

# TeNeS

**TeNeS Documentation**

*Release 2.0.0*

**Institute for Solid State Physics, University of Tokyo**

Nov 17, 2023



# CONTENTS

<b>1</b>	<b>What is TeNeS ?</b>	<b>1</b>
1.1	Overview . . . . .	1
1.2	Developers . . . . .	1
1.3	Version information . . . . .	1
1.4	License . . . . .	2
1.5	Papers . . . . .	2
1.6	Copyright . . . . .	2
<b>2</b>	<b>Install</b>	<b>3</b>
2.1	Download . . . . .	3
2.2	Prerequisites . . . . .	3
2.3	Install . . . . .	4
<b>3</b>	<b>Usage</b>	<b>7</b>
3.1	Usage of <code>tenes_simple</code> . . . . .	7
3.2	Usage of <code>tenes_std</code> . . . . .	9
3.3	Usage of <code>tenes</code> . . . . .	11
<b>4</b>	<b>Tutorial</b>	<b>13</b>
4.1	Ising model with transverse magnetic field . . . . .	13
4.2	Real-Time Evolution of the Transverse Field Ising Model . . . . .	17
4.3	Finite Temperature Calculations for the Transverse Field Ising Model . . . . .	19
4.4	Magnetization process of the Heisenberg model on triangular and square lattices . . . . .	25
4.5	Phase diagram of the hardcore boson model on a triangular lattice . . . . .	29
4.6	Definition of lattices, models, and operators using the standard mode . . . . .	31
<b>5</b>	<b>File format</b>	<b>41</b>
5.1	Short summary for input files of TeNeS . . . . .	41
5.2	Input file for <code>tenes_simple</code> . . . . .	42
5.3	Input file for <code>tenes_std</code> . . . . .	54
5.4	Input file for <code>tenes</code> . . . . .	66
5.5	Output files . . . . .	78
<b>6</b>	<b>Algorithm</b>	<b>89</b>
6.1	Tensor Network States . . . . .	89
6.2	Contraction of iTPS . . . . .	90
6.3	Optimization of iTPS . . . . .	93
6.4	Real-time evolution by iTPS . . . . .	96
6.5	Finite temperature simulation . . . . .	96
<b>7</b>	<b>FAQ</b>	<b>101</b>

<b>8</b>	<b>Acknowledgement</b>	<b>103</b>
<b>9</b>	<b>Contacts</b>	<b>105</b>

## 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. TeNeS can calculate finite temperature quantity and real-time evolution as well as the ground-state quantity. 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 (Faculty of Science and Technology, Keio University)
- 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. 2.0-beta: released on 2023-10-25.
- ver. 1.3.4: released on 2023-09-13.
- ver. 1.3.3: released on 2023-07-14.
- ver. 1.3.2: released on 2023-06-08.
- ver. 1.3.1: released on 2022-10-21.
- ver. 1.3.0: released on 2022-10-20.
- ver. 1.2.0: released on 2021-12-13.
- ver. 1.1.1: released on 2020-11-09.

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

## 1.4 License

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

## 1.5 Papers

When you publish the results by using TeNeS, we would appreciate if you cite the following paper:

Y. Motoyama, Tsuyoshi Okubo, Kazuyoshi Yoshimi, Satoshi Morita, Takeo Kato, and Naoki Kawashima, “TeNeS: Tensor Network Solver for Quantum Lattice Systems”, *Comput. Phys. Commun.* 279, 108437 (2022)

## 1.6 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.

## 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 downloading:

```
$ 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](#)

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 privileges, 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` . Additionally, the following python packages are also required.

1. `numpy`
2. `scipy`
3. `toml`

## 2.3 Install

1. Build TeNeS by typing the following commands (Some environment such as CentOS provides CMake3 as cmake3):

```
$ 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`.

---

### Parallel Build

The `make` command accepts `-j <num>` options and then uses `<num>` processes for a parallel building. This reduces the time to build TeNeS drastically.

The executable file `tenes` 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 in the following way,

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

---

### Specify ScaLAPACK

CMake detects your ScaLAPACK library automatically but may fail. In this case, you can specify the ScaLAPACK library (`<path>/lib/libscalapack.so`) in the following way,

```
$ cmake -DSCALAPACK_ROOT=<path> ../
```

---

### Use the pre-built mptensor



TeNeS is based on the parallelized tensor library `mptensor`. The build system of TeNeS installs this automatically, but if you want to use the specific version of the `mptensor` (`<path>/lib/libmptensor.a`), please add the following option in `cmake`.

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

---

---

### 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 in the following way,

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

---



## 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.

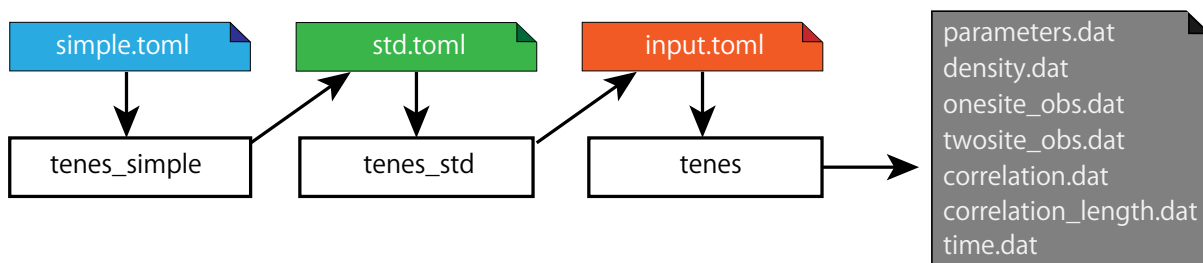


Fig. 3.1: Schematic calculation flow of TeNeS

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.
- --use-site-hamiltonian
  - \* Onsite terms in Hamiltonian like Zeeman term will be output as site Hamiltonians
  - \* If not specified, these terms will be absorbed into the nearest neighbor bond Hamiltonians

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
```

(continues on next page)

(continued from previous page)

```
[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  $\tau$ , 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
```

(continues on next page)

(continued from previous page)

```

# 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]]
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
""

```

(continues on next page)

(continued from previous page)

```

[[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
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 onsite 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.



## TUTORIAL

## 4.1 Ising model with transverse magnetic field

This section presents a calculation of the transverse magnetic field Ising model as an example. The Hamiltonian is

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

Please note that the model is defined using spin operators of size 1/2, not Pauli operators. By changing the variable `hx` 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 # Limit tensor elements in real (not complex)

[parameter.simple_update]
num_step = 1000 # Number of steps in simple update
tau = 0.01      # Imaginary time slice

[parameter.full_update]
num_step = 0    # Number of steps in full update
tau = 0.01     # Imaginary time slice

[parameter.ctm]
meanfield_env = false # Use meanfield environment to contract iTNS
iteration_max = 10    # Maximum number of iterations in CTMRG
dimension = 10       # Bond dimension of corner transfer matrix

[lattice]
type = "square lattice" # Type of lattice
L = 2                  # X length of unit cell
W = 2                  # Y length of unit cell
virtual_dim = 2        # Bond dimension of bulk tensors
initial = "ferro"      # Initial condition

[model]
type = "spin" # Type of model
Jz = -1.0     # Jz SzSz
Jx = 0.0      # Jx SxSx
Jy = 0.0      # Jy SySy
hx = 0.0      # hx Sx
```

In this case, since  $J_z = -1.0$ , the ferromagnetic state manifests itself as the ground state at  $h_x=0$ . 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 on-site operators
Save on-site observables to output_0/onsite_obs.dat
Start calculating two-site operators
Save two-site observables to output_0/twosite_obs.dat
Save observable densities to output_0/density.dat
Save elapsed times to output_0/time.dat

On-site observables per site:
Sz          = 0.5 0
Sx          = -1.28526262482e-13 0
Two-site 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
environment  = 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  $S_z$ ,  $S_x$  and two-site ones Hamiltonian, the nearest correlation  $S_z S_z$ ,  $S_x S_x$ ,  $S_y S_y$  are output. Finally, the calculation time for each process is output in units of seconds. `density.dat`, `parameters.dat`, `time.dat`, `onsite_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 `onsite_obs.dat`. By changing `hx` 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

MPI_cmd = "" # e.g., "mpiexec -np 1"

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(f"Calculation Process: {idx+1}/{num_hx}")
    with open("simple.toml") as f:
        dict_toml = toml.load(f)
        dict_toml["parameter"]["general"]["output"] = f"output_{idx}"
        dict_toml["model"]["hx"] = float(hx)

    simple_toml = f"simple_{idx}.toml"
    std_toml = f"std_{idx}.toml"
    input_toml = f"input_{idx}.toml"

    with open(simple_toml, "w") as f:
        toml.dump(dict_toml, f)
    cmd = f"tenes_simple {simple_toml} -o {std_toml}"
    subprocess.call(cmd.split())

    cmd = f"tenes_std {std_toml} -o {input_toml}"
    subprocess.call(cmd.split())

    cmd = f"{MPI_cmd} tenes {input_toml}"
    subprocess.call(cmd.split())
```

- `tutorial_read.py`

```
from os.path import join

import toml

num_hx = 16

print("# $1: h")
print("# $2: ene")
print("# $3: sz")
print("# $4: sx")
print()
```

(continues on next page)

(continued from previous page)

```

for idx in range(num_hx):
    try:
        with open(f"simple_{idx}.toml") 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(f"output_{idx}", "density.dat")) as f:
                for line in f:
                    words = line.split()
                    if words[0] == "Energy":
                        ene = words[2]
                    elif words[0] == "Sz":
                        mag_sz = words[2]
                    elif words[0] == "Sx":
                        mag_sx = words[2]
            print(f"{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, hx, energy,  $\langle S_z \rangle$  and  $\langle S_x \rangle$  are outputted in the standard output:

```
$ python tutorial_read.py
```

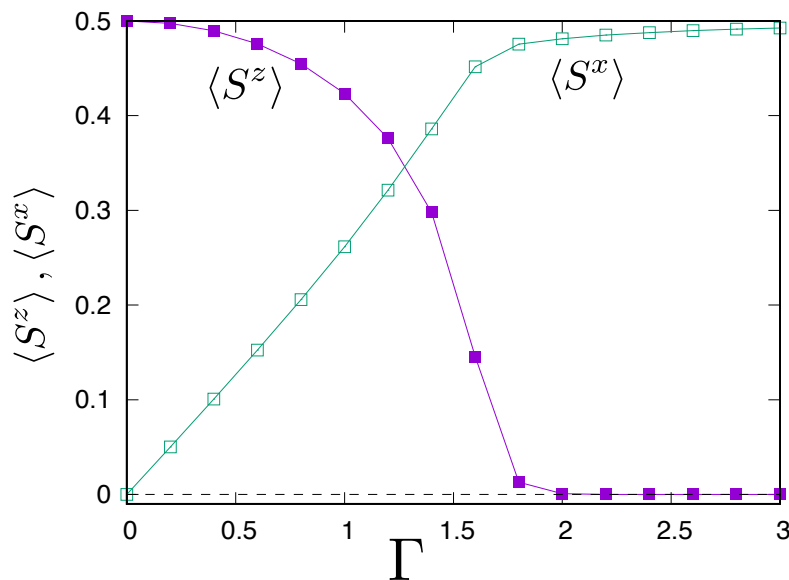


Fig. 4.1:  $\Gamma$  dependence of  $\langle S_z \rangle$  and  $\langle S_x \rangle$ .

As seen from Fig. 4.1, with increasing  $\Gamma$ , the  $\langle S_z \rangle$  decreases from 0.5 to 0, while the  $\langle S_x \rangle$  increases from 0 to 0.5.

## 4.2 Real-Time Evolution of the Transverse Field Ising Model

Here, we introduce a calculation example for the real-time evolution of the Ising model on a square lattice when a transverse magnetic field, denoted by  $h_x$ , is applied. The Hamiltonian is

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

Please note that the model is defined using spin operators of size  $1/2$ , not Pauli operators. The input and script files used in this tutorial can be found in `sample/02_time_evolution`.

Initially, we compute the ground state (refer to the `simple.toml` file) which serves as our starting state. Specifically, it's set as:

```
[parameter]
[parameter.general]
output = "output"
tensor_save = "save_tensor"

[parameter.simple_update]
num_step = 10
tau = 0.01

[parameter.full_update]
num_step = 0
tau = 0.0

[parameter.ctm]
meanfield_env = true
iteration_max = 10
dimension = 10

[lattice]
type = "square lattice"
L = 2
W = 2
virtual_dim = 3
initial = "ferro"

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

Given that  $J_z = -1.0$ , the ground state becomes ferromagnetic. We use the ground state as the initial state, and save the state tensor with `tensor_save = "save_tensor"`.

Next, we prepare the input file for the real-time evolution. This can be achieved by setting the mode to `time`. Below is a sample input file (`simple_te_strong.toml`):

```
[parameter]
[parameter.general]
output = "output_te_strong"
```

(continues on next page)

(continued from previous page)

```

tensor_load = "save_tensor"
mode = "time"

[parameter.simple_update]
num_step = 500
tau = 0.01

[parameter.full_update]
num_step = 0
tau = 0.0

[parameter.ctm]
meanfield_env = true
iteration_max = 10
dimension = 10

[lattice]
type = "square lattice"
L = 2
W = 2
virtual_dim = 3
initial = "ferro"

[model]
type = "spin"
Jz = -1.0
Jx = 0.0
Jy = 0.0
hx = 2.0

```

In this case, the transverse field is set to  $h_x = 2.0$ , and the time-step for evolution is  $\tau = 0.01$ . Moreover, since we are utilizing the ground state as our initial condition, we load the state tensor with `tensor_load = "save_tensor"`.

Once preparing the input file, we execute `tenes_simple`, `tenes_std`, and `tenes` in order. The results are saved in the `output_te_strong` directory. Basically, the output is the same as the ground state, but with the addition of time in the first column. For example, `FT_density.dat` records the expectation values of physical quantities over time:

```

# The meaning of each column is the following:
# $1: time
# $2: observable ID
# $3: real
# $4: imag
# The meaning of observable IDs are the following:
# 0: Energy
# 1: Sz
# 2: Sx
# 3: Sy
# 4: bond_hamiltonian
# 5: SzSz
# 6: SxSx
# 7: SySy

0.0000000000000000e+00 0 -5.00184764052080899e-01 0.0000000000000000e+00

```

(continues on next page)

(continued from previous page)

```
0.0000000000000000e+00 1 4.99999945646528332e-01 0.0000000000000000e+00
0.0000000000000000e+00 2 9.24306486797199186e-05 0.0000000000000000e+00
0.0000000000000000e+00 3 2.34088935337348195e-06 0.0000000000000000e+00
0.0000000000000000e+00 4 -5.00184764052080899e-01 3.47535331983321418e-21
0.0000000000000000e+00 5 4.99999902788251294e-01 -8.46256269499545126e-22
0.0000000000000000e+00 6 1.12653588020163689e-05 6.35907290717320676e-22
0.0000000000000000e+00 7 -1.12840199341671039e-05 -2.06527532941704114e-21
```

The second column represents the type of physical quantity, and in this case, 1 represents the longitudinal magnetization  $m^z = \langle S^z \rangle$ . We can extract the time evolution of the magnetization by extracting rows with the second column equal to 1:

```
awk '$2 == 1 {print $1, $3, $4}' output_te_strong/TE_density.dat > magnetization_strong.
↪ dat
```

For observing the time evolution with different transverse magnetic fields, we've also prepared sample input files named `simple_te_middle.toml` ( $h_x = 0.8$ ) and `simple_te_weak.toml` ( $h_x = 0.5$ ). Additionally, there's a script named `run.sh` to execute these calculations in one go. Ensure that paths to tools like `tenes` are set correctly, and then execute the calculations with:

```
sh run.sh
```

The computation will conclude in several seconds. Once done, launch `gnuplot` and enter:

```
load 'plot.plt'
```

This will plot the temporal evolution of magnetization,  $S_z$ . The result is displayed in [Fig. 4.2](#).

When the strength of the transverse-field exceeds the quantum phase transition point, the magnetization oscillates beyond 0 [\[DQPT\]](#).

As time evolution progresses, the entanglement increases. At a certain point, the tensor network's capacity may be insufficient to express the wave function. In our case, the jump at  $t=4.25$  for  $h_x=2.0$  indicates this issue. When applying this in practice, ensure no such discontinuities exist. If jumps are observed, steps like increasing the `virtual_dimension` might be necessary. For instance, adjusting it to `virtual_dimension = 10` and redoing the calculation as described above will eliminate the discontinuity, as can be seen in [Fig. 4.3](#).

## Reference

[DQPT] M. Heyl, A. Polkovnikov, and S. Kehrein, *Dynamical Quantum Phase Transitions in the Transverse-Field Ising Model*, Phys. Rev. Lett. **110**, 135704 (2013). [link](#)

## 4.3 Finite Temperature Calculations for the Transverse Field Ising Model

In this section, we present a calculation example of the ferromagnetic Ising model on a square lattice subjected to a transverse magnetic field, denoted by  $h_x$ , at finite temperatures. The Hamiltonian is

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

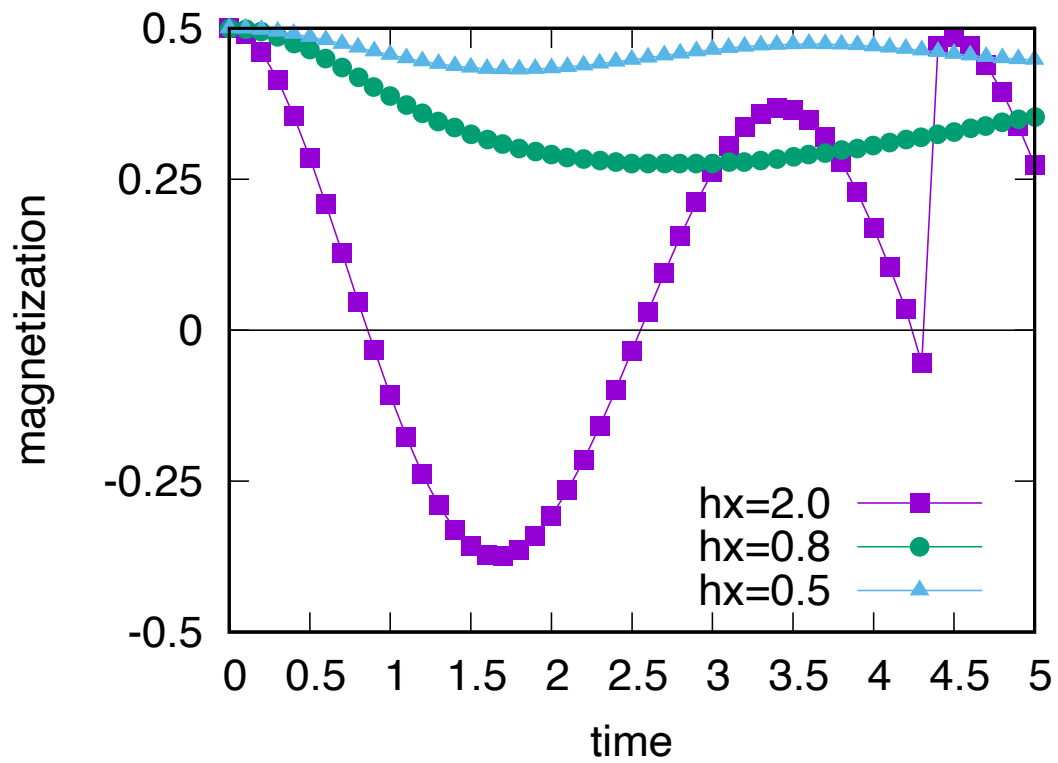


Fig. 4.2: Graph illustrating the real-time evolution of the Ising model. The vertical axis represents magnetization, and the horizontal axis represents time.



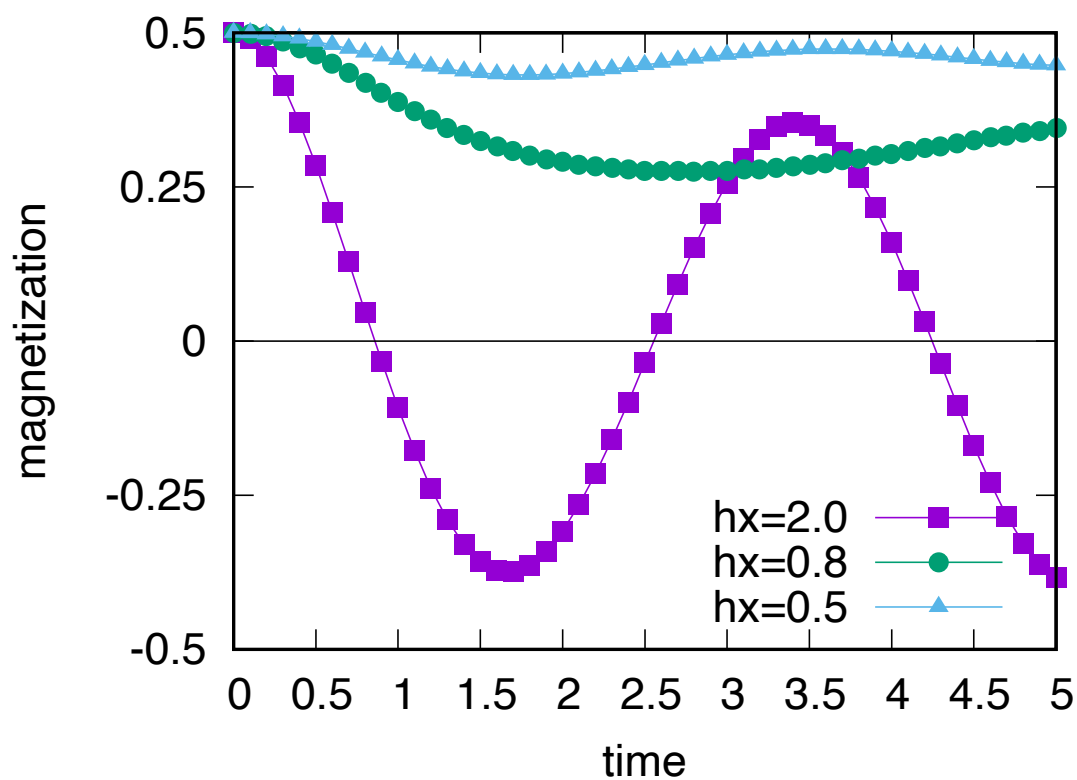


Fig. 4.3: Graph showcasing the real-time evolution of the Ising model. The vertical axis denotes magnetization, while the horizontal axis represents time. Results when `virtual_dimension = 10` are applied.

Please note that the model is defined using spin operators of size 1/2, not Pauli operators. The input and script files used in this tutorial are located in the `sample/03_finite_temperature` directory. Below is a sample input file (`simple_ft_strong.toml`):

```
[parameter]
[parameter.general]
mode = "finite"
is_real = false
output = "output_ft_strong"
measure_interval = [10, 10, 5]

[parameter.simple_update]
num_step = [50, 200, 10]
tau = [0.01, 0.005, 0.05]

[parameter.full_update]
num_step = 0
tau = 0.0

[parameter.ctm]
iteration_max = 10
dimension = 10

[lattice]
type = "square lattice"
L = 2
W = 2
virtual_dim = 3

[model]
type = "spin"
Jz = -1.0
Jx = 0.0
Jy = 0.0
hx = 2.0
```

To perform finite temperature calculations, set the `mode` to `finite`. Here, the transverse magnetic field is set to `hx = 2.0` with `tau = 0.01` (the inverse temperature step size is 2 times `tau`). Once preparing an input file of the simple mode, execute `tenes_simple`, `tenes_std`, and `tenes` in the same way as for the ground state calculation.

The results of the finite temperature calculations are output to the `output_ft_strong` directory. Basically, the output is the same as the ground state calculation, but the inverse temperature is added to the first column. For example, `FT_density.dat` is as follows:

```
# The meaning of each column is the following:
# $1: inverse temperature
# $2: observable ID
# $3: real
# $4: imag
# The meaning of observable IDs are the following:
# 0: Energy
# 1: Sz
# 2: Sx
# 3: Sy
```

(continues on next page)

(continued from previous page)

```
# 4: bond_hamiltonian
# 5: SzSz
# 6: SxSx
# 7: SySy

0.0000000000000000e+00 0 0.0000000000000000e+00 0.0000000000000000e+00
0.0000000000000000e+00 1 0.0000000000000000e+00 0.0000000000000000e+00
0.0000000000000000e+00 2 0.0000000000000000e+00 0.0000000000000000e+00
0.0000000000000000e+00 3 0.0000000000000000e+00 0.0000000000000000e+00
0.0000000000000000e+00 4 0.0000000000000000e+00 0.0000000000000000e+00
0.0000000000000000e+00 5 0.0000000000000000e+00 0.0000000000000000e+00
0.0000000000000000e+00 6 0.0000000000000000e+00 0.0000000000000000e+00
0.0000000000000000e+00 7 0.0000000000000000e+00 0.0000000000000000e+00

... continued ...
```

The second column indicates the type of physical quantity, and for example, 0 represents energy. Thus, you can extract the temperature dependence by extracting only the energy with awk:

```
awk '$2 == 0 {print $1, $3, $4}' output_ft_strong/FT_density.dat > energy_strong.dat
```

To observe the behavior at different transverse magnetic fields, we've provided additional sample input files: `simple_ft_middle.toml` ( $h_x = 0.8$ ), `simple_te_weak.toml` ( $h_x = 0.5$ ), and `simple_ft_zero.toml` ( $h_x = 0.0$ ). Moreover, a script named `run.sh` has been set up to execute all these calculations simultaneously. Ensure you've added tools like `tenes` to your `PATH`, then initiate the calculations with:

```
sh run.sh
```

The computation should complete in about a minute. Since specific heat is difficult to calculate directly, it is calculated from the energy by numerical differentiation. `calcspec.py` is a script to calculate specific heat from the energy by using the spline interpolation:

```
python3 calcspec.py
```

To visualize the results, scripts have been prepared to plot energy, heat capacity, and magnetization ( $S_x$ ,  $S_z$ ): `plot_e.plt`, `plot_c.plt`, `plot_mx.plt`, and `plot_mz.plt`. Running the following:

```
gnuplot -persist plot_e.plt
gnuplot -persist plot_c.plt
gnuplot -persist plot_mx.plt
gnuplot -persist plot_mz.plt
```

will display plots for energy, heat capacity, and magnetizations ( $m_x$  and  $m_z$ ). The resulting plots are illustrated in [Fig. 4.4](#). For comparison, results obtained using Quantum Monte Carlo calculations are also shown (using `ALPS/looper`).

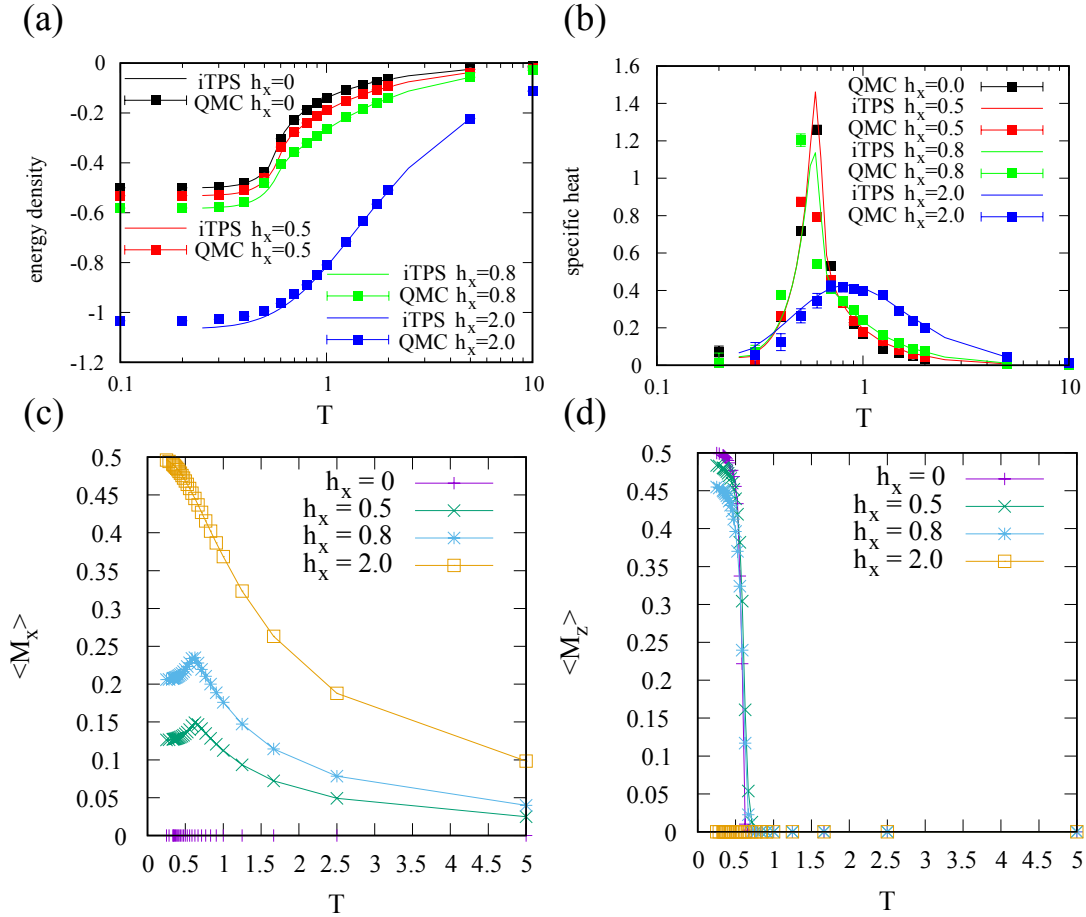


Fig. 4.4: Graphs for the finite temperature calculations of the Ising model: (a) energy, (b) heat capacity, (c)  $m_x$ , and (d)  $m_z$ . The vertical axis represents the physical quantity, and the horizontal axis denotes temperature.

## 4.4 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/04_magnetization` directory. The `basic.toml` file contains model settings and parameters.

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

[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  $3 \times 3$ . Here, in order to make the calculation lighter, only `simple update` is performed, and the imaginary time interval  $\tau$  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
```

(continues on next page)

(continued from previous page)

```

MPI_cmd = "" # e.g., "mpiexec -np 1"

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 f in (fmag, fene):
    f.write("# $1: hz\n")
    for i, num_step in enumerate(num_step_table, 2):
        f.write(f"# ${i}: num_step={num_step}\n")
    f.write("\n")

for idx, h in enumerate(np.linspace(min_h, max_h, num=num_h)):
    print(f"Calculation Process: {idx+1}/{num_h}")
    inum = 0
    num_pre = 0
    fmag.write(f"{h} ")
    fene.write(f"{h} ")
    for num_step in num_step_table:
        ns = num_step - num_pre
        print(f"Steps: {num_step}")
        with open("basic.toml") as f:
            dict_toml = toml.load(f)

        output_dir = f"output_{idx}_{num_step}"

        dict_toml["parameter"]["general"]["output"] = output_dir
        dict_toml["parameter"]["general"]["tensor_save"] = "tensor_save"
        dict_toml["model"]["hz"] = float(h)
        dict_toml["parameter"]["simple_update"]["num_step"] = ns
        if inum > 0:
            dict_toml["parameter"]["general"]["tensor_load"] = "tensor_save"

        simple_toml = f"simple_{idx}_{num_step}.toml"
        std_toml = f"std_{idx}_{num_step}.toml"
        input_toml = f"input_{idx}_{num_step}.toml"

        with open(simple_toml, "w") as f:
            toml.dump(dict_toml, f)
        cmd = f"tenes_simple {simple_toml} -o {std_toml}"
        subprocess.call(cmd.split())

        cmd = f"tenes_std {std_toml} -o {input_toml}"
        subprocess.call(cmd.split())

        cmd = f"{MPI_cmd} tenes {input_toml}"
        subprocess.call(cmd.split())

```

(continues on next page)

(continued from previous page)

```

ene = 0.0
mag_sz = 0.0
with open(join(output_dir, "density.dat")) as f:
    for line in f:
        name, vals = line.split("=")
        if name.strip() == "Energy":
            re, im = vals.split()
            ene += float(re)
        elif name.strip() == "Sz":
            re, im = vals.split()
            mag_sz += float(re)
fene.write(f"{ene} ")
fmag.write(f"{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. 4.5](#). In a similar way,

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

we obtain the ground-state energy as shown in the left panel of [Fig. 4.5](#).

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.

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/04_magnetization` directory. The content

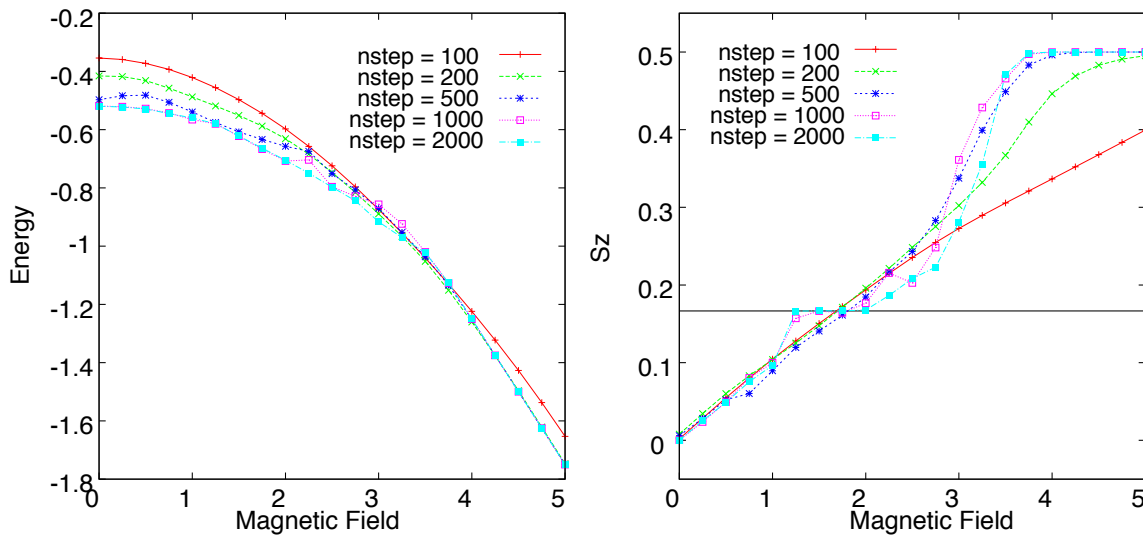


Fig. 4.5: Ground state energy (left figure) and magnetization (right figure) of the Heisenberg model on the triangular lattice.

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. 4.6 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. 4.6. 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.



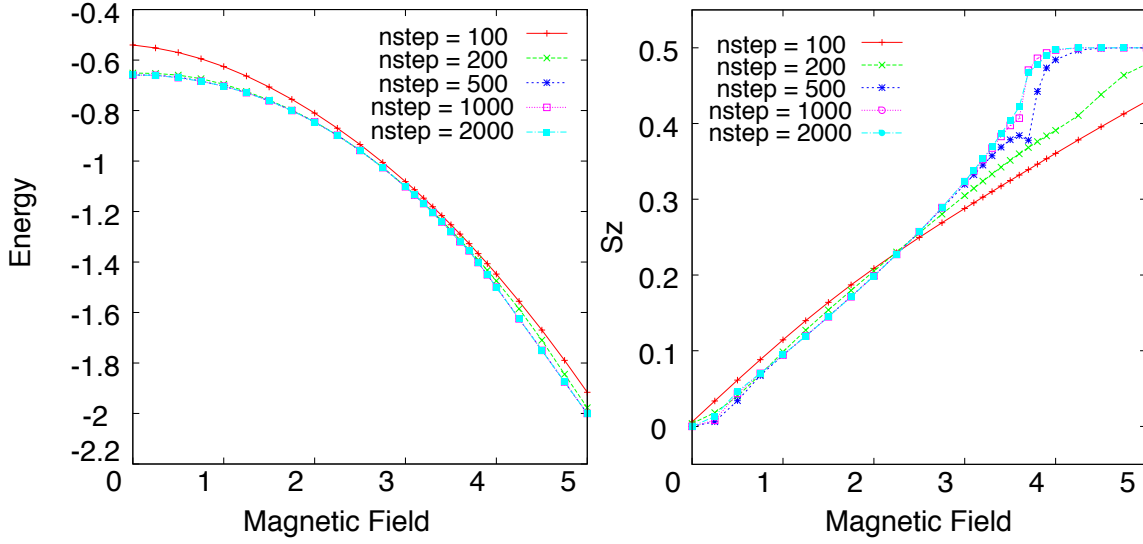


Fig. 4.6: Ground state energy (left figure) and magnetization (right figure) of the Heisenberg model on the square lattice.

## 4.5 Phase diagram of the hardcore boson model on a triangular lattice

Finally, let us consider a zero-temperature phase diagram of the hardcore boson model on a triangular lattice. The Hamiltonian of this model is given as

$$H = \sum_{\langle i,j \rangle} \left[ -t(b_i^\dagger b_j + b_j^\dagger b_i) + V n_i n_j \right] - \mu \sum_i n_i,$$

where  $\langle i, j \rangle$  indicates a pair of the nearest-neighbor sites,  $\mu$  is a chemical potential,  $t$  is a hopping energy,  $V$  is a strength of the nearest-neighbor interaction. For a hardcore boson system, the maximum number of bosons at each site is restricted to 0 or 1. It is known that several ordered phases characterized by two types of long-range order appear in this model [Wessel]. One is a superfluid order which is characterized by the offdiagonal order parameter  $|\langle b \rangle|$ . The other one is a solid-like order which exists at a  $1/3$  filling, where one of three sites is filled in a  $\sqrt{3} \times \sqrt{3}$  ordering with wave vector  $\mathbf{Q} = (4\pi/3, 0)$  (see the inset of Fig. 4.7). This long-range order is characterized by the structure factor  $S(\mathbf{Q}) = \sum_{i,j}^{N_{\text{sites}}} \langle n_i n_j \rangle \exp[-i\mathbf{Q} \cdot (\mathbf{r}_i - \mathbf{r}_j)] / N_{\text{sites}}$ .

To perform calculation for this system, the user can use toml files named `basic.toml`, `nn_obs.toml` and a python script file `run.py` in the direction `sample/05_hardcore_boson_triangular`. Here, `basic.toml` specifies the model and its parameters. This file is almost the same as the triangular Heisenberg model described in the previous section and differs from it only in the section `model` in the last part, where the line `type = "boson"` specifies the hardcore boson model and `t = 0.1`, `V = 1` determines the strength of the hopping and nearest-neighbor interaction.

To calculate the structure factor  $S(\mathbf{Q})$ , the correlations of densities at all the pairs of sites (in the unitcell)  $\langle n_i n_j \rangle$  are required. These observables are not defined by `tenes_simple`, users should define them by themselves. `nn_obs.toml` specifies them for  $3 \times 3$  unitcell, and the content of the file is appended to `std_XXX_YYY.toml` in `run.py`.

For larger unitcell, the computational cost increases, and hence the calculation of the structure factor gets difficult. On the other hand, TeNeS can calculate the correlation functions along the x and y directions (in the square lattice iTPS) at a low cost. Structure factor can be calculated using these correlation functions. Additionally, the Fourier transform of the density operator  $\langle n_i \rangle$  is also available. The ground state at  $1/3$  filling is three-fold degenerate, but one of them is selected in the finite bond dimension calculation, and calculated density has site dependence.

Let us execute calculation using the script `run.py`. After setting of the paths, execute calculation by typing the following command:

```
python run.py
```

The calculation will finish within several minutes or several tens of minutes. After the calculation, start `gnuplot` and execute the following command:

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

Then, we obtain a graph as shown in Fig. 4.7.  $S(Q)$  is the structure factor calculated from the density correlations at all the pairs of sites in the unitcell,  $S'(Q)$  is the structure factor calculated from the density correlations along the x-axis,  $n(Q)$  is the Fourier transform of the densities, and  $(|\langle b \rangle| + |\langle b^\dagger \rangle|)/2$  is the superfluid order parameter. We note that this calculation is not so accurate because the bond dimension used in the calculation is small. By increasing the bond dimensions specified in the beginning of the script `run.py`, we can perform more accurate calculation at the expense of execution time. From these figures, we find that there exists three phases for the ground state, i.e., (a) a superfluid phase ( $-0.5 \lesssim \mu/V \lesssim -0.2$ ), (b) a solid phase ( $-0.2 \lesssim \mu/V \lesssim 2.4$ ), and (c) a supersolid phase ( $2.4 \lesssim \mu/V$ ). This result is consistent with the previous work [Wessel] .

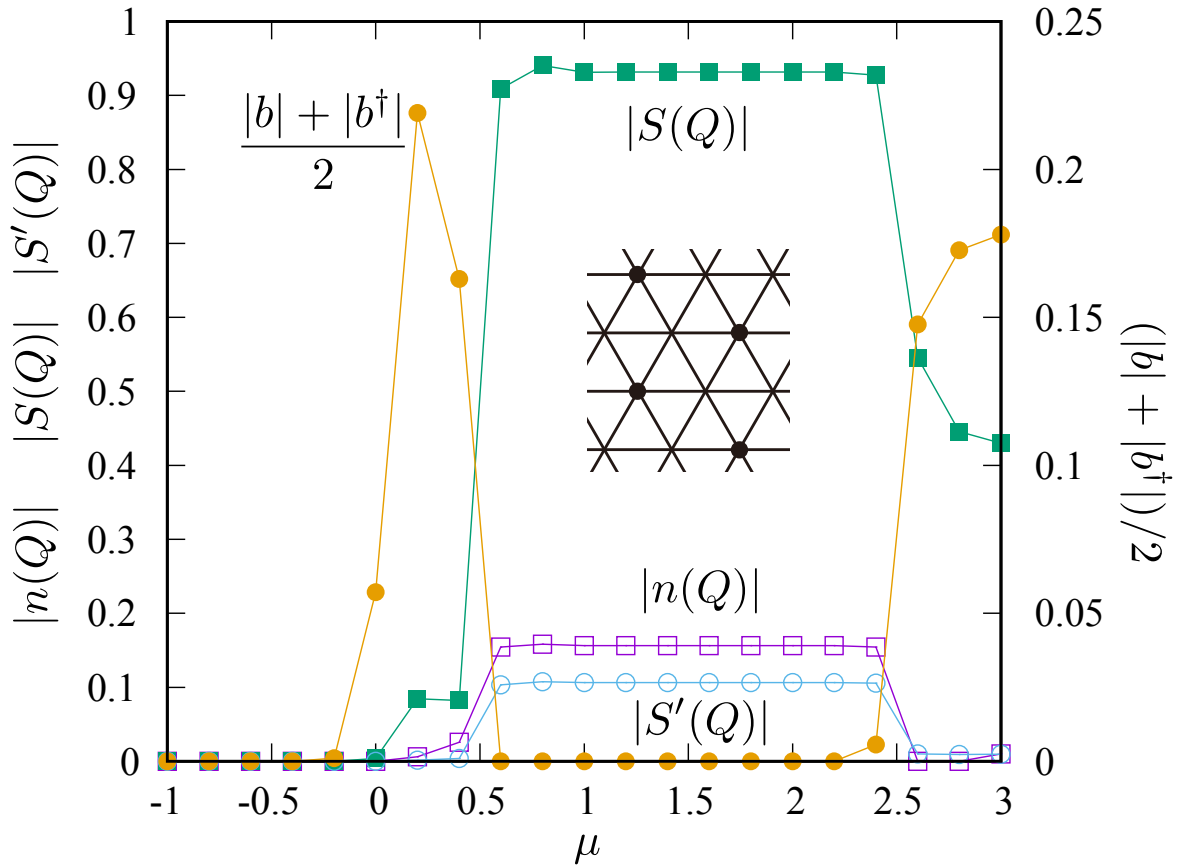


Fig. 4.7: Phase diagram of the hardcore boson model on a triangular lattice. Inset shows a particle pattern in a solid phase.

## Reference

[Wessel] S. Wessel, M. Troyer, *Supersolid hard-core boson on the triangular lattice*, Phys. Rev. Lett. **95**, 127205 (2005). [link](#).

## 4.6 Definition of lattices, models, and operators using the standard mode

By using the standard mode, users can define own lattices, models, and operators. In this section, we explain how to use the standard mode.

### 4.6.1 Definition of unit cell

Unit cells are defined using `[tensor]` and `[[tensor.unitcell]]`:

```
[tensor]
L_sub = [2, 2]          # 2x2 unitcell
skew = 0                # Displacement in x direction
                        # when go beyond a y-direction boundary

[[tensor.unitcell]]
virtual_dim = [4, 4, 4, 4] # Bond dimensions (←, ↑, →, ↓)
index = [0, 3]            # Indices of tensors in the unit cell
physical_dim = 2          # Physical bond dimensions
initial_state = [1.0, 0.0] # Initial state coefficients
noise = 0.01              # Fluctuation of elements in initial tensor
```

The initial state  $|\Psi\rangle$  is prepared as the direct product state of the per-site initial states  $|\psi\rangle_i: |\psi\rangle = \otimes_i |\psi_i\rangle$ .  $|\psi\rangle_i$  can be specified as follows, with the elements of the `initial_state = [a0, a1, ..., a_{d-1}]`,

$$|\psi\rangle_i \propto \sum_{k=0}^{d-1} a_k |k\rangle$$

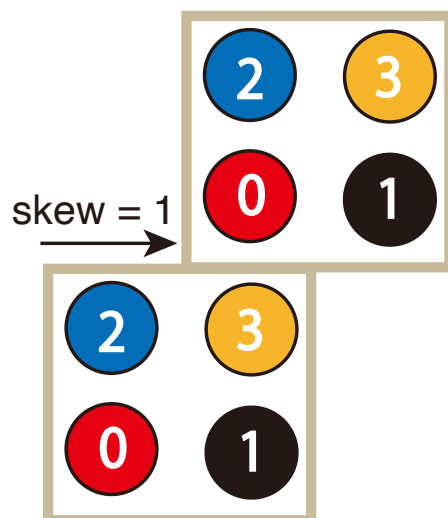
### 4.6.2 Definition of model (Hamiltonian)

TeNeS treats the Hamiltonian as the sum of bond Hamiltonians (two-site Hamiltonians) and site Hamiltonians (one-site Hamiltonians).

$$\mathcal{H} = \sum_{i,j} \mathcal{H}_{i,j} + \sum_i \mathcal{H}_i$$

These local Hamiltonians are defined as pairs of (nonzero) elements of matrix and site/bond it acts on. A bond is a directed pair of two sites: source and target. To define matrix elements is to define a model, and to define bonds is to define a lattice.

`[tensor]`



4-site unit cell ( $L_{\text{sub}} = [2, 2]$ )

`[[tensor.unitcell]]`

`virtual_dim = [4, 4, 4, 4]`

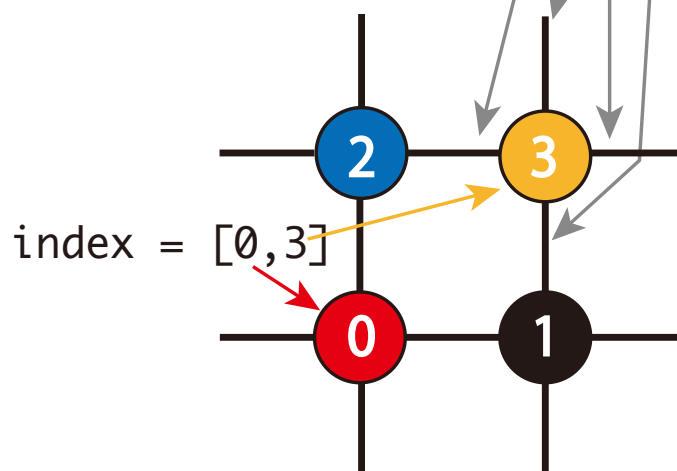
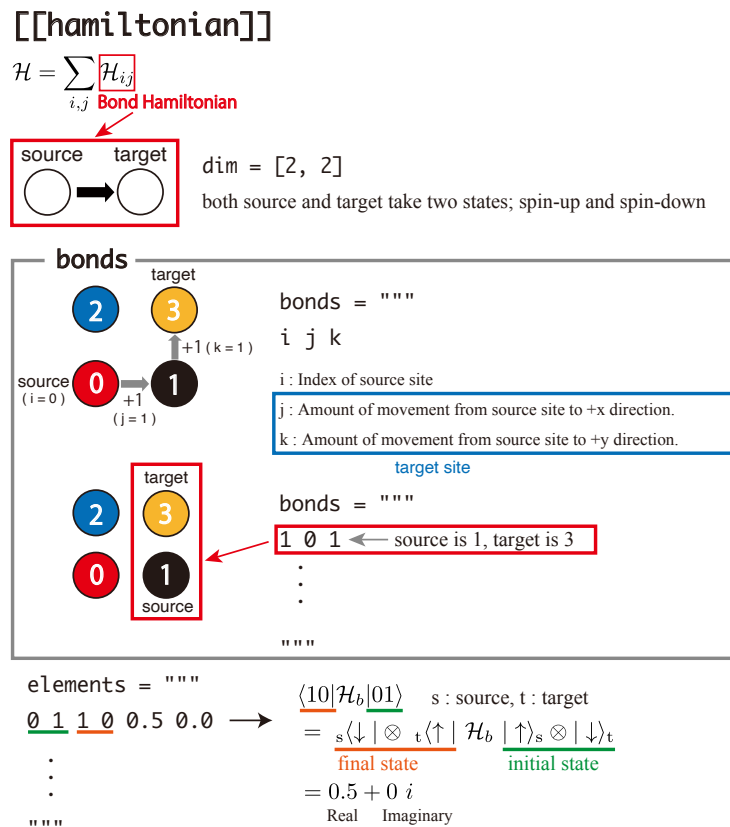


Fig. 4.8: `[tensor]` and `[[tensor.unitcell]]`



## Bond Hamiltonian

In the input file of the standard mode, say `std.toml`, each local Hamiltonian is specified as `[[hamiltonian]]`. The bonds where the bond Hamiltonian acts are specified by `bonds` string:

```
[[hamiltonian]]
bonds = "" # Set of acting bonds (1 bond per line)
0 1 0      # Row 1: Number of the source in the unit cell
1 1 0      # Row 2: x coordinate(displacement) of target from the source
2 1 0      # Row 3: y coordinate(displacement) of target from the source
3 1 0
0 0 1
1 0 1
2 0 1
3 0 1
""
```

One line of three integers corresponds one bond. The first integer is the index of the source site. The other two integers are x and y displacement of the target site from the source site. For example, `0 1 0` means the pair of the site 0 and the right neighbor ( $x+=1$  and  $y+=0$ ), the site 1, and `1 0 1` means the pair of the site 1 and the top neighbor ( $x+=0$  and  $y+=1$ ), the site 3.

The dimension of the bond Hamiltonian, i.e., the number of states of the source and target sites, is specified by `dim`, and the non-zero elements of the bond Hamiltonian is defined by `elements`:

```
dim = [2, 2]      # Number of possible states of the acting bond [source, target]
elements = ""      # (nonzero) matrix elements of the Hamiltonian (one element per row)
0 0 0 0 0.25 0.0  # Field 1: State of source before action
1 0 1 0 -0.25 0.0 # Field 2: State of target before action
0 1 1 0 0.5 0.0   # Field 3: State of source after action
1 0 0 1 0.5 0.0   # Field 4: State of target after action
0 1 0 1 -0.25 0.0 # Field 5: Real part of element
1 1 1 1 0.25 0.0  # Field 6: Imaginary part of element
""
```

One line of `elements` corresponds one element. The first two integers are the states of the source and target sites **before** the Hamiltonian acts on, and the following two integers are the states of the source and target sites **after** the Hamiltonian acts on. The remaining two numbers are the real and imaginary part of the element of the bond Hamiltonian.

## Site Hamiltonian

A site Hamiltonian is defined as a pair of (non-zero) matrix element and site it acts on.:

```
[[hamiltonian]]
dim = [2]
sites = []
elements = ""
1 0 -0.5 0.0
0 1 -0.5 0.0
""
```

Site is specified by `sites` as a list of indices. An empty list means all the sites.

Non-zero matrix elements are specified by `elements`, and how to define is the similar for bond Hamiltonians (Note that site Hamiltonians acts on only one site).

### 4.6.3 Definition of operators

`[[ observable . onesite ]]`

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

Specify only non-zero elements

```
elements = ""
0 0  0.5 0.0 ← ⟨0|Sz|0⟩ = 0.5
1 1 -0.5 0.0 ← ⟨1|Sz|1⟩ = -0.5
      Real Imaginary
""
```

Fig. 4.10: `[[observable.onesite]]`

Operators whose expected values are finally computed are defined in `[[observable]]`. The current version of TeNeS can evaluate onesite and twosites operators. The way to define operators is similar in Hamiltoninans, but an operator require the name (name) and identifier number (group). Although the energy operator is just the Hamiltonian (sum of the bond Hamiltonians), it should be defined in `[[observable]]` if users want to calculate. For convenience, `tenes_std` copies `[[hamiltonian]]` in `[[observable]]` when operator with `group = 0` is not defined.

For an example of onesite operator, the z-component of the spin operator

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

is defined as follows:

```
[[observable.onesite]] # onesite operator
name = "Sz"           # Name
group = 0              # 1-site operator identification number
sites = []             # Indices of tensors on which the operator acts ([] means all)
dim = 2                # Dimensions of operators
elements = ""         # Non-zero elements of operator matrix (one element per line)
0 0 0.5 0.0           # Fields 1 and 2: before and after action
1 1 -0.5 0.0          # Fields 3 and 4: Real and imaginary parts of the element
""
```

Non-zero elements of the matrix can be specified in the similar way of bond Hamiltonians.

Twosites operators can be defined in the similar way how to define the bond Hamiltonian. As an example of twosites

operator, spin-spin correlation on nearest neighbor bonds  $S_i^z S_j^z$  is defined as follows:

```
[[observable].twosite]] # twosite operator
name = "SzSz"          # Name
group = 1              # Index of twosite operator (independent of indices of on-site)
dim = [2, 2]           # Dimension
bonds = ""             # Bond that acts on
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] # When it can be written as a direct product of on-site operators,
              # their indices.
              # In this case, "Sz" is the on-site operator with index 0.
              # Matrix elements can also be written explicitly as elements
```

When the twosite operator is written as the direct product of the two on-site operators, `ops` can be used to specify them.

#### 4.6.4 Example: Antiferromagnetic Heisenberg model in staggered field

Let us consider the antiferromagnetic Heisenberg model in staggered field. The Hamiltonian is as follows

$$\mathcal{H} = J \sum_{\langle ij \rangle} S_i \cdot S_j - h \sum_{i \in A} S_i^z + h \sum_{j \in B} S_j^z,$$

where  $\sum_{\langle ij \rangle}$  is summation over the nearest neighbor bonds and  $A$  and  $B$  are the sublattices of the square lattice. The bond Hamiltonian  $\mathcal{H}_{ij}$  can be written as follows

$$\begin{aligned} \mathcal{H}_{ij} &= JS_i \cdot S_j \\ &= \begin{pmatrix} J/4 & 0 & 0 & 0 \\ 0 & -J/4 & J/2 & 0 \\ 0 & J/2 & -J/4 & 0 \\ 0 & 0 & 0 & J/4 \end{pmatrix} \end{aligned}$$

and the site Hamiltonian  $\mathcal{H}_i$  can be written as

$$\begin{aligned} \mathcal{H}_i &= -hS_i^z \\ &= \begin{pmatrix} -h/2 & 0 \\ 0 & h/2 \end{pmatrix} \end{aligned}$$

When  $J = 0$ ,  $h = 1$ , for example, an input file of `tenes_std`, `std.toml` (`sample/06_std_model/std.toml`), is as follows

```
[parameter]
[parameter.general]
is_real = true
tensor_save = "tensor"
[parameter.simple_update]
```

(continues on next page)



(continued from previous page)

```

num_step = 1000
tau = 0.01
[parameter.full_update]
num_step = 0
tau = 0.01
[parameter.ctm]
iteration_max = 10
dimension = 10

[tensor]
type = "square lattice"
L_sub = [2, 2]
skew = 0

[[tensor.unitcell]]
virtual_dim = [2, 2, 2, 2]
index = [0, 3]
physical_dim = 2
initial_state = [1.0, 0.0]
noise = 0.01

[[tensor.unitcell]]
virtual_dim = [2, 2, 2, 2]
index = [1, 2]
physical_dim = 2
initial_state = [0.0, 1.0]
noise = 0.01

[[hamiltonian]]
dim = [2]
sites = [0, 3]
elements = ""
0 0 -0.5 0.0
1 1 0.5 0.0
""

[[hamiltonian]]
dim = [2]
sites = [1, 2]
elements = ""
0 0 0.5 0.0
1 1 -0.5 0.0
""

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

```

(continues on next page)

(continued from previous page)

```

1 1 -0.5 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 = [1, 1]

```

We can calculate this model and obtain results as

```

$ tenes_std std.toml
$ tenes input.toml

... skipped ...

Onesite observables per site:
  hamiltonian = -0.5 0
  Sz          = 0 0
Twosite observables per site:
  SzSz        = -0.5 0

... skipped

```

Especially, the expectation values of onesite operators written in `output/onesite_obs.dat` are:

```

# $1: op_group
# $2: site_index
# $3: real
# $4: imag

0 0 -5.0000000000000000e-01 0.0000000000000000e+00
0 1 -5.0000000000000000e-01 0.0000000000000000e+00
0 2 -5.0000000000000000e-01 0.0000000000000000e+00
0 3 -5.0000000000000000e-01 0.0000000000000000e+00
1 0 5.0000000000000000e-01 0.0000000000000000e+00
1 1 -5.0000000000000000e-01 0.0000000000000000e+00
1 2 -5.0000000000000000e-01 0.0000000000000000e+00
1 3 5.0000000000000000e-01 0.0000000000000000e+00
-1 0 2.20256797875764860e+04 0.0000000000000000e+00
-1 1 2.20198975366861232e+04 0.0000000000000000e+00
-1 2 2.20294461413457539e+04 0.0000000000000000e+00
-1 3 2.20236290136460302e+04 0.0000000000000000e+00

```

Values of  $S^z$  (`op_group=`) show that spins on the A sublattice (`site_index=0,3`) are up (`0.5`) and those on the B (`site_index=1,2`) are down (`-0.5`). By imposing the staggered magnetic field ( $J = 0, h = 1$ ), a tensor product state representing the Neel state is obtained. Tensors are saved into the `tensor` directory because we set `tensor_save = "tensor"`, and therefore we can use them as the initial states of another calculation by setting `tensor_load = "tensor"`.



## FILE FORMAT

### 5.1 Short summary for input files of TeNeS

Input files of TeNeS are 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. Additionally, this copies `parameter`, `correlation`, and `correlation_length` sections.

The following table summarizes how each tool deal with sections.

Section	<code>tenes_simple</code>	<code>tenes_std</code>	<code>tenes</code>
<code>parameter</code>	copy	in / copy	in
<code>model</code>	in		
<code>lattice</code>	in		
<code>tensor</code>	out	in / copy	in
<code>observable</code>	out	copy	in
<code>correlation</code>	copy	copy	in
<code>correlation_length</code>	copy	copy	in
<code>hamiltonian</code>	out	in	
<code>evolution</code>		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

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

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

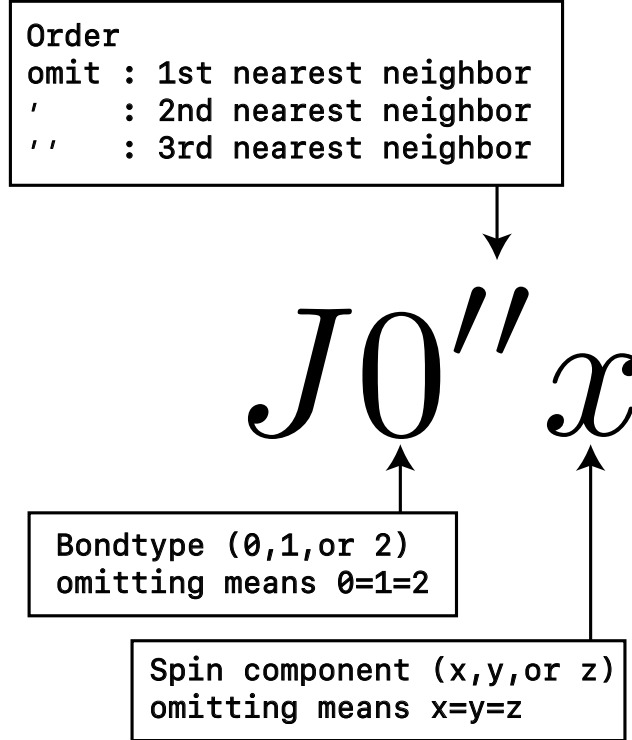
The exchange interaction  $J$  can have a bond dependency.

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

For the next nearest and third nearest neighbor bond, please surround the keyname with the double-quotation marks, ". 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 <b>0th</b> direction <b>nearest neighbor</b> bond	Real	0.0
B1	Biquadratic interaction of <b>1st</b> direction <b>nearest neighbor</b> bond	Real	0.0
B2	Biquadratic interaction of <b>2nd</b> direction <b>nearest neighbor</b> bond	Real	0.0
B0'	Biquadratic interaction of <b>0th</b> direction <b>next nearest neighbor</b> bond	Real	0.0
B1'	Biquadratic interaction of <b>1st</b> direction <b>next nearest neighbor</b> bond	Real	0.0
B2'	Biquadratic interaction of <b>2nd</b> direction <b>next nearest neighbor</b> bond	Real	0.0
B0''	Biquadratic interaction of <b>0th</b> direction <b>third nearest neighbor</b> bond	Real	0.0
B1''	Biquadratic interaction of <b>1st</b> direction <b>third nearest neighbor</b> bond	Real	0.0
B2''	Biquadratic interaction of <b>2nd</b> direction <b>third nearest neighbor</b> 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

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

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 ( $h^{\alpha}$  and  $D$  term) appear only in the nearest neighbor bonds, and  $z$  is the number of the coordinate 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 <b>0th</b> direction <b>nearest neighbor</b> bond	Real	0.0
t1	Hopping of <b>1st</b> direction <b>nearest neighbor</b> bond	Real	0.0
t2	Hopping of <b>2nd</b> direction <b>nearest neighbor</b> bond	Real	0.0
t0'	Hopping of <b>0th</b> direction <b>next nearest neighbor</b> bond	Real	0.0
t1'	Hopping of <b>1st</b> direction <b>next nearest neighbor</b> bond	Real	0.0
t2'	Hopping of <b>2nd</b> direction <b>next nearest neighbor</b> bond	Real	0.0
t0''	Hopping of <b>0th</b> direction <b>third nearest neighbor</b> bond	Real	0.0
t1''	Hopping of <b>1st</b> direction <b>third nearest neighbor</b> bond	Real	0.0
t2''	Hopping of <b>2nd</b> direction <b>third nearest neighbor</b> bond	Real	0.0
V0	Offsite repulsion of <b>0th</b> direction <b>nearest neighbor</b> bond	Real	0.0
V1	Offsite repulsion of <b>1st</b> direction <b>nearest neighbor</b> bond	Real	0.0
V2	Offsite repulsion of <b>2nd</b> direction <b>nearest neighbor</b> bond	Real	0.0
V0'	Offsite repulsion of <b>0th</b> direction <b>next nearest neighbor</b> bond	Real	0.0
V1'	Offsite repulsion of <b>1st</b> direction <b>next nearest neighbor</b> bond	Real	0.0
V2'	Offsite repulsion of <b>2nd</b> direction <b>next nearest neighbor</b> bond	Real	0.0
V0''	Offsite repulsion of <b>0th</b> direction <b>third nearest neighbor</b> bond	Real	0.0
V1''	Offsite repulsion of <b>1st</b> direction <b>third nearest neighbor</b> bond	Real	0.0
V2''	Offsite repulsion of <b>2nd</b> direction <b>third nearest neighbor</b> 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  $\mu$  term) appear only in the nearest neighbor bonds, and  $z$  is the number of the coordinate number.



## 5.2.2 lattice section

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

Name	Description	Type	Default
<code>type</code>	lattice name (square, triangular or honeycomb lattice)	String	–
<code>L</code>	Unit cell size in x direction	Integer	–
<code>W</code>	Unit cell size in y direction	Integer	L
<code>virtual_dim</code>	Bond dimension	Integer	–
<code>initial</code>	Initial state	String	random
<code>noise</code>	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.

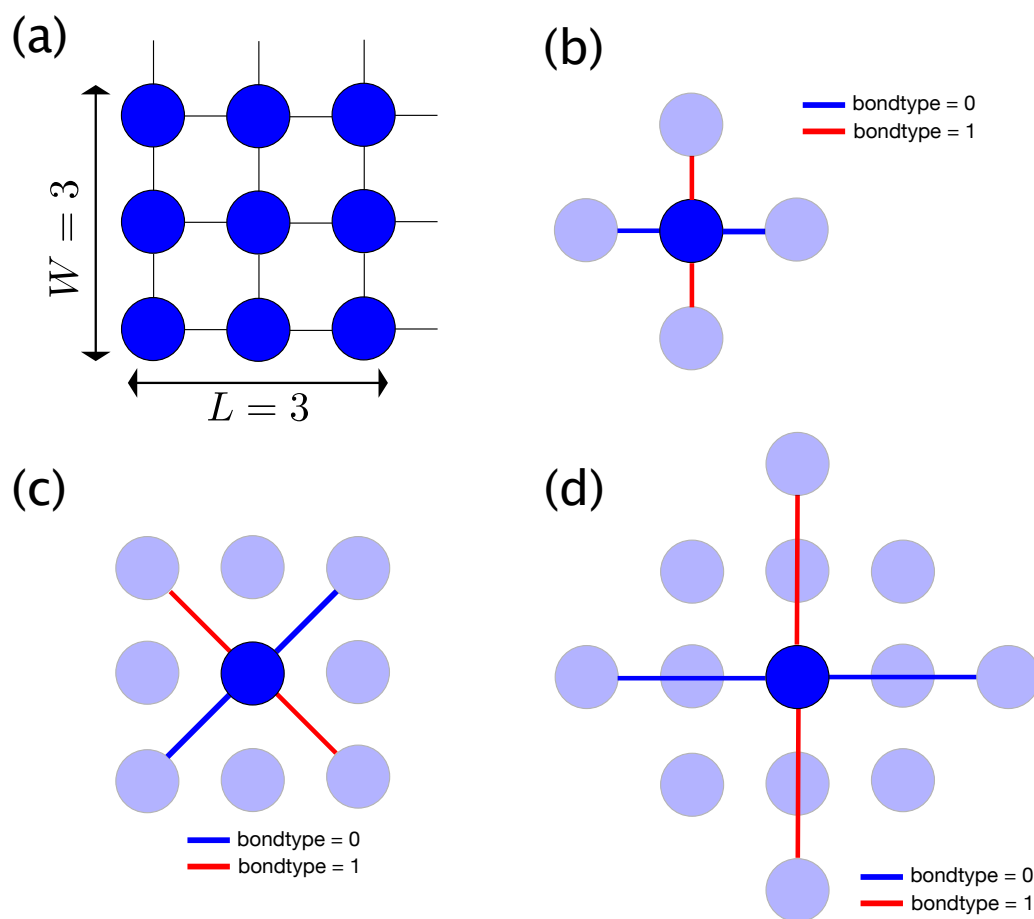


Fig. 5.1: Square lattice. (a) Site structure with  $L=3$ ,  $W=3$  (b) Nearest neighbor bonds. `bondtype=0` (blue) bond extends in the 0 degree direction and `bondtype=1` (red) one in the 90 degree direction. (c) Second nearest neighbor bonds. `bondtype=0` (blue) bond extends in the 45 degree direction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

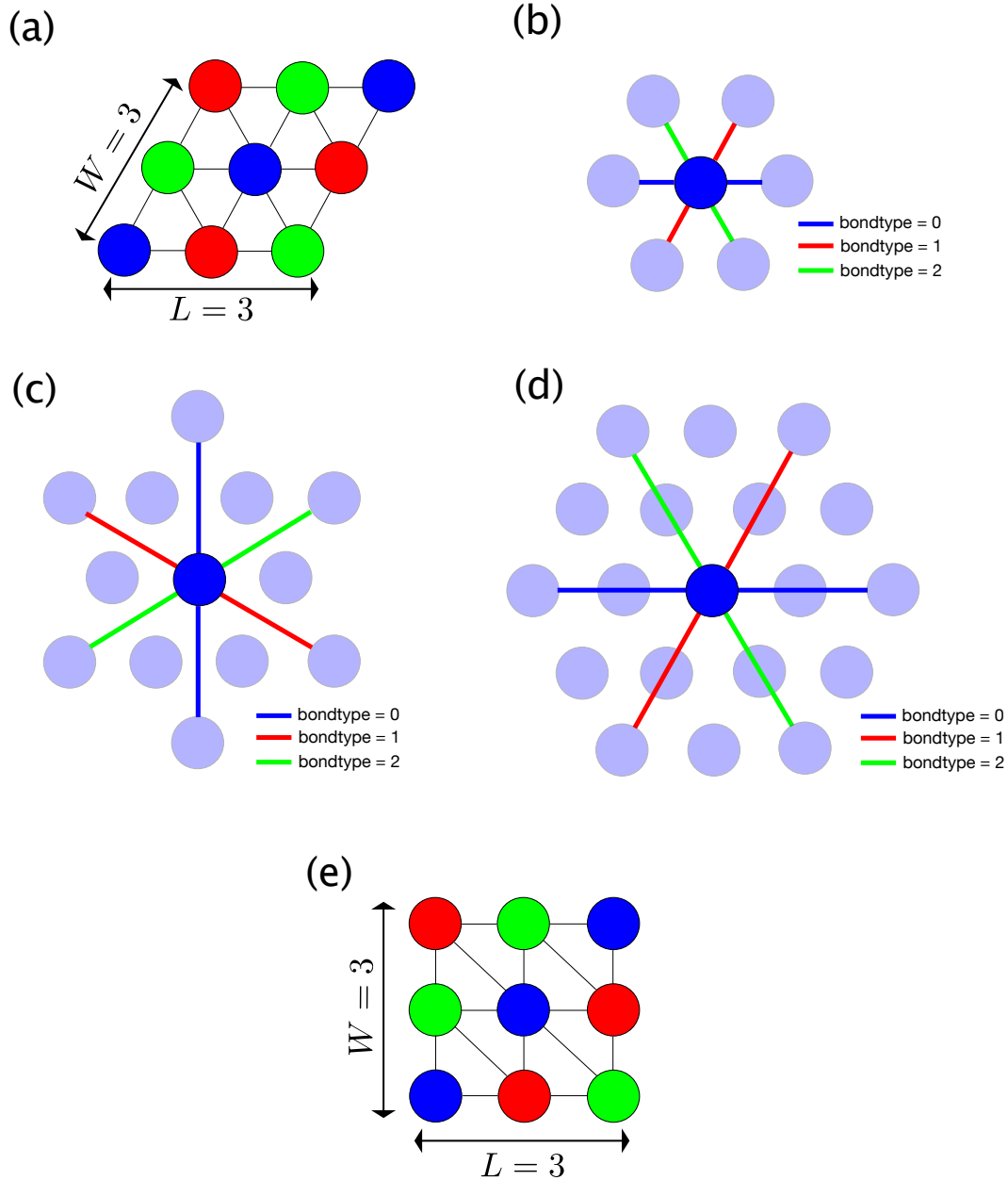


Fig. 5.2: Triangular lattice. (a) Site structure with  $L=3$ ,  $W=3$  (b) Nearest neighbor bonds. `bondtype=0` (blue) bond extends in the 0 degree direction, `bondtype=1` (red) one in the 60 degree direction, and `bondtype=2` (green) one in the 120 degree direction. (c) Second nearest neighbor bonds. `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) Corresponding 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`.

### `parameter.general`

General parameters for `tenes`.

Name	Description	Type	Default
<code>mode</code>	Calculation mode	String	<code>\ "ground state\ "</code>
<code>is_real</code>	Whether to limit all tensors to real valued ones	Boolean	<code>false</code>
<code>iszero_tol</code>	Absolute cutoff value for reading operators	Real	<code>0.0</code>
<code>measure</code>	Whether to calculate and save observables	Boolean	<code>true</code>
<code>measure_interval</code>	Interval of measurement in finite temperature calculation and time evolution process	Integer or list of integers	<code>10</code>
<code>output</code>	Directory for saving result such as physical quantities	String	<code>"output"</code>
<code>tensor_save</code>	Directory for saving optimized tensors	String	<code>""</code>
<code>tensor_load</code>	Directory for loading initial tensors	String	<code>""</code>

- `mode`
  - Specify the calculation mode
  - `"ground state"`
    - \* Search for the ground state of the Hamiltonian
    - \* `tenes_std` calculates the imaginary time evolution operator  $U(\tau) = e^{-\tau H}$  from the Hamiltonian  $H$
  - `"time evolution"`
    - \* Calculate the time evolution of the observables from the initial state
    - \* `tenes_std` calculates the time evolution operator  $U(t) = e^{-itH}$  from the Hamiltonian  $H$

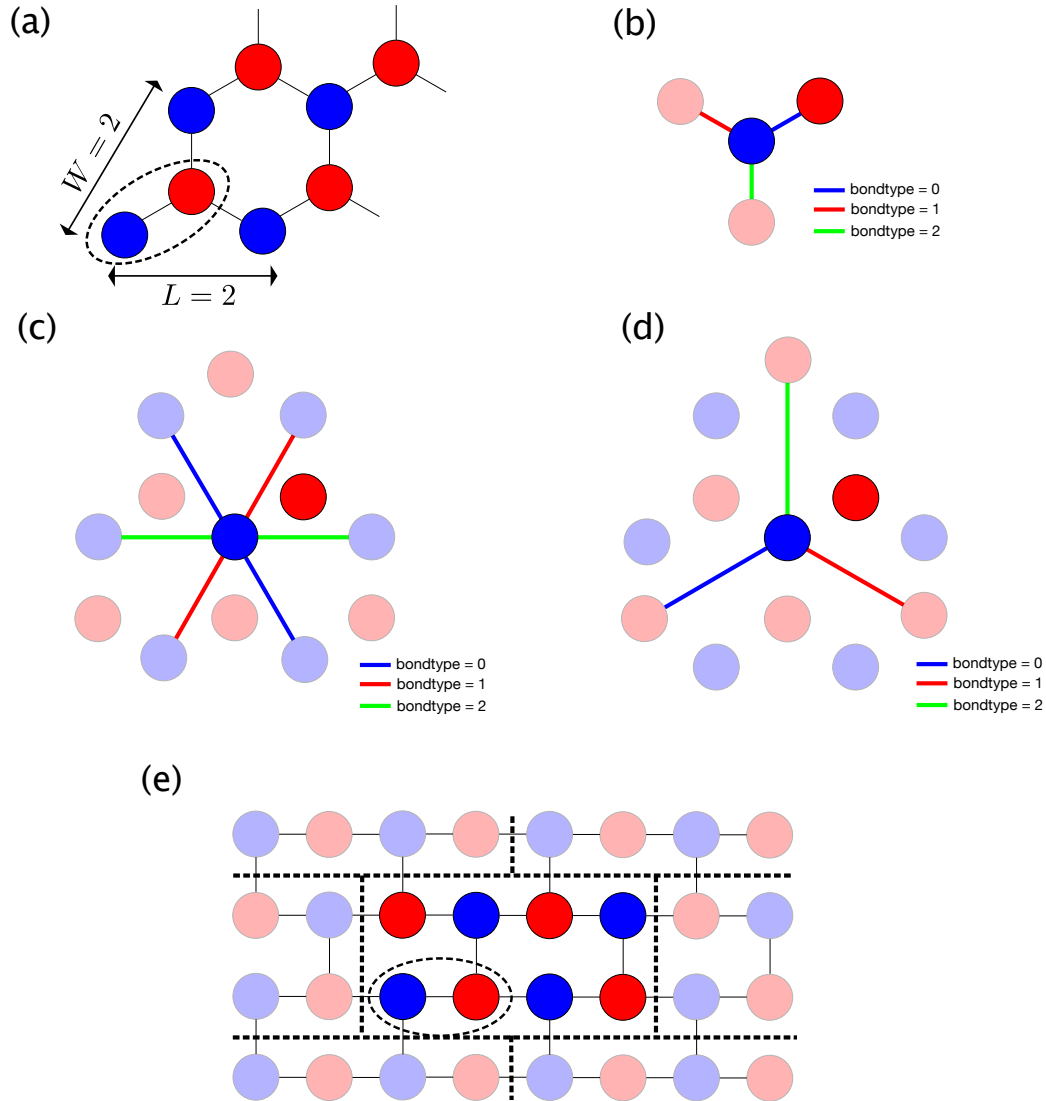


Fig. 5.3: Honeycomb lattice. (a) Site structure with  $L=2$ ,  $W=2$ . The dashed ellipse denotes one unit. (b) Nearest neighbor bonds. **bondtype=0** (blue) bond extends in the 30 degree direction, **bondtype=1** (red) one in the 150 degree direction, and **bondtype=2** (green) one in the -90 degree 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) Corresponding 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.

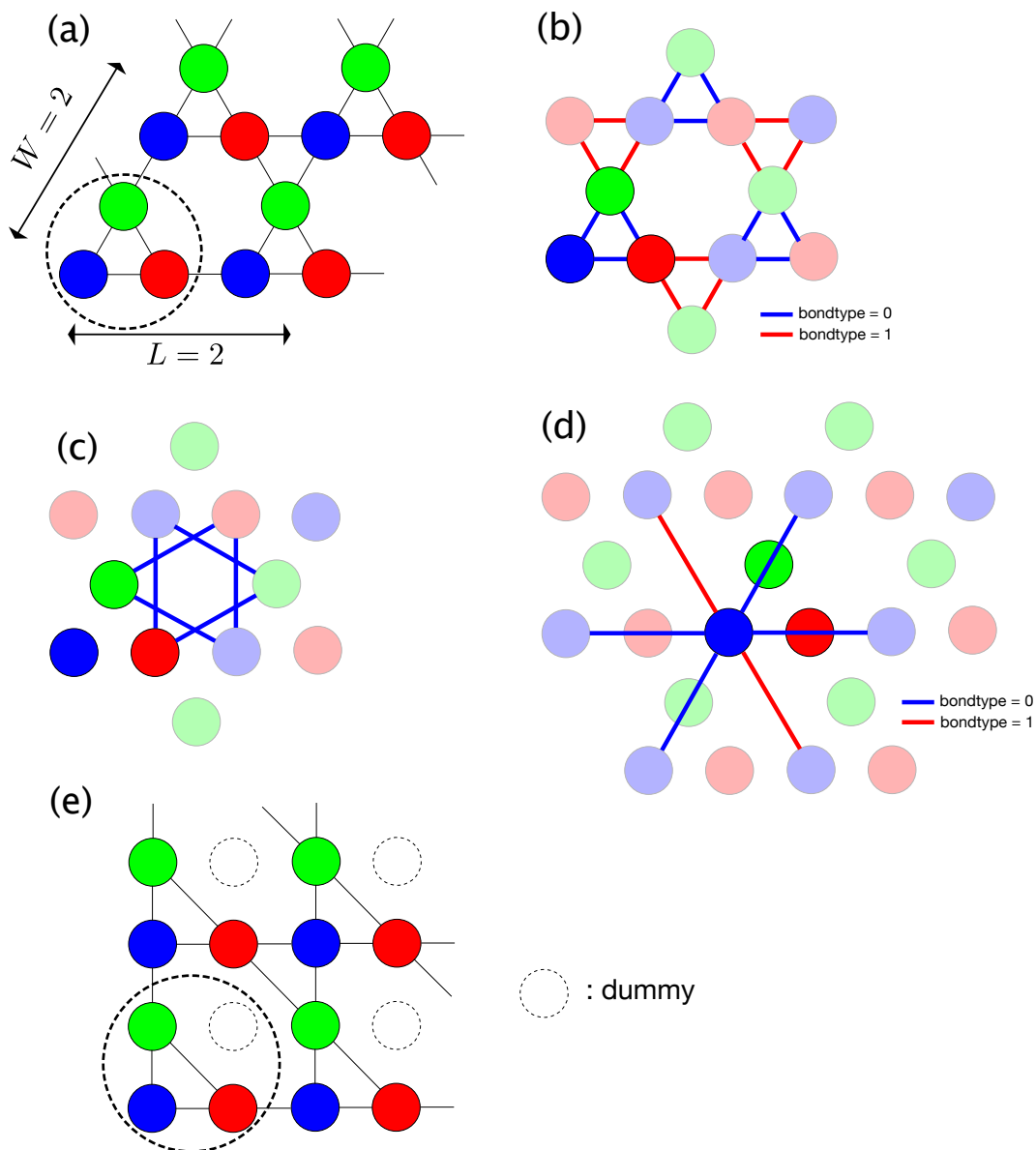


Fig. 5.4: Kagome lattice. (a) Site structure with  $L=2$ ,  $W=2$ . The dashed circle denotes one unit. (b) Nearest neighbor bonds.  $\text{bondtype}=0$  (blue) bonds form upper triangle and  $\text{bondtype}=1$  (red) bonds form lower triangle. (c) Second nearest neighbor bonds. (d) Third nearest neighbor bonds.  $\text{bondtype}=0$  (blue) bond passes over a site and  $\text{bondtype}=1$  (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.

- "finite temperature"
  - \* Calculate the finite temperature expectation values of the observables
  - \* `tenes_std` calculates the imaginary time evolution operator  $U(\tau) = e^{-\tau H}$  from the Hamiltonian  $H$
- `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
- `measure_interval`
  - Specify the interval of measurement in time evolution process and finite temperature Calculation
  - Physical quantities are calculated and saved each after `measure_interval` updates
- `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
<code>tau</code>	(Imaginary) time step $\tau$ in (imaginary) time evolution operator	Real or list of real	0.01
<code>num_step</code>	Number of simple updates	Integer or list of integers	0
<code>lambda_cutoff</code>	cutoff of the mean field to be considered zero in the simple update	Real	1e-12
<code>gauge_fix</code>	Whether the tensor gauge is fixed	Boolean	false
<code>gauge_maxiter</code>	Maximum number of iterations for fixing gauge	Integer	100
<code>gauge_converge_epsilon</code>	Convergence criteria of iterations for fixing gauge	Real	1e-2

- `tau`
  - Specify the (imaginary) time step  $\tau$  in (imaginary) time evolution operator
    - \* `tenes_std` uses it to calculate the imaginary time evolution operator  $e^{-\tau H}$  from the Hamiltonian
    - \* `tenes` uses it to calculate the time of each measurement
      - For finite temperature calculation, note that the inverse temperature increase  $2\tau$  at a step because  $\rho(\beta + 2\tau) = U(\tau)\rho(\beta)\bar{U}(\tau)$
  - When a list is specified, the time step can be changed for each group of time evolution operators
- `num_step`
  - Specify the number of simple updates
  - When a list is specified, the number of simple updates can be changed for each group of time evolution operators

### `parameter.full_update`

Parameters in the full update procedure.

Name	Description	Type	Default
<code>tau</code>	(Imaginary) time step $\tau$ in (imaginary) time evolution operator	Real or list of reals	0.01
<code>num_step</code>	Number of full updates	Integer or list of integers	0
<code>env_cutoff</code>	Cutoff of singular values to be considered as zero when computing environment through full updates	Real	1e-12
<code>inverse_precision</code>	Cutoff of singular values to be considered as zero when computing the pseudoinverse matrix with full update	Real	1e-12
<code>convergence_epsilon</code>	Convergence criteria for truncation optimization with full update	Real	1e-6
<code>iteration_max</code>	Maximum iteration number for truncation optimization on full updates	Integer	100
<code>gauge_fix</code>	Whether the tensor gauge is fixed	Boolean	true
<code>fastfullupdate</code>	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 $\chi$	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
meanfield_env	Use mean field environment obtained through simple update instead of CTM	Boolean	false

For Tensor renormalization 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\*](#).

### 5.2.5 correlation\_length section

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

This section describes how to calculate the correlation length  $\xi$ .

Name	Description	Type	Default
<code>measure</code>	Whether to calculate $\xi$ or not	Bool	true
<code>num_eigvals</code>	The number of eigenvalues of the transfer matrix to be calculated	Integer	4
<code>maxdim_dense_ei</code>	Maximum dimension of the transfer matrix where the diagonalization method for dense matrices is used	Integer	200
<code>arnoldi_maxdim</code>	Dimension of the Hessenberg matrix generated by the Arnoldi method	Integer	50
<code>arnoldi_restart</code>	The number of the initial vectors generated by the restart process of the IRA method	Integer	20
<code>arnoldi_maxiter</code>	Maximum number of iterations in the IRA method	Integer	1
<code>arnoldi_rtol</code>	Relative tolerance used in the Arnoldi method	Float	1e-10

The correlation length  $\xi$  will be calculated from the dominant eigenvalues of the transfer matrices. If the dimension of the transfer matrix is less than or equal to `maxdim_dense_eigensolver`, an eigensolver for dense matrices (LAPACK's `*geev` routines) will be used. If not, an iterative method, the implicit restart Arnoldi method (IRA method), will be used.

In the IRA method, a Hessenberg matrix with the size of `arnoldi_maxdim` is generated by the Arnoldi process. Its eigenvalues are approximants of the first `arnoldi_maxdim` eigenvalues of the original matrix. If not converged, the IRA method restarts the Arnoldi process with the newly generated `arnoldi_restartdim` initial vectors. In the many cases of the transfer matrices, such a process is not necessary (`arnoldi_maxiterations` = 1).

## 5.3 Input file for `tenes_std`

- File format: [TOML](#) format
- This file has 5 sections: `parameter`, `tensor`, `hamiltonian`, `observable`, `correlation`

### 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`.

#### `parameter.general`

General parameters for `tenes`.

Name	Description	Type	Default
<code>mode</code>	Calculation mode	String	<code>\ "ground state\ "</code>
<code>is_real</code>	Whether to limit all tensors to real valued ones	Boolean	<code>false</code>
<code>iszero_tol</code>	Absolute cutoff value for reading operators	Real	<code>0.0</code>
<code>measure</code>	Whether to calculate and save observables	Boolean	<code>true</code>
<code>measure_interval</code>	Interval of measurement in finite temperature calculation and time evolution process	Integer or list of integers	<code>10</code>
<code>output</code>	Directory for saving result such as physical quantities	String	<code>"output"</code>
<code>tensor_save</code>	Directory for saving optimized tensors	String	<code>""</code>
<code>tensor_load</code>	Directory for loading initial tensors	String	<code>""</code>

- `mode`
  - Specify the calculation mode
  - `"ground state"`
    - \* Search for the ground state of the Hamiltonian
    - \* `tenes_std` calculates the imaginary time evolution operator  $U(\tau) = e^{-\tau H}$  from the Hamiltonian  $H$
  - `"time evolution"`
    - \* Calculate the time evolution of the observables from the initial state
    - \* `tenes_std` calculates the time evolution operator  $U(t) = e^{-itH}$  from the Hamiltonian  $H$
  - `"finite temperature"`
    - \* Calculate the finite temperature expectation values of the observables
    - \* `tenes_std` calculates the imaginary time evolution operator  $U(\tau) = e^{-\tau H}$  from the Hamiltonian  $H$
- `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
- `measure_interval`

- Specify the interval of measurement in time evolution process and finite temperature Calculation
- Physical quantities are calculated and saved each after `measure_interval` updates
- `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
<code>tau</code>	(Imaginary) time step $\tau$ in (imaginary) time evolution operator	Real or list of real	0.01
<code>num_step</code>	Number of simple updates	Integer or list of integers	0
<code>lambda_cutoff</code>	cutoff of the mean field to be considered zero in the simple update	Real	1e-12
<code>gauge_fix</code>	Whether the tensor gauge is fixed	Boolean	false
<code>gauge_maxiter</code>	Maximum number of iterations for fixing gauge	Integer	100
<code>gauge_converge_epsilon</code>	Convergence criteria of iterations for fixing gauge	Real	1e-2

- `tau`
  - Specify the (imaginary) time step  $\tau$  in (imaginary) time evolution operator
    - \* `tenes_std` uses it to calculate the imaginary time evolution operator  $e^{-\tau H}$  from the Hamiltonian
    - \* `tenes` uses it to calculate the time of each measurement
      - For finite temperature calculation, note that the inverse temperature increase  $2\tau$  at a step because  $\rho(\beta + 2\tau) = U(\tau)\rho(\beta)\bar{U}(\tau)$
  - When a list is specified, the time step can be changed for each group of time evolution operators
- `num_step`
  - Specify the number of simple updates
  - When a list is specified, the number of simple updates can be changed for each group of time evolution operators

**parameter.full\_update**

Parameters in the full update procedure.

Name	Description	Type	Default
tau	(Imaginary) time step $\tau$ in (imaginary) time evolution operator	Real or list of reals	0.01
num_step	Number of full updates	Integer or list of integers	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 $\chi$	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
meanfield_env	Use mean field environment obtained through simple update instead of CTM	Boolean	false

For Tensor renormalization 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  $L_x$  times  $L_y$ . `lattice` section has an array of subsections `unitcell`.

Name	Description	Type	Default
<code>L_sub</code>	Unit cell size	Integer or a list of integer	–
<code>skew</code>	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  $L_x$  and the second one does  $L_y$ . A list of three or more elements causes an error. If `L_sub` is an integer, both  $L_x$  and  $L_y$  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.

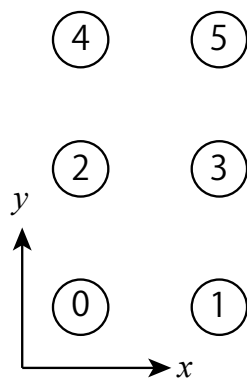


Fig. 5.5: An example for  $L_{\text{sub}} = [2, 3]$ .

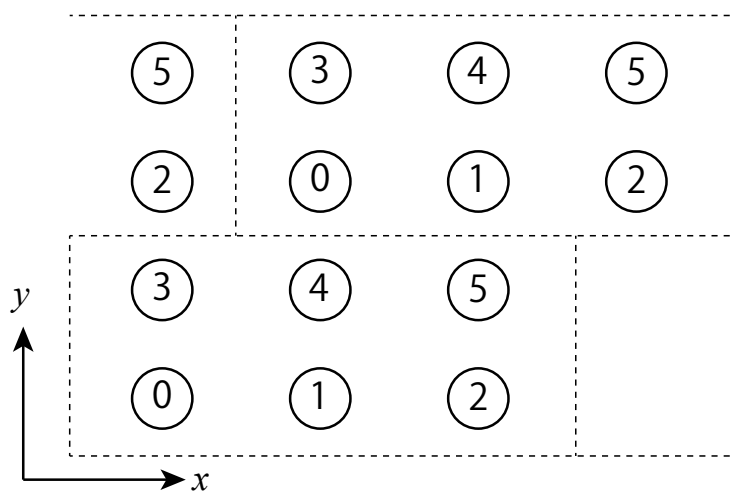


Fig. 5.6: An example for  $L_{\text{sub}} = [3, 2]$ ,  $\text{skew} = 1$  (ruled line is a separator for unit cell).

**tensor.unitcell subsection**

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

Name	Description	Type
<code>index</code>	Site number	Integer or a list of integer
<code>physical_dim</code>	Dimension of physical bond for a site tensor	Integer
<code>virtual_dim</code>	Dimension of virtual bonds $D$ for a site tensor	Integer or a list of integer
<code>initial_state</code>	Initial tensor	a list of real
<code>noise</code>	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  $A_{\alpha}$ , 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
<code>name</code>	Operator name	String
<code>group</code>	Identification number of operators	Integer
<code>sites</code>	Site number	Integer or a list of integer
<code>dim</code>	Dimension of an operator	Integer
<code>elements</code>	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 operator 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 on-site 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.onsite`. For example, if  $S^z$  is defined as `group = 0` in `observable.onsite`,  $S_i^z S_j^z$  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)

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

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 indices 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.

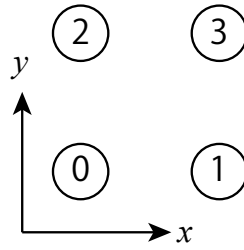


Fig. 5.7: Site indices of the  $S=1/2$  Heisenberg model on square lattice at  $L_{\text{sub}}=[2, 2]$ .

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
""
```

### observable.multisite

Define multi-body operators that indicate physical quantities defined on three or more sites. It is defined as a direct product of one-body operators defined in `observable.onsite`.

Name	Description	Type
name	Operator name	String
group	Identification number of operators	Integer
multisites	Sites	String
ops	Index of onsite operators	List of integers

`name` specifies an operator name.

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

`multisites` specifies a string representing the set of sets of sites on which the operator acts. One line consisting of integers means a set sites.

- The first integer is the number of the source site.
- The following integers are the coordinates (dx, dy) of the other sites from the source site.
  - `source_site dx2 dy2 dx3 dy3 ... dxN dyN` for N-site operator.
  - All sites must be within a square of size  $4 \times 4$ .

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

### 5.3.4 hamiltonian section

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

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

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

Name	Description	Type
dim	Dimension of an operator	A list of integers
sites	Site	A list of integers
bonds	Bond	String
elements	Non-zero elements of an operator	String

`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]`. `tenes_std` judges whether a local Hamiltonian is site one or bond one from the number of integers in `dims`; if one, a site Hamiltonian is defined and otherwise a bond one.

`sites`, a list of integers, specifies a set of sites where the site operator acts. An empty list (`[]`) means all the sites.

`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.

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

- For site Hamiltonian
  - The first integer is the index of the state of the site **before** the operator acts on.
  - The next one shows the index of the state of the site **after** the operator acts on.
  - The last two indicate the real and imaginary parts of the elements of the operator.
- For bond Hamiltonian
  - The first two integers are the indices of the states of the source site and target site **before** the operator acts on.
  - The next two show the indices of the states 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  $C = \langle A(\mathbf{r}_0)B(\mathbf{r}_0 + \mathbf{r}) \rangle$  is specified. If you omit this section, no correlation functions will be calculated.

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

$$\mathbf{r} = (0, 0), (1, 0), (2, 0), \dots, (r_{\max}, 0), (0, 1), (0, 2), \dots, (0, r_{\max})$$

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

Name	Description	Type
<code>r_max</code>	Maximum distance $r$ of the correlation function	Integer
<code>operators</code>	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]]
```

### 5.3.6 correlation\_length section

This section describes how to calculate the correlation length  $\xi$ .

Name	Description	Type	Default
<code>measure</code>	Whether to calculate $\xi$ or not	Bool	true
<code>num_eigvals</code>	The number of eigenvalues of the transfer matrix to be calculated	Integer	4
<code>maxdim_dense_ei</code>	Maximum dimension of the transfer matrix where the diagonalization method for dense matrices is used	Integer	200
<code>arnoldi_maxdim</code>	Dimension of the Hessenberg matrix generated by the Arnoldi method	Integer	50
<code>arnoldi_restart</code>	The number of the initial vectors generated by the restart process of the IRA method	Integer	20
<code>arnoldi_maxiter</code>	Maximum number of iterations in the IRA method	Integer	1
<code>arnoldi_rtol</code>	Relative tolerance used in the Arnoldi method	Float	1e-10

The correlation length  $\xi$  will be calculated from the dominant eigenvalues of the transfer matrices. If the dimension of the transfer matrix is less than or equal to `maxdim_dense_eigensolver`, an eigensolver for dense matrices (LAPACK's `*geev` routines) will be used. If not, an iterative method, the implicit restart Arnoldi method (IRA method), will be used.

In the IRA method, a Hessenberg matrix with the size of `arnoldi_maxdim` is generated by the Arnoldi process. Its eigenvalues are approximants of the first `arnoldi_maxdim` eigenvalues of the original matrix. If not converged, the IRA method restarts the Arnoldi process with the newly generated `arnoldi_restartdim` initial vectors. In the many cases of the transfer matrices, such a process is not necessary (`arnoldi_maxiterations` = 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`.

**parameter.general**

General parameters for `tenes`.

Name	Description	Type	Default
<code>mode</code>	Calculation mode	String	<code>\ "ground state\ "</code>
<code>is_real</code>	Whether to limit all tensors to real valued ones	Boolean	<code>false</code>
<code>iszero_tol</code>	Absolute cutoff value for reading operators	Real	<code>0.0</code>
<code>measure</code>	Whether to calculate and save observables	Boolean	<code>true</code>
<code>measure_interval</code>	Interval of measurement in finite temperature calculation and time evolution process	Integer or list of integers	<code>10</code>
<code>output</code>	Directory for saving result such as physical quantities	String	<code>"output"</code>
<code>tensor_save</code>	Directory for saving optimized tensors	String	<code>""</code>
<code>tensor_load</code>	Directory for loading initial tensors	String	<code>""</code>

- `mode`
  - Specify the calculation mode
  - `"ground state"`
    - \* Search for the ground state of the Hamiltonian
    - \* `tenes_std` calculates the imaginary time evolution operator  $U(\tau) = e^{-\tau H}$  from the Hamiltonian  $H$
  - `"time evolution"`
    - \* Calculate the time evolution of the observables from the initial state
    - \* `tenes_std` calculates the time evolution operator  $U(t) = e^{-itH}$  from the Hamiltonian  $H$
  - `"finite temperature"`
    - \* Calculate the finite temperature expectation values of the observables
    - \* `tenes_std` calculates the imaginary time evolution operator  $U(\tau) = e^{-\tau H}$  from the Hamiltonian  $H$
- `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
- `measure`
  - When set to `false`, the stages for measuring and saving observables will be skipped
  - Elapsed time `time.dat` is always saved
- `measure_interval`
  - Specify the interval of measurement in time evolution process and finite temperature Calculation
  - Physical quantities are calculated and saved each after `measure_interval` updates
- `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
<code>tau</code>	(Imaginary) time step $\tau$ in (imaginary) time evolution operator	Real or list of real	0.01
<code>num_step</code>	Number of simple updates	Integer or list of integers	0
<code>lambda_cutoff</code>	cutoff of the mean field to be considered zero in the simple update	Real	1e-12
<code>gauge_fix</code>	Whether the tensor gauge is fixed	Boolean	false
<code>gauge_maxiter</code>	Maximum number of iterations for fixing gauge	Integer	100
<code>gauge_converge_epsilon</code>	Convergence criteria of iterations for fixing gauge	Real	1e-2

- `tau`
  - Specify the (imaginary) time step  $\tau$  in (imaginary) time evolution operator
    - \* `tenes_std` uses it to calculate the imaginary time evolution operator  $e^{-\tau H}$  from the Hamiltonian
    - \* `tenes` uses it to calculate the time of each measurement
      - For finite temperature calculation, note that the inverse temperature increase  $2\tau$  at a step because  $\rho(\beta + 2\tau) = U(\tau)\rho(\beta)\bar{U}(\tau)$
  - When a list is specified, the time step can be changed for each group of time evolution operators
- `num_step`
  - Specify the number of simple updates
  - When a list is specified, the number of simple updates can be changed for each group of time evolution operators



**parameter.full\_update**

Parameters in the full update procedure.

Name	Description	Type	Default
tau	(Imaginary) time step $\tau$ in (imaginary) time evolution operator	Real or list of reals	0.01
num_step	Number of full updates	Integer or list of integers	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 $\chi$	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
meanfield_env	Use mean field environment obtained through simple update instead of CTM	Boolean	false

For Tensor renormalization 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  $L_x$  times  $L_y$ . `lattice` section has an array of subsections `unitcell`.

Name	Description	Type	Default
<code>L_sub</code>	Unit cell size	Integer or a list of integer	–
<code>skew</code>	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  $L_x$  and the second one does  $L_y$ . A list of three or more elements causes an error. If `L_sub` is an integer, both  $L_x$  and  $L_y$  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.

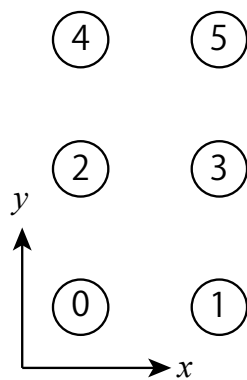


Fig. 5.8: An example for  $L_{\text{sub}} = [2, 3]$ .

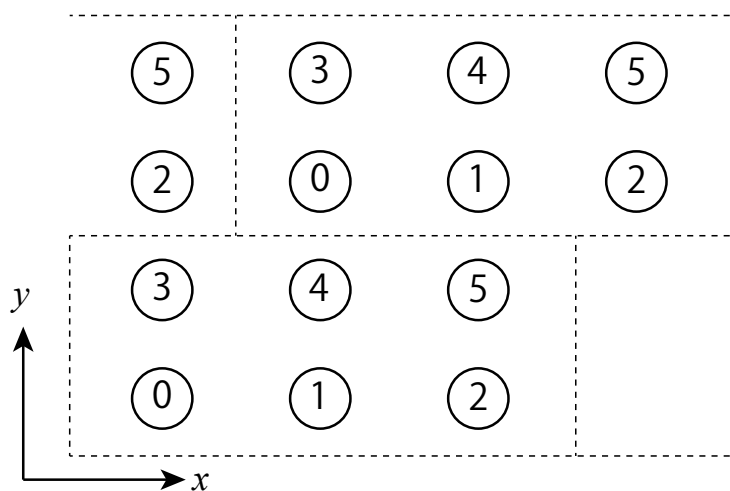


Fig. 5.9: An example for  $L_{\text{sub}} = [3, 2]$ ,  $\text{skew} = 1$  (ruled line is a separator for unit cell).

**tensor.unitcell subsection**

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

Name	Description	Type
<code>index</code>	Site number	Integer or a list of integer
<code>physical_dim</code>	Dimension of physical bond for a site tensor	Integer
<code>virtual_dim</code>	Dimension of virtual bonds $D$ for a site tensor	Integer or a list of integer
<code>initial_state</code>	Initial tensor	a list of real
<code>noise</code>	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  $A_{\alpha}$ , 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, `onsite` and `twosite`.

**observable.onsite**

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

Name	Description	type
<code>name</code>	Operator name	String
<code>group</code>	Identification number of operators	Integer
<code>sites</code>	Site number	Integer or a list of integer
<code>dim</code>	Dimension of an operator	Integer
<code>elements</code>	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 operator 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 on-site 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.onsite`. For example, if  $S^z$  is defined as `group = 0` in `observable.onsite`,  $S_i^z S_j^z$  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)

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

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 indices 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.

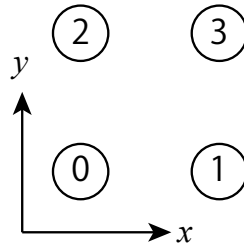


Fig. 5.10: Site indices of the  $S=1/2$  Heisenberg model on square lattice at  $L_{\text{sub}}=[2, 2]$ .

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
""
```

### observable.multisite

Define multi-body operators that indicate physical quantities defined on three or more sites. It is defined as a direct product of one-body operators defined in `observable.onsite`.

Name	Description	Type
name	Operator name	String
group	Identification number of operators	Integer
multisites	Sites	String
ops	Index of onsite operators	List of integers

name specifies an operator name.

group specifies an identification number of two sites operators.

multisites specifies a string representing the set of sets of sites on which the operator acts. One line consisting of integers means a set sites.

- The first integer is the number of the source site.
- The following integers are the coordinates (dx, dy) of the other sites from the source site.
  - source\_site dx2 dy2 dx3 dy3 ... dxN dyN for N-site operator.
  - All sites must be within a square of size  $4 \times 4$ .

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

## 5.4.4 evolution section

Specify the imaginary time evolution operators used in simple and full updates. One-site and two-sites (nearest neighbor bond) operators can be defined. This section has two subsections: `simple` and `full`.

Name	Description	Type
group	Group of the evolution operator	Integer (0-)
site	Index of site	Integer (0-)
source_site	Index of source site	Integer (0-)
source_leg	Direction from source site to target site	Integer (0-3)
dimensions	Dimension of a tensor of imaginary time evolution operator	A list of integers
elements	Non-zero elements of a tensor of imaginary time evolution operator	String

group specifies the group of the evolution operator (the default value is 0). It corresponds to the index of `tau` and `num_steps` in `parameter.simple_update` and `parameter.full_update`.

site is available for one-site operator, and `source_site` and `source_leg` are for two-site operator.

`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]

# One site
[[evolution.simple]]
site = 0
dimensions = [2, 2]
elements = ""
0 0  1.0012507815756226  0.0
1 1  0.9987507809245809  0.0
""

# Two site
[[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  $C = \langle A(\mathbf{r}_0)B(\mathbf{r}_0 + \mathbf{r}) \rangle$  is specified. If you omit this section, no correlation functions will be calculated.

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

$$\mathbf{r} = (0, 0), (1, 0), (2, 0), \dots, (r_{\max}, 0), (0, 1), (0, 2), \dots, (0, r_{\max})$$

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

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]]
```

### 5.4.6 correlation\_length section

This section describes how to calculate the correlation length  $\xi$ .

Name	Description	Type	Default
measure	Whether to calculate $\xi$ or not	Bool	true
num_eigvals	The number of eigenvalues of the transfer matrix to be calculated	Integer	4
maxdim_dense_ei	Maximum dimension of the transfer matrix where the diagonalization method for dense matrices is used	Integer	200
arnoldi_maxdim	Dimension of the Hessenberg matrix generated by the Arnoldi method	Integer	50
arnoldi_restart	The number of the initial vectors generated by the restart process of the IRA method	Integer	20
arnoldi_maxiter	Maximum number of iterations in the IRA method	Integer	1
arnoldi_rtol	Relative tolerance used in the Arnoldi method	Float	1e-10

The correlation length  $\xi$  will be calculated from the dominant eigenvalues of the transfer matrices. If the dimension of the transfer matrix is less than or equal to `maxdim_dense_eigensolver`, an eigensolver for dense matrices (LAPACK's `*geev` routines) will be used. If not, an iterative method, the implicit restart Arnoldi method (IRA method), will be used.

In the IRA method, a Hessenberg matrix with the size of `arnoldi_maxdim` is generated by the Arnoldi process. Its eigenvalues are approximants of the first `arnoldi_maxdim` eigenvalues of the original matrix. If not converged, the IRA method restarts the Arnoldi process with the newly generated `arnoldi_restartdim` initial vectors. In the many cases of the transfer matrices, such a process is not necessary (`arnoldi_maxiterations` = 1).

## 5.5 Output files

Output files are generated in the output directory.

### 5.5.1 For all modes

#### parameters.dat

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

Example:

```
simple_num_step = [10]
simple_tau = [0.01]
simple_inverse_lambda_cutoff = 1e-12
simple_gauge_fix = 0
simple_gauge_maxiter = 100
simple_gauge_convergence_epsilon = 0.01

full_num_step = [0]
full_inverse_projector_cutoff = 1e-12
full_inverse_precision = 1e-12
full_convergence_epsilon = 1e-06
full_iteration_max = 100
full_gauge_fix = true
full_fastfullupdate = true

ctm_dimension = 10
ctm_inverse_projector_cutoff = 1e-12
ctm_convergence_epsilon = 1e-06
ctm_iteration_max = 10
ctm_projector_corner = true
use_rsvd = false
rsvd_oversampling_factor = 2
meanfield_env = true

mode = ground state
simple
Lcor = 0
seed = 11
is_real = 0
iszero_tol = 0
measure = 1
tensor_load_dir =
tensor_save_dir = save_tensor
outdir = output

Lsub = [ 2 , 2 ]
skew = 0

start_datetime = 2023-06-08T16:41:50+09:00
```

**time.dat**

The calculation time is outputted.

Example:

```
time simple update = 1.64429
time full update   = 0
time environment   = 0.741858
time observable     = 0.104487
```

**5.5.2 For ground state calculation mode****density.dat**

The expectation value per site of each observable is outputted. When the name of the operator (name) is an empty, the index of the operator is written. Energy means the summation of site hamiltonian and bond hamiltonian.

Example:

```
Energy           = -5.00499902760266346e-01  0.000000000000000000e+00
site hamiltonian = -4.99999945662006270e-04  0.000000000000000000e+00
Sz               =  4.99999945662006284e-01  0.000000000000000000e+00
Sx               =  9.24214061616647275e-05  0.000000000000000000e+00
Sy               = -2.34065881671767322e-06  0.000000000000000000e+00
bond hamiltonian = -4.99999902814604325e-01  2.22346094146706503e-21
SzSz             =  4.99999902814604380e-01 -1.80051315353166456e-21
SxSx             =  1.12631053560300631e-05  6.08792260271591701e-21
SySy             = -1.12817627661272438e-05  4.76468712680822333e-21
```

**onsite\_obs.dat**

- The expected values of the site operator  $\langle \hat{A}_i^\alpha \rangle = \langle \Psi | \hat{A}_i^\alpha | \Psi \rangle / \langle \Psi | \Psi \rangle$  are outputted.
- Each row consists of four columns.
  1. Index of the operator  $\alpha$
  2. Index of the sites  $i$
  3. Real part of the expected value  $\text{Re}\langle \hat{A}_i^\alpha \rangle$
  4. Imag part of the expected value  $\text{Im}\langle \hat{A}_i^\alpha \rangle$
- In addition, norm of the wave function  $\langle \Psi | \Psi \rangle$  is outputted as an operator with index of -1.
  - If the imaginary part is finite, something is wrong. A typical cause is that the bond dimension of the CTM is too small.

Example:

```
# The meaning of each column is the following:
# $1: op_group
# $2: site_index
# $3: real
# $4: imag
```

(continues on next page)

(continued from previous page)

```
# The names of op_group are the following:
# 0: site hamiltonian
# 1: Sz
# 2: Sx
# 3: Sy
# -1: norm

0 0 -4.99999945520001373e-04 0.000000000000000000e+00
0 1 -4.99999967900088089e-04 0.000000000000000000e+00
0 2 -4.99999894622883147e-04 0.000000000000000000e+00
0 3 -4.99999974605052581e-04 0.000000000000000000e+00
1 0 4.99999945520001376e-01 0.000000000000000000e+00
1 1 4.99999967900088049e-01 0.000000000000000000e+00
1 2 4.99999894622883134e-01 0.000000000000000000e+00
1 3 4.99999974605052522e-01 0.000000000000000000e+00
... Skipped ...
-1 3 1.000000000000000044e+00 0.000000000000000000e+00
```

**twosite\_obs.dat**

- Expectation values for two-site operations are outputted.
- Each row consists of six columns.
  1. Index of the two-site operator
  2. Index of the source site
  3. x coordinate of the target site from the source site
  4. y coordinate of the target site from the source site
  5. Real part of the expected value
  6. Imaginary part of the expected value
- In addition, norm of the wave function  $\langle \Psi | \Psi \rangle$  is outputted as an operator with index of -1.
  - If the imaginary part is finite, something is wrong. A typical cause is that the bond dimension of the CTM is too small.

Example:

```
# The meaning of each column is the following:
# $1: op_group
# $2: source_site
# $3: dx
# $4: dy
# $5: real
# $6: imag
# The names of op_group are the following:
# 0: bond hamiltonian
# 1: SzSz
# 2: SxSx
# 3: SySy
# -1: norm
```

(continues on next page)

(continued from previous page)

```

0 0 0 1 -2.49999925774909121e-01 3.38316768671362694e-21
0 0 1 0 -2.49999967989907063e-01 4.24343236807659553e-22
0 1 0 1 -2.49999972903562101e-01 -2.06825262200104597e-25
0 1 1 0 -2.49999957625646446e-01 2.06789370628128221e-24
0 2 0 1 -2.49999931343147630e-01 3.11801499860976615e-28
0 2 1 0 -2.49999939447834718e-01 1.65429596395607220e-24
... Skipped ...
-1 3 1 0 1.000000000000000067e+00 0.00000000000000000e+00

```

**multisite\_obs\_#.dat**

- Expectation values for multi-site operations are outputted.
- # in the filename is replaced by the number of sites in the operator,  $N$ .
- Each row consists of  $4 + 2(N - 1)$  columns.
- The first column is the index of the operator.
- The second column is the index of the site, which is the origin of the coordinate.
- The following columns are the relative coordinates of the other sites.
- The last two columns are the real and imaginary parts of the expected value.

**correlation.dat**

- Correlation functions  $C_i^{\alpha\beta}(x, y) \equiv \langle \hat{A}^\alpha(x_i, y_i) \hat{A}^\beta(x_i + x, y_i + y) \rangle$  are outputted.
- Each row consists of seven columns.
  1. Index of the left operator  $\alpha$
  2. Index of the left site  $i$
  3. Index of the right operator  $\beta$
  4. x coordinate of the right site  $x$
  5. y coordinate of the right site  $y$
  6. Real part  $\text{Re}C$
  7. Imaginary part  $\text{Im}C$

Example:

```

# $1: left_op
# $2: left_site
# $3: right_op
# $4: right_dx
# $5: right_dy
# $6: real
# $7: imag

0 0 0 1 0 -1.71759992763061836e-01 1.36428299157186382e-14
0 0 0 2 0 1.43751794649139675e-01 -1.14110668277268192e-14

```

(continues on next page)

(continued from previous page)

```

0 0 0 3 0 -1.42375391377041444e-01 1.14103263451826963e-14
0 0 0 4 0 1.41835919840103741e-01 -1.11365361507372103e-14
0 0 0 5 0 -1.41783912096811515e-01 1.12856813523671142e-14
0 0 0 0 1 -1.72711348845767942e-01 1.40873628493918905e-14
0 0 0 0 2 1.43814797743900907e-01 -1.17958665742991377e-14
0 0 0 0 3 -1.42415176172922653e-01 1.22109610917000360e-14
0 0 0 0 4 1.41838862178711583e-01 -1.19321507524565005e-14
0 0 0 0 5 -1.41792935491960648e-01 1.23094733264734764e-14
1 0 1 1 0 -7.95389427681298805e-02 6.15901595234210079e-15
1 0 1 2 0 2.019160940009441903e-02 -1.27162373457160362e-15
... Skipped ...
2 3 2 0 5 -1.41888376278899312e-03 -2.38672137694415560e-16

```

### correlation\_length.dat

The correlation length  $\xi$  is outputted. Each row consists of 3+n columns.

1. Direction (0: x, 1: y)
2. When direction is 0 it is y coordinate, and otherwise x coordinate
3. Correlation length  $\xi = 1/e_1$

The 4th and the subsequent columns show the logarithm of the absolute value of the eigenvalues of the transfer matrix,  $e_i = -\log |\lambda_i/\lambda_0|$  ( $i > 0$ ). This information may be used to estimate the bond dimension dependence of the correlation length. See PRX **8**, 041033 (2018) and PRX **8**, 031030 (2018) for more information.

Example:

```

# The meaning of each column is the following:
# $1: direction 0: +x, 1: +y
# $2: y (dir=0) or x (dir=1) coordinates
# $3: correlation length xi = 1/e_1
# $4-: eigenvalues e_i = -log|t_i/t_0|
#       where i > 0 and t_i is i-th largest eigenvalue of T

0 0 2.18785686529154477e-01 4.57068291744370647e+00 4.57068291744370647e+00 4.
↪ 88102462824739991e+00
0 1 2.20658864940629751e-01 4.53188228022952533e+00 4.53188228022952533e+00 4.
↪ 56359469233104953e+00
1 0 2.23312072254469030e-01 4.47803824443704013e+00 4.47803824443704013e+00 6.
↪ 03413555039678595e+00
1 1 2.00830966658579996e-01 4.97931178960083720e+00 4.97931178960083720e+00 5.
↪ 08813099309339911e+00

```

### 5.5.3 For time evolution mode

#### TE\_density.dat

The expectation value per site of each observable is outputted. Each row consists of four columns.

1. Time  $t$
2. Operator ID  $\alpha$
3. Real part of the expected value  $\text{Re}\langle\hat{A}_i^\alpha\rangle$
4. Imag part of the expected value  $\text{Im}\langle\hat{A}_i^\alpha\rangle$

Example:

```
# The meaning of each column is the following:
# $1: time
# $2: observable ID
# $3: real
# $4: imag
# The meaning of observable IDs are the following:
# 0: Energy
# 1: site hamiltonian
# 2: Sz
# 3: Sx
# 4: Sy
# 5: bond hamiltonian
# 6: SzSz
# 7: SxSx
# 8: SySy

0.0000000000000000e+00 0 -5.00684745572451129e-01 0.0000000000000000e+00
0.0000000000000000e+00 1 -6.84842757985213292e-04 0.0000000000000000e+00
0.0000000000000000e+00 2 4.99999945661913914e-01 0.0000000000000000e+00
0.0000000000000000e+00 3 9.24214061616496842e-05 0.0000000000000000e+00
... Skipped ...
4.99999999999993783e+00 8 2.54571641402435656e-01 3.25677610112348483e-17
```

#### TE\_onesite\_obs.dat

The expected values of the site operators  $\langle\hat{A}_i^\alpha\rangle = \langle\Psi|\hat{A}_i^\alpha|\Psi\rangle/\langle\Psi|\Psi\rangle$  are outputted. Each row consists of five columns.

1. Time  $t$
2. Index of the operator  $\alpha$
3. Index of the sites  $i$
4. Real part of the expected value  $\text{Re}\langle\hat{A}_i^\alpha\rangle$
5. Imag part of the expected value  $\text{Im}\langle\hat{A}_i^\alpha\rangle$ 
  - In addition, norm of the wave function  $\langle\Psi|\Psi\rangle$  is outputted as an operator with index of -1.
    - If the imaginary part is finite, something is wrong. A typical cause is that the bond dimension of the CTM is too small.

Example:



```
# The meaning of each column is the following:
# $1: time
# $2: op_group
# $3: site_index
# $4: real
# $5: imag
# The names of op_group are the following:
# 0: site hamiltonian
# 1: Sz
# 2: Sx
# 3: Sy
# -1: norm

0.0000000000000000e+00 0 0 -6.43318936197596913e-04 0.0000000000000000e+00
0.0000000000000000e+00 0 1 -6.73418200262321655e-04 0.0000000000000000e+00
0.0000000000000000e+00 0 2 -9.89240026254938282e-04 0.0000000000000000e+00
0.0000000000000000e+00 0 3 -4.33393869225996210e-04 0.0000000000000000e+00
0.0000000000000000e+00 1 0 4.99999945519898625e-01 0.0000000000000000e+00
0.0000000000000000e+00 1 1 4.99999967900020936e-01 0.0000000000000000e+00
0.0000000000000000e+00 1 2 4.99999894622765451e-01 0.0000000000000000e+00
... Skipped ...
4.99999999999993783e+00 -1 3 9.9999999999999667e-01 0.0000000000000000e+00
```

### TE\_twosite\_obs.dat

- Expectation values for two-site operations are outputted.
- Each row consists of six columns.
  1. Time  $t$
  2. Index of the two-site operator
  3. Index of the source site
  4. x coordinate of the target site from the source site
  5. y coordinate of the target site from the source site
  6. Real part of the expected value
  7. Imaginary part of the expected value
- In addition, norm of the wave function  $\langle \Psi | \Psi \rangle$  is outputted as an operator with index of -1.
  - If the imaginary part is finite, something is wrong. A typical cause is that the bond dimension of the CTM is too small.

Example:

```
# The meaning of each column is the following:
# $1: time
# $2: op_group
# $3: source_site
# $4: dx
# $5: dy
# $6: real
```

(continues on next page)

(continued from previous page)

```
# $7: imag
# The names of op_group are the following:
# 0: bond hamiltonian
# 1: SzSz
# 2: SxSx
# 3: SySy
# -1: norm

0.0000000000000000e+00 0 0 0 1 -2.49999925774803150e-01 -1.01660465821037727e-20
0.0000000000000000e+00 0 0 1 0 -2.49999967989888300e-01 4.23516895582898471e-22
0.0000000000000000e+00 0 1 0 1 -2.49999972903488521e-01 -6.20403358955599675e-25
0.0000000000000000e+00 0 1 1 0 -2.49999957625561042e-01 4.13590865617858526e-25
0.0000000000000000e+00 0 2 0 1 -2.49999931343070220e-01 8.27316466562544801e-25
... Skipped ...
4.9999999999993783e+00 -1 3 1 0 9.9999999999999445e-01 1.38777878078144568e-17
```

**TE\_multisite\_obs\_#.dat**

- Expectation values for multi-site operations are outputted.
- # in the filename is replaced by the number of sites in the operator,  $N$ .
- Each row consists of  $5 + 2(N - 1)$  columns.
- The first column is the time  $t$ .
- The second column is the index of the operator.
- The third column is the index of the site, which is the origin of the coordinate.
- The following columns are the relative coordinates of the other sites.
- The last two columns are the real and imaginary parts of the expected value.

**TE\_correlation.dat**

- Correlation functions  $C_i^{\alpha\beta}(x, y) \equiv \langle \hat{A}^\alpha(x_i, y_i) \hat{A}^\beta(x_i + x, y_i + y) \rangle$  are outputted.
- Each row consists of eight columns.
  1. Time  $t$
  2. Index of the left operator  $\alpha$
  3. Index of the left site  $i$
  4. Index of the right operator  $\beta$
  5. x coordinate of the right site  $x$
  6. y coordinate of the right site  $y$
  7. Real part  $\text{Re}C$
  8. Imaginary part  $\text{Im}C$

Example:

```
# The meaning of each column is the following:
# $1: time
# $2: left_op
# $3: left_site
# $4: right_op
# $5: right_dx
# $6: right_dy
# $7: real
# $8: imag
# The names of operators are the following:
# 0: site hamiltonian
# 1: Sz
# 2: Sx
# 3: Sy

0.0000000000000000e+00 0 0 0 1 0 1.83422488349707711e-04 1.90382762094233524e-20
0.0000000000000000e+00 0 0 0 2 0 8.30943360551218668e-07 -4.19695835411528090e-23
0.0000000000000000e+00 0 0 0 3 0 4.12158436385765748e-07 -1.04903226091485958e-23
0.0000000000000000e+00 0 0 0 4 0 4.13819451426396547e-07 1.74438421668770658e-23
0.0000000000000000e+00 0 0 0 5 0 4.33224506806043380e-07 -8.71850465073480394e-24
... Skipped ...
4.99999999999993783e+00 2 3 2 0 5 3.96301355731331212e-02 -1.37659660157453792e-18
```

### TE\_correlation\_length.dat

The correlation length  $\xi$  is outputted. Each row consists of 4+n columns.

1. Time  $t$
2. Direction (0:  $x$ , 1:  $y$ )
3. When direction is 0 it is  $y$  coordinate, and otherwise  $x$  coordinate
4. Correlation length  $\xi = 1/e_1$

The 5th and the subsequent columns show the logarithm of the absolute value of the eigenvalues of the transfer matrix,  $e_i = -\log |\lambda_i/\lambda_0|$  ( $i > 0$ ). This information may be used to estimate the bond dimension dependence of the correlation length. See PRX **8**, 041033 (2018) and PRX **8**, 031030 (2018) for more information.

Example:

```
# The meaning of each column is the following:
# $1: time
# $2: direction 0: +x, 1: +y
# $3: y (dir=0) or x (dir=1) coordinates
# $4: correlation length xi = 1/e_1
# $5-: eigenvalues e_i = -log|t_i/t_0|
#       where i > 0 and t_i is i-th largest eigenvalue of T

0.0000000000000000e+00 0 0 2.18785686529220424e-01 4.57068291744232891e+00 4.
↪ 57068291744232891e+00 4.88102462824919758e+00
0.0000000000000000e+00 0 1 2.20658864940612931e-01 4.53188228022987083e+00 4.
↪ 53188228022987083e+00 4.56359469232955917e+00
0.0000000000000000e+00 1 0 2.23312072254560540e-01 4.47803824443520515e+00 4.
↪ 47803824443520515e+00 6.03413555040836602e+00
```

(continues on next page)

(continued from previous page)

```

0.0000000000000000e+00 1 1 2.00830966658709920e-01 4.97931178959761578e+00 4.
↳ 97931178959761667e+00 5.08813099310449513e+00
9.99999999999999917e-02 0 0 2.02379048126702904e-01 4.94122296382149528e+00 4.
↳ 94122296382149617e+00 6.74309974506451315e+00
9.99999999999999917e-02 0 1 2.20416567580991346e-01 4.53686404327366777e+00 4.
↳ 53686404327366777e+00 6.18101616573088020e+00
9.99999999999999917e-02 1 0 2.12137154053103655e-01 4.71393143960851368e+00 4.
↳ 71393143960851368e+00 7.17220113786375002e+00
9.99999999999999917e-02 1 1 1.90367314703518503e-01 5.25300260476656966e+00 5.
↳ 25300260476656966e+00 7.61893825410630487e+00
2.000000000000000039e-01 0 0 1.96835348300227503e-01 5.08038829730281805e+00 5.
↳ 08038829730281805e+00 7.35176717846311778e+00
2.000000000000000039e-01 0 1 2.02355022722768896e-01 4.94180963014702801e+00 4.
↳ 94180963014702801e+00 6.57691315725687975e+00
2.000000000000000039e-01 1 0 2.05314677188187883e-01 4.87057239986509760e+00 4.
↳ 87057239986509760e+00 7.90951918842309798e+00
2.000000000000000039e-01 1 1 1.63323696507474692e-01 6.12281023136305169e+00 6.
↳ 12281023136305169e+00 7.83104916294462416e+00
... Skipped ...
4.9999999999999993783e+00 1 1 4.61585992965019176e-01 2.16644355600232430e+00 2.
↳ 16644355600232430e+00 2.29497956495965427e+00

```

## 5.5.4 For finite temperature calculation mode

The formats of the files are the same as those in the real time evolution mode. The only difference is that the file name starts with FT\_ instead of TE\_, and the first column is the inverse temperature  $\beta = 1/T$  instead of the time  $t$ .

## 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

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

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

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

where  $\text{tTr}[\dots]$  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  $\text{tTr}[\dots]$  becomes usual matrix products as

$$\Psi_{s_1, s_2, \dots, s_N}^{\text{MPS}} = T^{(1)}[s_1] T^{(2)}[s_2] \dots T^{(N)}[s_N],$$

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:

$$\Psi^{\text{MPS}} = \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---}$$

$$T_{ij}[s] = \begin{array}{c} i \text{---} \bigcirc \text{---} j \\ | \\ s \end{array}$$

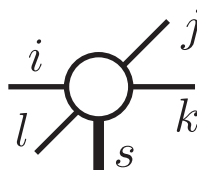
This MPS represents a wavefunction for a finite size system. Similarly, we can also consider an infinitely long MPS to represent an infinite system. Especially, when we assume a lattice translational symmetry, with a certain period, we can construct an infinite MPS (iMPS) with a few independent tensors.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

$$\Psi^{\text{iMPS}} = \dots \text{---} \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \dots$$

where tensors with the same color indicate identical tensors.

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

$$\Psi^{\text{iTPS}} = \dots \text{ [diagram of a 2D square lattice tensor network] } \dots$$

$T_{ijkl}[s] =$ 


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.

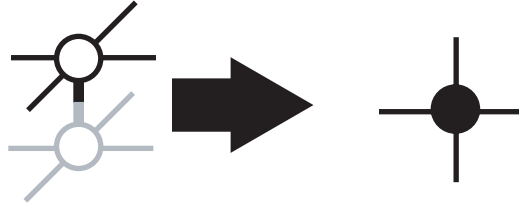
## 6.2 Contraction of iTPS

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

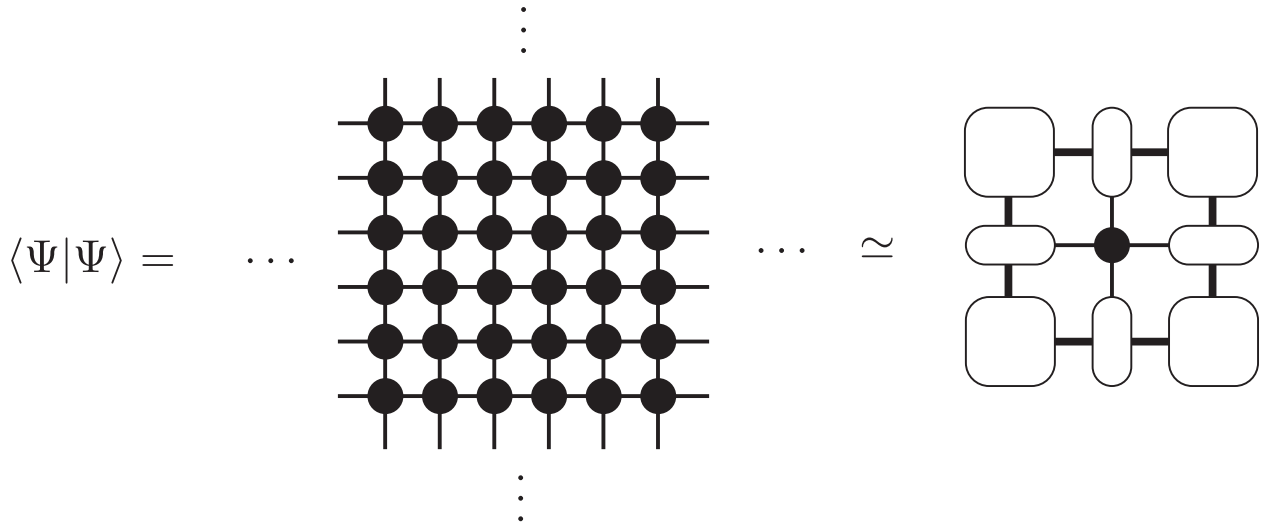
$$\langle \Psi | \Psi \rangle = \dots \text{ [diagram of a 2D square lattice tensor network with two overlapping layers, one solid and one faded] } \dots$$

which is often called a double layered tensor network. The contraction of a double layered tensor network often needs huge computation costs. 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*.

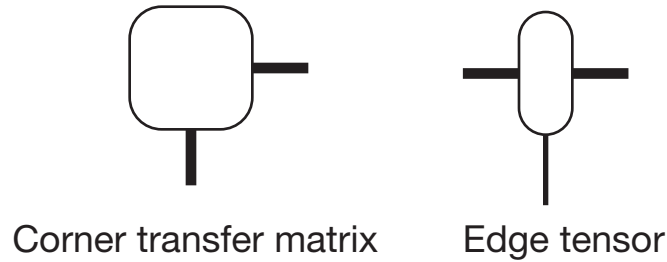
When we simplify the double layered tensor network by using a locally contracted tensor,



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

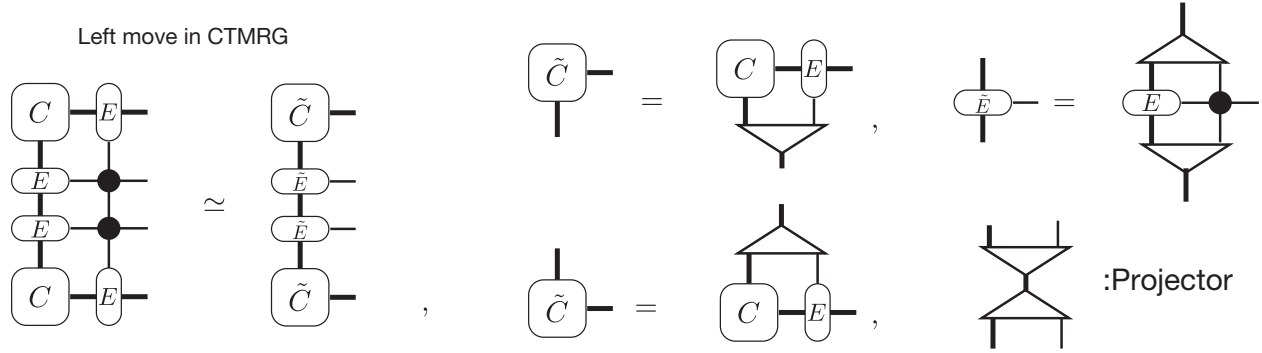


A corner transfer matrix and an edge tensor are defined as



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

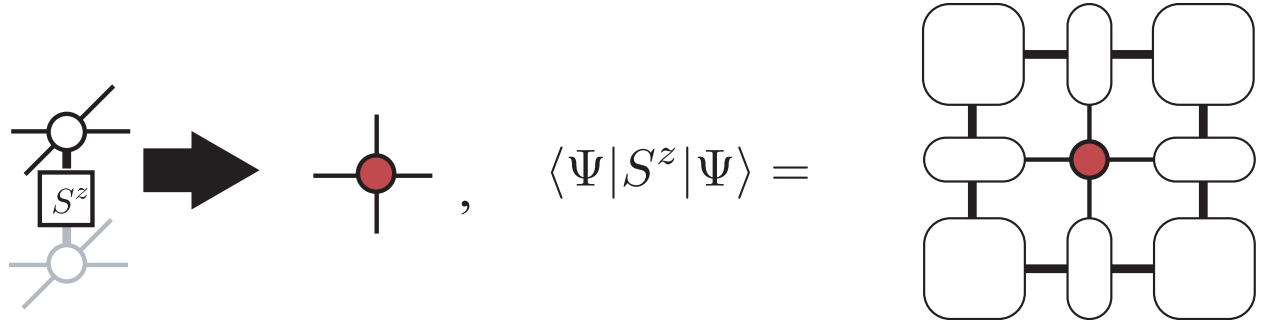
In the CTMRG algorithm, we iteratively optimise corner transfer matrices and edge tensors by *absorbing* local tensors until they converges. For example, an absorbing procedure, so called *left move*, is described as follows:



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

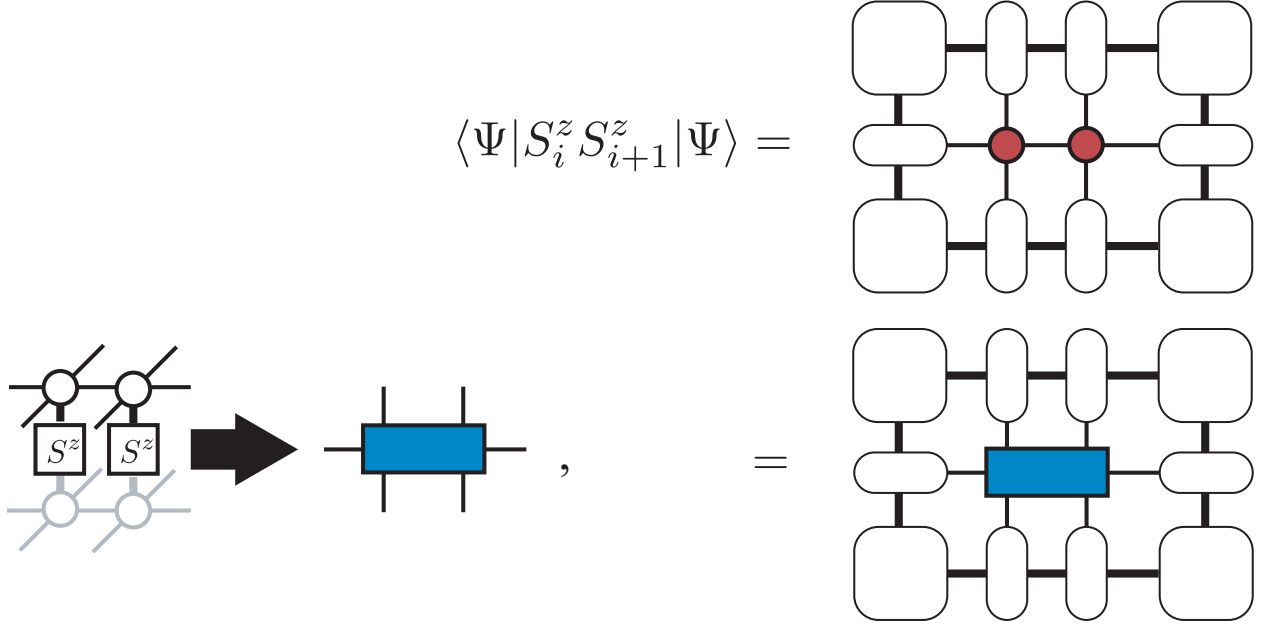
When we consider iTPS with the bond dimension  $D$  and CTMs with the bond dimension  $\chi$ , the leading computation cost of CTMRG scales as  $O(\chi^2 D^6)$  and  $O(\chi^3 D^4)$ . Notice that the bond dimension of the double layered tensor network becomes  $D^2$  by using locally contracted tensors. Thus, typically we increase  $\chi$  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 achieve 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 instead 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





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 operator, 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:

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

where  $|\Psi_0\rangle$  is an arbitrary initial iTPS. If  $T$  is sufficiently large, the left hand side,  $|\Psi^{\text{iTPS}}\rangle$ , 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

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

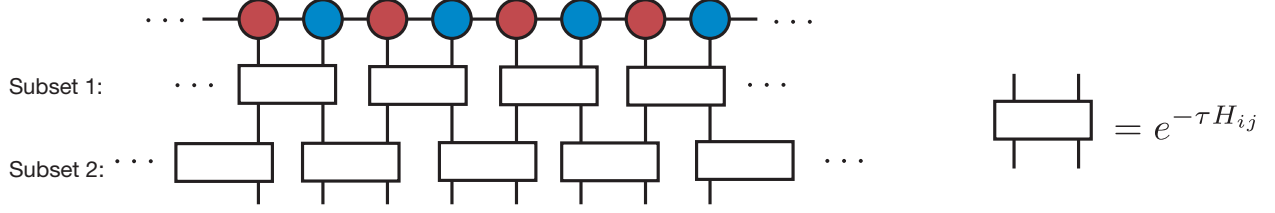
and apply Suzuki-Trotter decomposition to the ITE operator with small time step  $\tau$ :

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

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

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

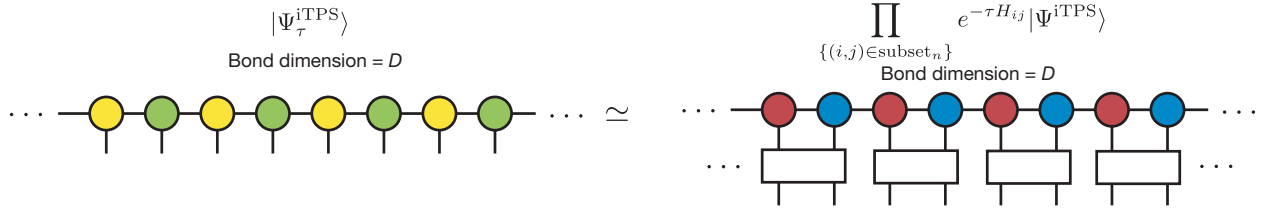
where  $N_\tau = T/\tau$  is the number of ITEs with sufficiently small  $\tau$ . In order to simulate the right hand side of the equation, we divide  $\prod_{\{(i,j)\}}$  into several subsets. In each subset, (local) ITE operators satisfy two properties: they commute with each other and they have the 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  $D$ :

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

where  $\prod_{\{(i,j) \in \text{subset}_n\}}$  means the product of operators in the  $n$ th subset, and  $|\Psi_\tau^{\text{iTPS}}\rangle$  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  $D$ .

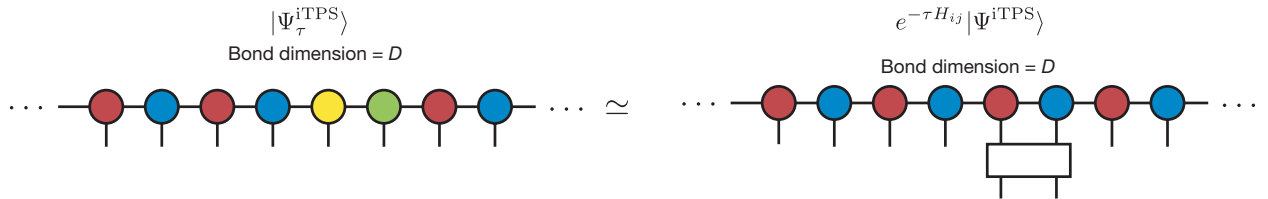
Naively, efficient truncation can be done by solving the minimization problem

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

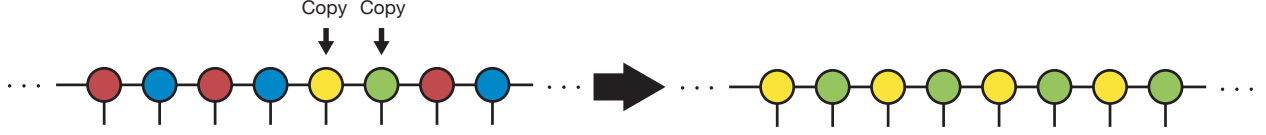
However, in practice, solving this minimization problem needs a huge computation cost because it is a highly nonlinear problem due to the translational symmetry of iTPS. Thus, instead, we usually consider an alternative local problem where we apply only a local ITE operator and try to find optimal iTPS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

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

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

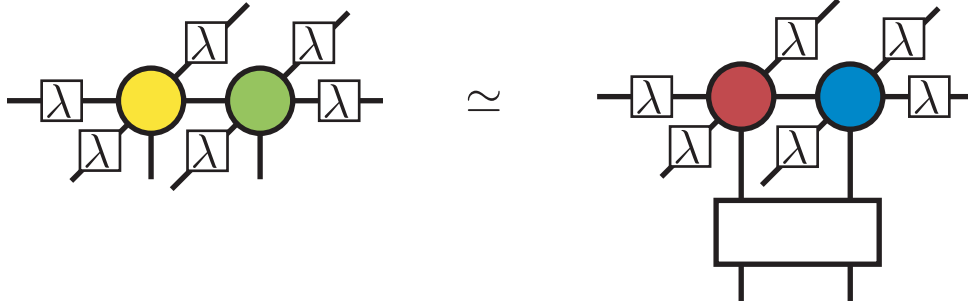
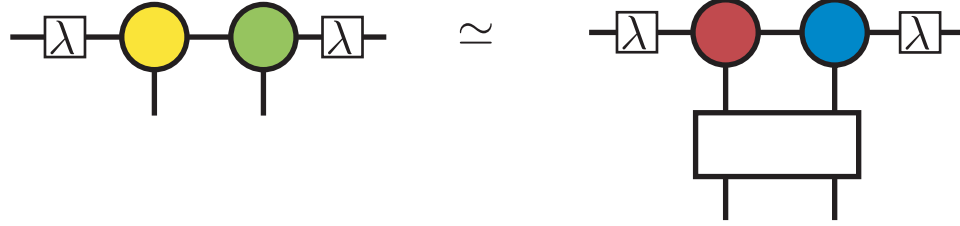


The squared norm  $\left\| |\Psi_\tau^{\text{iTPS}}\rangle - e^{-\tau H_{ij}} |\Psi^{\text{iTPS}}\rangle \right\|^2$  can be calculated by using, e.g., CTMRG and we can solve the minimization problem easily [ITE]. Although this new iTPS breaks translational symmetry, we make translationally symmetric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



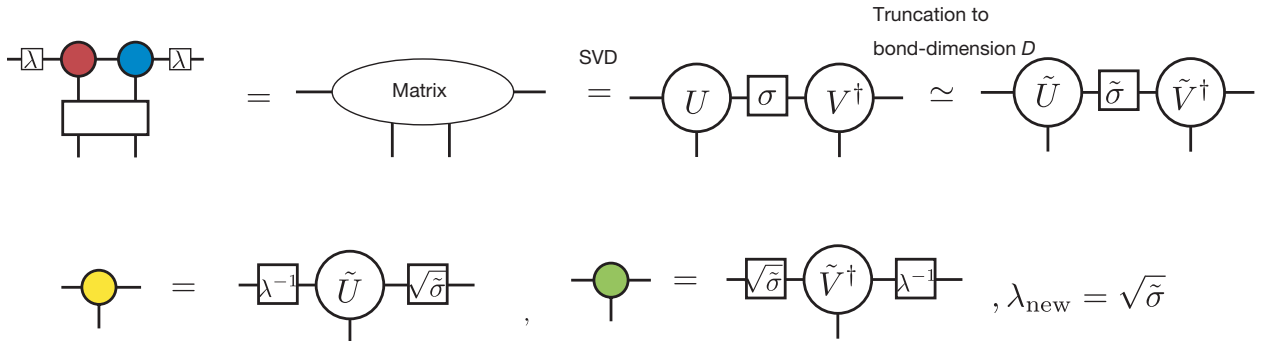
This ITE approach is often called as *full update*. The leading computation cost of the full update come from CTMRG and then it scales as  $O(D^{10})$  or  $O(D^{12})$  depending on SVD algorithms.

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:



$\boxed{\lambda}$  : Non-negative diagonal matrix

In this diagram,  $\lambda_i$  represents a non-negative diagonal matrix considered to be a mean field corresponding to the neglected environment beyond the bond  $i$ . The definition of  $\lambda_i$  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 are used as the mean field  $\lambda$  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 than 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.

## 6.4 Real-time evolution by iTPS

The algorithms of imaginary time evolution used for computing the ground state, such as the simple update method and the full update method, can also be used to calculate the real-time evolution of a quantum state. In TeNeS, similarly to the case of imaginary time evolution, the quantum state at time  $t$

$$|\Psi(t)\rangle = e^{-it\mathcal{H}}|\Psi_0\rangle,$$

is approximated by iTPS, which allows for the calculation of approximate time evolution. The difference between imaginary and real-time evolution lies only in whether the coefficient of the Hamiltonian  $\mathcal{H}$  in the exponent is  $-\tau$  or  $-it$ , hence real-time evolution can also be computed using the same simple update and full update methods applied in imaginary time evolution, by employing the Suzuki-Trotter decomposition.

Real-time evolution using iTPS (and other tensor network states) differs significantly from imaginary time evolution used for ground state calculation in two main aspects.

One major difference is the size of the quantum entanglement of the target quantum state. In imaginary time evolution, as the evolution progresses towards the ground state, the quantum entanglement of the state does not become excessively large. Thus, the description by iTPS works well. However, in real-time evolution, typically (unless the initial state's iTPS is an eigenstate of the Hamiltonian), quantum entanglement can increase over time. To maintain the approximation accuracy of iTPS, it is necessary to increase the bond dimension of iTPS as the time gets longer. Naturally, increasing the bond dimension also increases computational costs, so with realistic computational resources, accurately approximating real-time evolution using iTPS is limited to short times. The applicable time range depends on the model, but for example, in spin models, the limit is often around a time  $t = O(1/J)$  with respect to the typical interaction strength  $J$ .

Another difference is the characteristics of the physical phenomenon to be reproduced. When using imaginary time evolution to calculate the ground state, it is sufficient to reach the ground state after a sufficiently long evolution, so minor deviations from the correct path of imaginary time evolution are not a significant issue. On the other hand, in real-time evolution, there is often interest not only in the final state but also in the time evolution of the quantum state itself. To accurately approximate the path of time evolution, it is necessary to not only increase the bond dimension of iTPS but also to make the time increment  $\delta t$  of the Suzuki-Trotter decomposition sufficiently small. Depending on the situation, it may be more efficient to use higher-order Suzuki-Trotter decompositions. In TeNeS, it is possible to handle higher-order Suzuki-Trotter decompositions by editing the `evolution` section of the input file that is ultimately entered into TeNeS.

## 6.5 Finite temperature simulation

So far, we considered the tensor network representation of a pure state  $|\Psi\rangle$ , but similarly, we can consider the tensor network representation for a mixed state at finite temperature

$$\rho(\beta) = \frac{e^{-\beta\mathcal{H}}}{\text{Tr}e^{-\beta\mathcal{H}}}$$

where  $\beta$  represents the inverse temperature corresponding to temperature  $T$  as  $\beta = 1/T$ .

Similarly to pure states, if we consider a system of  $N$  quantum spins with  $S = 1/2$  at finite temperature, the mixed state can be expressed as

$$\rho(\beta) = \sum_{s_i=\uparrow,\downarrow, s'_i=\uparrow,\downarrow} (\rho(\beta))_{s_1, s_2, \dots, s_N}^{s'_1, s'_2, \dots, s'_N} |s'_1, s'_2, \dots, s'_N\rangle \langle s_1, s_2, \dots, s_N|$$

The expansion coefficients  $(\rho(\beta))_{s_1, s_2, \dots, s_N}^{s'_1, s'_2, \dots, s'_N}$  can be expressed, for example, using a Matrix Product Operator (MPO), generalized from MPS to matrices (operators), as

$$(\rho^{\text{MPO}}(\beta))_{s_1, s_2, \dots, s_N}^{s'_1, s'_2, \dots, s'_N} = T^{(1)}[s_1, s'_1] T^{(2)}[s_2, s'_2] \cdots T^{(N)}[s_N, s'_N]$$

and the corresponding diagram can be drawn as

$$\rho^{\text{MPO}} = \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---}$$

$$T_{ij}[s, s'] = \text{---} \begin{array}{c} s' \\ | \\ \bigcirc \\ | \\ s \end{array} \text{---}$$

For mixed states with translational symmetry, just like in the case of pure states, an infinite MPO (iMPO) can represent the state of an infinite system by repeating the same tensor infinitely. For example, for a one-dimensional, two-site translational symmetric state, the corresponding iMPO diagram would be

$$\rho^{\text{iMPO}} = \cdots \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \bigcirc \text{---} \cdots$$

As a tensor network representation of mixed states, in TeNeS, we handle a two-dimensional infinite tensor product operator (iTPO) [TPO], specifically assuming a square lattice network with translational symmetry. The diagram for such an iTPO can be written as

$$\rho^{\text{iTPO}} = \dots \begin{array}{c} \text{Diagram of a 2D tensor network for iTPO} \end{array} \dots$$

$T_{ijkl}[s, s'] =$

In TeNeS, the mixed state at finite temperature  $\rho(\beta)$  is computed using imaginary time evolution from the initial state corresponding to infinite temperature  $\rho(\beta = 0)$

$$\rho(\beta) = e^{-\frac{\beta}{2}\mathcal{H}}\rho(0)e^{-\frac{\beta}{2}\mathcal{H}}$$

Note that at infinite temperature, the density matrix is the identity matrix. From this property, for example, the iMPO representation of the state at infinite temperature becomes a tensor product of local identity matrices, and the diagram in this case would be drawn as

$$\rho^{\text{iMPO}}(0) = \dots \begin{array}{c} \text{Diagram of vertical lines representing local identity matrices} \end{array} \dots$$

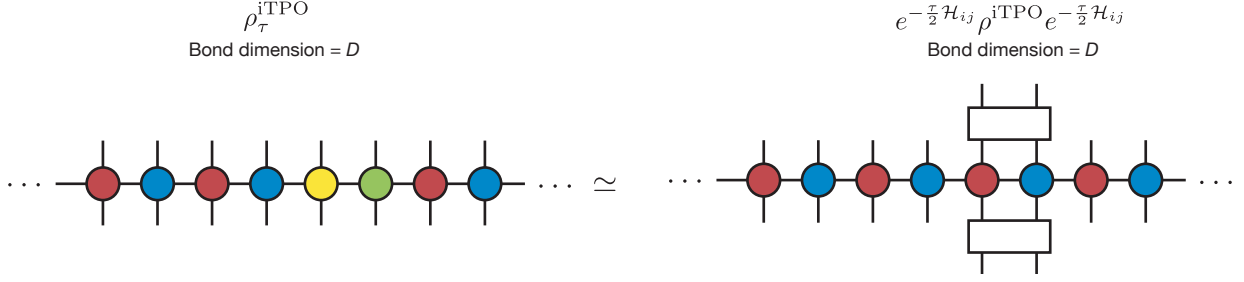
with “lines” corresponding to the local identity matrix.

The imaginary time evolution of a mixed state is calculated by a simple extension of the imaginary time evolution for pure states, as an approximate imaginary time evolution within the iTPO representation. The Suzuki-Trotter decomposition, simple update method, and full update method used for pure states can be almost directly applied to the case of mixed states. (TeNeS does not support the full update currently.)

The local minimization problem for mixed states can be described as

$$\min \left\| \rho_{\tau}^{\text{iTPO}} - e^{-\frac{\tau}{2}\mathcal{H}_{ij}/2} \rho^{\text{iTPO}} e^{-\frac{\tau}{2}\mathcal{H}_{ij}} \right\|^2$$

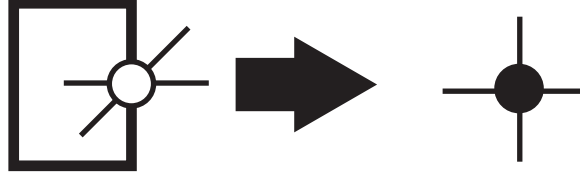
and the corresponding diagram, for clarity in the form of iMPO, would be



The biggest difference between the computations of finite temperature states by iTPO and pure states by iTPS appears in the tensor network for expectation value calculations. The expectation value of a physical quantity  $O$  for a given mixed state  $\rho$  is calculated as

$$\langle O \rangle_\rho = \frac{\text{Tr}(\rho O)}{\text{Tr}\rho}.$$

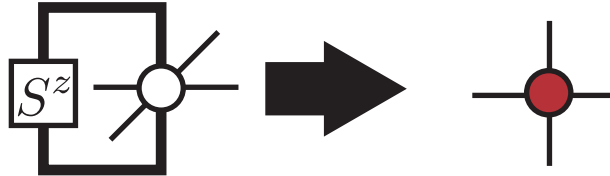
The trace  $\text{Tr}$  corresponds to connecting the corresponding upper and lower legs of the iTPO. Using a tensor obtained by connecting upper and lower legs of a local tensor in iTPO,



the denominator  $\text{Tr}\rho$  becomes the same structure as the two-dimensional square lattice diagram appeared in the expectation values for pure states. Thus, we can apply the same approximate calculation using corner transfer matrix representation and CTMRG.

The computation cost of CTMRG for the corner transfer matrix representation with bond dimension  $\chi$  and iTPO with bond dimension  $D$  scales with  $O(\chi^2 D^4)$  and  $O(\chi^3 D^3)$ . Note that this computation cost is smaller compared to CTMRG for pure states with the same bond dimension  $D$ . The difference is due to the bond dimension of the tensor indicated by the black circle being  $D^2$  in pure state calculations, while  $D$  for mixed states. Correspondingly, the bond dimension  $\chi$  of the corner transfer matrices can be increased proportionally to  $D$ , i.e.,  $\chi \propto O(D)$ . Under this condition, the computation cost of CTMRG becomes  $O(D^6)$ , and the required memory amount becomes  $O(D^4)$ . Thus, the computation cost of finite temperature calculations using iTPO is significantly lower than that of iTPS with the same  $D$ . It allows us to use larger bond dimensions  $D$  in finite temperature calculations.

Similarly to pure states, once the converged corner transfer matrices and edge tensors are computed,  $\text{Tr}(\rho O)$  can also be efficiently calculated. For example, when we define the tensor containing the operator as



the local magnetization  $\text{Tr}(\rho S_i^z)$  is calculated using the same diagram as  $\langle \Psi | S_i^z | \Psi \rangle$ .

Lastly, it is important to mention the drawbacks of approximation by iTPO. The density matrix of a mixed state is Hermitian and positive semidefinite, with non-negative eigenvalues. However, when approximating the density matrix with iTPO, this positive semidefiniteness is not guaranteed, and physical quantities calculated from the iTPO approximation might exhibit unphysical behavior, such as energies lower than the ground state energy. This is a problem of

iTPO representation, and cannot be avoided just by improving the accuracy of CTMRG in expectation value calculation by increasing the bond dimension  $\chi$ . To recover physical behavior, it is necessary to increase the bond dimension  $D$  of iTPO to improve the approximation accuracy of the density matrix.

As an alternative representation to avoid such unphysical behavior, a method has been proposed using purification of the density matrix, representing the purified density matrix with iTPO [Purification]. However, in this case, the diagram appearing in the expectation value calculation becomes a double-layer structure similar to pure states. This structure requires a larger computational cost, and the manageable bond dimension  $D$  becomes smaller than in the direct iTPO representation.

## 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. Schollwöck, *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](#)

[TPO] A. Kshetrimayum, M. Rizzi, J. Eisert, and R. Orús, *Tensor Network Annealing Algorithm for Two-Dimensional Thermal States*, Phys. Rev. Lett. **122**, 070502 (2019). [link](#)

[Purification] P. Czarnik, J. Dziarmaga, and P. Corboz, *Time evolution of an infinite projected entangled pair state: An efficient algorithm*, Phys. Rev. B **99**, 035115 (2019). [link](#); P. Czarnik and J. Dziarmaga, *Time evolution of an infinite projected entangled pair state: An algorithm from first principles*, Phys. Rev. B **98**, 045110 (2018). [link](#)



## FAQ

Q1. How can I set the number of full updates?

A1. The full update improves the accuracy of the calculation, but its computational time becomes longer. The number of full updates should be determined by balancing computer performance, bond dimensions, and required computational accuracy. One way to see what happens is to just do a simple update first. Also, if you do a simple update before the full update and approach the ground state, the full update can be done efficiently. However, if you have a complex quantum state (such as a spin liquid) and the simple update does not approach the correct ground state, you must do a full update from the beginning.

Q2. How can I set the number of the simple updates?

A2. Increasing the number of simple updates over time should bring the simulated quantum state closer to the ground state, but this is not necessarily the case; if the bond dimension is small, the calculation accuracy may deteriorate during the update. To see if the calculation works, plot the ground state energy against the number of simple updates. It is a good idea to increase the number of updates and adopt the minimum energy as the calculation result. Another strategy is to increase the number of the simple update so that the energy is almost the same, but that is not necessarily the minimum energy.

Q3. How do I get the bond dimension?

A3. Increasing the bond dimension improves calculation accuracy but increases calculation time. It is necessary to determine the bond dimension by considering the balance of the computer resources and the accuracy required for the physical quantity desired. It is also important to change `parameter.ctm.dimension` together when changing the bond dimension assigned by `lattice.virtual_dim`. Typically, the latter takes a value greater than or equal to the square of the former.

Q4. How should I test the correctness of the calculated ground state?

A4. It is difficult to guarantee that the calculated ground state is correct, but the easiest way is to check whether lower energy states are obtained or not by using multiple seed numbers. It is also useful to calculate the initial configuration guessed from several candidate ground states and compare their energies, though its computational cost is expensive. It is also important to change the shape of the unit cell to check for other low-energy states. Although there is a size issue, it is recommended to compare with other methods such as exact diagonalization. (The exact diagonalization can be easily performed using  $H \Phi$ .)



## **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.



## **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 `@`).