (tenes_simple and tenes_std) use python3 interpreter which is found in PATH via /usr/bin/env python3. Please make sure that python3 command invokes the interpreter which you want to use, for example, by using type python3.

If you want to fix the interpreter (or /usr/bin/env does not exist), you can specify the interpreter by the following way,

```
$ cmake -DTENES_PYTHON_EXECUTABLE=<path to your interpreter> ../
```

CHAPTER

THREE

USAGE

`tenes`, the main program of TeNeS, needs an input file to define the model, order of operations, etc. For ease of use to make the input file, the following script is provided (the schematic flow is shown Fig. 3.1):

- `tenes_std` : A tool that generates an input file to execute `tenes`. An input file of `tenes_std` defines a lattice model etc. by yourself according to a predetermined format.
- `tenes_simple`: A tool that generates input files for `tenes_std` from another simpler input file which specifies lattice model predefined.

In order to simulate other models and/or lattices than predefined ones, you should create the input file of `tenes_std` and convert it. Please see *File format* for details on the input files of TeNeS.

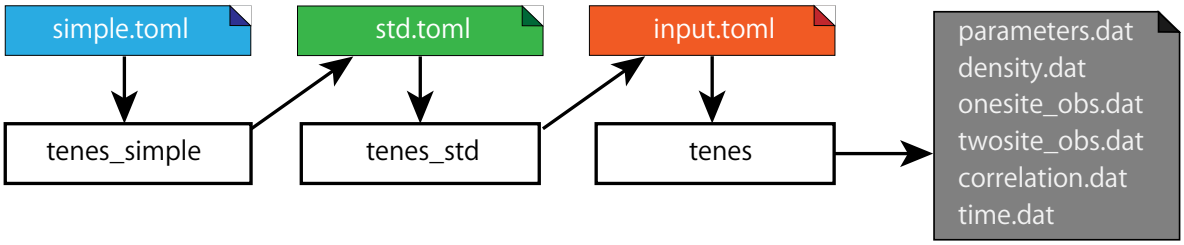


Fig. 3.1: Schematic calculation flow of TeNeS

The following sections describe how to use each script, and finally how to use `tenes`.

3.1 Usage of `tenes_simple`

`tenes_simple` is a tool that creates an input file of `tenes_std` for predefined models and lattices.

```
$ tenes_simple simple.toml
```

- Takes a file as an argument
- Output an input file for `tenes_std`
- Command line options are as follows
  - `--help`
    - Show help message

  - `--version`
    - Show version number
  - `--output=filename`
    - Specify the output file name `filename`
    - Default is `std.toml`
    - File name cannot be the same as the input file name
  - `--coordinatefile=coordfile`
    - Specify the output coordinate file name `coordfile`
    - Default is `coordinates.dat`
    - In a coordinate file, the first, second, and third columns denote site index, x coordinate, and y coordinate (in Cartesian), respectively.

The currently defined models and lattices are as follows:

- Model
  - Spin system
- Lattice
  - Square lattice
  - Triangular lattice
  - Honeycomb lattice
  - Kagome lattice

See *Input file for tenes_simple* for details of the input file. Below, a sample file for the S=1/2 Heisenberg model on the square lattice is shown.

```
[lattice]
type = "square lattice" # type of lattice
L = 2                   # size of unitcell
W = 2                   # size of unitcell
virtual_dim = 3         # bond dimension
initial = "antiferro"   # initial state

[model]
type = "spin" # type of model
J = 1.0       # Heisenberg interaction

[parameter]
[parameter.general]
is_real = true # use real tensor

[parameter.simple_update]
num_step = 1000  # number of steps
tau = 0.01       # imaginary time step

[parameter.full_update]
num_step = 0    # number of steps
tau = 0.01      # imaginary time step

[parameter.ctm]
dimension = 9       # bond dimension
```

3.2 Usage of `tenes_std`

`tenes_std` is a tool to calculate imaginary time evolution operators $\exp(-\tau\mathcal{H}_{ij})$ from a given Hamiltonian $\mathcal{H}$ and an imaginary time step τ, and to generate an input file for `tenes`.

```
$ tenes_std std.toml
```

- Takes a file as an argument
- Output an input file for `tenes`
- **Command line options are as follows**
  - **`--help`**
    - Show help message
  - **`--version`**
    - Show version number
  - **`--output=filename`**
    - Specify the output file name `filename`
    - Default is `input.toml`
    - File name cannot be the same as the input file name

By making and editing input files, users can simulate on other models and lattices than predefined ones. See *Input file for tenes_std* for details of the input file. Below, a sample file for the S=1/2 Heisenberg model on the square lattice is shown.

```
[parameter]
[parameter.general]
is_real = true   # limit tensors as real-valued ones
[parameter.simple_update]
num_step = 1000  # number of steps
tau = 0.01       # imaginary time step
[parameter.full_update]
num_step = 0     # number of steps
tau = 0.01       # imaginary time step
[parameter.ctm]
dimension = 9    # bond dimension

[tensor]
type = "square lattice"
L_sub = [2, 2]   # unitcell size
skew = 0         # boundary condition

# tensors in unitcell
[[tensor.unitcell]]
index = [0, 3]   # index of tensors
physical_dim = 2 # physical bond dimension
virtual_dim = [3, 3, 3, 3]
                 # virtual bond dimension
noise = 0.01     # noise in initial tensor
initial_state = [1.0, 0.0]
                 # initial state

[[tensor.unitcell]]
```

(continues on next page)

(continued from previous page)

```
index = [1, 2]
physical_dim = 2
virtual_dim = [3, 3, 3, 3]
noise = 0.01
initial_state = [0.0, 1.0]


# (bond) hamiltonian
[[hamiltonian]]
dim = [2, 2]    # physical bond dimensions
bonds = """     # bond information
0 1 0           # first: index of one site
1 1 0           # second: x coord of the other
2 1 0           # third:  y coord of the other
3 1 0
0 0 1
1 0 1
2 0 1
3 0 1
"""
elements = """     # nonzero elements of tensor
0 0 0 0 0.25 0.0   # first:  initial state of one site
1 0 1 0 -0.25 0.0  # second: initial state of the other
0 1 1 0 0.5 0.0    # third:  final state of one site
1 0 0 1 0.5 0.0    # fourth: final state of the other
0 1 0 1 -0.25 0.0  # fifth:  real part
1 1 1 1 0.25 0.0   # sixth:  imag part
"""

# observables
[observable]
[[observable.onesite]]
name = "Sz"        # name
group = 0          # index
sites = []         # sites to be acted
dim = 2            # dimension
elements = """     # nonzero elements
0 0 0.5 0.0
1 1 -0.5 0.0
"""

[[observable.twosite]]
name = "hamiltonian"
group = 0
dim = [2, 2]
bonds = """
0 1 0
1 1 0
2 1 0
3 1 0
0 0 1
1 0 1
2 0 1
3 0 1
"""
elements = """
0 0 0 0 0.25 0.0
```

(continues on next page)

(continued from previous page)

```
1 0 1 0 -0.25 0.0
0 1 1 0 0.5 0.0
1 0 0 1 0.5 0.0
0 1 0 1 -0.25 0.0
1 1 1 1 0.25 0.0
"""

[[observable.twosite]]
name = "SzSz"
group = 1
dim = [2, 2]
bonds = """
0 1 0
1 1 0
2 1 0
3 1 0
0 0 1
1 0 1
2 0 1
3 0 1
"""
ops = [0, 0]  # index of onesite operators
```

3.3 Usage of `tenes`

`tenes` is the main program of TeNeS.

```
$ tenes input.toml
```

- Take the input file name as an argument
- The command line options are:
  - `--help` - Show help messages.
  - `--version` - Show the version number.
  - `--quiet` - Do not print any messages to the standard output.

In many cases, users do not have to edit the input file directly. See *Input file for tenes* for details of the input file.

CHAPTER

FOUR

TUTORIAL

4.1 Ising model with transverse magnetic field

This section presents a calculation of the transverse magnetic field Ising model as an example. By changing the variable `G` in the input file, the magnitude of the transverse magnetic field will be modified. For example, when the transverse magnetic field is 0, the input file is

```
[parameter]
[parameter.general]
is_real = true

[parameter.simple_update]
num_step = 1000
tau = 0.01

[parameter.full_update]
num_step = 0
tau = 0.01

[parameter.ctm]
iteration_max = 10
dimension = 10

[lattice]
type = "square lattice"
L = 2
W = 2
virtual_dim = 2
initial = "ferro"

[model]
type = "spin"
Jz = -1.0
Jx = 0.0
Jy = 0.0
G  = 0.0
```

In this case, since `Jz = -1.0` , the ferromagnetic state manifests itself as the ground state at `G=0`. When the input file name is `simple.toml` , type the following commands to execute `tenes` (before typing them, please install TeNeS and set PATH properly.):

```
$ tenes_simple simple.toml
$ tenes_std std.toml
$ tenes input.toml
```

Then, the following logs are output:

```
Number of Processes: 1
Number of Threads / Process: 1
Tensor type: real
Start simple update
10% [100/1000] done
20% [200/1000] done
30% [300/1000] done
40% [400/1000] done
50% [500/1000] done
60% [600/1000] done
70% [700/1000] done
80% [800/1000] done
90% [900/1000] done
100% [1000/1000] done
Start calculating observables
Start updating environment
Start calculating onesite operators
Save onesite observables to output_0/onesite_obs.dat
Start calculating twosite operators
Save twosite observables to output_0/twosite_obs.dat
Save observable densities to output_0/density.dat
Save elapsed times to output_0/time.dat

Onesite observables per site:
Sz          = 0.5 0
Sx          = -1.28526262482e-13 0
Twosite observables per site:
hamiltonian = -0.5 0
SzSz        = 0.5 0
SxSx        = -1.7374919982e-18 0
SySy        = 1.73749202733e-18 0
Wall times [sec.]:
simple update = 3.545813509
full update   = 0
environmnent  = 0.123170523
observable    = 0.048149856

Done.
```

First, the information of parallelization and the tensors (complex or not) is displayed. Next, the execution status of the calculation process is displayed. After finishing the calculation, the expected values per site of the one-site operators `Sz`, `Sx` and two-site ones Hamiltonian, the nearest correlation `SzSz`, `SxSx`, `SySy` are output. Finally, the calculation time for each process is output in units of seconds. `density.dat`, `parameters.dat`, `time.dat`, `onesite_obs.dat`, and `twosite_obs.dat` are saved to the output directory. For details on each output file, see *Output files*. For example, the value of `<Sz>` can be read from `onesite_obs.dat`. By changing `G` in increments of 0.2 from 0 to 3.0 and running `tenes_simple` and `tenes`, the following result is obtained. As an example of the sample script, `tutorial_example.py` , `tutorial_read.py` are prepared in the `sample/01_transverse_field_ising` directory.

- `tutorial_example.py`

```
import subprocess

import numpy as np

import toml
```

(continues on next page)

(continued from previous page)

```
num_hx = 16
min_hx = 0.0
max_hx = 3.0

total = 0
for idx, hx in enumerate(np.linspace(min_hx, max_hx, num=num_hx)):
    print("Caclulation Process: {}/{}".format(idx+1, num_hx))
    with open("simple.toml") as f:
        dict_toml = toml.load(f)
    dict_toml["parameter"]["general"]["output"] = "output_{}".format(idx)
    dict_toml["model"]["hx"] = float(hx)
    with open("simple_{}.toml".format(idx), 'w') as f:
        toml.dump(dict_toml, f)
    cmd = "tenes_simple simple_{}.toml -o std_{}.toml".format(idx, idx)
    subprocess.call(cmd.split())
    cmd = "tenes_std std_{}.toml -o input_{}.toml".format(idx, idx)
    subprocess.call(cmd.split())
    cmd = "tenes input_{}.toml".format(idx)
    subprocess.call(cmd.split())
```

- `tutorial_read.py`

```
from os.path import join

import numpy as np

import toml


num_hx = 16

for idx in range(num_hx):
    try:
        with open("simple_{}.toml".format(idx)) as f:
            dict_toml = toml.load(f)
        hx = dict_toml["model"]["hx"]
        ene = 0.0
        mag_sz = 0.0
        mag_sx = 0.0
        with open(join("output_{}".format(idx), "density.dat")) as f:
            for line in f:
                words = line.split()
                if words[0] == 'hamiltonian':
                    ene = words[2]
                elif words[0] == 'Sz':
                    mag_sz = words[2]
                elif words[0] == 'Sx':
                    mag_sx = words[2]
        print("{} {} {} {}".format(hx, ene, mag_sz, mag_sx))
    except:
        continue
```

The calculation will be done by typing the following command:

```
$ python tutorial_example.py
```

For MacBook2017 (1.4 GHz Intel Core i7), the calculation was finished in a few minutes. By typing the following command, G, energy, <Sz> and <Sx> are outputted in the standard output:

```
$ python tutorial_read.py
```

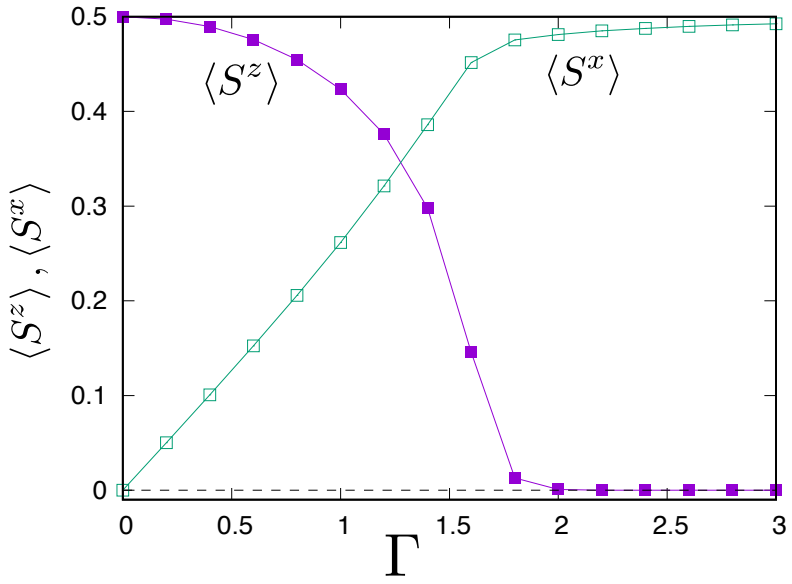


Fig. 4.1: G dependence of <Sz> and <Sx>.

As seen from Fig. 4.1 , with increasing G, the <Sz> decreases from 0.5 to 0, while the <Sx> increases from 0 to 0.5.

4.2 Magnetization process of the Heisenberg model on triangular and square lattices

Next, we introduce the calculation of the magnetization process of the quantum Heisenberg model with spin $S = 1/2$ defined on a triangular lattice. The Hamiltonian looks like this:

$$H = J \sum_{\langle i,j \rangle} \sum_{\alpha}^{x,y,z} S_i^\alpha S_j^\alpha - h \sum_i S_i^z$$

Here, $\langle i, j \rangle$ represents the pair of nearest neighbor sites, and h represents the magnitude of the external magnetic field applied in the z direction. Let's calculate the ground state of this model and find $\langle S_z \rangle \equiv \frac{1}{N_u} \sum_i^{N_u} \langle S_i^z \rangle$, where N_u is the total number of sites in the unit cell, as a function of the magnetic field h. To do this, the toml file basic.toml and the python script tutorial_magnetization.py are prepared in the sample/05_magnetization directory. The basic.toml file contains model settings and parameters.

```
[parameter]
[parameter.general]
is_real = true
```

(continues on next page)

(continued from previous page)

```
[parameter.simple_update]
num_step = 200
tau = 0.01

[parameter.full_update]
num_step = 0
tau = 0.01

[parameter.ctm]
iteration_max = 100
dimension = 10

[lattice]
type = "triangular lattice"
L = 3
W = 3
virtual_dim = 2
initial = "random"

[model]
type = "spin"
J = 1.0
```

The `lattice` section specifies a triangular lattice with the unit cell size of 3×3. Here, in order to make the calculation lighter, only `simple update` is performed, and the imaginary time interval τ is assumed to be $\tau = 0.01$. For simplicity, $J = 1$. Using this basic setting file, tutorial_magnetization.py calculates the magnetization when the magnetic field is swept.

```
import subprocess
from os.path import join
import numpy as np
import toml

num_h = 21
min_h = 0.0
max_h = 5.0
num_step_table = [100, 200, 500, 1000, 2000]

fmag = open("magnetization.dat","w")
fene = open("energy.dat","w")
for idx, h in enumerate(np.linspace(min_h, max_h, num=num_h)):
    print("Caclulation Process: {}/{}".format(idx+1, num_h))
    inum = 0
    num_pre = 0
    fmag.write("{} ".format(h))
    fene.write("{} ".format(h))
    for num_step in num_step_table:
        ns = num_step - num_pre
        print("Steps: {}".format(num_step))
        with open("basic.toml") as f:
            dict_toml = toml.load(f)
        dict_toml["parameter"]["general"]["output"] = "output_{}_{}".format(idx,num_
↪step)
        dict_toml["parameter"]["general"]["tensor_save"] = "tensor_save".format(idx,
↪num_step)
        dict_toml["model"]["hz"] = float(h)
```

(continues on next page)

(continued from previous page)

```
        dict_toml["parameter"]["simple_update"]["num_step"] = ns
        if inum > 0:
            dict_toml["parameter"]["general"]["tensor_load"] = "tensor_save".
↪format(idx,num_pre)
        with open("simple_{}_{}.toml".format(idx,num_step), 'w') as f:
            toml.dump(dict_toml, f)
        cmd = "tenes_simple simple_{}_{}.toml -o std_{}_{}.toml".format(idx,num_step,
↪idx,num_step)
        subprocess.call(cmd.split())
        cmd = "tenes_std std_{}_{}.toml -o input_{}_{}.toml".format(idx,num_step,idx,
↪num_step)
        subprocess.call(cmd.split())
        cmd = "tenes input_{}_{}.toml".format(idx,num_step)
        subprocess.call(cmd.split())
        with open(join("output_{}_{}".format(idx,num_step), "density.dat")) as f:
            lines = f.readlines()
            mag_sz = lines[0].split('=')[1].strip()
            ene = lines[2].split('=')[1].strip()
        fene.write("{} ".format(ene))
        fmag.write("{} ".format(mag_sz))
        inum = inum + 1
        num_pre = num_step
    fene.write("\n")
    fmag.write("\n")
fene.close()
fmag.close()
```

In this script, the magnetic field h is changed in steps of 0.25 from 0 to 5, and the ground state energy and $\langle S_z \rangle$ are calculated and output to `energy.dat` and `magnetization.dat`, respectively. In order to see what happens when the number of time steps for simple update is changed, calculations are also performed with 100, 200, 500, 1000, and 2000 steps for each magnetic field. In order to reduce the amount of calculation, the information of the wave function obtained with a small number of steps is stored in `tensor_save`, and this is used as the initial state for the calculation of a larger number of steps. For example, the python script first performs a calculation with the number of time steps set to 100, and output the result. Then, it perform a calculation with the number of time steps set to 200 using the wave function at the end of the calculation of the number of steps 100. The script consequently reduce the amount of the calculation by 100 steps for the latter in the directory.

Let's actually run it. After passing through a path to tenes in advance, execute calculation by typing as follows.

```
python tutorial_magnetization.py
```

The calculation will finish within a few hours if you use a notebook PC using a single processor. After the calculation is completed, start up gnuplot and type

```
load 'plot.gp'
```

to obtain the magnetization curve as shown in the right panel of Fig. 4.2. In a similar way,

```
load 'plot_ene.gp'
```

we obtain the ground-state energy as shown in the left panel of Fig. 4.2 .

As can be seen from the result for a sufficiently large number of steps (for example, 2000 steps), a plateau structure occurs in the magnetization process at the magnetization of $1/3$ of the saturation magnetization $\langle S_z \rangle = 0.5$. On this plateau, spins on the three lattices form a periodic magnetic structure with $\uparrow$, $\uparrow$, $\downarrow$, and a spin gap is generated. This plateau structure is unique to the triangular lattice. To see whether the accuracy of calculation is enough or not, it is helpful to check the step dependence of energy. In principle, the ground-state energy should decrease as the number

of steps increases, but in some magnetic fields, the calculated energy increases. This is a sign that the calculation accuracy is not good. It is presumed that it is necessary to increase the bond dimension.

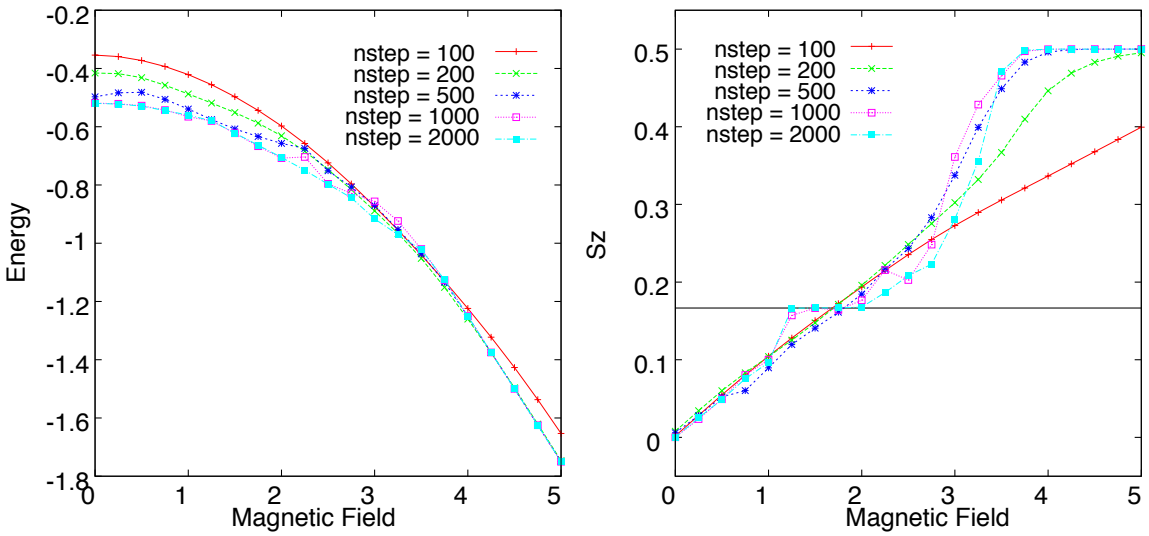


Fig. 4.2: Ground state energy (left figure) and magnetization (right figure) of the Heisenberg model on the triangular lattice.

Next, let's perform the calculation for a model on a square lattice. Use the toml file `basic_square.toml` and the python script `tutorial_magnetization_square.py` in the `sample/05_magnetization` directory. The content of `basic_square.toml` is the same as `basic.toml` except that the `lattice` section has been changed as follows.

```
[lattice]
type = "square lattice"
L = 2
W = 2
```

To perform the calculation, type:

```
python tutorial_magnetization_square.py
```

After the calculation is completed, start up gnuplot and type

```
load 'plot_square.gp'
```

Then, the magnetization curve shown in the right panel of Fig. 4.3 is obtained. In a similar way, by typing the following command,

```
load 'plot_ene_square.gp'
```

you will obtain the ground-state energy as shown in the left panel of Fig. 4.3. The calculation is almost converged at 2000 steps, and it can be seen that the plateau structure does not appear unlike the triangular lattice Heisenberg model. Since the energy generally decreases as the number of steps is increased, it is assumed that the calculation accuracy is sufficiently high.

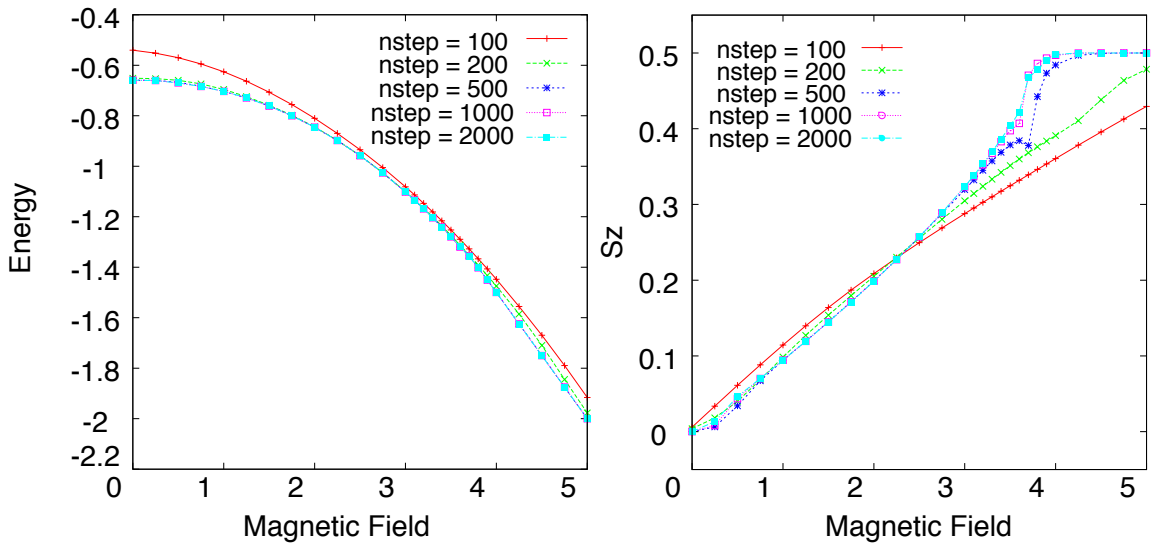


Fig. 4.3: Ground state energy (left figure) and magnetization (right figure) of the Heisenberg model on the square lattice.

CHAPTER

FIVE

FILE FORMAT

5.1 Short summary for input files of TeNeS

Input files of TeNeS is written in TOML format and each file has some sections. `tenes_simple` and `tenes_std` read some sections and generate an input file for `tenes_std` and `tenes`, respectively. `tenes` reads some sections and performs simulation.

For example, `tenes_simple` reads `model` and `lattice` sections and generates `tensor`, `observable`, and `hamiltonian` ones. Additionary, this copies `parameter` and `correlation` sections.

The following table summarizes how each tool deal with sections.

Section	`tenes_simple`	`tenes_std`	`tenes`
`parameter`	copy	in / copy	in
`model`	in		
`lattice`	in		
`tensor`	out	in / copy	in
`observable`	out	copy	in
`correlation`	copy	copy	in
`hamiltonian`	out	in	
`evolution`		out	in

- "in"
  - Tool uses this section as input
- "out"
  - Tool generates this section in output (= next input)
- "copy"
  - Tool copies this section into output (= next input)

5.2 Input file for `tenes_simple`

- File format is TOML format.
- The input file has four sections : `model`, `parameter`, `lattice`, `correlation` .
  - The `parameter` section is copied to the standard mode input.

5.2.1 `model` section

Specify the model to calculate. In this version, spin system (`"spin"`) and bosonic system (`"boson"`) are defined.

Name	Description	Type	Default
`type`	Model type (“spin” or “boson”)	String	–

The parameter names such as interactions depend on the model type.

Spin system: `"spin"`

Hamiltonian is described as

$$\mathcal{H} = \sum_{\langle ij \rangle} \left[\sum_{\alpha}^{x,y,z} J_{ij}^{\alpha} S_i^{\alpha} S_j^{\alpha} + B \left(\vec{S}_i \cdot \vec{S}_j \right)^2 \right] - \sum_i \sum_{\alpha}^{x,y,z} h^{\alpha} S_i^{\alpha} - \sum_i D \left(S_i^z \right)^2$$

The parameters of the one-body terms are defined as follows.

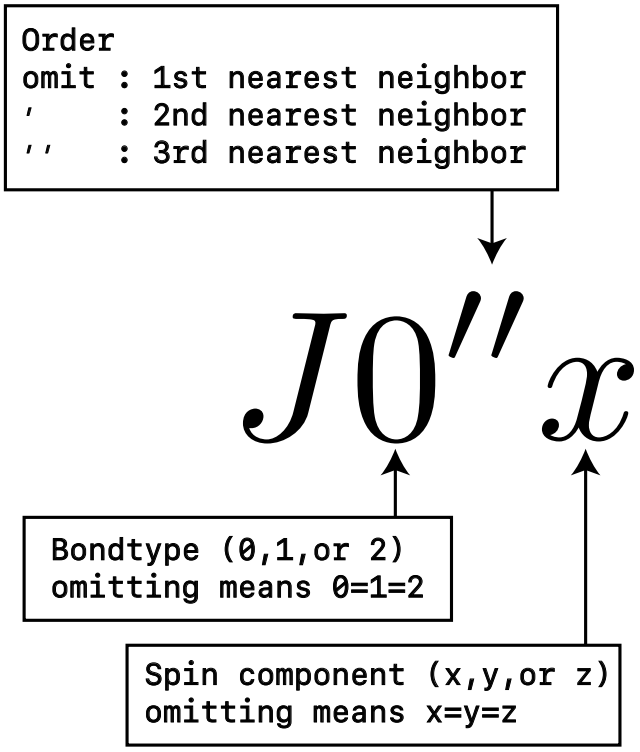
Name	Description	Type	Default
`S`	Magnitude of the local spin	Real (integer or half integer)	0.5
`hx`	Magnetic field along S^x, h^x	Real	0.0
`hy`	Magnetic field along S^y, h^y	Real	0.0
`hz`	Magnetic field along S^z, h^z	Real	0.0
`D`	On-site spin anisotropy D	Real	0.0

The exchange interaction J can have a bond dependency.

Name	Description	Type	Default
`J0`	Exchange interaction of **0th** direction **nearest neighbor** bond	Real	0.0
`J1`	Exchange interaction of **1st** direction **nearest neighbor** bond	Real	0.0
`J2`	Exchange interaction of **2nd** direction **nearest neighbor** bond	Real	0.0
`J0'`	Exchange interaction of **0th** direction **next nearest neighbor** bond	Real	0.0
`J1'`	Exchange interaction of **1st** direction **next nearest neighbor** bond	Real	0.0
`J2'`	Exchange interaction of **2nd** direction **next nearest neighbor** bond	Real	0.0
`J0''`	Exchange interaction of **0th** direction **third nearest neighbor** bond	Real	0.0
`J1''`	Exchange interaction of **1st** direction **third nearest neighbor** bond	Real	0.0
`J2''`	Exchange interaction of **2nd** direction **third nearest neighbor** bond	Real	0.0

The bond direction depends on the lattice defined in the `lattice` section. For a square lattice, for example, coupling constants along two bond directions can be defined, x-direction (0) and y-direction (1). By omitting the direction number, you can specify all directions at once. You can also specify Ising-like interaction by adding one character of *xyz* at the end. If the same bond or component is specified twice or more, an error will occur.

To summarize,



The biquadratic interaction B can also have a bond dependency like as J.

Name	Description	Type	Default
`B0`	Biquadratic interaction of **0th** direction **nearest neighbor** bond	Real	0.0
`B1`	Biquadratic interaction of **1st** direction **nearest neighbor** bond	Real	0.0
`B2`	Biquadratic interaction of **2nd** direction **nearest neighbor** bond	Real	0.0
`B0'`	Biquadratic interaction of **0th** direction **next nearest neighbor** bond	Real	0.0
`B1'`	Biquadratic interaction of **1st** direction **next nearest neighbor** bond	Real	0.0
`B2'`	Biquadratic interaction of **2nd** direction **next nearest neighbor** bond	Real	0.0
`B0''`	Biquadratic interaction of **0th** direction **third nearest neighbor** bond	Real	0.0
`B1''`	Biquadratic interaction of **1st** direction **third nearest neighbor** bond	Real	0.0
`B2''`	Biquadratic interaction of **2nd** direction **third nearest neighbor** bond	Real	0.0

One-site operators S^z and S^x are automatically defined. If `parameter.general.is_real = false`, S^y is also defined. In addition, bond Hamiltonian

$$\mathcal{H}_{ij} = \left[\sum_{\alpha}^{x,y,z} J_{ij}^{\alpha} S_i^{\alpha} S_j^{\alpha} + B\left(\vec{S}_i \cdot \vec{S}_j\right)^2\right] - \frac{1}{z}\left[\sum_{\alpha}^{x,y,z} h^{\alpha}\left(S_i^{\alpha} + S_j^{\alpha}\right) + D\left(\left(S_i^z\right)^2 + \left(S_j^z\right)^2\right)\right],$$

and spin correlations on nearest neighbor bonds $S_i^{\alpha} S_j^{\alpha}$ ($\alpha = x, y, z$) are automatically defined as two-site operators. In the bond Hamiltonian, one body terms (h^{α} and D term) appear only in the nearest neighbor bonds, and z is the number of the coodinate number.

Bosonic system: `"boson"`

Hamiltonian is described as

$$\mathcal{H} = \sum_{i<j} \left[-t_{ij} \left(b_i^\dagger b_j + b_j^\dagger b_i \right) + V_{ij} n_i n_j \right] + \sum_i \left[U \frac{n_i(n_i-1)}{2} - \mu n_i \right],$$

where $b^\dagger$ and b are the creation and the annihilation operators of a boson, and $n = b^\dagger b$ is the number operator.

The parameters of the one-body terms are defined as follows.

Name	Description	Type	Default
`nmax`	Maximum number of particles on a site	Integer	1
`U`	Onsite repulsion	Real	0.0
`mu`	Chemical potential	Real	0.0

The hopping constant t and the offsite repulsion V can have a bond dependency.

Name	Description	Type	Default
`t0`	Hopping of **0th** direction **nearest neighbor** bond	Real	0.0
`t1`	Hopping of **1st** direction **nearest neighbor** bond	Real	0.0
`t2`	Hopping of **2nd** direction **nearest neighbor** bond	Real	0.0
`t0'`	Hopping of **0th** direction **next nearest neighbor** bond	Real	0.0
`t1'`	Hopping of **1st** direction **next nearest neighbor** bond	Real	0.0
`t2'`	Hopping of **2nd** direction **next nearest neighbor** bond	Real	0.0
`t0''`	Hopping of **0th** direction **third nearest neighbor** bond	Real	0.0
`t1''`	Hopping of **1st** direction **third nearest neighbor** bond	Real	0.0
`t2''`	Hopping of **2nd** direction **third nearest neighbor** bond	Real	0.0
`V0`	Offsite repulsion of **0th** direction **nearest neighbor** bond	Real	0.0
`V1`	Offsite repulsion of **1st** direction **nearest neighbor** bond	Real	0.0
`V2`	Offsite repulsion of **2nd** direction **nearest neighbor** bond	Real	0.0
`V0'`	Offsite repulsion of **0th** direction **next nearest neighbor** bond	Real	0.0
`V1'`	Offsite repulsion of **1st** direction **next nearest neighbor** bond	Real	0.0
`V2'`	Offsite repulsion of **2nd** direction **next nearest neighbor** bond	Real	0.0
`V0''`	Offsite repulsion of **0th** direction **third nearest neighbor** bond	Real	0.0
`V1''`	Offsite repulsion of **1st** direction **third nearest neighbor** bond	Real	0.0
`V2''`	Offsite repulsion of **2nd** direction **third nearest neighbor** bond	Real	0.0

The bond direction depends on the lattice defined in the `lattice` section. For a square lattice, for example, coupling constants along two bond directions can be defined, x-direction (0) and y-direction (1). By omitting the direction number, you can specify all directions at once.

One-site operators n, b, and $b^\dagger$ are automatically defined. In addition, bond Hamiltonian

$$\mathcal{H}_{ij} = \left[-t_{ij} \left(b_i^\dagger b_j + b_j^\dagger b_i \right) + V_{ij} n_i n_j \right] + \frac{1}{z} \left[\left(U \frac{n_i(n_i-1)}{2} - \mu n_i \right) + (i \leftrightarrow j) \right]$$

and short range correlations on nearest neighbor bonds $n_i n_j$, $b_i^\dagger b_j$, and $b_i b_j^\dagger$ are automatically defined as two-site operators. In the bond Hamiltonian, one body terms (U and μ term) appear only in the nearest neighbor bonds, and z is the number of the coodinate number.

5.2.2 `lattice` section

Specify the lattices to calculate. Square, triangular, honeycomb, and Kagome lattices are defined.

Name	Description	Type	Default
`type`	lattice name (square, triangular or honeycomb lattice)	String	–
`L`	Unit cell size in x direction	Integer	–
`W`	Unit cell size in y direction	Integer	`L`
`virtual_dim`	Bond dimension	Integer	–
`initial`	Inital state	String	random
`noise`	Noise for elements in initial tensor	Real	1e-2

`initial` and `noise` are parameters that determine the initial state of the wave function. If `tensor_load` is set in `parameter.general`, `initial` is ignored.

- `initial`
  - `"ferro"` : Ferromagnetic state
    - In spin system, all sites has $S^z = S$
    - In bosonic system, all sites has $n = n_{\max}$ particles
  - `"antiferro"` : Antiferromagnetic state
    - In spin system, for square lattice and honeycomb lattice, the Neel order state ($S^z = S$ for the A sublattice and $S^z = -S$ for the B sublattice), and for triangular lattice and kagome lattice, the 120 degree order state (spins on sites belonging to the A, B, and C sublattice are pointing to $(\theta, \phi) = (0, 0), (2\pi/3, 0)$ and $(2\pi/3, \pi)$ direction, respectively.)
    - In bosonic system, sites belonging to one sublattice have $n_{\max}$ particles and the other sites have no particles.
  - `"random"` : Random state
- `noise`
  - The amount of fluctuation in the elements of the initial tensor

Square lattice

A square lattice `type = "square lattice"` consists of `L` sites in the $(1, 0)$ direction and `W` sites in the $(0, 1)$ direction. As a concrete example, Fig. 5.1 (a) shows the structure for `L=3, W=3`. In addition, the definitions of the first, second and third nearest neighbor bonds are shown in Fig. 5.1 (b), (c), and (d), respectively. The blue line represents a bond of `bondtype = 0` and the red line represents a bond of `bondtype = 1`.

Triangular lattice

A triangular lattice `type = "triangular lattice"` consists of `L` sites in the $(1, 0)$ direction and `W` sites in the $(1/2, \sqrt{3}/2)$ direction. As a concrete example, Fig. 5.2 (a) shows the structure for `L=3, W=3`. In addition, the definitions of the first, second and third nearest neighbor bonds are shown in Fig. 5.2 (b), (c), and (d), respectively. The blue, red, and green lines represent bonds of `bondtype = 0, 1`, and `2`, respectively. (e) shows the corresponding square TPS with `L=3, W=3`.

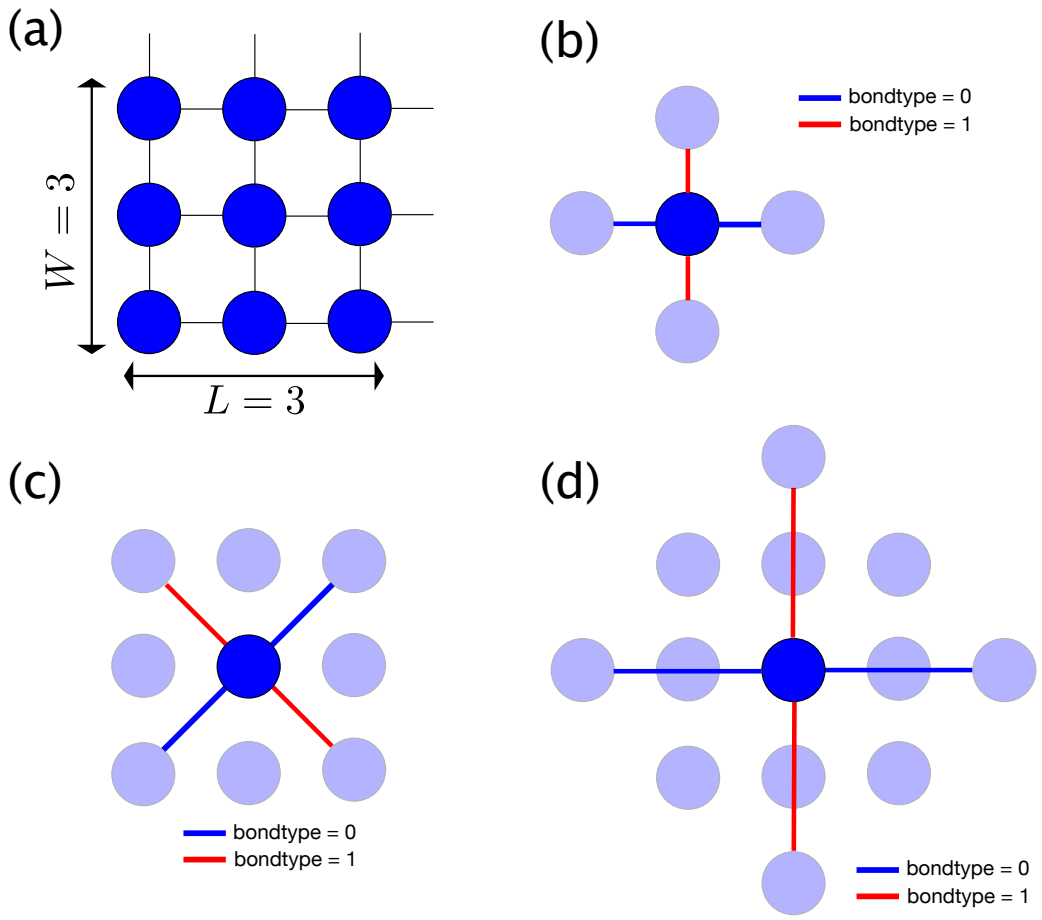


Fig. 5.1: Square lattice. (a) Site structure with `L=3, W=3` (b) Nearest neighbor bonds. `bondtype=0` (blue) bond extends in the 0 degree direction and `bondtype=1` (red) one in the 90 degree direction. (c) Second nearest neighbor bonds. `bondtype=0` (blue) bond extends in the 45 degree direction and `bondtype=1` (red) one in the -45 degree direction. (d) Third nearest neighbor bonds. `bondtype=0` (blue) bond extends in the 0 degree direction and `bondtype=1` (red) one in the 90 degree direction.

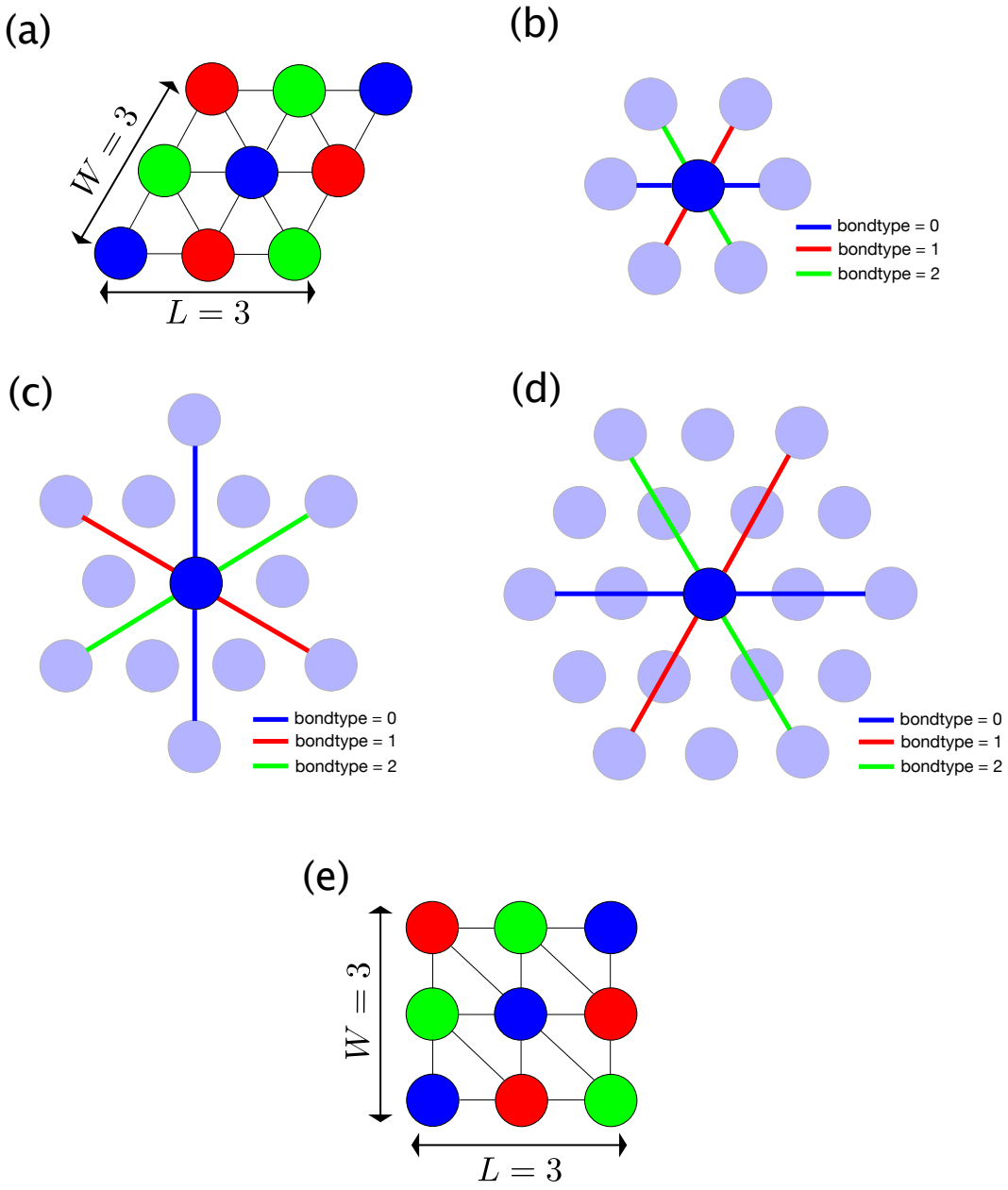


Fig. 5.2: Triangular lattice. (a) Site structure with `L=3, W=3` (b) Nearest neighbor bonds. `bondtype=0` (blue) bond extends in the 0 degree direction, `bondtype=1` (red) one in the 60 degree direction, and `bondtype=2` (green) one in the 120 degree direction. (c) Second nearest neighbor bonds. `bondtype=0` (blue) bond extends in the 90 degree direction, `bondtype=1` (red) one in the -30 degree direction, and `bondtype=2` (green) one in the 30 degree direction. (d) Third nearest neighbor bonds. `bondtype=0` (blue) bond extends in the 0 degree direction, `bondtype=1` (red) one in the 60 degree direction, and `bondtype=2` (green) one in the 120 degree direction. (e) Correnssponding square TPS of the triangular lattice with `L=3, W=3`.

Honeycomb lattice

In a honeycomb lattice `type = "honeycomb lattice"`, units consisting of two sites of coordinates $(0, 0)$ and $(\sqrt{3}/2, 1/2)$ are arranged with `L` units in the $(\sqrt{3}, 0)$ direction and `W` units in the $(1/2, 3/2)$ direction. As a concrete example, Fig. 5.3 (a) shows the structure for `L=2, W=2`. In addition, the definitions of the first, second and third nearest neighbor bonds are shown in Fig. 5.3 (b), (c), and (d), respectively. The blue, red, and green lines represent bonds of `bondtype = 0, 1`, and `2`, respectively. (e) shows the corresponding square TPS with `L=2, W=2`.

Kagome lattice

In a kagome lattice `type = "kagome lattice"`, units consisting of three sites of coordinates $(0, 0)$, $(1, 0)$, and $(1/2, \sqrt{3}/2)$ are arranged with `L` units in the $(2, 0)$ direction and `W` units in the $(1, \sqrt{3})$ direction. As a concrete example, Fig. 5.4 (a) shows the structure for `L=2, W=2`. In addition, the definitions of the first, second and third nearest neighbor bonds are shown in Fig. 5.4 (b), (c), and (d), respectively. The blue and the red lines represent bonds of `bondtype = 0`, and `1`, respectively. (e) shows the corresponding square TPS with `L=2, W=2`.

5.2.3 `parameter` section

Parameters defined in this section is not used in `tenes_simple` but they are copied to the input file of `tenes_std`.

Set various parameters that appear in the calculation, such as the number of updates. This section has five subsections: `general`, `simple_update`, `full_update`, `ctm`, `random`.

Imaginary-time step τ for simple update `parameter.simple_update.tau` and that for full update `parameter.full_update.tau` are used only in standard mode `tenes_std`, not used in `tenes`.

`parameter.general`

General parameters for `tenes`.

Name	Description	Type	Default
`is_real`	Whether to limit all tensors to real valued ones	Boolean	false
`iszero_tol`	Absolute cutoff value for reading operators	Real	0.0
`measure`	Whether to calculate and save observables	Boolean	true
`output`	Directory for saving result such as physical quantities	String	“output”
`tensor_save`	Directory for saving optimized tensors	String	“”
`tensor_load`	Directory for loading initial tensors	String	“”

- `is_real`
  - When set to `true`, the type of elements of the tensor becomes real.
  - If one complex operator is defined at least, calculation will end in errors before starting.
- `iszero_tol`
  - When the absolute value of operator elements loaded is less than `iszero_tol`, it is regarded as zero
- `meaure`
  - When set to `false`, the stages for measuring and saving observables will be skipped
  - Elapsed time `time.dat` is always saved
- `output`

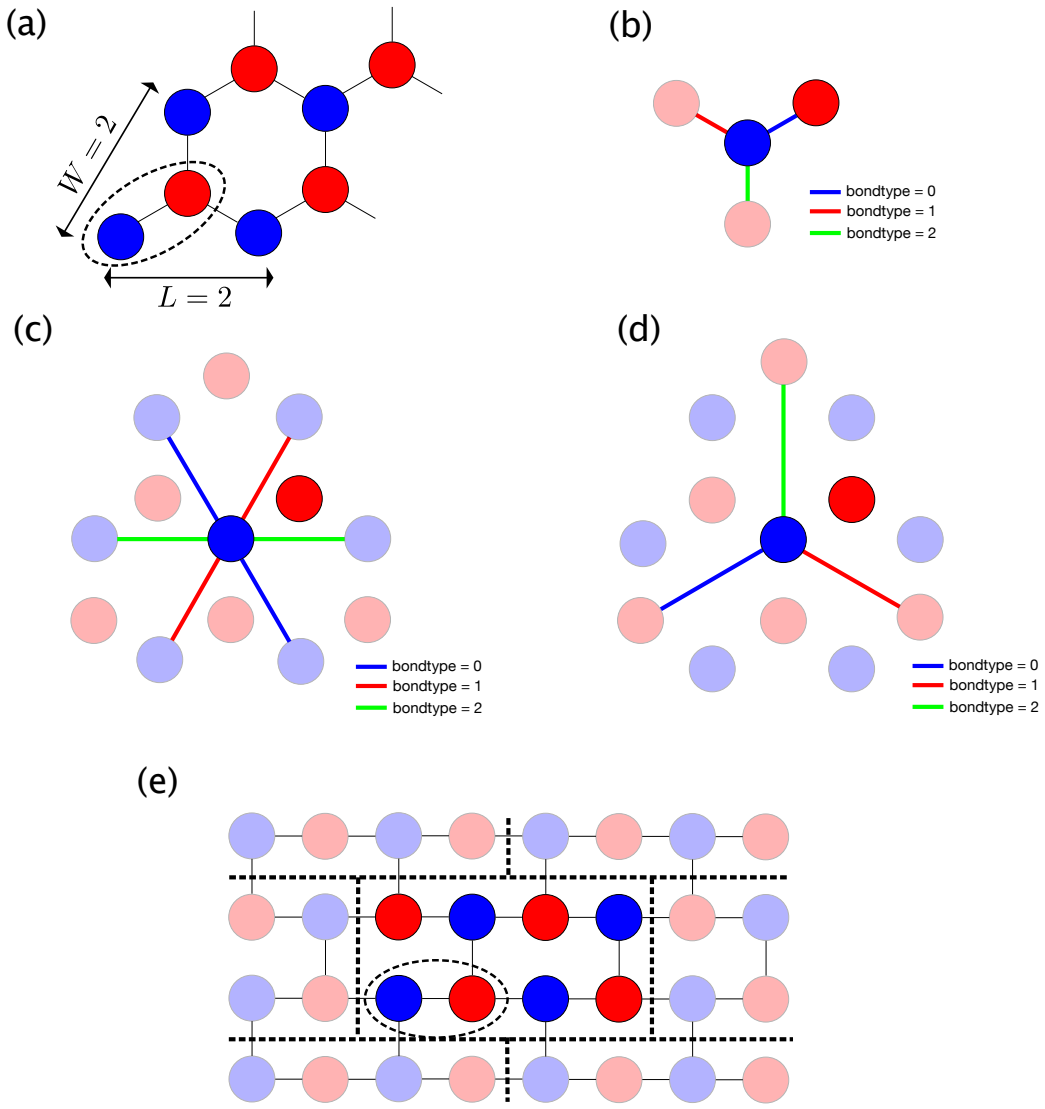


Fig. 5.3: Honeycomb lattice. (a) Site structure with `L=2, W=2`. The dashed ellipse denotes one unit. (b) Nearest neighbor bonds. `bondtype=0` (blue) bond extends in the 30 degree direction, `bondtype=1` (red) one in the 150 degree direction, and `bondtype=2` (green) one in the -90 degree direction. (c) Second nearest neighbor bonds. `bondtype=0` (blue) bond extends in the 120 degree direction, `bondtype=1` (red) one in the 60 degree direction, and `bondtype=2` (green) one in the 0 degree direction. (d) Third nearest neighbor bonds. `bondtype=0` (blue) bond extends in the -30 degree direction, `bondtype=1` (red) one in the -150 degree direction, and `bondtype=2` (green) one in the 90 degree direction. (e) Corrensponding square TPS of the honeycomb lattice with `L=2, W=2`. Note that the most top-right red tensor in the honeycomb lattice moves to the most top-left position, and the boundary condition is skewed.

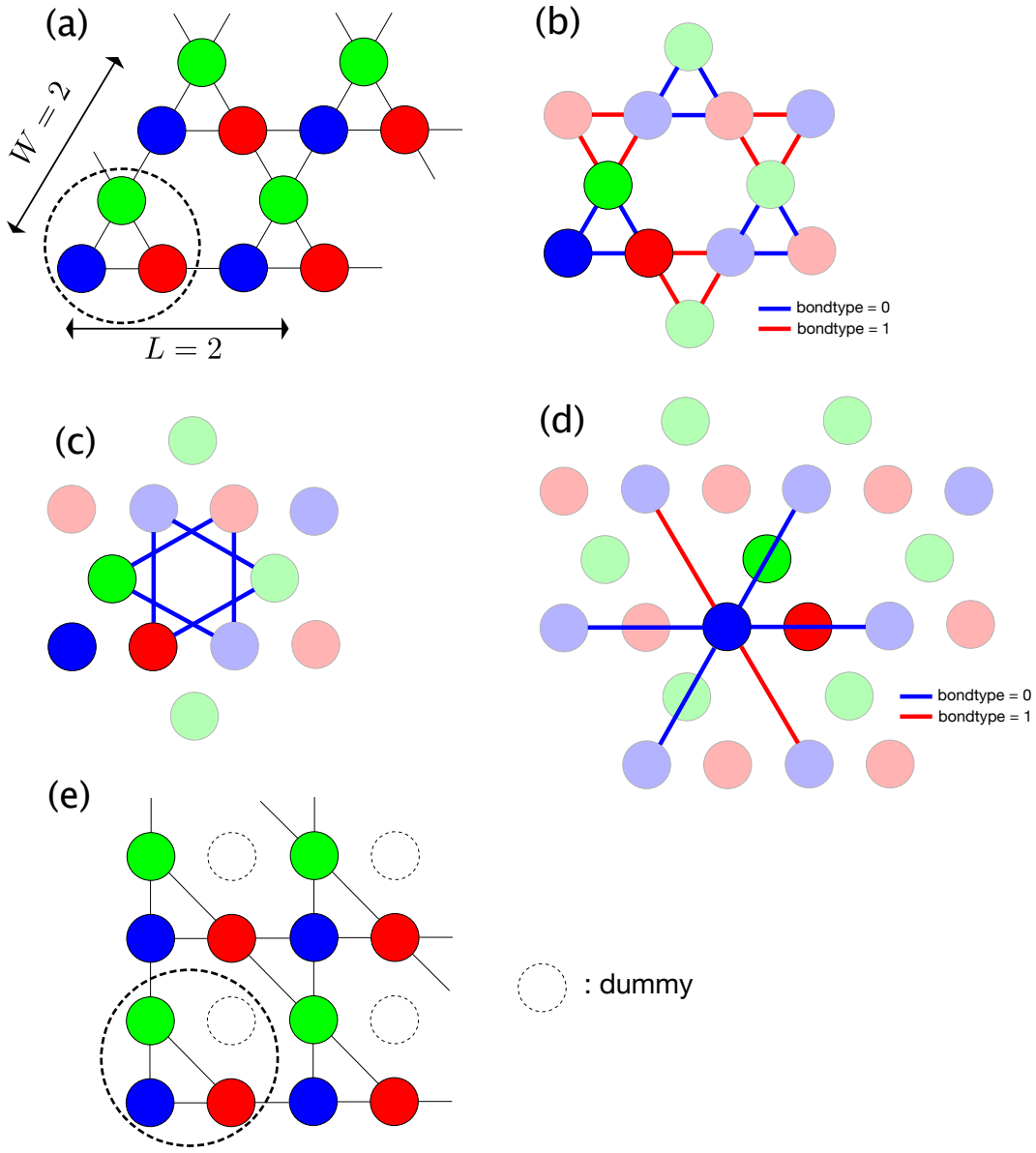


Fig. 5.4: Kagome lattice. (a) Site structure with `L=2, W=2`. The dashed circle denotes one unit. (b) Nearest neighbor bonds. `bondtype=0` (blue) bonds form upper triangle and `bondtype=1` (red) bonds form lowertriangle. (c) Second nearest neighbor bonds. (d) Third nearest neighbor bonds. `bondtype=0` (blue) bond passes over a site and `bondtype=1` (red) one does not. (e) Corresponding square TPS of the kagome lattice with `L=2, W=2`. The white circles are the dummy tensors with bonds of dimension one.

  - Save numerical results such as physical quantities to files in this directory
  - Empty means `"."` (current directory)
- `tensor_save`
  - Save optimized tensors to files in this directory
  - If empty no tensors will be saved
- `tensor_load`
  - Read initial tensors from files in this directory
  - If empty no tensors will be loaded

`parameter.simple_update`

Parameters in the simple update procedure.

Name	Description	Type	Default
`tau`	Imaginary time step τ in imaginary time evolution operator	Real	0.01
`num_step`	Number of simple updates	Integer	0
`lambda_cutoff`	cutoff of the mean field to be considered zero in the simple update	Real	1e-12

`parameter.full_update`

Parameters in the full update procedure.

Name	Description	Type	Default
`tau`	Imaginary time step τ in imaginary time evolution operator	Real	0.01
`num_step`	Number of full updates	Integer	0
`env_cutoff`	Cutoff of singular values to be considered as zero when computing environment through full updates	Real	1e-12
`inverse_precision`	Cutoff of singular values to be considered as zero when computing the pseudoinverse matrix with full update	Real	1e-12
`convergence_epsilon`	Convergence criteria for truncation optimization with full update	Real	1e-6
`iteration_max`	Maximum iteration number for truncation optimization on full updates	Integer	100
`gauge_fix`	Whether the tensor gauge is fixed	Boolean	true
`fastfullupdate`	Whether the fast full update is adopted	Boolean	true

parameter.ctm

Parameters for corner transfer matrices, CTM.

Name	Description	Type	Default
dimension	Bond Dimension of CTM χ	Integer	4
projector_cutoff	Cutoff of singular values to be considered as zero when computing CTM projectors	Real	1e-12
convergence_epsilon	CTM convergence criteria	Real	1e-6
iteration_max	Maximum iteration number of convergence for CTM	Integer	100
projector_corner	Whether to use only the 1/4 corner tensor in the CTM projector calculation	Boolean	true
use_rsvd	Whether to replace SVD with random SVD	Boolean	false
rsvd_oversampling_factor	Ratio of the number of the oversampled elements to that of the obtained elements in random SVD method	Real	2.0

For Tensor renomalization group approach using random SVD, please see the following reference, S. Morita, R. Igarashi, H.-H. Zhao, and N. Kawashima, Phys. Rev. E 97, 033310 (2018) .

parameter.random

Parameters for random number generators.

Name	Description	Type	Default
seed	Seed of the pseudo-random number generator used to initialize the tensor	Integer	11

Each MPI process has the own seed as `seed` plus the process ID (MPI rank).

Example

```
[parameter]
[parameter.general]
is_real = true
[parameter.simple_update]
num_step = 100
tau = 0.01
[parameter.full_update]
num_step = 0  # No full update
tau = 0.01
[parameter.ctm]
iteration_max = 10
dimension = 9 # CHI
```

5.2.4 `correlation` section

For `tenes_simple` , correlation functions $C = \langle A(0)B(r)\rangle$ are not calculated by default. For calculating correlation functions, they have to be specified in the same file format as the input file of `tenes`. For details, See `correlation` section *Input file for tenes*.

5.3 Input file for `tenes_std`

- File format: TOML format
- This file has 5 sections: `parameter`, `tensor`, `hamiltonian`, `observable`, `correlation`
  - The four sections other than `hamiltonian` are identical to the `tenes` input file format, with the following exceptions, and are copied to the `tenes` input file.
  - By setting a real number for `parameter.simple_update.tau` and `parameter.full_update.tau`, the imaginary time step for the imaginary time evolution operator can be specified.

5.3.1 `parameter` section

Set various parameters that appear in the calculation, such as the number of updates. This section has five subsections: `general`, `simple_update`, `full_update`, `ctm`, `random`.

Imaginary-time step τ for simple update `parameter.simple_update.tau` and that for full update `parameter.full_update.tau` are used only in standard mode `tenes_std`, not used in `tenes`.

`parameter.general`

General parameters for `tenes`.

Name	Description	Type	Default
`is_real`	Whether to limit all tensors to real valued ones	Boolean	false
`iszero_tol`	Absolute cutoff value for reading operators	Real	0.0
`measure`	Whether to calculate and save observables	Boolean	true
`output`	Directory for saving result such as physical quantities	String	“output”
`tensor_save`	Directory for saving optimized tensors	String	“”
`tensor_load`	Directory for loading initial tensors	String	“”

- `is_real`
  - When set to `true`, the type of elements of the tensor becomes real.
  - If one complex operator is defined at least, calculation will end in errors before starting.
- `iszero_tol`
  - When the absolute value of operator elements loaded is less than `iszero_tol`, it is regarded as zero
- `meaure`
  - When set to `false`, the stages for measuring and saving observables will be skipped
  - Elapsed time `time.dat` is always saved

- `output`
  - Save numerical results such as physical quantities to files in this directory
  - Empty means `"."` (current directory)
- `tensor_save`
  - Save optimized tensors to files in this directory
  - If empty no tensors will be saved
- `tensor_load`
  - Read initial tensors from files in this directory
  - If empty no tensors will be loaded

parameter.simple_update

Parameters in the simple update procedure.

Name	Description	Type	Default
`tau`	Imaginary time step τ in imaginary time evolution operator	Real	0.01
`num_step`	Number of simple updates	Integer	0
`lambda_cutoff`	cutoff of the mean field to be considered zero in the simple update	Real	1e-12

parameter.full_update

Parameters in the full update procedure.

Name	Description	Type	Default
`tau`	Imaginary time step τ in imaginary time evolution operator	Real	0.01
`num_step`	Number of full updates	Integer	0
`env_cutoff`	Cutoff of singular values to be considered as zero when computing environment through full updates	Real	1e-12
`inverse_precision`	Cutoff of singular values to be considered as zero when computing the pseudoinverse matrix with full update	Real	1e-12
`convergence_epsilon`	Convergence criteria for truncation optimization with full update	Real	1e-6
`iteration_max`	Maximum iteration number for truncation optimization on full updates	Integer	100
`gauge_fix`	Whether the tensor gauge is fixed	Boolean	true
`fastfullupdate`	Whether the fast full update is adopted	Boolean	true

parameter.ctm

Parameters for corner transfer matrices, CTM.

Name	Description	Type	Default
dimension	Bond Dimension of CTM χ	Integer	4
projector_cutoff	Cutoff of singular values to be considered as zero when computing CTM projectors	Real	1e-12
convergence_epsilon	CTM convergence criteria	Real	1e-6
iteration_max	Maximum iteration number of convergence for CTM	Integer	100
projector_corner	Whether to use only the 1/4 corner tensor in the CTM projector calculation	Boolean	true
use_rsvd	Whether to replace SVD with random SVD	Boolean	false
rsvd_oversampling_factor	Ratio of the number of the oversampled elements to that of the obtained elements in random SVD method	Real	2.0

For Tensor renomalization group approach using random SVD, please see the following reference, S. Morita, R. Igarashi, H.-H. Zhao, and N. Kawashima, Phys. Rev. E 97, 033310 (2018) .

parameter.random

Parameters for random number generators.

Name	Description	Type	Default
seed	Seed of the pseudo-random number generator used to initialize the tensor	Integer	11

Each MPI process has the own seed as `seed` plus the process ID (MPI rank).

Example

```
[parameter]
[parameter.general]
is_real = true
[parameter.simple_update]
num_step = 100
tau = 0.01
[parameter.full_update]
num_step = 0  # No full update
tau = 0.01
[parameter.ctm]
iteration_max = 10
dimension = 9 # CHI
```

5.3.2 `tensor` section

Specify the unit cell information (Information of bonds is given in the `hamiltonian` (`tenes_std`) and `evolution` (`tenes`) sections.). Unit cell has a shape of a rectangular with the size of `Lx` times `Ly`. `lattice` section has an array of subsections `unitcell` .

Name	Description	Type	Default
`L_sub`	Unit cell size	Integer or a list of integer	–
`skew`	Shift value in skew boundary condition	Integer	0

When a list of two integers is passed as `L_sub`, the first element gives the value of `Lx` and the second one does `Ly`. A list of three or more elements causes an error. If `L_sub` is an integer, both `Lx` and `Ly` will have the same value.

Sites in a unit cell are indexed starting from 0. These are arranged in order from the x direction.

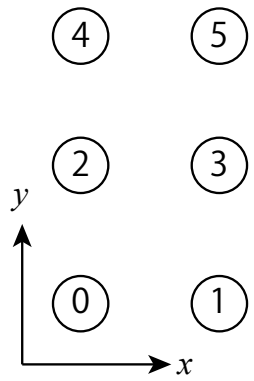


Fig. 5.5: An example for `L_sub = [2,3]`.

`skew` is the shift value in the x direction when moving one unit cell in the y direction.

`tensor.unitcell` subsection

The information of site tensors $T^{(n)}_{ijkl\alpha}$ is specified. Here, i, j, k, l indicate the index of the virtual bond, α indicates the index of the physical bond, and n indicates the site number.

Name	Description	Type
`index`	Site number	Integer or a list of integer
`physical_dim`	Dimension of physical bond for a site tensor	Integer
`virtual_dim`	Dimension of virtual bonds D for a site tensor	Integer or a list of integer
`initial_state`	Initial tensor	a list of real
`noise`	Noise for initial tensor	Real

Multiple sites can be specified at once by setting a list to `index`. An empty list `[]` means all sites.

By setting a list to `virtual_dim`, individual bond dimensions in four directions can be specified. The order is left (-x), top (+y), right (+x), and bottom (-y).

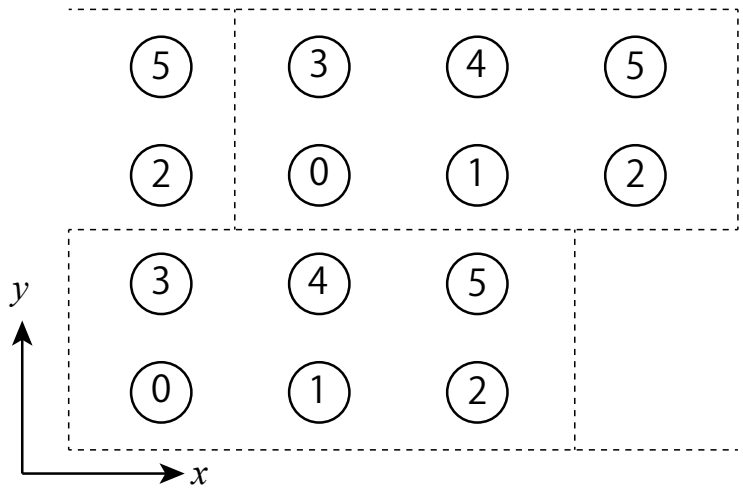


Fig. 5.6: An example for `L_sub = [3,2], skew = 1` (ruled line is a separator for unit cell).

An initial state of a system $|\Psi\rangle$ is represented as the direct product state of the initial states at each site i, $|\Psi_i\rangle$:

$$|\Psi\rangle = \otimes_i |\Psi_i\rangle,$$

where $|\Psi_i\rangle = \sum_\alpha A_\alpha |\alpha\rangle_i$ is the initial state at i site. Site tensors are initialized to realize this product state with some noise. `initial_state` specifies (real) values of expansion coefficient A_α, which will be automatically normalized. The tensor itself is initialized such that all elements with a virtual bond index of 0 are $T_{0000\alpha} = A_\alpha$. The other elements are independently initialized by a uniform random number of `[-noise, noise)`. For example, in the case of $S = 1/2$, set `initial_state = [1.0, 0.0]` when you want to set the initial state as the state $|\Psi_i\rangle = |\uparrow\rangle = |0\rangle$. When you want to set the initial state as the state $|\Psi_i\rangle = (|\uparrow\rangle + |\downarrow\rangle)/\sqrt{2}$, set `initial_state = [1.0, 1.0]`.

When an array consisting of only zeros is passed as `initil_state`, all the elements of the initial tensor will be initialized independently by uniform random value `[-noise, noise)` .

5.3.3 `observable` section

Define various settings related to physical quantity measurement. This section has two types of subsections, `onesite` and `twosite`.

`observable.onesite`

Define one-body operators that indicate physical quantities defined at each site i.

Name	Description	type
`name`	Operator name	String
`group`	Identification number of operators	Integer
`sites`	Site number	Integer or a list of integer
`dim`	Dimension of an operator	Integer
`elements`	Non-zero elements of an operator	String

`name` specifies an operator name.

`group` specifies an identification number of one-site operators.

`sites` specifies a site number where an operater acts on. By using a list, the operators can be defined on the multiple sites at the same time. An empty list `[]` means all sites.

`dim` specifies a dimension of an operator.

`elements` is a string specifying the non-zero element of an operator. One element is specified by one line consisting of two integers and two floating-point numbers separated by spaces.

- The first two integers are the state numbers before and after the act of the operator, respectively.
- The latter two floats indicate the real and imaginary parts of the elements of the operator, respectively.

Example

As an example, the case of S^z operator for S=1/2

$$S^z = \begin{pmatrix} 0.5 & 0.0 \\ 0.0 & -0.5 \end{pmatrix}$$

is explained.

First, set the name to `name = "Sz"` and the identification number to `group = 0`.

Next, if the same operator is used at all sites, set `sites = []`. Otherwise, for example, if there are sites with different spin length S, specify a specific site number such as `sites = [0,1]`.

The dimension of the operator is `dim = 2`, because it is the size of the matrix shown above.

Finally, the operator element is defined. When we label two basis on site as $|\uparrow\rangle = |0\rangle$ and $|\downarrow\rangle = |1\rangle$, non-zero elements of S^z are represented as

```
elements = """
0 0   0.5 0.0
1 1  -0.5 0.0
"""
```

As a result, S^z operator for S=1/2 is defined as follows:

```
[[observable.onesite]]
name = "Sz"
group = 0
sites = []
dim = 2
elements = """
0 0  0.5 0.0
1 1  -0.5 0.0
"""
```

observable.twosite

Define two-body operators that indicate physical quantities defined on two sites.

Name	Description	Type
name	Operator name	String
group	Identification number of operators	Integer
bonds	Bond	String
dim	Dimension of an operator	Integer
elements	Non-zero elements of an operator	String
ops	Index of onesite operators	A list of integer

name specifies an operator name.

group specifies an identification number of two sites operators.

bonds specifies a string representing the set of site pairs on which the operator acts. One line consisting of three integers means one site pair.

- The first integer is the number of the source site.
- The last two integers are the coordinates (dx, dy) of the other site (target site) from the source site.
  - Both dx and dy must be in the range $-3 \leq dx \leq 3$.

dim specifies a dimension of an operator. In other words, the number of possible states of the site where the operator acts on. In the case of interaction between two $S = 1/2$ spins, for example, dim = [2, 2] .

elements is a string specifying the non-zero element of an operator. One element consists of one line consisting of four integers and two floating-point numbers separated by spaces.

- The first two integers are the status numbers of the source site and target site **before** the operator acts on.
- The next two integers show the status numbers of the source site and target site **after** the operator acts on.
- The last two floats indicate the real and imaginary parts of the elements of the operator.

Using ops, a two-body operator can be defined as a direct product of the one-body operators defined in observable.onesite. For example, if S^z is defined as group = 0 in observable.onesite, $S_i^z S_j^z$ can be expressed as ops = [0,0].

If both elements and ops are defined, the process will end in error.

Example

As an example, for the calculation of the energy of the bond Hamiltonian for S=1/2 Heisenberg model on square lattice at Lsub=[2,2] , the way to define two site operators (equal to the Hamiltonian)

$$\mathcal{H}_{ij} = S_i^z S_j^z + \frac{1}{2}\left[S_i^+ S_j^- + S_i^- S_j^+\right]$$

is explained below.

First, the name and identification number is set as name = "hamiltonian" and group = 0. dim = [2,2] because the state of each site is a superposition of the two states $|\uparrow\rangle$ and $|\downarrow\rangle$.

Next, let's define the bonds. In this case, site indecies are given as shown in bond_22 . The bond connecting 0 and 1 is represented as 0 1 0 because 1 is located at (1,0) from 0. Similarly, The bond connecting 1 and 3 is represented as 1 0 1 because 3 is located at (0,1) from 1.

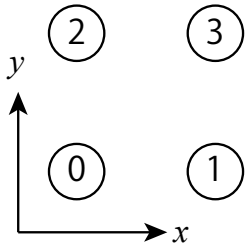


Fig. 5.7: Site indecies of the S=1/2 Heisenberg model on square lattice at `Lsub=[2,2]` .

Finally, how to define the elements of the operator is explained. First, the basis of the site is needed to be labeled. Here, we label $|\uparrow\rangle$ as 0 and $|\downarrow\rangle$ as 1. Using this basis and label number, for example, one of diagonal elements $\langle\uparrow_i\uparrow_j|\mathcal{H}_{ij}|\uparrow_i\uparrow_j\rangle = 1/4$ is specified by `0 0 0 0 0.25 0.0`. Likewise, one of off-diagonal elements $\langle\uparrow_i\downarrow_j|\mathcal{H}_{ij}|\downarrow_i\uparrow_j\rangle = 1/2$ is specified by `1 0 0 1 0.5 0.0`.

As a result, the Heisenberg Hamiltonian for S=1/2 is defined as follows:

```
[[observable.twosite]]
name = "hamiltonian"
group = 0
dim = [2, 2]
bonds = """
0 0 1
0 1 0
1 0 1
1 1 0
2 0 1
2 1 0
3 0 1
3 1 0
"""
elements = """
0 0 0 0  0.25 0.0
1 0 1 0  -0.25 0.0
0 1 1 0  0.5 0.0
1 0 0 1  0.5 0.0
0 1 0 1  -0.25 0.0
1 1 1 1  0.25 0.0
"""
```

5.3.4 `hamiltonian` section

Let the whole Hamiltonian be the sum of the bond Hamiltonian (two-site Hamiltonian).

$$\mathcal{H} = \sum_{i,j} \mathcal{H}_{ij}$$

In `hamiltonian` section, each two-site Hamiltonian is defined. The format is similar to that of the two-site operator specified in `observable.twosite`.

Name	Description	Type
`bonds`	Bond	String
`dim`	Dimension of an operator	A list of integer
`elements`	Non-zero elements of an operator	String

`bonds` specifies a string representing the set of site pairs on which the operator acts. One line consisting of three integers means one site pair.

- The first integer is the number of the source site.
- The last two integers are the coordinates (dx, dy) of the destination site (target) from the source site.

`dim` specifies a dimension of an operator. In other words, the number of possible states of the site where the operator acts on. In the case of interaction between two $S = 1/2$ spin, for example, `dim = [2,2]` .

`elements` is a string specifying the non-zero element of an operator. One element consists of one line consisting of four integers and two floating-point numbers separated by spaces. The first two are the status numbers of the source site and target site before the operator acts on. The next two show the status numbers of the source site and target site after the operator acts on. The last two indicate the real and imaginary parts of the elements of the operator.

5.3.5 `correlation` section

In this section, the parameters about the site-site correlation function $C = \langle A(\boldsymbol{r}_0)B(\boldsymbol{r}_0 + \boldsymbol{r})\rangle$ is specified. If you omit this section, no correlation functions will be calculated.

Coordinates $\boldsymbol{r}, \boldsymbol{r}_0$ measured in the system of square lattice TNS. For example, the coordinate of the right neighbor tensor is $\boldsymbol{r} = (1, 0)$ and that of the top neighbor one is $\boldsymbol{r} = (0, 1)$. TeNeS calculates the correlation functions along the positive direction of x and y axis, that is,

$$\boldsymbol{r} = (0,0), (1,0), (2,0), \ldots, (r_{\text{max}}, 0), (0,1), (0,2), \ldots, (0, r_{\text{max}})$$

The coordinate of each site of the unitcell is used as the center coordinate, $\boldsymbol{r}_0$.

Name	Description	Type
`r_max`	Maximum distance r of the correlation function	Integer
`operators`	Indices of operators A and B to be measured	A list of integer

The operators defined in the `observable.onesite` section are used.

Example

For example, if S^z is defined as 0th operator and S^x is defined as 1st one, then $S^z(0)S^z(r), S^z(0)S^x(r), S^x(0)S^x(r)$ for $0 \leq r \leq 5$ are measured by the following definition:

```
[correlation]
r_max = 5
operators = [[0,0], [0,1], [1,1]]
```

5.4 Input file for `tenes`

- File format is TOML format.
- The input file has five sections: `parameter`, `tensor`, `evolution`, `observable`, `correlation`.

5.4.1 `parameter` section

Set various parameters that appear in the calculation, such as the number of updates. This section has five subsections: `general`, `simple_update`, `full_update`, `ctm`, `random`.

Imaginary-time step τ for simple update `parameter.simple_update.tau` and that for full update `parameter.full_update.tau` are used only in standard mode `tenes_std`, not used in `tenes`.

`parameter.general`

General parameters for `tenes`.

Name	Description	Type	Default
`is_real`	Whether to limit all tensors to real valued ones	Boolean	false
`iszero_tol`	Absolute cutoff value for reading operators	Real	0.0
`measure`	Whether to calculate and save observables	Boolean	true
`output`	Directory for saving result such as physical quantities	String	"output"
`tensor_save`	Directory for saving optimized tensors	String	""
`tensor_load`	Directory for loading initial tensors	String	""

- `is_real`
  - When set to `true`, the type of elements of the tensor becomes real.
  - If one complex operator is defined at least, calculation will end in errors before starting.
- `iszero_tol`
  - When the absolute value of operator elements loaded is less than `iszero_tol`, it is regarded as zero
- `meaure`
  - When set to `false`, the stages for measuring and saving observables will be skipped
  - Elapsed time `time.dat` is always saved
- `output`
  - Save numerical results such as physical quantities to files in this directory
  - Empty means `"."` (current directory)
- `tensor_save`
  - Save optimized tensors to files in this directory
  - If empty no tensors will be saved
- `tensor_load`
  - Read initial tensors from files in this directory
  - If empty no tensors will be loaded

parameter.simple_update

Parameters in the simple update procedure.

Name	Description	Type	Default
tau	Imaginary time step τ in imaginary time evolution operator	Real	0.01
num_step	Number of simple updates	Integer	0
lambda_cutoff	cutoff of the mean field to be considered zero in the simple update	Real	1e-12

parameter.full_update

Parameters in the full update procedure.

Name	Description	Type	Default
tau	Imaginary time step τ in imaginary time evolution operator	Real	0.01
num_step	Number of full updates	Integer	0
env_cutoff	Cutoff of singular values to be considered as zero when computing environment through full updates	Real	1e-12
inverse_precision	Cutoff of singular values to be considered as zero when computing the pseudoinverse matrix with full update	Real	1e-12
convergence_epsilon	Convergence criteria for truncation optimization with full update	Real	1e-6
iteration_max	Maximum iteration number for truncation optimization on full updates	Integer	100
gauge_fix	Whether the tensor gauge is fixed	Boolean	true
fastfullupdate	Whether the fast full update is adopted	Boolean	true

parameter.ctm

Parameters for corner transfer matrices, CTM.

Name	Description	Type	Default
dimension	Bond Dimension of CTM χ	Integer	4
projector_cutoff	Cutoff of singular values to be considered as zero when computing CTM projectors	Real	1e-12
convergence_epsilon	CTM convergence criteria	Real	1e-6
iteration_max	Maximum iteration number of convergence for CTM	Integer	100
projector_corner	Whether to use only the 1/4 corner tensor in the CTM projector calculation	Boolean	true
use_rsvd	Whether to replace SVD with random SVD	Boolean	false
rsvd_oversampling_factor	Ratio of the number of the oversampled elements to that of the obtained elements in random SVD method	Real	2.0

For Tensor renomalization group approach using random SVD, please see the following reference, S. Morita, R. Igarashi, H.-H. Zhao, and N. Kawashima, Phys. Rev. E 97, 033310 (2018) .

`parameter.random`

Parameters for random number generators.

Name	Description	Type	Default
`seed`	Seed of the pseudo-random number generator used to initialize the tensor	Integer	11

Each MPI process has the own seed as `seed` plus the process ID (MPI rank).

Example

```
[parameter]
[parameter.general]
is_real = true
[parameter.simple_update]
num_step = 100
tau = 0.01
[parameter.full_update]
num_step = 0  # No full update
tau = 0.01
[parameter.ctm]
iteration_max = 10
dimension = 9 # CHI
```

5.4.2 `tensor` section

Specify the unit cell information (Information of bonds is given in the `hamiltonian` (`tenes_std`) and `evolution` (`tenes`) sections.). Unit cell has a shape of a rectangular with the size of `Lx` times `Ly`. `lattice` section has an array of subsections `unitcell` .

Name	Description	Type	Default
`L_sub`	Unit cell size	Integer or a list of integer	–
`skew`	Shift value in skew boundary condition	Integer	0

When a list of two integers is passed as `L_sub`, the first element gives the value of `Lx` and the second one does `Ly`. A list of three or more elements causes an error. If `L_sub` is an integer, both `Lx` and `Ly` will have the same value.

Sites in a unit cell are indexed starting from 0. These are arranged in order from the x direction.

`skew` is the shift value in the x direction when moving one unit cell in the y direction.

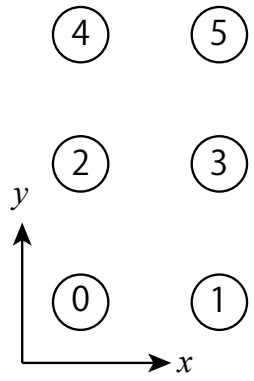


Fig. 5.8: An example for `L_sub = [2,3]`.

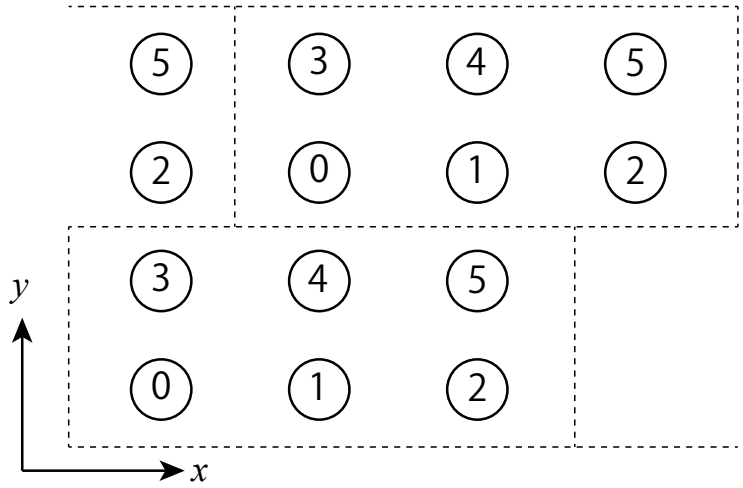


Fig. 5.9: An example for `L_sub = [3,2], skew = 1` (ruled line is a separator for unit cell).

`tensor.unitcell` subsection

The information of site tensors $T^{(n)}_{ijkl\alpha}$ is specified. Here, i, j, k, l indicate the index of the virtual bond, α indicates the index of the physical bond, and n indicates the site number.

Name	Description	Type
`index`	Site number	Integer or a list of integer
`physical_dim`	Dimension of physical bond for a site tensor	Integer
`virtual_dim`	Dimension of virtual bonds D for a site tensor	Integer or a list of integer
`initial_state`	Initial tensor	a list of real
`noise`	Noise for initial tensor	Real

Multiple sites can be specified at once by setting a list to `index`. An empty list `[]` means all sites.

By setting a list to `virtual_dim`, individual bond dimensions in four directions can be specified. The order is left (-x), top (+y), right (+x), and bottom (-y).

An initial state of a system $|\Psi\rangle$ is represented as the direct product state of the initial states at each site i, $|\Psi_i\rangle$:

$$|\Psi\rangle = \otimes_i |\Psi_i\rangle,$$

where $|\Psi_i\rangle = \sum_\alpha A_\alpha |\alpha\rangle_i$ is the initial state at i site. Site tensors are initialized to realize this product state with some noise. `initial_state` specifies (real) values of expansion coefficient A_α, which will be automatically normalized. The tensor itself is initialized such that all elements with a virtual bond index of 0 are $T_{0000\alpha} = A_\alpha$. The other elements are independently initialized by a uniform random number of `[-noise, noise)`. For example, in the case of $S = 1/2$, set `initial_state = [1.0, 0.0]` when you want to set the initial state as the state $|\Psi_i\rangle = |\uparrow\rangle = |0\rangle$. When you want to set the initial state as the state $|\Psi_i\rangle = (|\uparrow\rangle + |\downarrow\rangle)/\sqrt{2}$, set `initial_state = [1.0, 1.0]`.

When an array consisting of only zeros is passed as `initil_state`, all the elements of the initial tensor will be initialized independently by uniform random value `[-noise, noise)` .

5.4.3 `observable` section

Define various settings related to physical quantity measurement. This section has two types of subsections, `onesite` and `twosite`.

`observable.onesite`

Define one-body operators that indicate physical quantities defined at each site i.

Name	Description	type
`name`	Operator name	String
`group`	Identification number of operators	Integer
`sites`	Site number	Integer or a list of integer
`dim`	Dimension of an operator	Integer
`elements`	Non-zero elements of an operator	String

`name` specifies an operator name.

`group` specifies an identification number of one-site operators.

`sites` specifies a site number where an operater acts on. By using a list, the operators can be defined on the multiple sites at the same time. An empty list `[]` means all sites.

`dim` specifies a dimension of an operator.

`elements` is a string specifying the non-zero element of an operator. One element is specified by one line consisting of two integers and two floating-point numbers separated by spaces.

- The first two integers are the state numbers before and after the act of the operator, respectively.
- The latter two floats indicate the real and imaginary parts of the elements of the operator, respectively.

Example

As an example, the case of S^z operator for S=1/2

$$S^z = \begin{pmatrix} 0.5 & 0.0 \\ 0.0 & -0.5 \end{pmatrix}$$

is explained.

First, set the name to `name = "Sz"` and the identification number to `group = 0`.

Next, if the same operator is used at all sites, set `sites = []`. Otherwise, for example, if there are sites with different spin length S, specify a specific site number such as `sites = [0,1]`.

The dimension of the operator is `dim = 2`, because it is the size of the matrix shown above.

Finally, the operator element is defined. When we label two basis on site as $|\uparrow\rangle = |0\rangle$ and $|\downarrow\rangle = |1\rangle$, non-zero elements of S^z are represented as

```
elements = """
0 0   0.5 0.0
1 1  -0.5 0.0
"""
```

As a result, S^z operator for S=1/2 is defined as follows:

```
[[observable.onesite]]
name = "Sz"
group = 0
sites = []
dim = 2
elements = """
0 0  0.5 0.0
1 1  -0.5 0.0
"""
```

observable.twosite

Define two-body operators that indicate physical quantities defined on two sites.

Name	Description	Type
`name`	Operator name	String
`group`	Identification number of operators	Integer
`bonds`	Bond	String
`dim`	Dimension of an operator	Integer
`elements`	Non-zero elements of an operator	String
`ops`	Index of onesite operators	A list of integer

`name` specifies an operator name.

`group` specifies an identification number of two sites operators.

`bonds` specifies a string representing the set of site pairs on which the operator acts. One line consisting of three integers means one site pair.

- The first integer is the number of the source site.
- The last two integers are the coordinates (dx, dy) of the other site (target site) from the source site.
  - Both dx and dy must be in the range $-3 \leq dx \leq 3$.

`dim` specifies a dimension of an operator. In other words, the number of possible states of the site where the operator acts on. In the case of interaction between two $S = 1/2$ spins, for example, `dim = [2, 2]` .

`elements` is a string specifying the non-zero element of an operator. One element consists of one line consisting of four integers and two floating-point numbers separated by spaces.

- The first two integers are the status numbers of the source site and target site **before** the operator acts on.
- The next two integers show the status numbers of the source site and target site **after** the operator acts on.
- The last two floats indicate the real and imaginary parts of the elements of the operator.

Using `ops`, a two-body operator can be defined as a direct product of the one-body operators defined in `observable.onesite`. For example, if S^z is defined as `group = 0` in `observable.onesite`, $S^z_i S^z_j$ can be expressed as `ops = [0,0]`.

If both `elements` and `ops` are defined, the process will end in error.

Example

As an example, for the calculation of the energy of the bond Hamiltonian for S=1/2 Heisenberg model on square lattice at `Lsub=[2,2]` , the way to define two site operators (equal to the Hamiltonian)

$$\mathcal{H}_{ij} = S^z_i S^z_j + \frac{1}{2}\left[S^+_i S^-_j + S^-_i S^+_j\right]$$

is explained below.

First, the name and identification number is set as `name = "hamiltonian"` and `group = 0`. `dim = [2,2]` because the state of each site is a superposition of the two states $|\uparrow\rangle$ and $|\downarrow\rangle$.

Next, let's define the bonds. In this case, site indecies are given as shown in `bond_22` . The bond connecting 0 and 1 is represented as `0 1 0` because 1 is located at (1,0) from 0. Similarly, The bond connecting 1 and 3 is represented as `1 0 1` because 3 is located at (0,1) from 1.

Finally, how to define the elements of the operator is explained. First, the basis of the site is needed to be labeled. Here, we label $|\uparrow\rangle$ as 0 and $|\downarrow\rangle$ as 1. Using this basis and label number, for example, one of diagonal elements $\langle\uparrow_i\uparrow_j|\mathcal{H}_{ij}|\uparrow_i\uparrow_j\rangle = 1/4$ is specified by `0 0 0 0 0.25 0.0`. Likewise, one of off-diagonal elements $\langle\uparrow_i\downarrow_j|\mathcal{H}_{ij}|\downarrow_i\uparrow_j\rangle = 1/2$ is specified by `1 0 0 1 0.5 0.0`.

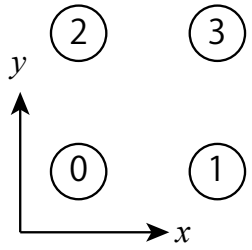


Fig. 5.10: Site indecies of the S=1/2 Heisenberg model on square lattice at `Lsub=[2,2]` .

As a result, the Heisenberg Hamiltonian for S=1/2 is defined as follows:

```
[[observable.twosite]]
name = "hamiltonian"
group = 0
dim = [2, 2]
bonds = """
0 0 1
0 1 0
1 0 1
1 1 0
2 0 1
2 1 0
3 0 1
3 1 0
"""
elements = """
0 0 0 0  0.25 0.0
1 0 1 0  -0.25 0.0
0 1 1 0  0.5 0.0
1 0 0 1  0.5 0.0
0 1 0 1  -0.25 0.0
1 1 1 1  0.25 0.0
"""
```

5.4.4 `evolution` section

Specify the imaginary time evolution opetrators used in simple and full updates. This section has two subsections: `simple` and `full`.

Name	Description	Type
`source_site`	Index of source site	Integer
`source_leg`	Direction from source site to target site	Integer
`dimensions`	Dimension of a tensor of imaginary time evolution operator	A list of integer
`elements`	Non-zero elements of a tensor of imaginary time evolution operator	String

`source_leg` is specified as an integer from 0 to 3. Defined as `0: -x, 1: +y, 2: +x, 3: -y` in the clockwise order from the -x direction.

`dimensions` is different from `dim` in `observable` section, so you need to specify the dimensions of all legs. The order of the legs is `source_initial, target_initial, source_final, target_final`, just like `elements`.

Example

```
[evolution]
[[evolution.simple]]
source_site = 0
source_leg = 2
dimensions = [2, 2, 2, 2]
elements = """
0 0 0 0  0.9975031223974601 0.0
1 0 1 0  1.0025156589209967 0.0
0 1 1 0  -0.005012536523536871 0.0
1 0 0 1  -0.005012536523536871 0.0
0 1 0 1  1.0025156589209967 0.0
1 1 1 1  0.9975031223974601 0.0
"""
```

5.4.5 `correlation` section

In this section, the parameters about the site-site correlation function $C = \langle A(\boldsymbol{r}_0)B(\boldsymbol{r}_0 + \boldsymbol{r})\rangle$ is specified. If you omit this section, no correlation functions will be calculated.

Coordinates $\boldsymbol{r}, \boldsymbol{r}_0$ measured in the system of square lattice TNS. For example, the coordinate of the right neighbor tensor is $\boldsymbol{r} = (1, 0)$ and that of the top neighbor one is $\boldsymbol{r} = (0, 1)$. TeNeS calculates the correlation functions along the positive direction of x and y axis, that is,

$$\boldsymbol{r} = (0,0), (1,0), (2,0), \ldots, (r_{\text{max}}, 0), (0,1), (0,2), \ldots, (0, r_{\text{max}})$$

The coordinate of each site of the unitcell is used as the center coordinate, $\boldsymbol{r}_0$.

Name	Description	Type
`r_max`	Maximum distance r of the correlation function	Integer
`operators`	Indices of operators A and B to be measured	A list of integer

The operators defined in the `observable.onesite` section are used.

Example

For example, if S^z is defined as 0th operator and S^x is defined as 1st one, then $S^z(0)S^z(r), S^z(0)S^x(r), S^x(0)S^x(r)$ for $0 \leq r \leq 5$ are measured by the following definition:

```
[correlation]
r_max = 5
operators = [[0,0], [0,1], [1,1]]
```

5.5 Output files

Output files are generated in the `output` directry.

5.5.1 `parameters.dat`

Paramters in the `parameter` and `lattice` sections defined in the input file are outputted.

5.5.2 `energy.dat`

The energy of each site is output.

5.5.3 `site_obs.dat`

- The expected values of the site operator are outputted.
- Each row consists of four columns.
  1. Index of the operator
  2. Index of the sites
  3. Real part of the expected value
  4. Imaginary part of the expected value

Example

```
# $1: op_index
# $2: site_index
# $3: real
# $4: imag

0 0 1.92549465249573365e-02 0.00000000000000000e+00
0 1 -1.92620814130195529e-02 0.00000000000000000e+00
0 2 -1.95243093055922252e-02 0.00000000000000000e+00
0 3 1.91619477632061150e-02 0.00000000000000000e+00
1 0 4.07206063348768799e-01 0.00000000000000000e+00
1 1 -4.07243511737157671e-01 0.00000000000000000e+00
1 2 -4.07255967738734126e-01 0.00000000000000000e+00
1 3 4.07308918791554009e-01 0.00000000000000000e+00
```

5.5.4 `neighbor_obs.dat`

- Nearest neighbor correlations for site operations are outputted.
- Each row consists of five columns.
  1. Index of the operator
  2. Index of the sites
  3. Index of the sites

4. Real part of the expected value
5. Imaginary part of the expected value

```
# $1: op_index
# $2: source_site
# $3: target_site
# $4: real
# $5: imag

0 0 1 -7.0592761506496890e-02 0.0000000000000000e+00
0 0 2 -7.2706845643005127e-02 0.0000000000000000e+00
0 1 0 -7.1328438595739229e-02 0.0000000000000000e+00
0 1 3 -7.1952334925611358e-02 0.0000000000000000e+00
0 2 3 -7.1261036489548304e-02 0.0000000000000000e+00
0 2 0 -7.1973150756101195e-02 0.0000000000000000e+00
0 3 2 -7.0563355823021006e-02 0.0000000000000000e+00
0 3 1 -7.2680375080734049e-02 0.0000000000000000e+00
1 0 1 -1.8594286923710335e-01 0.0000000000000000e+00
1 0 2 -1.8716473167754519e-01 0.0000000000000000e+00
1 1 0 -1.8636038255007659e-01 0.0000000000000000e+00
1 1 3 -1.8676845108636669e-01 0.0000000000000000e+00
1 2 3 -1.8638418190980594e-01 0.0000000000000000e+00
1 2 0 -1.8674757673269352e-01 0.0000000000000000e+00
1 3 2 -1.8597508952501360e-01 0.0000000000000000e+00
1 3 1 -1.8719652291687905e-01 0.0000000000000000e+00
```

5.5.5 `correlation.dat`

- Correlation functions are outputted.
- Each row consists of eight columns.
  1. Index of the left operator
  2. Site index of the left operator
  3. Index of the right operator
  4. Site index of the right operator
  5. Unit cell offset of the right operator (x)
  6. Unit cell offset of the right operator (y)
  7. Real part of the expected value
  8. Imaginary part of the expected value

Example

```
# $1: left_op
# $2: left_site
# $3: right_op
# $4: right_site
# $5: offset_x
# $6: offset_y
# $7: real
```

(continues on next page)

(continued from previous page)

```
# $8: imag

0 0 0 1 0 0 -7.0592761506496792e-02 0.0000000000000000e+00
0 0 0 0 1 0 1.1966884322676102e-02 0.0000000000000000e+00
0 0 0 1 1 0 -2.4308622932000586e-03 0.0000000000000000e+00
0 0 0 0 2 0 7.4272919452849631e-04 0.0000000000000000e+00
0 0 0 1 2 0 -4.3879481941688542e-04 0.0000000000000000e+00
0 0 0 2 0 0 -7.2706845643005114e-02 0.0000000000000000e+00
0 0 0 0 0 1 1.2333984574662128e-02 0.0000000000000000e+00
0 0 0 2 0 1 -2.5011118624440735e-03 0.0000000000000000e+00
0 0 0 0 0 2 7.5460780658739152e-04 0.0000000000000000e+00
0 0 0 2 0 2 -4.4773455996967955e-04 0.0000000000000000e+00
1 0 1 1 0 0 -1.8594286923710324e-01 0.0000000000000000e+00
...
1 3 1 1 0 3 -1.6587424589146155e-01 0.0000000000000000e+00
```

5.5.6 time.dat

The calculation time is outputted.

CHAPTER

SIX

ALGORITHM

6.1 Tensor Network States

Tensor network states (TNS) are variational wavefunctions represented as products of small tensors *[TNS]*. For example, in the case of $S = 1/2$ spin system with N sites, a wavefunction can be represented by using the product state basis as

$$|\Psi\rangle = \sum_{s_i \pm \uparrow,\downarrow} \Psi_{s_1,s_2,\ldots,s_N} |s_1, s_2, \ldots, s_N\rangle$$

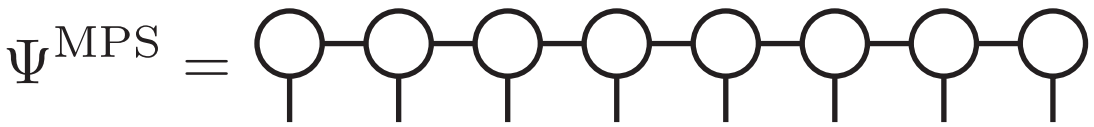
In a tensor network state, $\Psi_{s_1,s_2,\ldots,s_N}$ is represented as a tensor network, e.g,

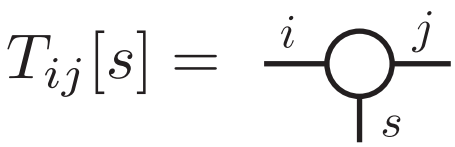
$$\Psi_{s_1,s_2,\ldots,s_N} = \mathrm{tTr}\left[T^{(1)}[s_1]T^{(2)}[s_2]\cdots T^{(N)}[s_N]\right],$$

where $\mathrm{tTr}[\ldots]$ represents tensor network contraction and $T^{(i)}[s_i]$ is a tensor. In the case of a matrix product state (MPS) *[MPS]* , $T^{(i)}[s_i]$ becomes a matrix for a given s_i and $\mathrm{tTr}[\ldots]$ becomes usual matrix products as

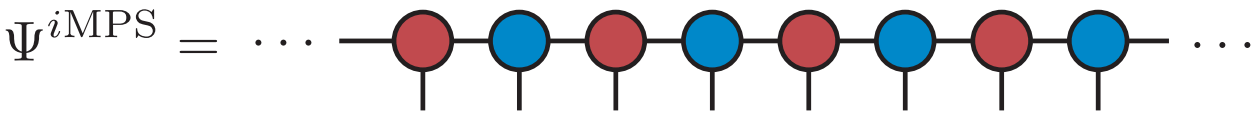
$$\Psi^{\mathrm{MPS}}_{s_1,s_2,\ldots,s_N} = T^{(1)}[s_1]T^{(2)}[s_2]\cdots T^{(N)}[s_N],$$

where we assume that shapes of $T^{(1)}[s_1]$, $T^{(i)}[s_i](i \neq 1, N)$, and $T^{(N)}[s_N]$ are $1 \times D_1$ $D_{i-1} \times D_i$,and $D_{N-1} \times 1$, respectively. When we use TNS in order to approximate the ground state wavefunction, the accuracy is determined by D_i. D_i is usually called as *bond dimension*. By using a tensor network diagram, MPS is represented as follows:





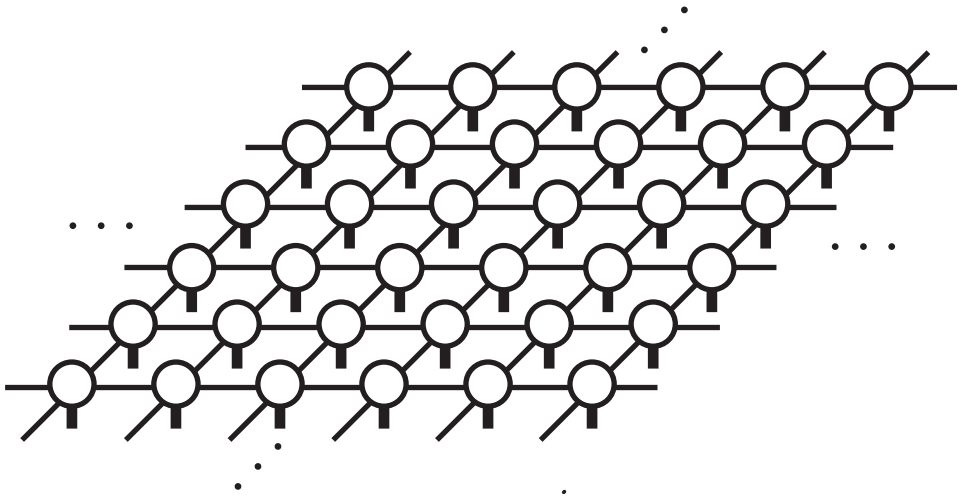
This MPS represents a wavefunction for a finite size system. Similarly, we can also consider an infinitely long MPS to represent an infinite system. Especially, when we assume a lattice transrational symmetry, with a certain period, we can construct an infinite MPS (iMPS) with a few independent tensors. In the case of two-site periodicity, an iMPS looks like as



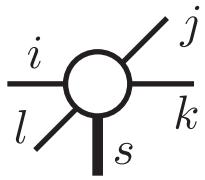
where tensors with the same color indicate identical tensors.

In TeNeS, we consider two-dimensional infinite tensor product states (iTNS), which are natural extension of iMPS to higher dimensions. We assume a square lattice tensor network with a translational symmetry, whose diagram is shown as

$\Psi^{\mathrm{iTPS}} =$



$T_{ijkl}[s] =$

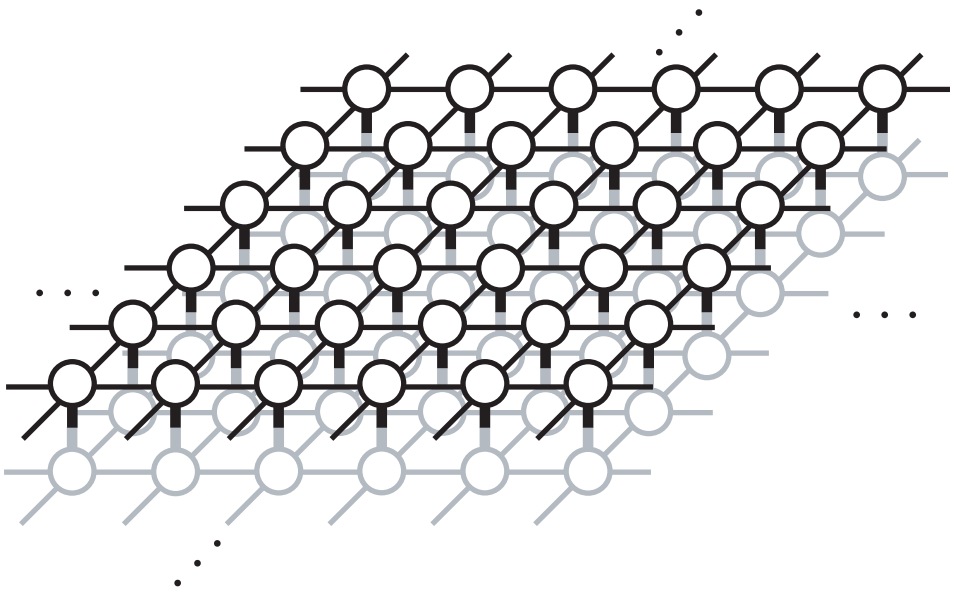


and try to find an approximate ground state wavefunction of two-dimensional quantum many-body systems. Notice that square lattice tensor networks can represent lattices other than the square lattice, such as the honeycomb and the triangular lattices, by considering proper mapping.

6.2 Contraction of iTPS

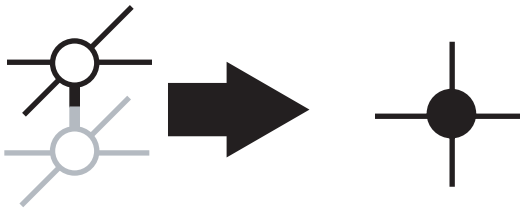
In order to calculate expectation values over a TNS, $\langle\Psi|O|\Psi\rangle/\langle\Psi|\Psi\rangle$, generally we need to contract tensor networks corresponding to $\langle\Psi|O|\Psi\rangle$ and $\langle\Psi|\Psi\rangle$. For example, a tensor network corresponding to $\langle\Psi|\Psi\rangle$ is given by

$\langle\Psi|\Psi\rangle =$



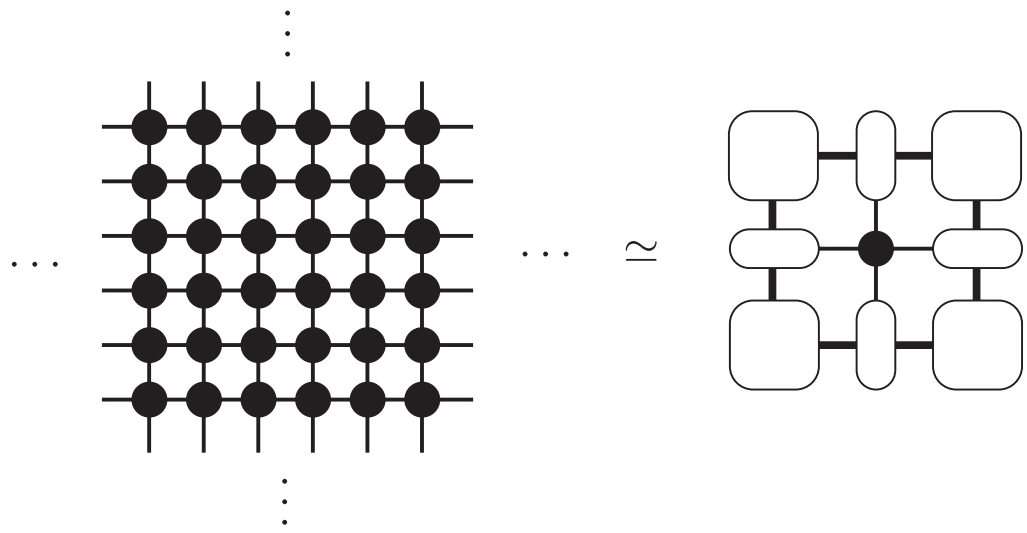
which is often called as a double layered tensor network. The contraction of a double layered tensor network often needs huge computation cost. In the case of MPS (and iMPS), fortunately, we can contract it efficiently, *e.g*, by considering a transfer matrix which consists of local tensors. However, in the case of TPS (and iTPS), exact contraction is impossible except for small finite size systems (or infinite cylinders) and we often use approximate contraction methods. Among several efficient methods for contracting iTPS in two-dimension, TeNeS supports corner transfer matrix renormalization group (CTMRG) method *[CTMRG]*, which expresses an infinitely extended double layered tensor network by using *corner transfer matrices* and *edge tensors*.

When we simplify the double layered tensor network by using a locally contracted tensor,

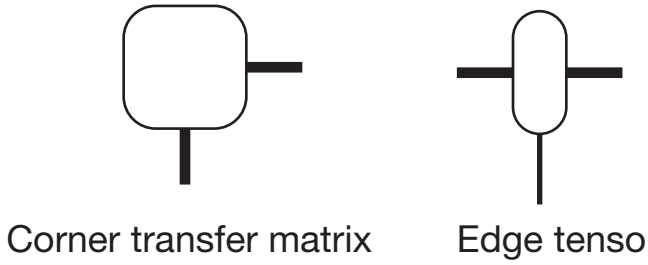


a tensor network diagram for the corner transfer matrix representation is given as

$\langle\Psi|\Psi\rangle =$

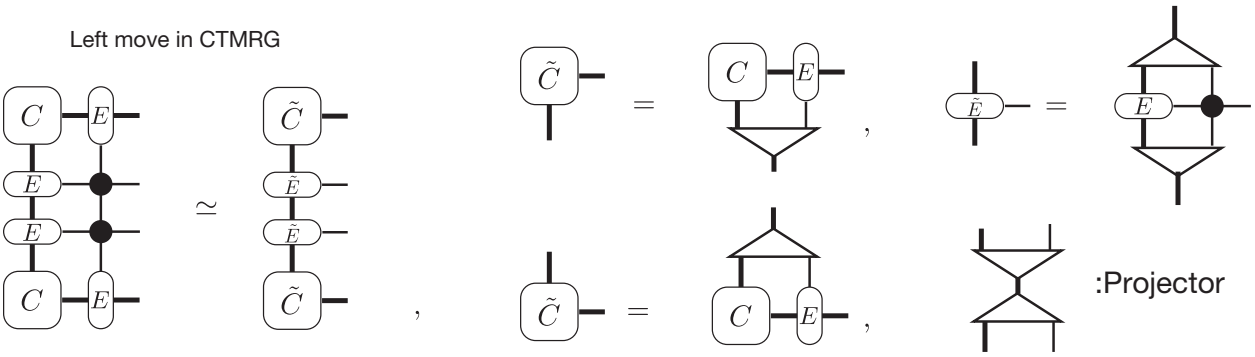


A corner transfer matrix and an edge tensor are defined as



The accuracy of the corner transfer matrix representation is determined by the bond dimension χ of corner transfer matrices, which is indicated as thick lines in the diagrams.

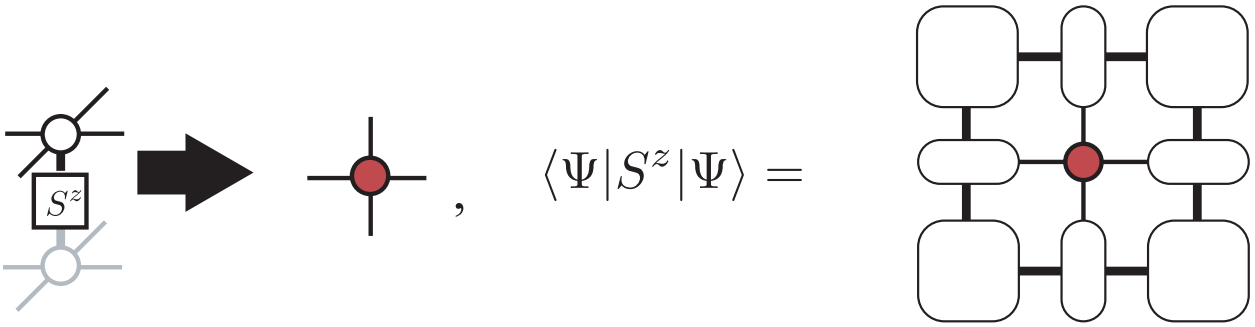
In the CTMRG algorithm, we iteratively optimise corner transfer matrices and edge tensors by *absorbing* local tensors until they converges. For example, an absorbing procedure, so called *left move*, is described as follows:



The projectors in the above diagram is calculated in several ways *[CTMRG]* and they reduces the degree of freedoms to χ.

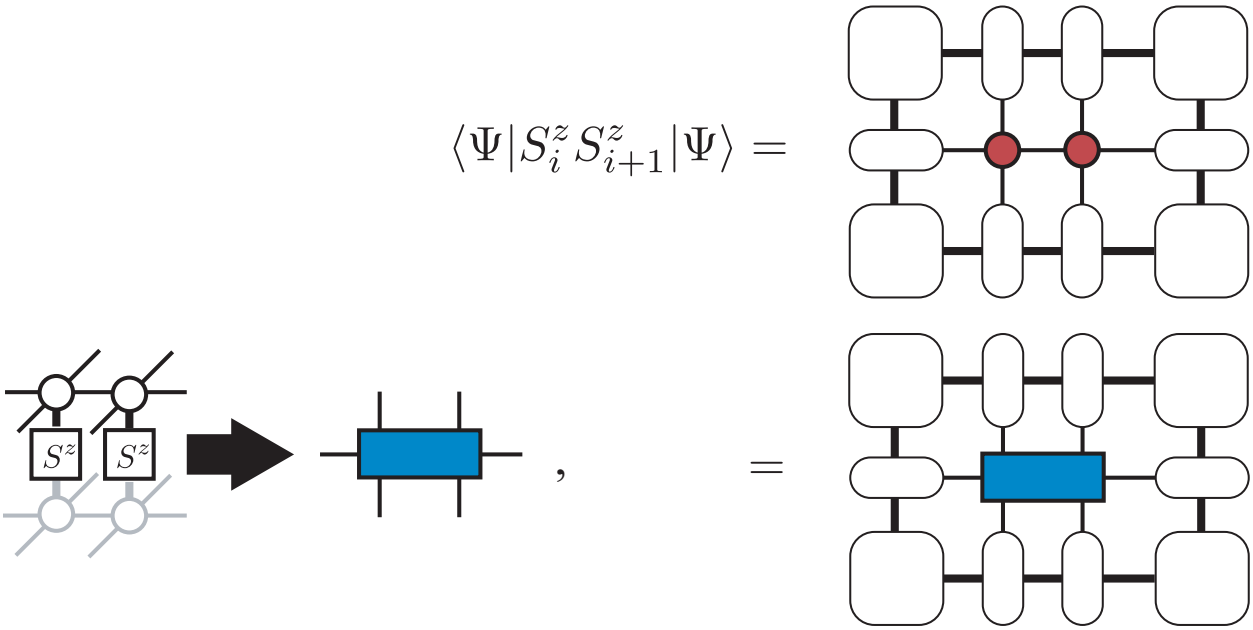
When we consider iTPS with the bond dimension D and CTMs with the bond dimension χ, the leading computation cost of CTMRG scales as $O(\chi^2 D^6)$ and $O(\chi^3 D^4)$. Notice that the bond dimension of the double layered tensor network becomes D^2 by using locally contracted tensors. Thus, typically we increase χ as $\chi \propto O(D^2)$. In this setup, the leading computation cost of CTMRG algorithm is reduced to $O(D^{10})$, while the memory usage scales $O(D^8)$. In order to achive the computation cost discussed above, we need to use a partial singular value decomposition (SVD) (or the truncated SVD) technique. When we use the full SVD insted of the partial SVD, the computation cost becomes $O(D^{12})$.

Once we obtain the corner transfer matrices and edge tensors, we can also calculate $\langle\Psi|O|\Psi\rangle$ efficiently. For example, a local magnetization $\langle\Psi|S_i^z|\Psi\rangle$ is represented as



and similarly the nearest neighbor correlation $\langle\Psi|S_i^z S_{i+1}^z|\Psi\rangle$ is represented as

$$\langle\Psi|S_i^z S_{i+1}^z|\Psi\rangle =$$



Notice that by using the second representation, we can calculate expectation values of any two-site operators. Although we can generalize such a diagram for any operators, the computation cost to contract the tensor network becomes huge for larger clusters.

6.3 Optimization of iTPS

In order to use iTPS as variational wavefunctions for the ground state, we need to optimize it so that it give us the minimum energy expectation value,

$$E = \frac{\langle\Psi|\mathcal{H}|\Psi\rangle}{\langle\Psi|\Psi\rangle},$$

where $\mathcal{H}$ represents the Hamiltonian of the target system. Among two types of popular optimization algorithms, the imaginary evolution (ITE) and the variational optimization, we support the ITE in TeNeS. In TeNeS, we consider approximate ITE within the iTPS ansatz:

$$|\Psi^{\mathrm{iTPS}}\rangle \simeq e^{-T\mathcal{H}}|\Psi_0\rangle,$$

where $|\Psi_0\rangle$ is an arbitrary initial iTPS. If T is sufficiently large, the left hand side, $|\Psi^{\mathrm{iTPS}}\rangle$, is expected to be a good approximation of the ground state.

In TeNeS, we assume that the Hamiltonian can be represented as a sum of short range two-body interactions as

$$\mathcal{H} = \sum_{\{(i,j)\}} H_{ij},$$

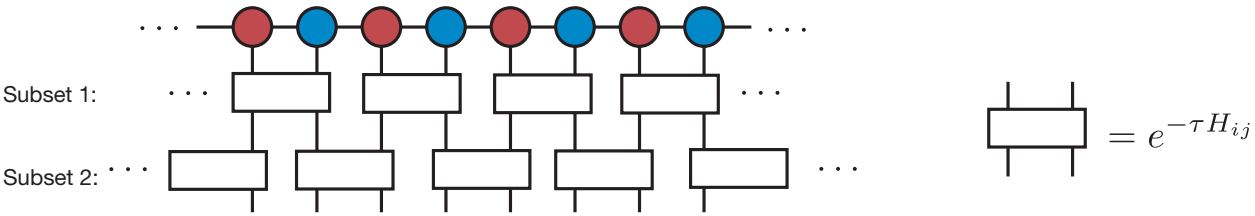
and apply Suzuki-Trotter decomposition to the ITE operator with small time step τ:

$$e^{-\tau\mathcal{H}} = \prod_{\{(i,j)\}} e^{-\tau H_{ij}} + O(\tau^2).$$

We can also consider higher order Suzuki-Trotter decomposition. By using the Suzuki-Trotter decomposition form, the ITE is represented as

$$e^{-T\mathcal{H}}|\Psi_0\rangle = \left(\prod_{\{(i,j)\}} e^{-\tau H_{ij}}\right)^{N_\tau} |\Psi_0\rangle + O(\tau),$$

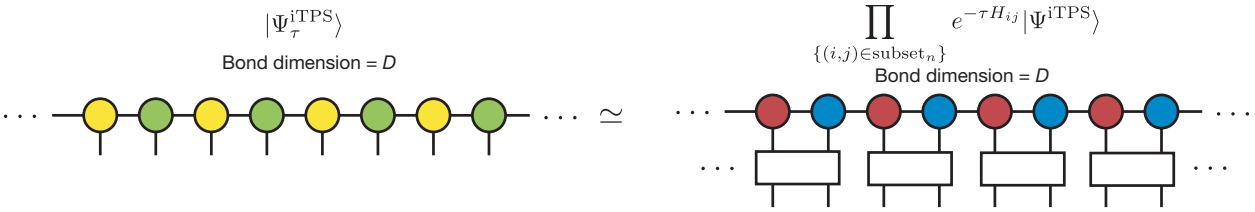
where $N_\tau = T/\tau$ is the number of ITEs with sufficiently small τ. In order to simulate the right hand side of the equation, we divide $\prod_{\{(i,j)\}}$ into several subsets. In each subset, (local) ITE operators satisfy two properties: they commute with each other and they have the same translation symmetry with the iTPS ansatz. For example, in the case of two-site iMPS for the one-dimensional nearest-neighbor interaction Hamiltonian, we have two subsets:



Then, we approximate the wavefunction after multiplication of each ITE-operator subset as an iTPS with the bond dimension D:

$$|\Psi_\tau^{\mathrm{iTPS}}\rangle \simeq \prod_{\{(i,j)\in \mathrm{subset}_n\}} e^{-\tau H_{ij}} |\Psi^{\mathrm{iTPS}}\rangle,$$

where $\prod_{\{(i,j)\in \mathrm{subset}_n\}}$ means the product of operators in the nth subset, and $|\Psi_\tau^{\mathrm{iTPS}}\rangle$ is a new iTPS. By using a diagram, it is represented as follows:



Notice that by applying $e^{-\tau H_{ij}}$ the bond dimension of the exact iTPS representation generally increases. In order to continue the simulation stably, we need to *truncate* the bond dimension to a constant D.

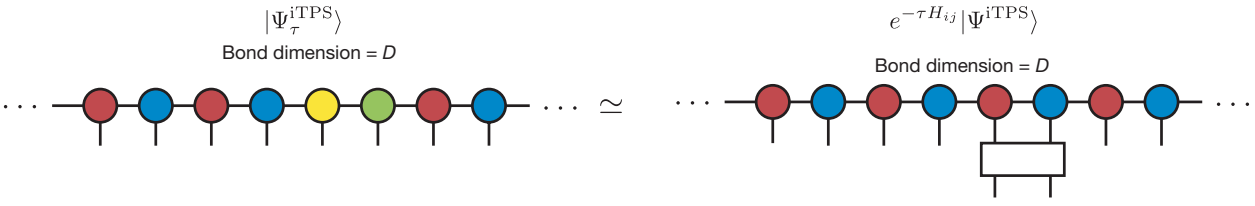
Naively, efficient truncation can be done by solving the minimization problem

$$\min \left\| |\Psi_\tau^{\mathrm{iTPS}}\rangle - \prod_{\{(i,j)\in \mathrm{subset}_n\}} e^{-\tau H_{ij}} |\Psi^{\mathrm{iTPS}}\rangle \right\|^2 .$$

However, in practice, solving this minimization problem needs huge computation cost because it is a highly nonlinear problem due to the translational symmetry of iTPS. Thus, instead, we usually consider an alternative local problem where we apply only a local ITE operator and try to find optimal iTPS $|\Psi_\tau^{\mathrm{iTPS}}\rangle$ in which only a few local tensors are modified from the original $|\Psi^{\mathrm{iTPS}}\rangle$. This minimization problem is written as

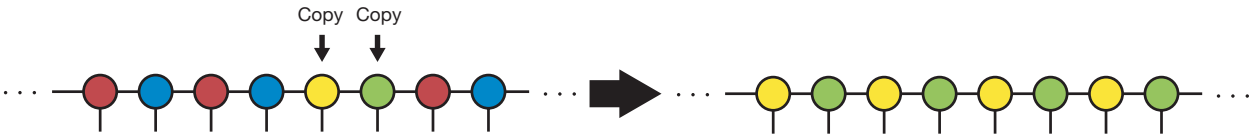
$$\min \left\| |\Psi_\tau^{\mathrm{iTPS}}\rangle - e^{-\tau H_{ij}} |\Psi^{\mathrm{iTPS}}\rangle \right\|^2 .$$

In the case of the nearest-neighbor interaction on the one-dimensional chain, the diagrams corresponding to this minimization problems are



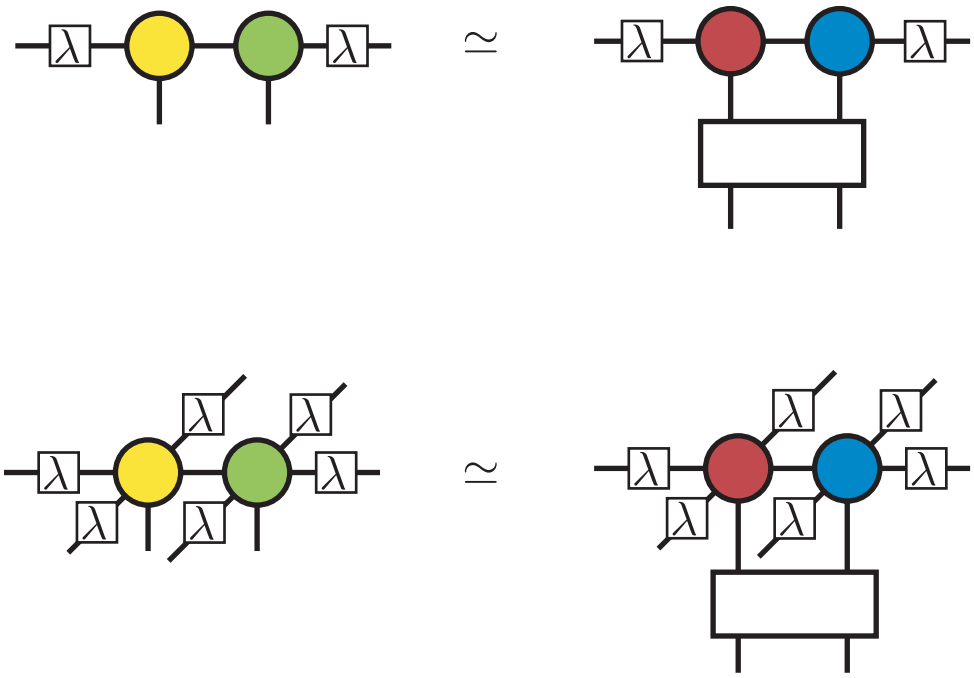
The squared norm $\left\| |\Psi_\tau^{\mathrm{iTPS}}\rangle - e^{-\tau H_{ij}} |\Psi^{\mathrm{iTPS}}\rangle \right\|^2$ can be calculated by using, *e.g.*, CTMRG and we can solve the minimization problem easily *[ITE]*. Although this new iTPS breaks translational symmetry, we make translationally

symmetric iTPS by *copying* updated local tensors to other parts so that the obtained iTPS can be considered as an approximated solution of the original minimization problem:

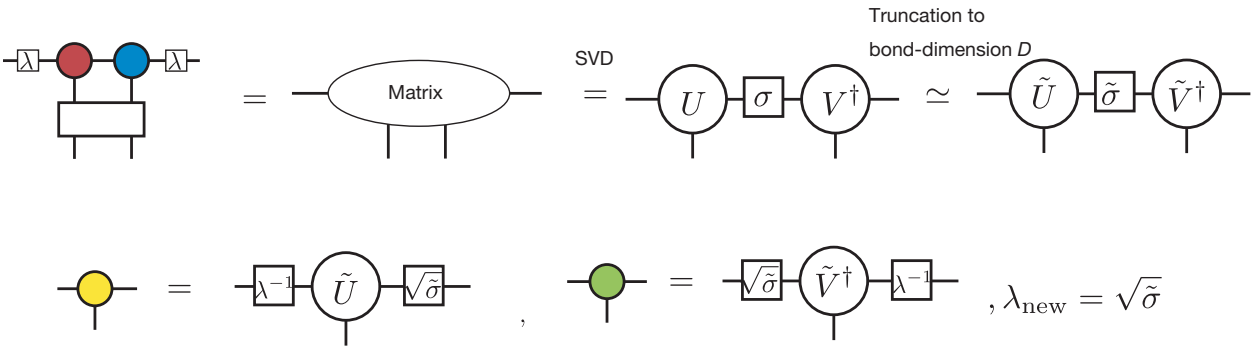


This ITE approach is often called as *full update*. The leading computation cost of the full update come from CTMRG and then it scales as $O(D^{10})$ or $O(D^{12})$ depending on SVD algorithms.

The *simple update* (or *simplified update*) is a cheaper version of ITE optimization. In order to avoid expensive environment calculation by CTMRG, we consider a part of the tensor network instead to treat the whole *[SimpleUpdate]* in the simple update. For example, in the case of the nearest-neighbor interaction, we consider the following local optimization problem:



In this diagram, λ_i represents a non-negative diagonal matrix considered to be a mean field corresponding to the neglected environment beyond the bond i. The definition of λ_i will be given later. This optimization problem can be viewed as the low rank approximation of a matrix consisting of two tensors and a ITE operator, and then we can solve it by SVD. The procedure of the simple update is given in the following diagram:



The singular values obtained from the SVD of the matrix is used as the mean field λ in the next step. The computation cost of the simple update is $O(D^5)$, if we use QR decomposition before we construct the matrix *[QR]*. Thus, it is much cheaper that that of the full update.

Although the computation cost of the simple update is cheaper than that of the full update, it is known that the simple update shows strong initial state dependence and it tends to overestimate the local magnetization. Thus, for complicated problems, we need to carefully check results obtained by the simple update.

References

[TNS] R. Orús, *A practical introduction to tensor networks: Matrix product states and projected entangled pair states*, Annals. of Physics **349**, 117 (2014). link; R. Orús, *Tensor networks for complex quantum systems*, Nature Review Physics **1**, 538 (2019). link.

[MPS] U. Schollwcök, *The density-matrix renormalization group in the age of matrix product states*, Annals. of Physics **326**, 96 (2011). link

[CTMRG] T. Nishino and K. Okunishi, *Corner Transfer Matrix Renormalization Group Method*, J. Phys. Soc. Jpn. **65**, 891 (1996).; R. Orús and G. Vidal, *Simulation of two-dimensional quantum systems on an infinite lattice revisited: Corner transfer matrix for tensor contraction*, Phys. Rev. B **80**, 094403 (2009). link ; P. Corboz *et al.*, *Competing States in the t-J Model: Uniform d-Wave State versus Stripe State*, Phys. Rev. Lett. **113**, 046402 (2014). link

[ITE] J. Jordan *et al.*, *Classical Simulation of Infinite-Size Quantum Lattice Systems in Two Spatial Dimensions*, Phys. Rev. Lett. **101**, 250602, (2008). link; R. Orús and G. Vidal, *Simulation of two-dimensional quantum systems on an infinite lattice revisited: Corner transfer matrix for tensor contraction*, Phys. Rev. B **80**, 094403 (2009). link

[SimpleUpdate] H. G. Jiang *et al.*, *Accurate Determination of Tensor Network State of Quantum Lattice Models in Two Dimensions*, Phys. Rev. Lett. **101**, 090603 (2008). link

[QR] L. Wang *et al.*, *Monte Carlo simulation with tensor network states*, Phys. Rev. B **83**, 134421 (2011). link

CHAPTER

SEVEN

ACKNOWLEDGEMENT

TeNeS was supported by MEXT as “Exploratory Challenge on Post-K computer” (Frontiers of Basic Science: Challenging the Limits) and “Priority Issue on Post-K computer” (Creation of New Functional Devices and High-Performance Materials to Support Next-Generation Industries). We also would also like to express our thanks for the support of the “*Project for advancement of software usability in materials science*” of The Institute for Solid State Physics, The University of Tokyo, for the development of TeNeS.

CHAPTER

EIGHT

CONTACTS

- Report bugs

  Please report all problems and bugs on the GitHub Issues page

  Follow these guidelines when reporting:

  - Please specify the version of TeNeS, OS, and compiler you are using.
  - If there are problems for installation, please include input / output of `cmake` and `make`, and CMakeCache.txt (one of the output file of `cmake`).
  - If a problem occurs during execution, please show the input file used and obtained output.

  Thank you for your cooperation.

- Others

  If you have any questions about topics related to your research that are difficult to consult in public (e.g., at Issue page on GitHub), please send an e-mail to the following address:

  E-mail: `tenes-dev__at__issp.u-tokyo.ac.jp` (replace `__at__` by `@`).