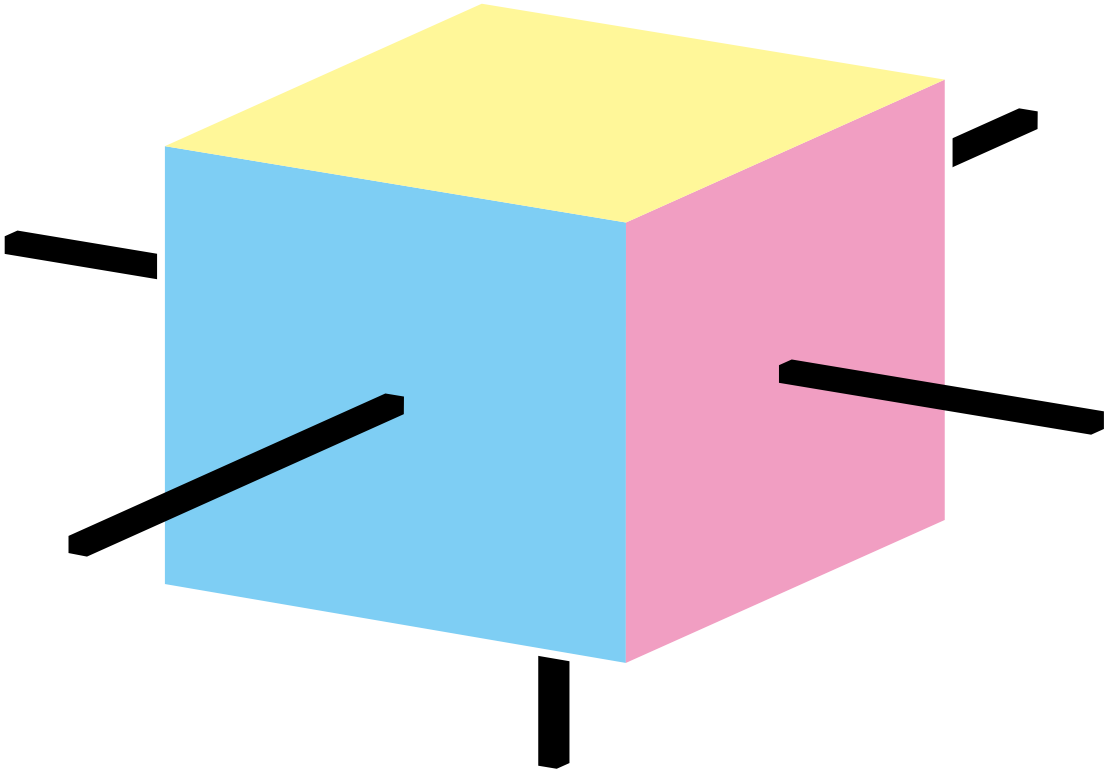


TeNeS

TeNeS ユーザーマニュアル

リリース ***1.0.0***

東京大学物性研究所

2020 年 **04** 月 **18** 日

目次

第 1 章

TeNeS とは？

1.1 概要

TeNeS (**Te** nsor **Ne** twork **S** olver) はテンソルネットワーク法に基づく多体量子状態計算のためのオープンソースのプログラムパッケージです。二次元格子上で定義された量子スピン模型などの多体ハミルトニアンの基底状態波動関数を求め、磁化や相関関数などの物理量を計算します。あらかじめ定義された模型・格子に対しては、ユーザーが簡単に入力ファイルを作成するためのツールがあり、気軽に体験できます。OpenMP/MPI ハイブリッド並列に対応しており、大規模計算機による大規模計算が可能です。

1.2 開発者

TeNeS は以下のメンバーで開発しています。

- ver 1.0
  - 大久保 毅 (東京大学大学院 理学系研究科)
  - 森田 悟史 (東京大学 物性研究所)
  - 本山 裕一 (東京大学 物性研究所)
  - 吉見 一慶 (東京大学 物性研究所)
  - 加藤 岳生 (東京大学 物性研究所)
  - 川島 直輝 (東京大学 物性研究所)

1.3 バージョン履歴

- ver. 1.0.0: 2020-04-17 にリリース。
- ver. 1.0-beta: 2020-03-30 にリリース。
- ver. 0.1: 2019-12-04 にリリース。

1.4 ライセンス

本ソフトウェアのプログラムパッケージおよびソースコード一式は GNU General Public License version 3（GPL v3）に準じて配布されています。

1.5 コピーライト

本ソフトウェアの一部は 2019 年度 東京大学物性研究所 ソフトウェア高度化プロジェクトの支援を受け開発されており、その著作権は東京大学が所持しています。

第 2 章

インストール方法

2.1 ダウンロード

TeNeS のソースコードは GitHub page からダウンロードできます。git がインストールされている環境で、以下のコマンドを打つとダウンロードが開始されます。

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

2.2 必要なライブラリ・環境

TeNeS をコンパイルするには以下のライブラリ・環境が必要です。

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

TeNeS は以下のライブラリに依存していますが、自動でダウンロードおよびビルドがされます。

1. mptensor
2. cpptoml
3. sanitizers-cmake

MPI および ScaLAPACK を利用することでテンソル演算を並列化できます。MPI, ScaLAPACK については自身でインストールする必要があります。たとえば Debian GNU/Linux （やその派生ディストリビューション）をお使いで、root 権限をお持ちの場合は、

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

でインストールすることが可能です。それ以外の方は、Open MPI などの MPI 実装ならびに ScaLAPACK のホームページを参照の上、インストールをしてください。

入力ファイル作成ツールの使用には Python (>= 3.0.0) および以下の Python パッケージが必要です。

1. numpy
2. scipy
3. toml

2.3 インストール

1. TeNeS のディレクトリに移動したのち、以下の手順に従ってビルドを行います。

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

<path to install to> のデフォルト値は /usr/local です。

（ CentOS など、環境によっては cmake3 とする必要があります。）

なお、上記のコマンドで build/src ディレクトリに実行ファイル tests が作成されます。

```
$ make test
```

と打つとテストを実行することができます。

2. 次にインストールします。

```
$ make install
```

実行ファイル tenes, tenes_std, tenes_simple が <path to install to>/bin にインストールされます。

MPI/ScaLAPACK 並列化の無効化

MPI および ScaLAPACK を利用しない場合には、 -DENABLE_MPI=OFF オプションを cmake コマンドに追加してください。macOS では ScaLAPACK の一部関数とシステムの BLAS, LAPACK とで相性が悪く、エラー終了するのを確認しています。MPI 並列の無効化を推奨しています。

コンパイラの指定

CMake では自動でコンパイラを検出してビルドを行います。コンパイラを指定したい場合には, 以下のようにオプションを追加してください。

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

mptensor の指定

TeNeS は並列テンソル演算ライブラリ mptensor を利用しています。CMake は自動で mptensor をダウンロード・ビルドしますが、ユーザーが事前にインストールした mptensor を使用したい場合には、以下のようにオプションを追加してください。

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

Python インタープリタの指定

TeNeS ツールは Python3 で書かれており、 パスの通った `python3` コマンドを自動的に起動します。ツールの実行がうまく行かない場合には、 `type python3` などを利用して、 `python3` コマンドにパスが通っているかどうか確認してください。

使うインタープリタを固定したい場合（あるいは `/usr/bin/env` コマンドが実行できずにエラーが出る場合）には、 以下のように CMake オプションを追加してください。

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

第 3 章

使用方法

TeNeS のメインプログラム tenes を利用するには、模型や演算順などを定義するための入力ファイルを作成する必要があります。入力ファイルの作成をしやすいように、

- tenes_std: 所定のフォーマットに従い、自分で格子模型などを定義した入力ファイルを作成し実行することで、 tenes を実行するための入力ファイルを生成するツール
- tenes_simple: あらかじめ定義された格子模型に対して, 簡単な入力ファイルから tenes_std を実行するための入力ファイルを生成するツール

が用意されています (図 3.1 参照)。任意の模型や格子に対応させたい場合には、 tenes_std の入力ファイルを直接作成することで対応できます。 TeNeS の各種入力ファイルの詳細については、ファイルフォーマット をご覧ください。

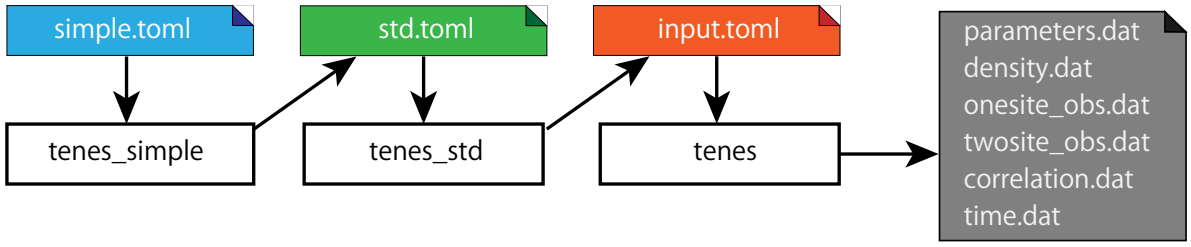


図 3.1　TeNeS の計算フロー概要図

以下、それぞれのツールの使用方法について記載し、最後に tenes の使用方法について説明します。

3.1 tenes_simple の使用方法

tenes_simple は定義済みの模型、格子に対する tenes_std の入力ファイルを生成するツールです。

```
$ tenes_simple simple.toml
```

- 引数としてファイルを取ります
- tenes_std の入力ファイルを出力します
- コマンドラインオプションは以下の通りです
  - --help
    - ヘルプメッセージの表示
  - --version
    - バージョン番号の表示
  - --output=filename
    - 出力するファイルの名前 filename を指定します
    - デフォルトは std.toml
    - 入力ファイル名と同じファイル名にすることはできません
  - --coordinatefile=coordfile
    - サイトの座標情報を出力するファイル名 coordfile を指定します
    - デフォルトは coordinates.dat
    - 座標ファイルは 1 列目にサイト番号を、 2, 3 列目に x, y 座標（デカルト座標系）を含みます

現在定義されている模型・格子は次の通り。

- 模型
  - スピン系
- 格子
  - 正方格子
  - 三角格子
  - 蜂の巣格子
  - かごめ格子

模型・格子や入力ファイルの詳細は *tenes_simple の入力ファイル* を参照してください。以下、正方格子上で定義されたスピン 1/2 のハイゼンベルグ模型の入力ファイル例です。

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

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

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

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

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

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

3.2 tenes_std の使用方法

tenes_std は与えられたハミルトニアンから虚時間刻み τ の虚時間発展演算子 $\exp\left(-\tau\mathcal{H}_{ij}\right)$ を導出し、tenes の入力ファイルを生成するツールです。

```
$ tenes_std std.toml
```

- 引数としてファイルを取ります
- tenes の入力ファイルを出力します
- コマンドラインオプションは以下の通りです
  - --help
    - ヘルプメッセージの表示
  - --version

        * バージョン番号の表示

    - `--output=filename`

        * 出力するファイルの名前 `filename` を指定します

        * デフォルトは `input.toml`

        * 入力ファイル名と同じファイル名にすることはできません

入力ファイルは `tenes_simple` を用いて生成できます。さらに、入力ファイルを編集することで、定義されていない模型・格子での計算が行なえます。入力ファイルの詳細は *tenes_std の入力ファイル* を参照してください。以下、正方格子上で定義されたスピン 1/2 のハイゼンベルグ模型の入力ファイル例です。

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

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

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

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

(次のページに続く)

(前のページからの続き)

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

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

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

(次のページに続く)

(前のページからの続き)

```
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 onesite operators
```

3.3 tenes の使用方法

tenes はメインプログラムです。

```
$ tenes input.toml
```

- 引数として入力ファイル名を取ります
- コマンドラインオプションは以下の通りです
  - --help - ヘルプメッセージの表示
  - --version - バージョン情報の表示
  - --quiet - 標準出力に何も書き出さないようにします

多くの場合において、ユーザーが入力ファイルを直接編集する必要はありません。入力ファイルの詳細は *tenes の入力ファイル* を参照してください。

第 4 章

チュートリアル

4.1 横磁場イジングモデル

ここでは横磁場イジングモデルに対して、横磁場を変化させた場合の計算例について紹介します。入力ファイルの変数 G を用いることで横磁場の大きさを調整することが可能です。例えば、横磁場が 0 の場合には、

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

[parameter.simple_update]
num_step = 1000
tau = 0.01

[parameter.full_update]
num_step = 0
tau = 0.01

[parameter.ctm]
iteration_max = 10
dimension = 10

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

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

(次のページに続く)

(前のページからの続き)

```
G  = 0.0
```

とします (Jz = -1.0 なので、 G=0 では強磁性状態になります)。入力ファイルを simple.toml とした場合、

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

を実行することで計算が開始されます。（あらかじめ TeNeS をインストールしたのち、環境変数 PATH を適切に設定してください。）計算を実行すると、

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

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

(次のページに続く)

(前のページからの続き)

```
Done.
```

のように計算が実行されます。最初に並列化の情報およびテンソルの実虚が表示されます。次に計算プロセスの実行状況が表示されます。計算終了後、 1 サイト演算子 Sz, Sx およびハミルトニアン hamiltonian , 最近接相関 SzSz, SxSx, SySy のサイトあたりの期待値が出力されます。最後にフェーズごとの計算時間が出力されます (単位は秒)。計算終了後は output ディレクトリに density.dat, parameters.dat, time.dat, onesite_obs.dat, twosite_obs.dat がそれぞれ出力されます。各出力ファイルの詳細は、 出力ファイル をご覧ください。例えば <Sz> の値は、 onesite_obs.dat から読み取ることが可能です。G をパラメータとして 0.2 刻みで 0-3.0 まで振ったときの結果を下図に表示します。

なお、サンプルスクリプトの例として、 sample/01_transverse_field_ising フォルダ内に tutorial_example.py , tutorial_read.py があります。

- tutorial_example.py の中身

```
import subprocess

import numpy as np

import toml


num_g = 16
min_g = 0.0
max_g = 3.0

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

- tutorial_read.py の中身

```
from os.path import join

import numpy as np

import toml


num_g = 16

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

あらかじめ tenes などにパスを通した上で

```
$ python tutorial_example.py
```

として実行できます (MacBook2017, 1.4 GHz Intel Core i7 で数分程度で計算が全て終了します)。得られた結果は

```
$ python tutorial_read.py
```

とすることで集計でき、 G, エネルギー、 <Sz> 、 <Sx> が出力されます。

図 4.1 から G が大きくなるにつれ、 <Sz> が 0.5 から徐々に小さくなり最終的には 0 になる一方、 <Sx> は 0 から大きくなり最終的には 0.5 になることが分かります。

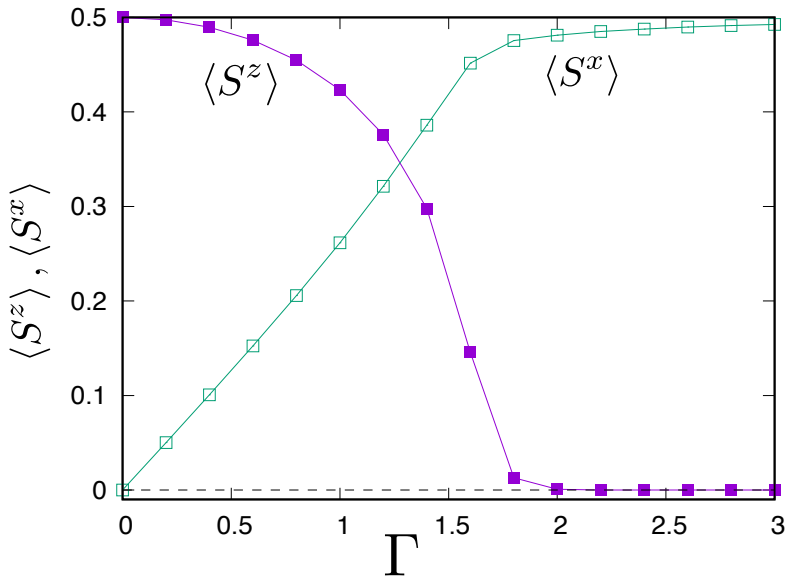


図 4.1　<Sz> , <Sx> の G 依存性

4.2 三角格子・正方格子ハイゼンベルク模型の磁化過程

次に三角格子上で定義されたスピン $S = 1/2$ の量子ハイゼンベルク模型の磁化過程の計算を紹介します。ハミルトニアンは以下のようになります:

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

ここで $\langle i, j \rangle$ は隣接サイトの組を表し、h は z 方向にかけられた外部磁場の大きさを表します。この模型の基底状態を計算し、ユニットセルの平均磁化 $\langle S_z \rangle \equiv \frac{1}{N_u} \sum_i^{N_u} \langle S_i^z \rangle$ を磁場 h の関数として求めてみましょう (N_u はユニットセル内のサイト数)。

この計算を行うには、 sample/05_magnetization のディレクトリ内にある toml ファイル basic.toml と、python スクリプト tutorial_magnetization.py を利用します。 basic.toml ファイルには、模型の設定やパラメータなどが記述されています。

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

[parameter.simple_update]
num_step = 200
tau = 0.01

[parameter.full_update]
num_step = 0
tau = 0.01
```

(次のページに続く)

(前のページからの続き)

```
[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
```

lattice セクションで三角格子を指定しており、ユニットセルの大きさは 3×3 を指定しています。ここでは計算を軽くするために、 simple update だけを行っており、虚時間の刻み幅 τ は $\tau = 0.01$ としています。また簡単のため、$J = 1$ としています。この基本設定ファイルを用いて、 tutorial_magnetization.py では磁場を掃引したときの磁化を計算します。

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

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

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

(次のページに続く)

(前のページからの続き)

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

このスクリプトでは、磁場 h を 0 から 5 まで 0.25 刻みで変化させ、基底状態のエネルギーと $\langle S_z \rangle$ を計算して、`energy.dat` および `magnetization.dat` に出力します。 `simple update` の時間ステップ数を 100, 200, 500, 1000, 2000 と変化させたときの様子を見るために、各磁場でステップ数を変えた計算も行っています。計算量を減らすために、少ないステップ数で得られた波動関数の情報を `tensor_save` に保存し、それをより多いステップ数の計算の初期状態としてとっています。例えば、最初に時間ステップ数を 100 とした計算を行って結果を出力したあと、ステップ数 100 の計算終了時の波動関数からさらにステップ数 100 の計算を行って、結果的にステップ数 200 の計算結果を得ます。

実際に実行してみましょう。あらかじめ `tenes` などにパスを通した上で

```
python tutorial_magnetization.py
```

により計算を実行します。ノート PC(シングルプロセッサ) では数時間程度の計算量となります。計算が終了したら、gnuplot を起動し、

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

とすれば、図 4.2 の右図のような磁化カーブが得られます。同様に

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

とすれば、図 4.2 の左図のような基底エネルギーのグラフが得られます。

十分なステップ数 (例えばステップ数 2000) の計算結果からわかるように、磁化過程には飽和磁化 $\langle S_z \rangle = 0.5$ の 1/3 の磁化のところで、プラトー構造が生じます。このプラトー上では、3 つの格子上のスピンが $\uparrow, \uparrow, \downarrow$ と磁化した周期構造を形成し、スピンギャップが生じています。このプラトー構造は三角格子特有のものです。実際に計算精度がでているかどうかをみるには、エネルギーのステップ依存性が参考になります。理想的にはステップ数を増やすほど基底エネルギーが下がるはずですが、一部の磁場領域では逆に基底エネルギーが増加します。これは計算精度があまりでていない兆候です。ボンド次元を増やすなどして、より計算精度を高める必要があると推測されます。

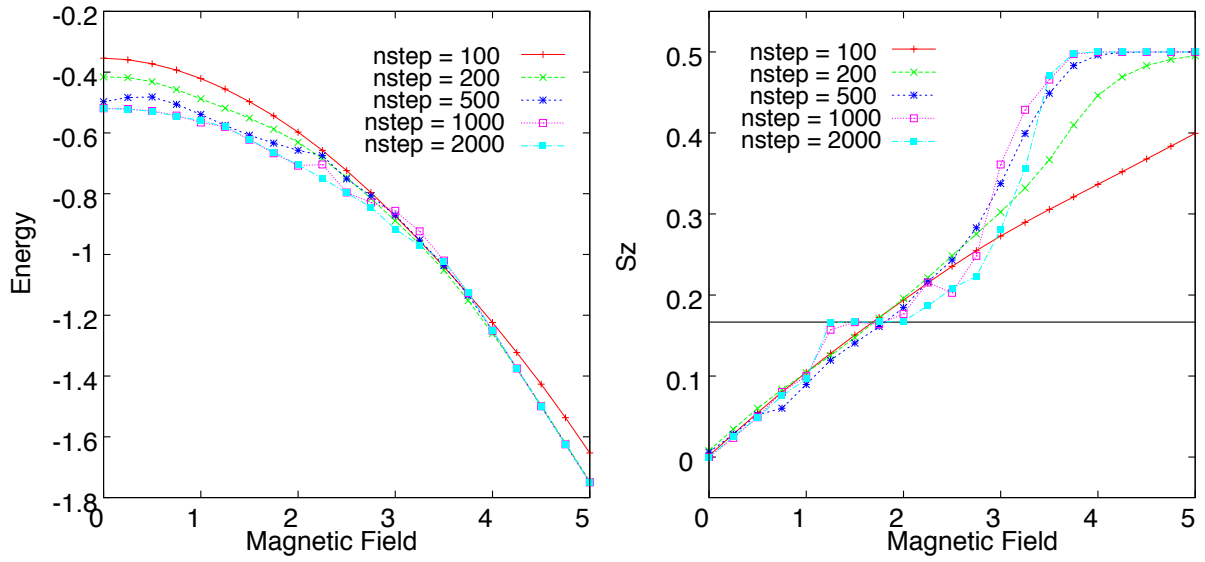


図 4.2　三角格子量子ハイゼンベルク模型のエネルギー (左図) と磁化過程 (右図)

では正方格子でも同じことをやってみましょう。 `sample/05_magnetization` のディレクトリ中にある toml ファイル `basic_square.toml` と、python スクリプト `tutorial_magnetization_square.py` を利用します。`basic_square.toml` は、 `lattice` セクションが以下のように変更されているほかは `basic.toml` と同じ内容です。

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

実際に計算を行うには、

```
python tutorial_magnetization_square.py
```

とします。計算が終了したら、gnuplot を起動し、

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

とすれば、 図 4.3 の右図のような磁化カーブが得られます。同様に

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

とすれば、図 4.3 の左図のような基底エネルギーのグラフが得られます。

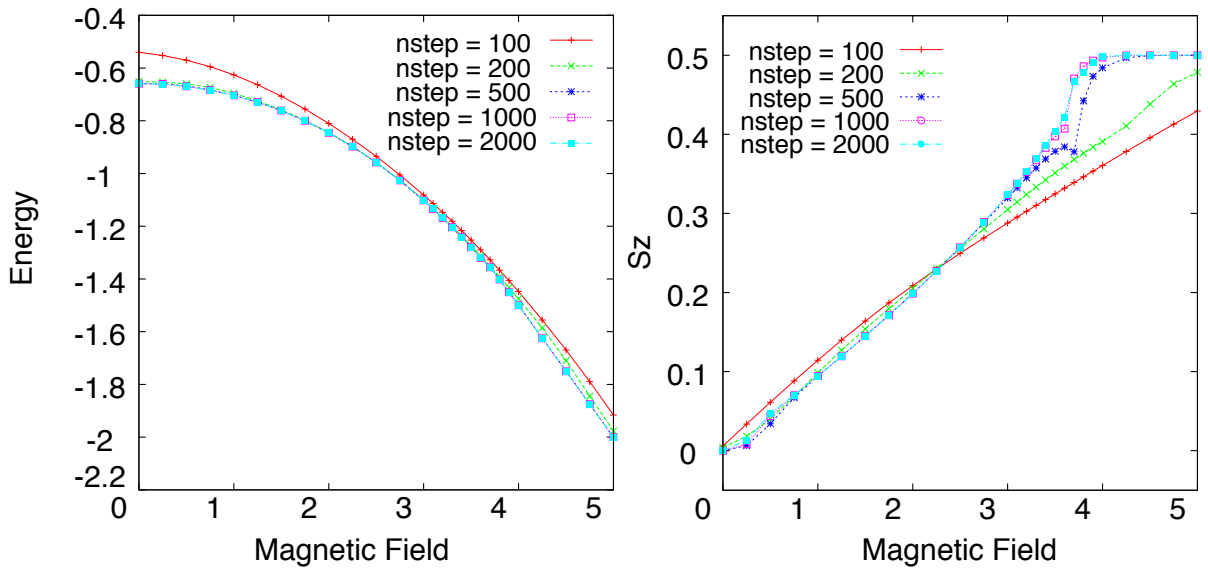


図 4.3　正方格子量子ハイゼンベルク模型のエネルギー (左図) と磁化過程 (右図)

ステップ数 2000 でほぼ収束しており、三角格子ハイゼンベルク模型と異なり、プラトー構造は現れないことがわかります。エネルギーは概ね、ステップ数を増加させると減少するため、ある程度計算精度がでていると推測されます。

第 5 章

ファイルフォーマット

5.1 TeNeS の入力ファイルの簡易まとめ

TeNeS の入力ファイルは TOML 形式 で書かれており、入力ファイルはいくつかのセクションに分かれています。`tenes_simple` と `tenes_std` は自分が必要とするセクションの情報を入力として読み取り、それぞれ `tenes_std` と `tenes` の入力ファイルを生成します。`tenes` は入力ファイルの各セクションに書かれた情報を元に実際の計算を行います。

例えば `tenes_simple` は `model` と `lattice` の情報から `tensor`, `observable`, `hamiltonian` の情報を生成し、さらに `parameter`, `correlation` はそのままコピーして、 `tenes_std` の入力ファイルとして出力します。

次表は各セクションの簡単な説明および各ツールがどう扱うかを示しています。

セクション名	説明	`tenes_simple`	`tenes_std`	`tenes`
`parameter`	計算パラメータ	copy	in / copy	in
`model`	模型パラメータ	in		
`lattice`	格子パラメータ	in		
`tensor`	テンソル	out	in / copy	in
`observable`	測定する演算子	out	copy	in
`correlation`	相関関数	copy	copy	in
`hamiltonian`	ハミルトニアン	out	in	
`evolution`	虚時間発展演算子		out	in

- "in"
  - ツールはこのセクションの情報を利用します
- "out"
  - ツールはこのセクションを新たに生成し、出力します

- "copy"
  - ツールはこのセクションを変更せずにそのまま出力します

5.2 tenes_simple の入力ファイル

- ファイルフォーマットは TOML 形式
- model, lattice, parameter, correlation の 4 つのセクションを持ちます。
  - parameter セクションはそのままスタンダードモードの入力へとコピーされます。

5.2.1 model セクション

計算する模型を指定します。スピン系 (spin) が定義済みです。

名前	説明	型	デフォルト
type	模型の種類	文字列	--

模型の種類によって相互作用などのパラメータ名が変わります。

スピン系 spin

スピン系

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

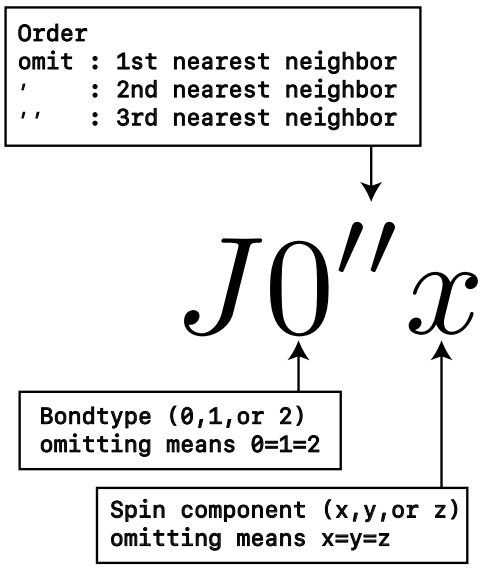
一体項のパラメータは次の通り。

名前	説明	型	デフォルト
S	局所スピンの大きさ	実数（整数もしくは半整数）	0.5
h	縦磁場 h	実数	0.0
G	横磁場 Γ	実数	0.0
D	オンサイトスピン異方性 D	実数	0.0

交換相互作用 J にはボンド依存性をもたせることができます。

名前	説明	型	デフォルト
J0	最近接・第 0 方向ボンドの交換相互作用	実数	0.0
J1	最近接・第 1 方向ボンドの交換相互作用	実数	0.0
J2	最近接・第 2 方向ボンドの交換相互作用	実数	0.0
J0'	次近接・第 0 方向ボンドの交換相互作用	実数	0.0
J1'	次近接・第 1 方向ボンドの交換相互作用	実数	0.0
J2'	次近接・第 2 方向ボンドの交換相互作用	実数	0.0
J0''	三次近接・第 0 方向ボンドの交換相互作用	実数	0.0
J1''	三次近接・第 1 方向ボンドの交換相互作用	実数	0.0
J2''	三次近接・第 2 方向ボンドの交換相互作用	実数	0.0

ボンドの方向は lattice セクションで定義される格子に依存します。例えば正方格子では x 方向 (0) と y 方向 (1) の 2 種類のボンド方向ごとに定義できます。方向を示す番号を省略することで、すべての方向について一度に指定することもできます。また、最後に *xyz* のうち一文字を追加するとイジング的な相互作用を指定できます。同一ボンド・同一成分を 2 回以上指定するとエラー終了します。



双二次相互作用 B も J と同様にボンド依存性をもたせらることもできます。

名前	説明	型	デフォルト
B0	最近接・第 0 方向ボンドの双二次相互作用	実数	0.0
B1	最近接・第 1 方向ボンドの双二次相互作用	実数	0.0
B2	最近接・第 2 方向ボンドの双二次相互作用	実数	0.0
B0'	次近接・第 0 方向ボンドの双二次相互作用	実数	0.0
B1'	次近接・第 1 方向ボンドの双二次相互作用	実数	0.0
B2'	次近接・第 2 方向ボンドの双二次相互作用	実数	0.0
B0''	三次近接・第 0 方向ボンドの双二次相互作用	実数	0.0
B1''	三次近接・第 1 方向ボンドの双二次相互作用	実数	0.0
B2''	三次近接・第 2 方向ボンドの双二次相互作用	実数	0.0

物理量測定に使われる 1 サイト物理量として、 S^z と S^x 、(`parameter.general.is_real = false` ならば) S^y を自動的に定義されます。また、2 サイト物理量として、ボンドハミルトニアン

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

および最近接ボンド上の相関 $S_i^{\alpha} S_j^{\alpha}$ ($\alpha = x, y, z$) が自動的に定義されます。

5.2.2 `lattice` セクション

計算する格子を指定します。正方格子 (square) と 三角格子 (triangular), 蜂の巣格子 (honeycomb) , かごめ格子 (kagome) が定義されています。

名前	説明	型	デフォルト
type	格子名 (square, triangular, honeycomb, もしくは kagome)	文字列	--
L	ユニットセルの x 方向の大きさ	整数	--
W	ユニットセルの y 方向の大きさ	整数	L
virtural_dim	ボンド次元	整数	--
initial	初期状態	文字列	random
noise	初期テンソルの揺らぎ	実数	1e-2

initial と noise は波動関数の初期状態を決めるパラメータです。なお、 parameter.general で tensor_load が設定されている場合には、そちらが優先され、テンソルをファイルから読み込みます。

- initial
  - "ferro"
    - 強磁性状態。 各サイトで $S^z = S$ となる状態。
  - "antiferro"
    - 反強磁性状態。正方格子、蜂の巣格子では $S^z = S$ と $S^z = -S$ が互いに並んだ Neel 秩序。三角格子、かごめ格子では スピンが $(\theta, \phi) = (0, 0), (2\pi/3, 0), (2\pi/3, \pi)$ 方向に向いた 120 度秩序。
  - "random"
    - 各サイトバラバラなランダム状態。
- noise
  - テンソルの要素に付与されるゆらぎの大きさ。

正方格子 square lattice

正方格子 type = "square lattice" では、 サイトが $(1, 0)$ 方向に L 個、 $(0, 1)$ 方向に W 個並びます。具体例として、 L=3, W=3 のときのサイトの並びを 図 5.1 (a) に示します。また、最近接、次近接、三次近接のボンドタイプの定義を図 5.1 (b), (c), (d) にそれぞれ示します。青線は bondtype = 0 のボンドを, 赤線は bondtype = 1 のボンドを表します。

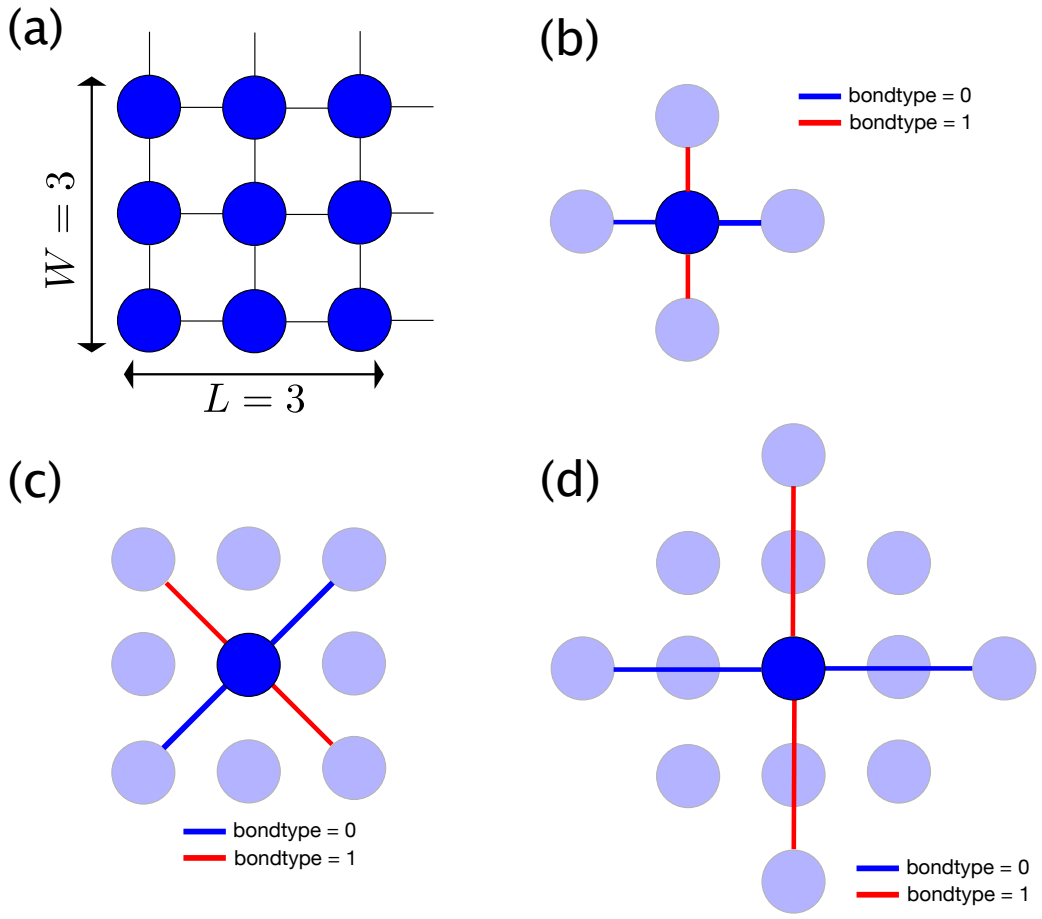


図 5.1　正方格子 (square) のサイト・ボンド。(a) `L=3`, `W=3` としたときのサイトの並び。(b) 最近接ボンド。`bondtype=0` (青) は 0 度方向に、`bondtype=1` (赤) は 90 度方向に伸びる。(c) 次近接ボンド。`bondtype=0` (青) は 45 度方向に、`bondtype=1` (赤) は -45 度方向に伸びる。(d) 三次近接ボンド。`bondtype=0` (青) は 0 度方向に、`bondtype=1` (赤) は 90 度方向に伸びる。

三角格子 triangular lattice

三角格子 `type = "triangular lattice"` では、サイトが $(1, 0)$ 方向に `L` 個、$(1/2, \sqrt{3}/2)$ 方向に `W` 個並びます。具体例として、`L=3`, `W=3` のときのサイトの並びを 図 5.2 (a) に示します。また、最近接、次近接、三次近接のボンドタイプの定義を図 5.2 (b), (c), (d) にそれぞれ示します。青線は `bondtype = 0` のボンドを、赤線は `bondtype = 1` のボンドを、緑線は `bondtype = 2` のボンドを、それぞれ表します。

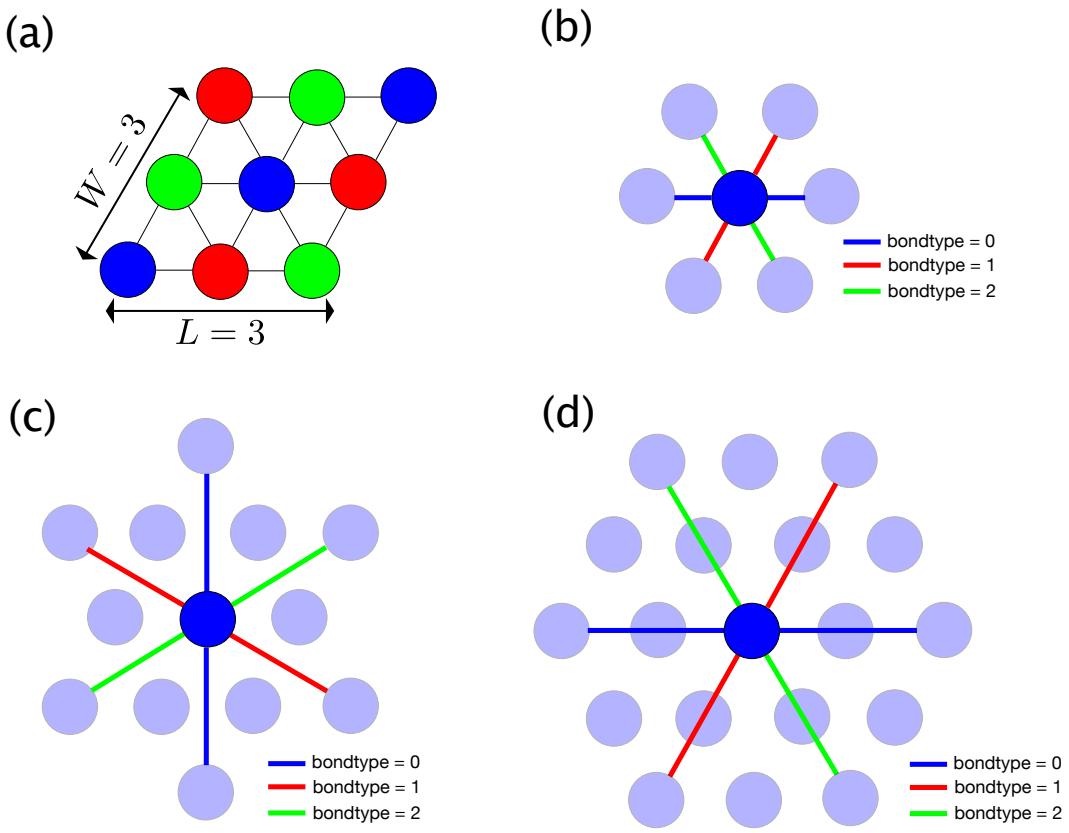


図 5.2　三角格子 (triangular) のサイト・ボンド。(a) `L=3`, `W=3` としたときのサイトの並び。(b) 最近接ボンド。 `bondtype=0` (青) は 0 度方向に、 `bondtype=1` (赤) は 60 度方向に、 `bondtype=2` (緑) は 120 度方向にそれぞれ伸びる。(c) 次近接ボンド。 `bondtype=0` (青) は 90 度方向に、 `bondtype=1` (赤) は -30 度方向に、 `bondtype=2` (緑) は 30 度方向にそれぞれ伸びる。(d) 三次近接ボンド。 `bondtype=0` (青) は 0 度方向に、 `bondtype=1` (赤) は 60 度方向に、 `bondtype=2` (緑) は 120 度方向にそれぞれ伸びる。

蜂の巣格子 honeycomb lattice

蜂の巣格子 `type = "honeycomb lattice"` では、 座標 $(0,0)$ と $(\sqrt{3}/2, 1/2)$ の 2 つのサイトからなるユニットが、 $(\sqrt{3}, 0)$ 方向に `L` 個、 $(1/2, 3/2)$ 方向に `W` 個並びます。具体例として、`L=3`, `W=3` のときのサイトの並びを 図 5.3 (a) に示します。破線はユニットを表します。また、最近接、次近接、三次近接のボンドタイプの定義を図 5.3 (b), (c), (d) にそれぞれ示します。青線は `bondtype = 0` のボンドを、 赤線は `bondtype = 1` のボンドを、 緑線は `bondtype = 2` のボンドを、それぞれ表します。

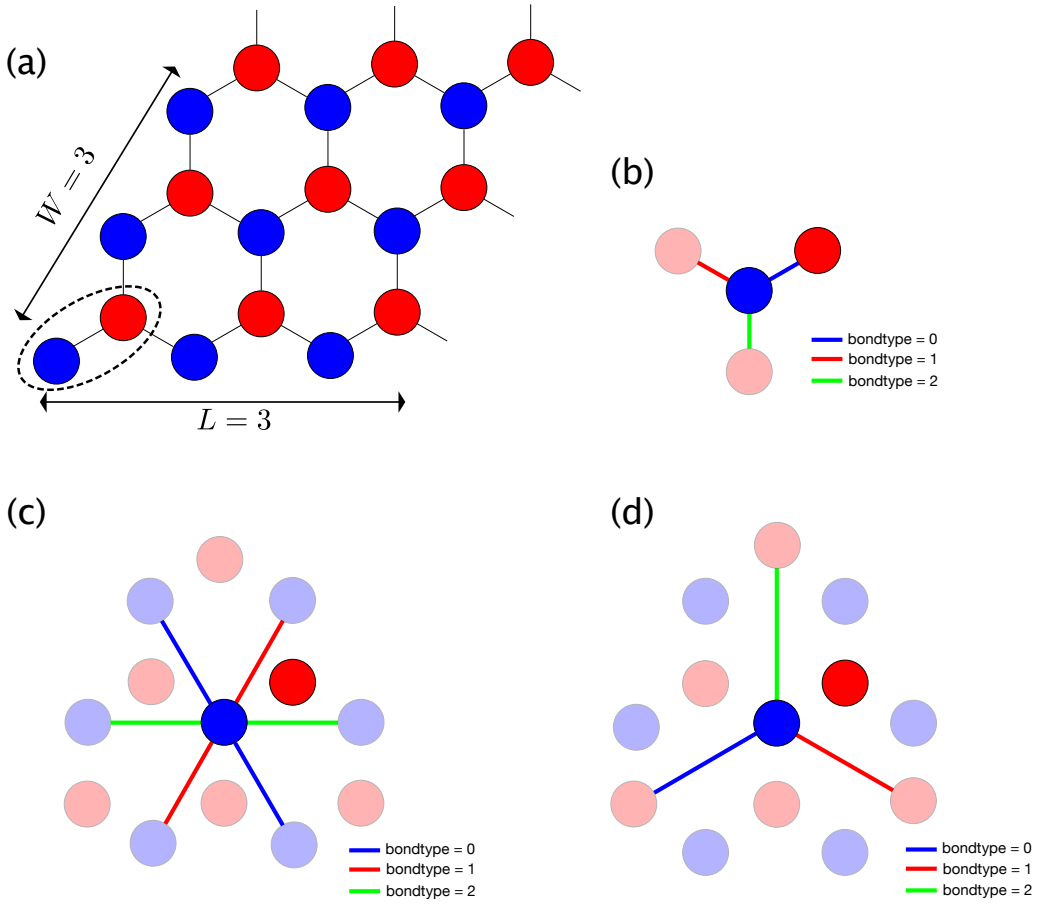


図 5.3　蜂の巣格子 (honeycomb) のサイト・ボンド。(a) `L=3, W=3` としたときのサイトの並び。 破線で表されるユニットが `L` かける `W` 個並ぶ。(b) 最近接ボンド。 `bondtype=0` (青) は 30 度方向に、 `bondtype=1` (赤) は 150 度方向に、 `bondtype=2` (緑) は -90 度方向にそれぞれ伸びる。(c) 次近接ボンド。 `bondtype=0` (青) は 120 度方向に、 `bondtype=1` (赤) は 60 度方向に、 `bondtype=2` (緑) は 0 度方向にそれぞれ伸びる。(d) 三次近接ボンド。 `bondtype=0` (青) は -30 度方向に、 `bondtype=1` (赤) は -150 度方向に、 `bondtype=2` (緑) は 90 度方向にそれぞれ伸びる。

かごめ格子 kagome lattice

かごめ格子 `type = "kagome lattice"` では、 座標 $(0,0)$, $(1,0)$, $(1/2,\sqrt{3}/2)$ の 3 つのサイトからなるユニット（上向き三角）が、 $(2,0)$ 方向に `L` 個、 $(1,\sqrt{3})$ 方向に `W` 個並びます。具体例として、`L=3, W=3` のときのサイトの並びを 図 5.4 (a) に示します。破線はユニットは破線を表します。また、最近接、次近接、三次近接のボンドタイプの定義を図 5.4 (b), (c), (d) にそれぞれ示します。青線は `bondtype = 0` のボンドを、 赤線は `bondtype = 1` のボンドを表します。

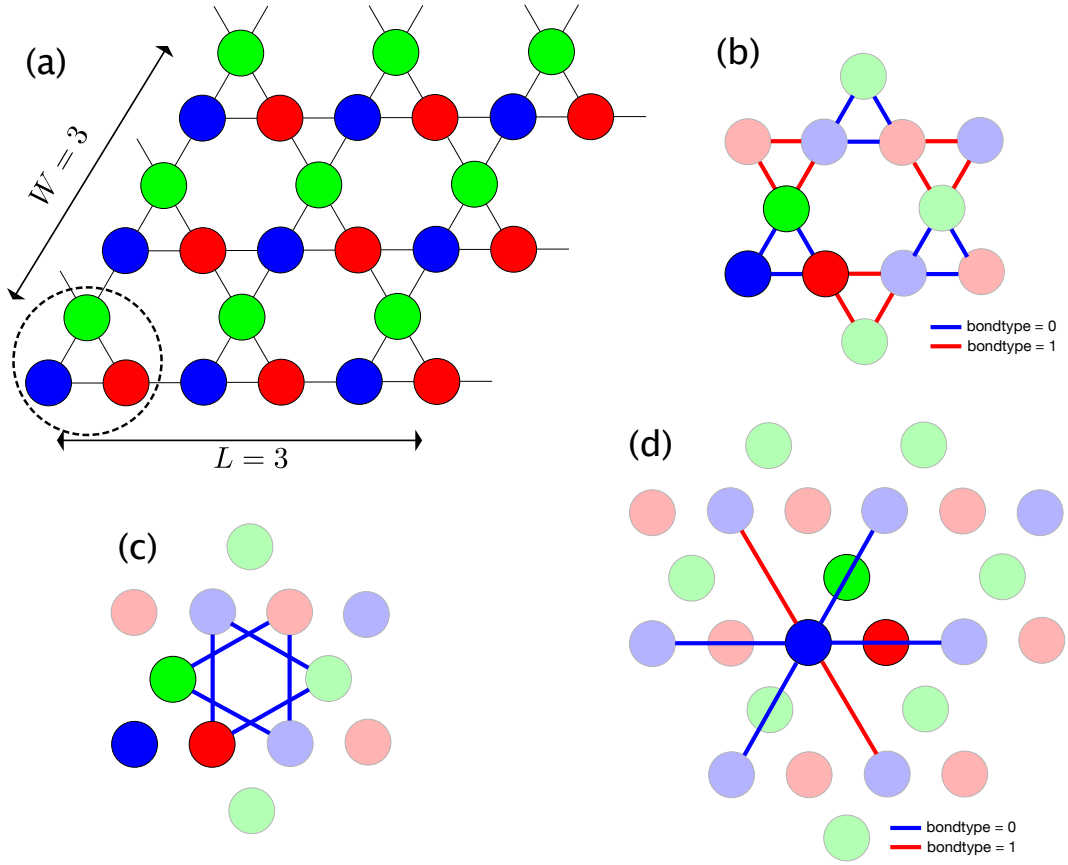


図 5.4　かごめ格子 (kagome) のサイト・ボンド。(a) `L=3`, `W=3` としたときのサイトの並び。 破線で表されるユニットが `L` かける `W` 個並ぶ。(b) 最近接ボンド。 `bondtype=0` (青) は 上向き三角形を、 `bondtype=1` (赤) は 下向き三角形を作る。(c) 次近接ボンド。(d) 三次近接ボンド。 `bondtype=0` (青) はサイトを横切り、 `bondtype=1` (赤) は横切らない。

5.2.3 `parameter` セクション

`tenes_simple` では使われず、 `tenes_std` の入力ファイルとしてそのままコピーされます。

更新回数など、 計算にあらわれる種々のパラメータを記述します。サブセクションとして `general`, `simple_update`, `full_update`, `ctm`, `random` を持ちます。

simple update および full update の虚時間刻み `parameter.simple_update.tau` と `parameter.full_update.tau` のみ、 `tenes` 本体ではなくスタンダードモード `tenes_std` で使われるパラメータです。

parameter.general

tenes の全般的な設定パラメータ

名前	説明	型	デフォルト
is_real	すべてのテンソルを実数に制限するかどうか	真偽値	false
iszero_tol	演算子テンソルの読み込みにおいてゼロとみなす絶対値カットオフ	実数	0.0
measure	物理量測定をするかどうか	真偽値	true
output	物理量などを書き込むディレクトリ	文字列	"output"
tensor_save	最適化後のテンソルを書き込むディレクトリ	文字列	""
tensor_load	初期テンソルを読み込むディレクトリ	文字列	""

- is_real
  - true にするとテンソルの要素を実数に制限して計算を行います
  - 一つでも複素演算子があるとエラー終了します
- iszero_tol
  - 各種演算子テンソル要素の実部・虚部の読み込みにおいて、絶対値が iszero_tol 以下はゼロとみなします
- measure
  - false にすると物理量計算・保存をスキップします
  - 実行時間 time.dat は常に保存されます
- output
  - 物理量などの計算結果をこのディレクトリ以下に保存します
  - 空文字列の場合はカレントディレクトリに保存します
- tensor_save
  - 最適化後のテンソルをこのディレクトリ以下に保存します
  - 空文字列の場合は保存しません
- tensor_load
  - 各種テンソルをこのディレクトリ以下から読み込みます
  - 空文字列の場合は読み込みません

parameter.simple_update

simple update に関するパラメータ

名前	説明	型	デフォルト
tau	虚時間発展演算子における虚時間刻み τ	実数	0.01
num_step	simple update の回数	整数	0
lambda_cutoff	simple update において平均場 λ の切り捨て閾値	実数	1e-12

parameter.full_update

full update に関するパラメータ

名前	説明	型	デフォルト
tau	虚時間発展演算子における虚時間刻み τ	実数	0.01
num_step	full update の回数	整数	0
env_cutoff	full update で環境テンソルを計算する際にゼロとみなす特異値の cutoff	実数	1e-12
inverse_precision	full update で擬似逆行列を計算する際にゼロとみなす特異値の cutoff	実数	1e-12
convergence_epsilon	full update で truncation の最適化を行う際の収束判定値	実数	1e-6
iteration_max	full update で truncation の最適化を行う際の iteration の最大回数	整数	100
gauge_fix	テンソルのゲージを固定するかどうか	真偽値	true
fastfullupdate	Fast full update にするかどうか	真偽値	true

parameter.ctm

角転送行列 (CTM) に関するパラメータ

名前	説明	型	デフォルト
dimension	CTM のボンド次元 χ	整数	4
projector_cutoff	CTM の projector を計算する際にゼロとみなす特異値の cutoff	実数	1e-12
convergence_epsilon	CTM の収束判定値	実数	1e-6
iteration_max	CTM の収束 iteration の最大回数	整数	100
projector_corner	CTM の projector 計算で 1/4 角のテンソルのみを使う	真偽値	true
use_rsvd	SVD を 乱択 SVD で置き換えるかどうか	真偽値	false
rsvd_oversampling_factor	乱択 SVD 中に計算する特異値の数の、最終的に用いる数に対する比率	実数	2.0

乱拓 SVD を用いたテンソル繰り込み群の手法については、 S. Morita, R. Igarashi, H.-H. Zhao, and N. Kawashima, Phys. Rev. E 97, 033310 (2018) を参照してください。

parameter.random

疑似乱数生成器に関するパラメータ

名前	説明	型	デフォルト
seed	テンソルの初期化や乱択 SVD に用いる疑似乱数生成器のシード	整数	11

MPI 並列において、各プロセスは seed にプロセス番号を足した数を実際のシードとして持ちます。

例

```
[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 セクション

tenes_simple では相関関数 C = <A(0)B(r)> はデフォルトでは計算されません。相関関数を計算したい場合は、 tenes の入力ファイルで指定する correlation セクションと共通のフォーマットで指定することができます。詳細は、*tenes の入力ファイル* の correlation セクションをご覧ください。

5.3 tenes_std の入力ファイル

- ファイルフォーマットは TOML 形式
- parameter, tensor, hamiltonian, observable, correlation の 5 つのセクションを持ちます。
  - hamiltonian 以外の 4 つは以下に挙げる例外を除き、tenes の入力ファイルフォーマットと同一であり、そのまま tenes の入力ファイルとしてコピーされます。
  - parameter.simple_update.tau および parameter.full_update.tau に実数を渡すことで、虚時間発展演算子における虚時間刻みを指定できます。

5.3.1 parameter セクション

更新回数など、 計算にあらわれる種々のパラメータを記述します。サブセクションとして general, simple_update, full_update, ctm, random を持ちます。

simple update および full update の虚時間刻み parameter.simple_update.tau と parameter.full_update.tau のみ、 tenes 本体ではなくスタンダードモード tenes_std で使われるパラメータです。

parameter.general

tenes の全般的な設定パラメータ

名前	説明	型	デフォルト
is_real	すべてのテンソルを実数に制限するかどうか	真偽値	false
iszero_tol	演算子テンソルの読み込みにおいてゼロとみなす絶対値カットオフ	実数	0.0
measure	物理量測定をするかどうか	真偽値	true
output	物理量などを書き込むディレクトリ	文字列	"output"
tensor_save	最適化後のテンソルを書き込むディレクトリ	文字列	""
tensor_load	初期テンソルを読み込むディレクトリ	文字列	""

- is_real

  - `true` にするとテンソルの要素を実数に制限して計算を行います
  - 一つでも複素演算子があるとエラー終了します
- `iszero_tol`
  - 各種演算子テンソル要素の実部・虚部の読み込みにおいて、絶対値が `iszero_tol` 以下はゼロとみなします
- `measure`
  - `false` にすると物理量計算・保存をスキップします
  - 実行時間 `time.dat` は常に保存されます
- `output`
  - 物理量などの計算結果をこのディレクトリ以下に保存します
  - 空文字列の場合はカレントディレクトリに保存します
- `tensor_save`
  - 最適化後のテンソルをこのディレクトリ以下に保存します
  - 空文字列の場合は保存しません
- `tensor_load`
  - 各種テンソルをこのディレクトリ以下から読み込みます
  - 空文字列の場合は読み込みません

parameter.simple_update

simple update に関するパラメータ

名前	説明	型	デフォルト
`tau`	虚時間発展演算子における虚時間刻み τ	実数	0.01
`num_step`	simple update の回数	整数	0
`lambda_cutoff`	simple update において平均場 λ の切り捨て閾値	実数	1e-12

parameter.full_update

full update に関するパラメータ

名前	説明	型	デフォルト
tau	虚時間発展演算子における虚時間刻み τ	実数	0.01
num_step	full update の回数	整数	0
env_cutoff	full update で環境テンソルを計算する際にゼロとみなす特異値の cutoff	実数	1e-12
inverse_precision	full update で擬似逆行列を計算する際にゼロとみなす特異値の cutoff	実数	1e-12
convergence_epsilon	full update で truncation の最適化を行う際の収束判定値	実数	1e-6
iteration_max	full update で truncation の最適化を行う際の iteration の最大回数	整数	100
gauge_fix	テンソルのゲージを固定するかどうか	真偽値	true
fastfullupdate	Fast full update にするかどうか	真偽値	true

parameter.ctm

角転送行列 (CTM) に関するパラメータ

名前	説明	型	デフォルト
dimension	CTM のボンド次元 χ	整数	4
projector_cutoff	CTM の projector を計算する際にゼロとみなす特異値の cutoff	実数	1e-12
convergence_epsilon	CTM の収束判定値	実数	1e-6
iteration_max	CTM の収束 iteration の最大回数	整数	100
projector_corner	CTM の projector 計算で 1/4 角のテンソルのみを使う	真偽値	true
use_rsvd	SVD を 乱択 SVD で置き換えるかどうか	真偽値	false
rsvd_oversampling_factor	乱択 SVD 中に計算する特異値の数の、最終的に用いる数に対する比率	実数	2.0

乱拓 SVD を用いたテンソル繰り込み群の手法については、 S. Morita, R. Igarashi, H.-H. Zhao, and N. Kawashima, Phys. Rev. E 97, 033310 (2018) を参照してください。

parameter.random

疑似乱数生成器に関するパラメータ

名前	説明	型	デフォルト
seed	テンソルの初期化や乱択 SVD に用いる疑似乱数生成器のシード	整数	11

MPI 並列において、各プロセスは seed にプロセス番号を足した数を実際のシードとして持ちます。

例

```
[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 セクション

「ユニットセル」の情報を記述します (ボンドの情報は hamiltonian (tenes_std) や evolution (tenes) で与えます)。ユニットセルは Lx かける Ly の大きさをもつ長方形の形をしています。また、サブセクション unitcell を持ちます。

名前	説明	型	デフォルト
L_sub	ユニットセルの大きさ	整数または整数のリスト	--
skew	skew 境界条件におけるシフト値	整数	0

L_sub として 2 つの整数からなるリストを渡した場合、はじめの要素が Lx に、もう片方が Ly になります。3 つ以上の要素からなるリストを渡した場合にはエラー終了します。L_sub として整数を渡した場合、 Lx と Ly とが等しくなります。

ユニットセル内のサイトは 0 から順番に番号付けされます。 x 方向から順に並びます。

skew は y 方向にユニットセル 1 つ分動いたときの x 方向のズレです。

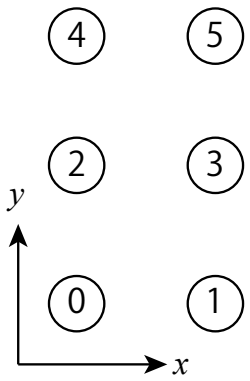


図 5.5 `L_sub = [2,3]` としたときの例

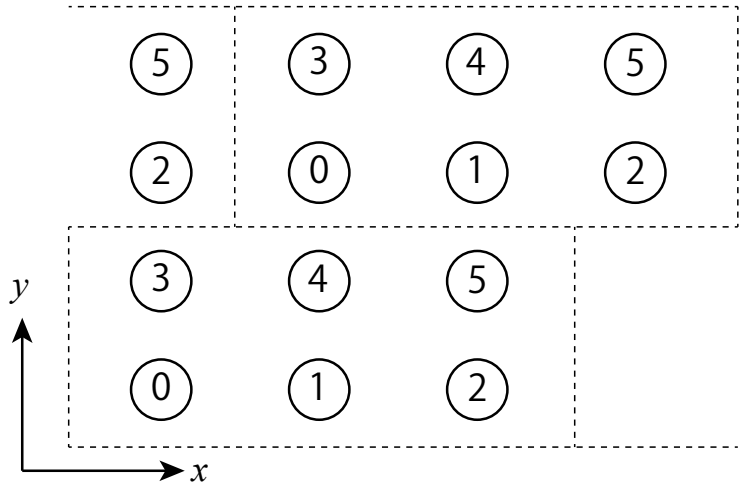


図 5.6 `L_sub = [3,2], skew = 1` としたときの例 (罫線はユニットセルの区切り)

`tensor.unitcell` サブセクション

サイトテンソル $T^{(n)}_{ijkl\alpha}$ の情報を指定します。ここで i,j,k,l は virtual bond のインデックス、 α は physical bond のインデックス、 n はサイト番号を意味します。

名前	説明	型
`index`	サイト番号	整数 or 整数のリスト
`physical_dim`	サイトテンソルの physical bond の次元	整数
`virtual_dim`	サイトテンソルの virtual bond の次元 D	整数 or 整数のリスト
`initial_state`	初期状態	実数のリスト
`noise`	初期テンソルのゆらぎの大きさ	実数

index にリストを渡すことによって、複数のサイトを同時に指定できます。空のサイト [] は全サイトを意味します。

virtual_dim にリストを渡すことで、4 方向のボンド次元を個別に指定できます。順番は、左 (-x)、上 (+y)、右 (+x)、下 (-y) の順番です。

系全体の初期状態 $|\Psi\rangle$ は、各サイト i の初期状態 $|\Psi_i\rangle$ の直積で与えられます。

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

サイトテンソルはこの直積状態を表現するように初期化されます。initial_state では各サイト i の初期状態 $|\Psi_i\rangle = \sum_\alpha A_\alpha |\alpha\rangle_i$ における実展開係数 A_α の値を指定します。係数は自動的に規格化されます。テンソル自体は、すべての virtual ボンドインデックスが 0 である要素が、$T_{0000\alpha} = A_\alpha$ のように初期化されます。他の要素には [-noise, noise) の一様乱数が互いに独立に入力されます。たとえば、$S = 1/2$ のとき、S^z 方向に向いた状態 $|\Psi_i\rangle = |\uparrow\rangle = |0\rangle$ を初期値にしたい場合には initial_state = [1.0, 0.0] に、S^x 方向に向いた状態 $|\Psi_i\rangle = (|\uparrow\rangle + |\downarrow\rangle)/\sqrt{2}$ を初期値にしたい場合には initial_state = [1.0, 1.0] とします。

initial_state にゼロのみからなる配列を渡した場合、テンソルのすべての要素が独立に [-noise, noise) で乱数初期化されます。

5.3.3 observable セクション

物理量測定に関する諸々を記述します。onesite と twosite の 2 種類のサブセクションを持ちます。

observable.onesite

ひとつのサイト上で定義される物理量を示す一体演算子を定義します。

名前	説明	型
name	演算子の名前	文字列
group	演算子の識別番号	整数
sites	サイト番号	整数 or 整数のリスト
dim	演算子の次元	整数
elements	演算子の非ゼロ要素	文字列

name は演算子の名前です。

group は onesite 演算子の識別番号です。

sites は演算子が作用するサイト番号です。リストを渡すことで複数同時に定義できます。空リスト [] は全サイトを意味します。

dim は演算子の次元です。

elements は演算子の非ゼロ要素を指定する文字列です。1 つの要素は、空白で区切られた 2 つの整数と 2 つの浮動小数点数からなる 1 つの行で表されます。

- 最初の 2 つはそれぞれ演算子が作用する前と後の状態番号を示します。
- あとの 2 つはそれぞれ演算子の要素の実部と虚部を示します。

例

S=1/2 の Sz 演算子

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

を具体例として説明します。

まず、名前は `name = "Sz"` として、識別番号は `group = 0` としておきます。

次に、演算子の作用するサイトですが、すべてのサイトで同一の演算子を用いる場合には `sites = []` とします。そうではない場合、例えばスピンの大きさが異なるサイトがある場合には、`sites = [0,1]` などと具体的なサイト番号を指定します。

演算子の次元は、上に示した行列表示のサイズなので、 `dim = 2` です。

最後に演算子の要素です。非ゼロ要素について、そのインデックス（ゼロ始まり）と要素を順番に並べれば良いので、

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

となります。

結果として、 S=1/2 の Sz 演算子は次のように定義されます。

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

observable.twosite

ふたつのサイト上で定義される物理量を示す演算子を定義します。

名前	説明	型
name	演算子の名前	文字列
group	演算子の識別番号	整数
bonds	ボンド	文字列
dim	演算子の次元	整数のリスト
elements	演算子の非ゼロ要素	文字列
ops	onesite 演算子の識別番号	整数のリスト

name は演算子の名前です。

group は twosites 演算子の識別番号です。

bonds は演算子が作用するサイト対の集合を表す文字列です。3 つの整数からなる 1 行が 1 つのサイト対を意味します。

- 最初の整数は 始点サイト (source) の番号です。
- あとの 2 つの整数は source site からみた終点サイト (target) の座標 (dx, dy) です。
  - dx, dy ともに $-3 \leq dx \leq 3$ の範囲に収まる必要があります。

dim は演算子の次元、すなわち作用するサイトの取りうる状態数です。例として、2 つの $S = 1/2$ スピンの相互作用の場合は、 `dim = [2,2]` です。

elements は演算子の非ゼロ要素を指定する文字列です。1 つの要素は 4 つの整数と 2 つの浮動小数点数を空白区切りからなる 1 つの行からなります。

- 最初の 2 つは演算子が作用する **前** の source site, target site の状態番号を示します。
- つぎの 2 つは演算子が作用した **後** の source site, target site の状態番号を示します。
- 最後の 2 つはそれぞれ演算子の要素の実部と虚部を示します。

ops を使うと observable.onesite で定義した 1 体演算子の直積として 2 体演算子を定義できます。例えば observable.onesite の group=0 として S^z を定義していた場合には、`ops = [0,0]` として $S_i^z S_j^z$ を表現できます。

elements と ops を同時に定義した場合にはエラー終了します。

例

ここでは具体例として、Lsub=[2,2] の正方格子 S=1/2 ハイゼンベルグ模型のボンドハミルトニアンのエネルギーを求めるため、ハミルトニアン

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

を 2 体演算子として設定する例を説明します。

まず、名前と識別番号はそれぞれ name = "hamiltonian" と group = 0 としておきます。それぞれのサイトの状態は $|\uparrow\rangle$ と $|\downarrow\rangle$ の 2 状態の重ね合わせとなるため、次元は 2 となり、dim = [2,2] となります。

次にボンドです。サイトは 図 5.7 のように並んでいます。0 番と 1 番をつなぐボンドは、 1 番は 0 番から見て (1,0) の位置にあるので 0 1 0 と表現されます。同様に 1 番と 3 番をつなぐボンドは、 3 番が 1 番から見て (0,1) の位置にあるので 1 0 1 と表現されます。

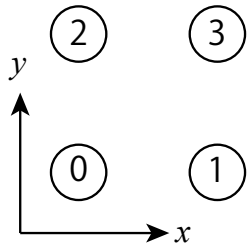


図 5.7　Lsub=[2,2] の正方格子 S=1/2 ハイゼンベルグ模型のサイトの並び順

最後に演算子の要素です。まずはサイトの基底を番号付ける必要がありますが、ここでは $|\uparrow\rangle$ を 0, $|\downarrow\rangle$ を 1 とします。この基底と番号を用いると、例えば対角項の 1 つ $\langle\uparrow_i\uparrow_j|\mathcal{H}_{ij}|\uparrow_i\uparrow_j\rangle = 1/4$ は 0 0 0 0 0.25 0.0 と表現されます。他に、非対角項の 1 つ $\langle\uparrow_i\downarrow_j|\mathcal{H}_{ij}|\downarrow_i\uparrow_j\rangle = 1/2$ は 1 0 0 1 0.5 0.0 と表現されます。

結果として、 S=1/2 のハイゼンベルグハミルトニアンは次のように定義されます。

```
[[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
"""
```

(次のページに続く)

(前のページからの続き)

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

5.3.4 hamiltonian セクション

ハミルトニアン全体をボンドハミルトニアン (2 サイトハミルトニアン) の和

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

であると捉えて、個々のボンドハミルトニアンを定義します。定義のやりかたは `observable.twosite` でなされる 2 サイト演算子と同様です。

名前	説明	型
`bonds`	ボンド	文字列
`dim`	演算子の次元	整数のリスト
`elements`	演算子の非ゼロ要素	文字列

`bonds` は演算子が作用するサイト対の集合を表す文字列です。3 つの整数からなる 1 行が 1 つのサイト対を意味します。最初の整数は 始点サイト (source) の番号です。あとの 2 つの整数は source site からみた終点サイト (target) の座標 (dx, dy) です。

`dim` は演算子の次元、すなわち作用するサイトの取りうる状態数です。例として、2 つの $S = 1/2$ スピンの相互作用の場合は、 `dim = [2,2]` です。

`elements` は演算子の非ゼロ要素を指定する文字列です。1 つの要素は 4 つの整数と 2 つの浮動小数点数を空白区切りからなる 1 つの行からなります。

- 最初の 2 つは演算子が作用する **前** の source site, target site の状態番号を示します。
- つぎの 2 つは演算子が作用した **後** の source site, target site の状態番号を示します。
- 最後の 2 つはそれぞれ演算子の要素の実部と虚部を示します。

5.3.5 correlation セクション

サイト演算子の相関関数 $C = \langle A(\boldsymbol{r}_0)B(\boldsymbol{r}_0 + \boldsymbol{r})\rangle$ に関する情報を指定するセクションです。本セクションを省略した場合、相関関数は計算されません。

座標は正方格子 TNS の座標系で測られます。すなわち、右隣のテンソルは $\boldsymbol{r} = (1, 0)$ で、真上は $\boldsymbol{r} = (0, 1)$ です。中心座標 $\boldsymbol{r}_0$ として、ユニットセル内のすべてのサイトが用いられます。また、$\boldsymbol{r}$ は x ないし y 軸に平行な方向に、正の向きにのみ動きます。すなわち、

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

です。

名前	説明	型
r_max	相関関数の距離 r の最大値	整数
operators	相関関数を測る 1 体演算子 A, B を表す番号	整数のリストのリスト

演算子は observable.onesite セクションで指定したものが用いられます。

例

例えば S^z が 0 番で、 S^x が 1 番として定義されている場合、

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

では相関関数 $S^z(0)S^z(r), S^z(0)S^x(r), S^x(0)S^x(r)$ が、 $0 \leq r \leq 5$ の範囲で測定されます。

5.4 tenes の入力ファイル

- ファイルフォーマットは TOML 形式
- parameter, tensor, evolution, observable, correlation の 5 つのセクションを持ちます。

5.4.1 parameter セクション

更新回数など、 計算にあらわれる種々のパラメータを記述します。サブセクションとして general, simple_update, full_update, ctm, random を持ちます。

simple update および full update の虚時間刻み parameter.simple_update.tau と parameter.full_update.tau のみ、 tenes 本体ではなくスタンダードモード tenes_std で使われるパラメータです。

parameter.general

tenes の全般的な設定パラメータ

名前	説明	型	デフォルト
is_real	すべてのテンソルを実数に制限するかどうか	真偽値	false
iszero_tol	演算子テンソルの読み込みにおいてゼロとみなす絶対値カットオフ	実数	0.0
measure	物理量測定をするかどうか	真偽値	true
output	物理量などを書き込むディレクトリ	文字列	"output"
tensor_save	最適化後のテンソルを書き込むディレクトリ	文字列	""
tensor_load	初期テンソルを読み込むディレクトリ	文字列	""

- is_real
  - true にするとテンソルの要素を実数に制限して計算を行います
  - 一つでも複素演算子があるとエラー終了します
- iszero_tol
  - 各種演算子テンソル要素の実部・虚部の読み込みにおいて、絶対値が iszero_tol 以下はゼロとみなします
- measure
  - false にすると物理量計算・保存をスキップします
  - 実行時間 time.dat は常に保存されます
- output
  - 物理量などの計算結果をこのディレクトリ以下に保存します
  - 空文字列の場合はカレントディレクトリに保存します
- tensor_save

  - 最適化後のテンソルをこのディレクトリ以下に保存します
  - 空文字列の場合は保存しません
- `tensor_load`
  - 各種テンソルをこのディレクトリ以下から読み込みます
  - 空文字列の場合は読み込みません

`parameter.simple_update`

simple update に関するパラメータ

名前	説明	型	デフォルト
`tau`	虚時間発展演算子における虚時間刻み τ	実数	0.01
`num_step`	simple update の回数	整数	0
`lambda_cutoff`	simple update において平均場 λ の切り捨て閾値	実数	1e-12

`parameter.full_update`

full update に関するパラメータ

名前	説明	型	デフォルト
`tau`	虚時間発展演算子における虚時間刻み τ	実数	0.01
`num_step`	full update の回数	整数	0
`env_cutoff`	full update で環境テンソルを計算する際にゼロとみなす特異値の cutoff	実数	1e-12
`inverse_precision`	full update で擬似逆行列を計算する際にゼロとみなす特異値の cutoff	実数	1e-12
`convergence_epsilon`	full update で truncation の最適化を行う際の収束判定値	実数	1e-6
`iteration_max`	full update で truncation の最適化を行う際の iteration の最大回数	整数	100
`gauge_fix`	テンソルのゲージを固定するかどうか	真偽値	true
`fastfullupdate`	Fast full update にするかどうか	真偽値	true

parameter.ctm

角転送行列 (CTM) に関するパラメータ

名前	説明	型	デフォルト
dimension	CTM のボンド次元 χ	整数	4
projector_cutoff	CTM の projector を計算する際にゼロとみなす特異値の cutoff	実数	1e-12
convergence_epsilon	CTM の収束判定値	実数	1e-6
iteration_max	CTM の収束 iteration の最大回数	整数	100
projector_corner	CTM の projector 計算で 1/4 角のテンソルのみを使う	真偽値	true
use_rsvd	SVD を 乱択 SVD で置き換えるかどうか	真偽値	false
rsvd_oversampling_factor	乱択 SVD 中に計算する特異値の数の、最終的に用いる数に対する比率	実数	2.0

乱拓 SVD を用いたテンソル繰り込み群の手法については、 S. Morita, R. Igarashi, H.-H. Zhao, and N. Kawashima, Phys. Rev. E 97, 033310 (2018) を参照してください。

parameter.random

疑似乱数生成器に関するパラメータ

名前	説明	型	デフォルト
seed	テンソルの初期化や乱択 SVD に用いる疑似乱数生成器のシード	整数	11

MPI 並列において、各プロセスは seed にプロセス番号を足した数を実際のシードとして持ちます。

例

```
[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
```

(次のページに続く)

(前のページからの続き)

```
[parameter.ctm]
iteration_max = 10
dimension = 9 # CHI
```

5.4.2 tensor セクション

「ユニットセル」の情報を記述します (ボンドの情報は hamiltonian (tenes_std) や evolution (tenes) で与えます)。ユニットセルは Lx かける Ly の大きさをもつ長方形の形をしています。また、サブセクション unitcell を持ちます。

名前	説明	型	デフォルト
L_sub	ユニットセルの大きさ	整数または整数のリスト	--
skew	skew 境界条件におけるシフト値	整数	0

L_sub として 2 つの整数からなるリストを渡した場合、はじめの要素が Lx に、もう片方が Ly になります。3 つ以上の要素からなるリストを渡した場合にはエラー終了します。L_sub として整数を渡した場合、 Lx と Ly とが等しくなります。

ユニットセル内のサイトは 0 から順番に番号付けされます。 x 方向から順に並びます。

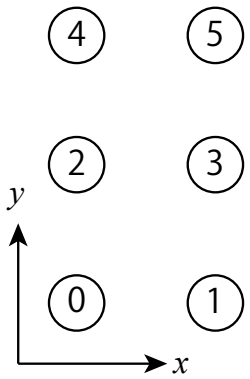


図 5.8 L_sub = [2,3] としたときの例

skew は y 方向にユニットセル 1 つ分動いたときの x 方向のズレです。

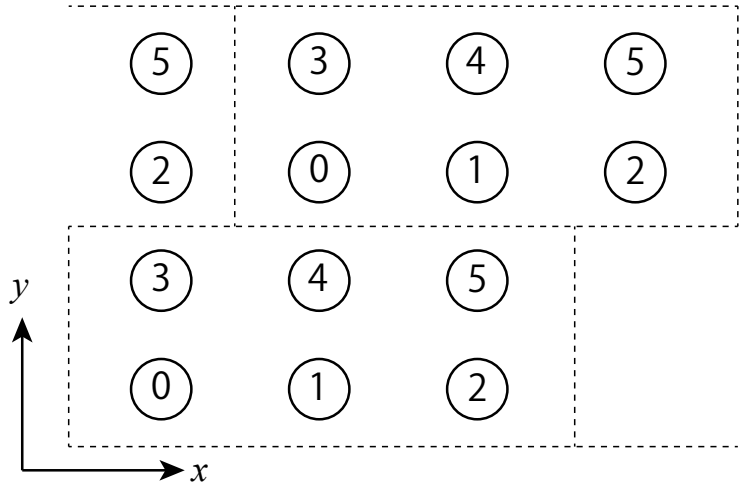


図 5.9 `L_sub = [3,2], skew = 1` としたときの例 (罫線はユニットセルの区切り)

`tensor.unitcell` サブセクション

サイトテンソル $T^{(n)}_{ijkl\alpha}$ の情報を指定します。ここで i,j,k,l は virtual bond のインデックス、 α は physical bond のインデックス、 n はサイト番号を意味します。

名前	説明	型
`index`	サイト番号	整数 or 整数のリスト
`physical_dim`	サイトテンソルの physical bond の次元	整数
`virtual_dim`	サイトテンソルの virtual bond の次元 D	整数 or 整数のリスト
`initial_state`	初期状態	実数のリスト
`noise`	初期テンソルのゆらぎの大きさ	実数

`index` にリストを渡すことによって、複数のサイトを同時に指定できます。空のサイト `[]` は全サイトを意味します。

`virtual_dim` にリストを渡すことで、4 方向のボンド次元を個別に指定できます。順番は、左 (-x)、上 (+y)、右 (+x)、下 (-y) の順番です。

系全体の初期状態 $|\Psi\rangle$ は、各サイト i の初期状態 $|\Psi_i\rangle$ の直積で与えられます。

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

サイトテンソルはこの直積状態を表現するように初期化されます。`initial_state` では各サイト i の初期状態 $|\Psi_i\rangle = \sum_\alpha A_\alpha |\alpha\rangle_i$ における実展開係数 A_α の値を指定します。係数は自動的に規格化されます。テンソル自体は、 すべての virtual ボンドインデックスが 0 である要素が、 $T_{0000\alpha} = A_\alpha$ のように初期化されます。他の要素に

は `[-noise, noise)` の一様乱数が互いに独立に入力されます。たとえば、$S = 1/2$ のとき、S^z 方向に向いた状態 $|\Psi_i\rangle = |\uparrow\rangle = |0\rangle$ を初期値にしたい場合には `initial_state = [1.0, 0.0]` に、S^x 方向に向いた状態 $|\Psi_i\rangle = (|\uparrow\rangle + |\downarrow\rangle)/\sqrt{2}$ を初期値にしたい場合には `initial_state = [1.0, 1.0]` とします。

`initial_state` にゼロのみからなる配列を渡した場合、テンソルのすべての要素が独立に `[-noise, noise)` で乱数初期化されます。

5.4.3 observable セクション

物理量測定に関する諸々を記述します。`onesite` と `twosite` の 2 種類のサブセクションを持ちます。

observable.onesite

ひとつのサイト上で定義される物理量を示す一体演算子を定義します。

名前	説明	型
`name`	演算子の名前	文字列
`group`	演算子の識別番号	整数
`sites`	サイト番号	整数 or 整数のリスト
`dim`	演算子の次元	整数
`elements`	演算子の非ゼロ要素	文字列

`name` は演算子の名前です。

`group` は onesite 演算子の識別番号です。

`sites` は演算子が作用するサイト番号です。リストを渡すことで複数同時に定義できます。空リスト `[]` は全サイトを意味します。

`dim` は演算子の次元です。

`elements` は演算子の非ゼロ要素を指定する文字列です。1 つの要素は、空白で区切られた 2 つの整数と 2 つの浮動小数点数からなる 1 つの行で表されます。

- 最初の 2 つはそれぞれ演算子が作用する前と後の状態番号を示します。
- あとの 2 つはそれぞれ演算子の要素の実部と虚部を示します。

例

S=1/2 の Sz 演算子

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

を具体例として説明します。

まず、名前は `name = "Sz"` として、識別番号は `group = 0` としておきます。

次に、演算子の作用するサイトですが、すべてのサイトで同一の演算子を用いる場合には `sites = []` とします。そうではない場合、例えばスピンの大きさが異なるサイトがある場合には、`sites = [0,1]` などと具体的なサイト番号を指定します。

演算子の次元は、上に示した行列表示のサイズなので、 `dim = 2` です。

最後に演算子の要素です。非ゼロ要素について、そのインデックス（ゼロ始まり）と要素を順番に並べれば良いので、

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

となります。

結果として、 S=1/2 の Sz 演算子は次のように定義されます。

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

observable.twosite

ふたつのサイト上で定義される物理量を示す演算子を定義します。

名前	説明	型
name	演算子の名前	文字列
group	演算子の識別番号	整数
bonds	ボンド	文字列
dim	演算子の次元	整数のリスト
elements	演算子の非ゼロ要素	文字列
ops	onesite 演算子の識別番号	整数のリスト

name は演算子の名前です。

group は twosites 演算子の識別番号です。

bonds は演算子が作用するサイト対の集合を表す文字列です。3 つの整数からなる 1 行が 1 つのサイト対を意味します。

- 最初の整数は 始点サイト (source) の番号です。
- あとの 2 つの整数は source site からみた終点サイト (target) の座標 (dx, dy) です。
  - dx, dy ともに $-3 \leq dx \leq 3$ の範囲に収まる必要があります。

dim は演算子の次元、すなわち作用するサイトの取りうる状態数です。例として、2 つの $S = 1/2$ スピンの相互作用の場合は、 `dim = [2,2]` です。

elements は演算子の非ゼロ要素を指定する文字列です。1 つの要素は 4 つの整数と 2 つの浮動小数点数を空白区切りからなる 1 つの行からなります。

- 最初の 2 つは演算子が作用する **前** の source site, target site の状態番号を示します。
- つぎの 2 つは演算子が作用した **後** の source site, target site の状態番号を示します。
- 最後の 2 つはそれぞれ演算子の要素の実部と虚部を示します。

ops を使うと observable.onesite で定義した 1 体演算子の直積として 2 体演算子を定義できます。例えば observable.onesite の group=0 として S^z を定義していた場合には、`ops = [0,0]` として $S_i^z S_j^z$ を表現できます。

elements と ops を同時に定義した場合にはエラー終了します。

例

ここでは具体例として、Lsub=[2,2] の正方格子 S=1/2 ハイゼンベルグ模型のボンドハミルトニアンのエネルギーを求めるため、ハミルトニアン

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

を 2 体演算子として設定する例を説明します。

まず、名前と識別番号はそれぞれ name = "hamiltonian" と group = 0 としておきます。それぞれのサイトの状態は $|\uparrow\rangle$ と $|\downarrow\rangle$ の 2 状態の重ね合わせとなるため、次元は 2 となり、dim = [2,2] となります。

次にボンドです。サイトは 図 5.7 のように並んでいます。0 番と 1 番をつなぐボンドは、 1 番は 0 番から見て (1,0) の位置にあるので 0 1 0 と表現されます。同様に 1 番と 3 番をつなぐボンドは、 3 番が 1 番から見て (0,1) の位置にあるので 1 0 1 と表現されます。

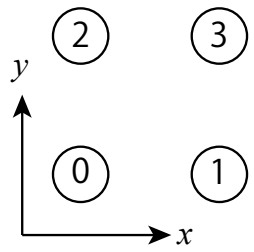


図 5.10　Lsub=[2,2] の正方格子 S=1/2 ハイゼンベルグ模型のサイトの並び順

最後に演算子の要素です。まずはサイトの基底を番号付ける必要がありますが、ここでは $|\uparrow\rangle$ を 0, $|\downarrow\rangle$ を 1 とします。この基底と番号を用いると、例えば対角項の 1 つ $\langle\uparrow_i\uparrow_j|\mathcal{H}_{ij}|\uparrow_i\uparrow_j\rangle = 1/4$ は 0 0 0 0 0.25 0.0 と表現されます。他に、非対角項の 1 つ $\langle\uparrow_i\downarrow_j|\mathcal{H}_{ij}|\downarrow_i\uparrow_j\rangle = 1/2$ は 1 0 0 1 0.5 0.0 と表現されます。

結果として、 S=1/2 のハイゼンベルグハミルトニアンは次のように定義されます。

```
[[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
"""
```

(次のページに続く)

(前のページからの続き)

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

5.4.4 evolution セクション

simple update, full update で使う 2 サイト虚時間発展演算子を記述します。次のようなフィールドを持つ simple, full の 2 つのサブセクションを持ちます。

名前	説明	型
source_site	source site の番号	整数
source_leg	source site から見た target site の方向	整数
dimensions	虚時間発展演算子テンソルの次元	整数のリスト
elements	虚時間発展演算子テンソルの非ゼロ要素	文字列

source_leg は 0 から 3 までの整数で指定します。-x 方向から順番に時計回りに、 0:-x, 1:+y, 2:+x, 3:-y として定義されています。

dimensions は observable の dim と異なり、すべての足の次元を指定する必要があります。足の順番は elements と同様に、 source_initial, target_initial, source_final, target_final の順番です。

例

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

5.4.5 correlation セクション

サイト演算子の相関関数 $C = \langle A(\boldsymbol{r}_0)B(\boldsymbol{r}_0 + \boldsymbol{r})\rangle$ に関する情報を指定するセクションです。本セクションを省略した場合、相関関数は計算されません。

座標は正方格子 TNS の座標系で測られます。すなわち、右隣のテンソルは $\boldsymbol{r} = (1, 0)$ で、真上は $\boldsymbol{r} = (0, 1)$ です。中心座標 $\boldsymbol{r}_0$ として、ユニットセル内のすべてのサイトが用いられます。また、$\boldsymbol{r}$ は x ないし y 軸に平行な方向に、正の向きにのみ動きます。すなわち、

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

です。

名前	説明	型
r_max	相関関数の距離 r の最大値	整数
operators	相関関数を測る 1 体演算子 A, B を表す番号	整数のリストのリスト

演算子は observable.onesite セクションで指定したものが用いられます。

例

例えば S^z が 0 番で、 S^x が 1 番として定義されている場合、

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

では相関関数 $S^z(0)S^z(r), S^z(0)S^x(r), S^x(0)S^x(r)$ が、 $0 \leq r \leq 5$ の範囲で測定されます。

5.5 出力ファイル

各種計算結果は output ディレクトリ以下に保存されます。また、入力ファイルとして使ったファイルがコピーされます。

5.5.1 parameters.dat

実際に計算に使われたパラメータおよび実行日時が出力されます。

例

```
simple_num_step = 1000
simple_inverse_lambda_cutoff = 1e-12

full_num_step = 0
full_inverse_projector_cutoff = 1e-12
full_inverse_precision = 1e-12
full_convergence_epsilon = 1e-12
full_iteration_max = 1000
full_gauge_fix = true
full_fastfullupdate = true

ctm_dimension = 9
ctm_inverse_projector_cutoff = 1e-12
ctm_convergence_epsilon = 1e-10
ctm_iteration_max = 10
ctm_projector_corner = false
use_rsvd = false
rsvd_oversampling_factor = 2

seed = 11
is_real = 1
iszero_tol = 0
tensor_load_dir =
tensor_save_dir =
outdir = output

Lsub = [ 2 , 2 ]
skew = 0

start_datetime =  2020-03-21T17:04:06+09:00
finish_datetime = 2020-03-21T17:04:08+09:00
```

5.5.2 density.dat

各種演算子のサイトあたりの期待値 (実部と虚部) が出力されます。各演算子の名前 name が空だった場合はかわりに演算子番号が出力されます。

例

```
Sz          =  6.11647102532908438e-03  0.0000000000000000e+00
Sx          = -1.18125085038094907e-01  0.0000000000000000e+00
hamiltonian = -5.43684776153081639e-01  0.0000000000000000e+00
SzSz        = -3.16323622995942133e-01  0.0000000000000000e+00
SxSx        = -8.55704529153783616e-02  0.0000000000000000e+00
SySy        = -1.41790700241760936e-01  0.0000000000000000e+00
```

5.5.3 onesite_obs.dat

onesite 演算子の期待値 $\langle \hat{A}_i^\alpha \rangle$ が出力されます。各行 4 列からなります。

1. 演算子の識別番号 α
2. サイトの番号 i
3. 期待値の実部 $\mathrm{Re}\langle \hat{A}_i^\alpha \rangle$
4. 期待値の虚部 $\mathrm{Im}\langle \hat{A}_i^\alpha \rangle$

例

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

0 0 3.7653061689963413e-01 -2.9960378455674193e-14
0 1 -3.7650975472291992e-01 3.0068068151572541e-14
0 2 -3.7653055335765684e-01 3.0287620090471537e-14
0 3 3.7650969120374933e-01 -2.9799344104397859e-14
1 0 2.8309786789638672e-05 -6.2106666026872947e-16
1 1 -2.8346642046177013e-05 -8.1536805775606980e-17
1 2 -2.8348210012438907e-05 3.4850466772163179e-16
1 3 2.8308215524673213e-05 -1.5314665837228706e-16
2 0 -5.1090831684806589e-05 -3.1844760311397194e-16
2 1 5.1150013284430505e-05 2.5518615259402604e-17
2 2 5.1152842229474105e-05 6.3910132047015778e-16
2 3 -5.1087995797809094e-05 -2.6554762165511758e-16
```

5.5.4 twosite_obs.dat

twosite 演算子の期待値が出力されます。各行 6 列からなります。

1. twosite 演算子の識別番号
2. source サイトの番号
3. source からみた target の x 変位
4. source からみた target の y 変位
5. 期待値の実部
6. 期待値の虚部

例

```
# $1: op_group
# $2: source_site
# $3: dx
# $4: dy
# $5: real
# $6: imag

0 0 0 1 -3.34687360117867760e-01 2.68843730436328272e-14
0 0 1 0 -3.30837879477270169e-01 2.57989265090129366e-14
0 1 0 1 -3.34681128826821883e-01 2.69020083537152420e-14
0 1 1 0 -3.32798087400507758e-01 2.66387741317542961e-14
0 2 0 1 -3.36648943667345379e-01 2.61735500972987307e-14
0 2 1 0 -3.30837927854314540e-01 2.64427835764587303e-14
0 3 0 1 -3.36688835303625589e-01 2.64550560558367253e-14
0 3 1 0 -3.32798142125971141e-01 2.64082512640410446e-14
```

5.5.5 correlation.dat

相関関数 $C_i^{\alpha\beta}(x,y) \equiv \langle \hat{A}^\alpha(x_i,y_i)\hat{A}^\beta(x_i+x,y_i+y)\rangle$ が出力されます。各行 7 列から構成されます。

1. 左演算子の識別番号 α
2. 左演算子のサイト番号 i
3. 右演算子の識別番号 β
4. 右演算子の x 方向変位 x
5. 右演算子の y 方向変位 y

6. 演算子の実部 $\mathrm{Re}C$

7. 演算子の虚部 $\mathrm{Im}C$

例

```
# $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
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 2.01916094009441903e-02 -1.27162373457160362e-15
... Skipped ...
2 3 2 0 5 -1.41888376278899312e-03 -2.38672137694415560e-16
```

5.5.6 time.dat

計算時間が出力されます。

例

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

第 6 章

アルゴリズム

6.1 テンソルネットワーク状態

テンソルネットワーク状態 (Tensor network states (TNS)) とは小さなテンソルの積、繋がりで表現された変分波動関数です *[TNS]* 。例えば、N 個の $S = 1/2$ 量子スピン系では、その波動関数は直積状態の基底を用いて、

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

と表せます。テンソルネットワーク状態では、この展開係数 $\Psi_{s_1,s_2,\ldots,s_N}$ はテンソルのネットワークで表現され、例えば

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

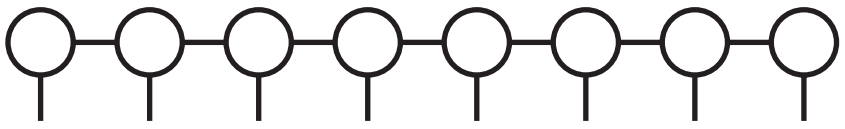
と書くことができます。ここで、 $\mathrm{tTr}[\ldots]$ はテンソルネットワークの縮約を表し、 $T^{(i)}[s_i]$ はテンソルを表しています。 行列積状態 (matrix product state (MPS)) と呼ばれるテンソルネットワーク状態の場合 *[MPS]* , $T^{(i)}[s_i]$ は s_i が与えられると行列になっていて、 $\mathrm{tTr}[\ldots]$ はこの場合、通常の行列積になります：

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

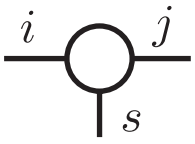
ここで、 $T^{(1)}[s_1]$ 、 $T^{(i)}[s_i](i \neq 1, N)$ 、 $T^{(N)}[s_N]$ の形状はそれぞれ、 $1 \times D_1$ 、 $D_{i-1} \times D_i$ 、 $D_{N-1} \times 1$ であると仮定しました。

このような TNS を基底状態波動関数の近似に用いる場合、その精度は D_i によって制限されます。 D_i は **ボンド次元** と呼ばれています。テンソルネットワークのダイアグラム表記を用いると、 MPS は

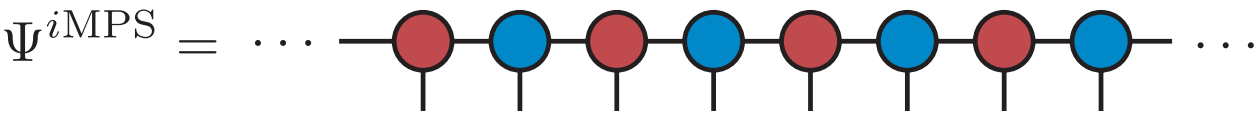
$\Psi^{\mathrm{MPS}} =$



$T_{ij}[s] =$

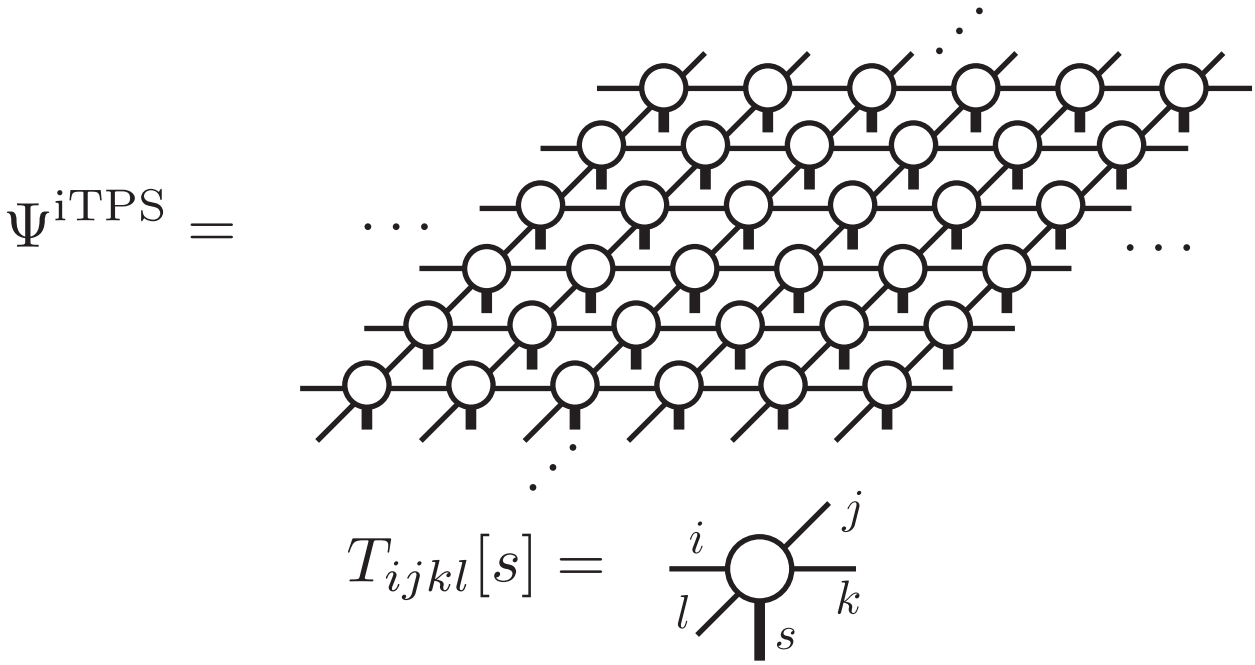


のように描くことができます。この MPS は有限系の波動関数を表していますが、同様にして、無限系の波動関数を表わす、無限に長い MPS を考えることもできます。特に、波動関数が格子の（ある長さの）並進に対して対称性を持っている場合、少数の独立なテンソルを繰り返すことで、無限系の MPS (infinite MPS (iMPS)) を作ることができます。 2 サイトの並進対称性の場合、このような iMPS は



と表せます。ここで、同じ色のテンソルは、同じ要素を持つテンソルです。

TeNeS では、2 次元の無限に広がったテンソル積状態 (infinite tensor product states (iTNS)) を取り扱います。この iTPS は iMPS の高次元への自然な拡張になっています。 TeNeS では、並進対称性をもった正方格子テンソルネットワークを仮定しており、ダイヤグラムでは、

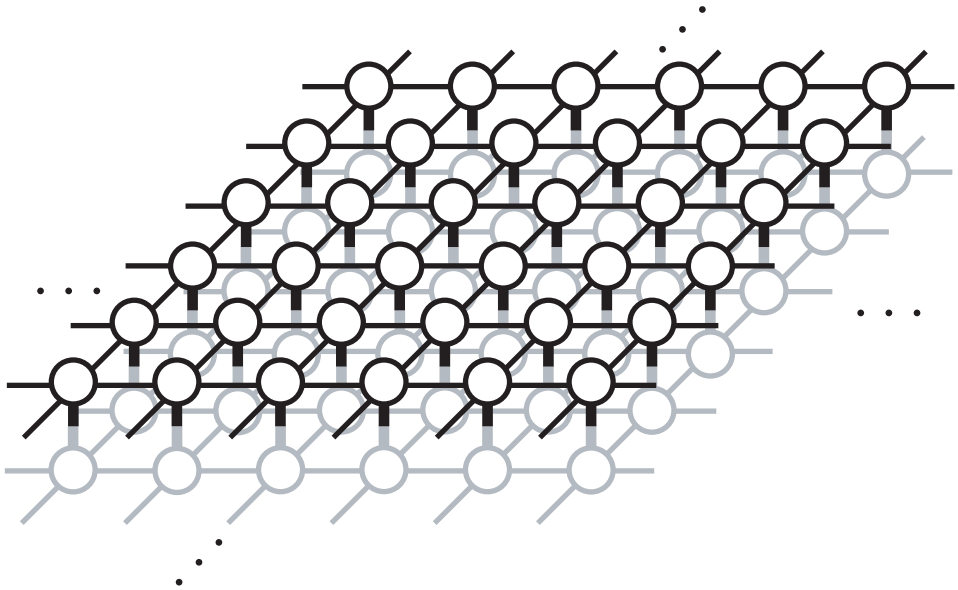


のように描けます。TeNeS ではこの iTPS を用いて、2 次元量子多体系の基底状態を近似的に計算します。なお、正方格子テンソルネットワークは、正方格子模型だけでなく、適切なマッピングにより、ハニカム格子模型、三角格子模型など様々な 2 次元格子模型に適用できます。

6.2 iTPS の縮約

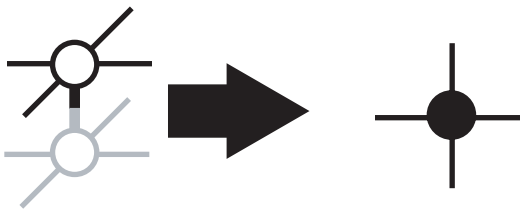
ある TNS が与えられた時に、その TNS での期待値、$\langle\Psi|O|\Psi\rangle/\langle\Psi|\Psi\rangle$ を計算するためには、一般に、$\langle\Psi|O|\Psi\rangle$ と $\langle\Psi|\Psi\rangle$ という二つの量に対応するテンソルネットワークの縮約計算が必要になります。例えば、$\langle\Psi|\Psi\rangle$ に対応するテンソルネットワークは

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

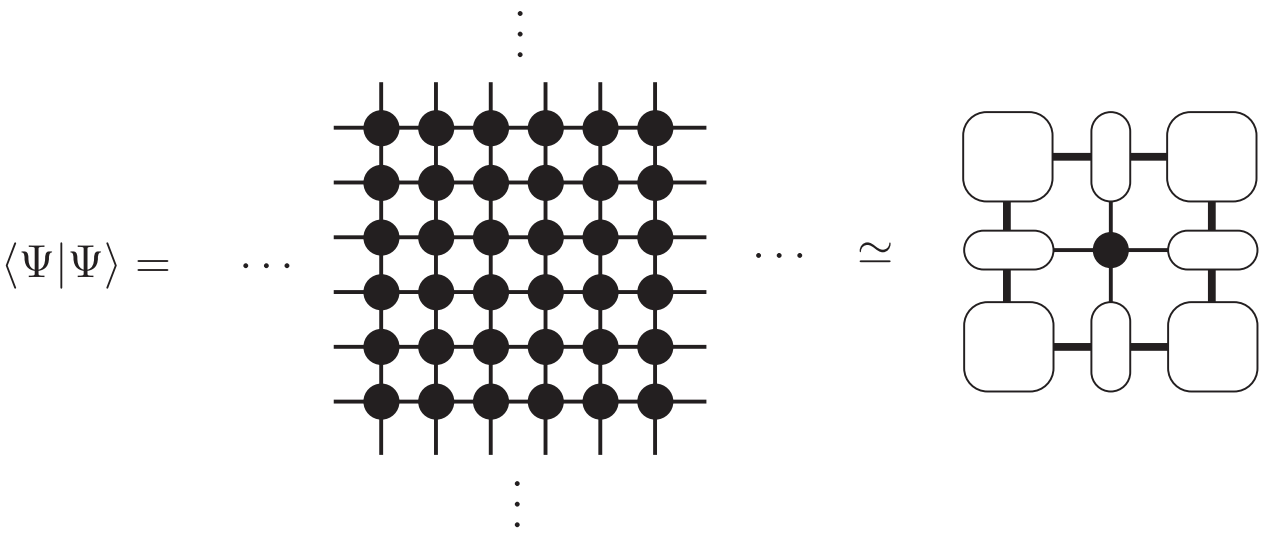


で与えられます。この形のテンソルネットワークは、しばしば、ダブルレイヤー（double layered）テンソルネットワークと呼ばれます。 ダブルレイヤーテンソルネットワークの縮約計算は、通常、非常に大きな計算コストが必要です。MPS（や iMPS）の場合には、幸いにも、局所的なテンソルで構成される転送行列を考えることなどによって、効率的に計算することができます。しかし、TPS（や iTPS）の場合、厳密な縮約計算は小さいクラスター（又は小さい半径の無限シリンダー）を除いてほぼ不可能で、通常、近似的な縮約計算法を用います。TeNeS では、角転送行列繰り込み群法（corner transfer matrix renormalization group (CTMRG) *[CTMRG]* と呼ばれる、無限に広がったダブルレイヤーテンソルネットワークを **角転送行列** と **エッジテンソル** を用いて近似する方法を採用しています。

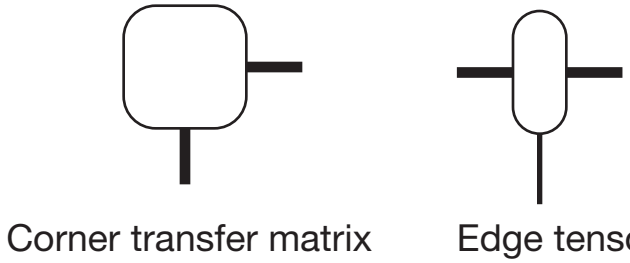
ダブルレイヤーテンソルネットワークを局所的に縮約したテンソル



を使って単純化すると、角転送行列表現に対応するテンソルネットワークダイアグラムは、

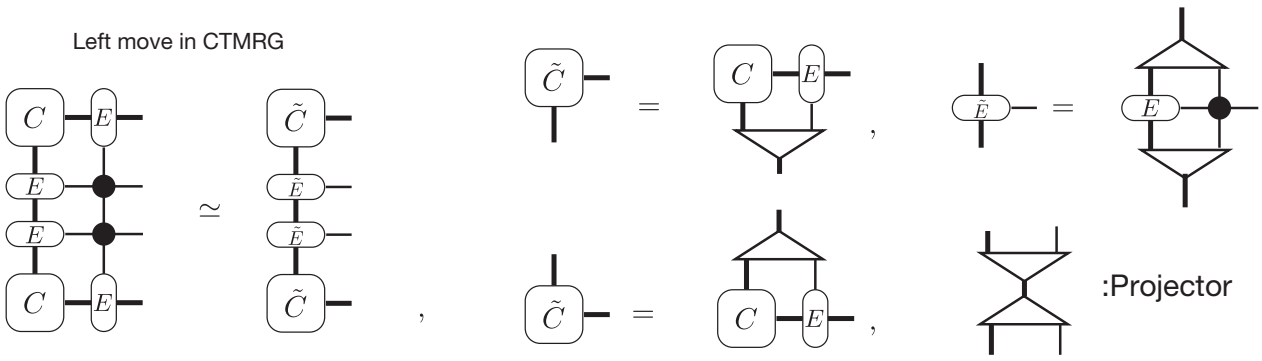


と表されます。角転送行列とエッジテンソルは、



のように定義されています。角転送行列表現の精度は、ダイアグラム中で太線で表現した、角転送行列のボンド次元 χ によって制限されます。

CTMRG のアルゴリズムでは、 角転送行列とエッジテンソルに局所的なテンソルを **吸収** していくことでそれらをアップデートし、結果が収束するまで繰り返します。例えば、 *left move* と呼ばれる吸収手続きは、ダイアグラムでは

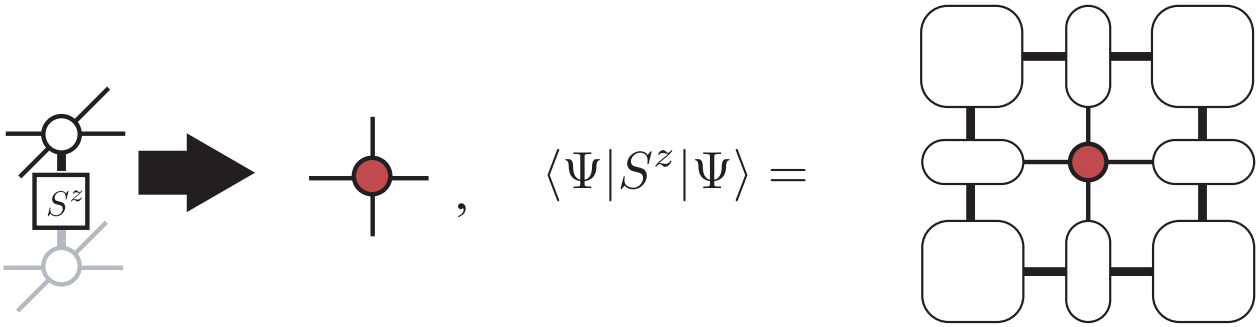


と表されます。このダイアグラムに現れる **プロジェクター** は、いくつかの方法で計算することができ *[CTMRG]* 、自由度を χ に減らす働きをします。

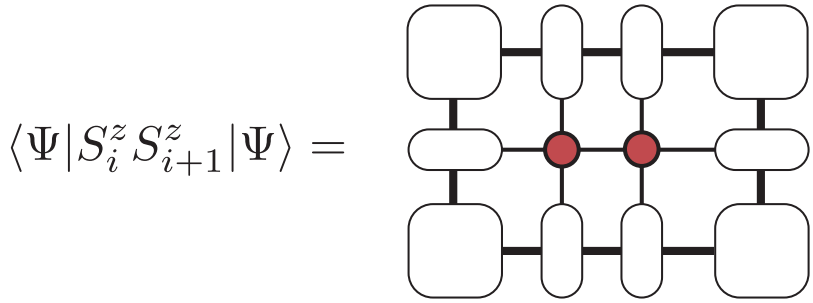
ボンド次元 D の iTPS を用いて、ボンド次元 χ の角転送行列表現を考える場合、CTMRG の計算コストは、$O(\chi^2 D^6)$ と $O(\chi^3 D^4)$ の大きな方でスケールします。 ここで、ダブルレイヤーテンソルネットワークのボンド次

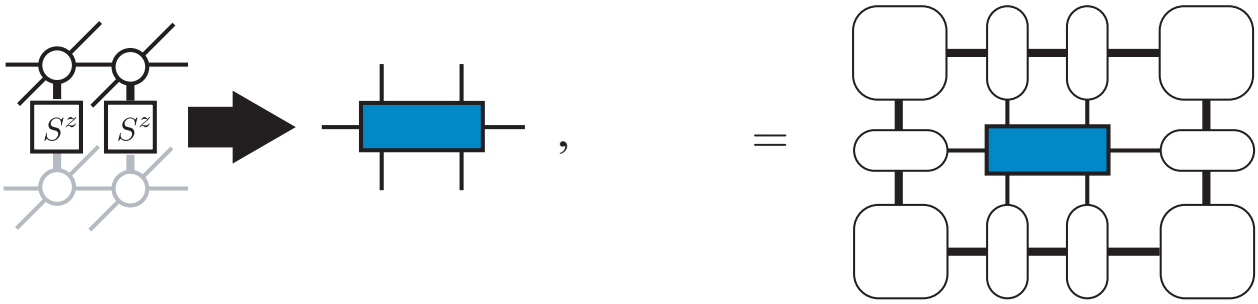
元は、局所縮約したテンソルを用いる表現では、 D^2 になっていることに注意してください。このため、通常、 χ は $\chi \propto O(D^2)$ のように D^2 に比例して増やします。この条件では、CTMRG の計算コストは $O(D^{10})$ になり、メモリ量は $O(D^8)$ になります。 なお、ここで述べた計算コストを得るためには、疎行列の特異値分解（SVD）を用いる必要があります、代わりに、密行列の SVD を用いる場合、計算コストは $O(D^{12})$ となります。

いったん収束した角転送行列とエッジテンソルを得れば、$\langle\Psi|O|\Psi\rangle$ も効率的に計算することができます。例えば、局所磁化 $\langle\Psi|S_i^z|\Psi\rangle$ は、



のように表わされ、同様に最近接相関 $\langle\Psi|S_i^z S_{i+1}^z|\Psi\rangle$ は





と表現することができます。また、ダイアグラムの 2 番目の表記を用いることで、任意の 2 サイト演算子の期待値も計算できることがわかります。このようなダイグラムを任意の演算子に対して描くことは可能ですが、クラスターが大きくなるとその縮約計算に必要となる計算コストが莫大になることに注意してください。

6.3 iTPS の最適化

iTPS を基底状態の変分波動関数として用いる場合、iTPS が最小のエネルギー期待値

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

を与えるように、テンソルを最適化する必要があります。ここで、 $\mathcal{H}$ は対象系のハミルトニアンを表しています。TeNeS では、虚時間発展（the imaginary evolution (ITE)）法と変分最適化（the variational optimization）法という二つの手法のうち、前者の ITE を採用しています。TeNeS では、iTPS の範囲での近似的な虚時間発展

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

を考えます。ここで、 $|\Psi_0\rangle$ は任意の初期 iTPS です。. もし、 T が十分に大きければ、左辺の $|\Psi^{\mathrm{iTPS}}\rangle$ は基底状態の良い近似になっていると考えることができます。

TeNeS では、ハミルトニアンは短距離の二体相互作用の和で

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

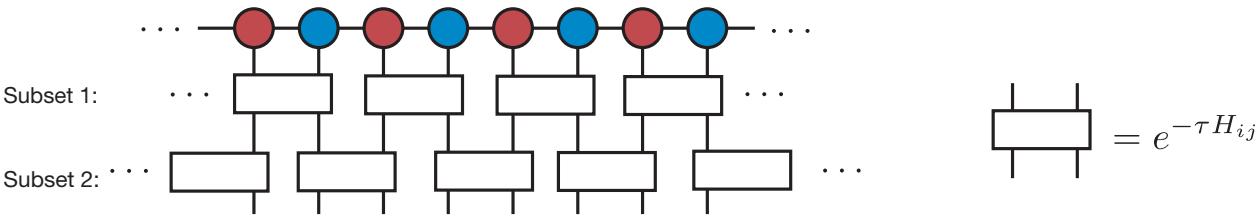
のように表されていると仮定し、小さな時間刻み τ の虚時間発展演算子に対して Suzuki-Trotter 分解

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

を適用します。ここでは、一次の近似を考えましたが、より高次の分解を考えることもできます。Suzuki-Trotter 分解の形を用いることで、虚時間発展は

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

のように書き下すことができます。ここで、 $N_\tau = T/\tau$ は十分に小さな τ での虚時間発展のステップ数です。この式の右辺を計算するために、 $\prod_{\{(i,j)\}}$ の積をいくつかの部分集合に分解します。それぞれの部分集合内では,（局所的な）虚時間発展演算子はお互いに交換し、考えている iTPS と同じ並進対称性を持っているとします。例えば、 2 サイトの iMPS で、 1 元系の最近接相互作用ハミルトニアンを考えた場合、二つの部分集合を用いて、

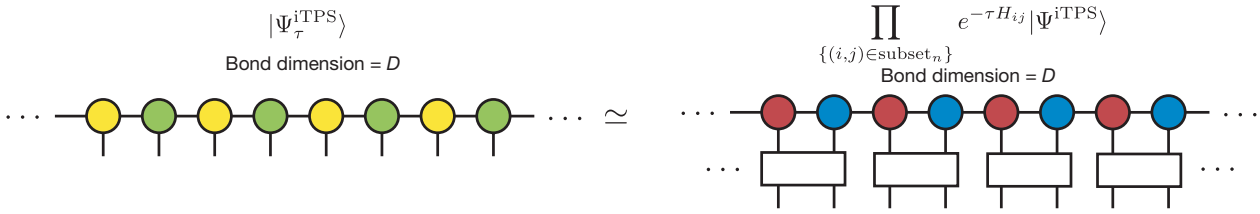


のように虚時間発展を分解することができます。

次に、それぞれの虚時間発展演算子の部分集合を適用した波動関数を、ボンド次元 D: の新しい iTPS として

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

のように近似します。ここで $\prod_{\{(i,j)\in \mathrm{subset}_n\}}$ は n 番目の部分集合ないの演算子の積を表し、$|\Psi_\tau^{\mathrm{iTPS}}\rangle$ は新しい iTPS です。ダイアグラムを用いるとこの式は、



のように表現できます。一般に、$e^{-\tau H_{ij}}$ をかけることで 厳密な iTPS 表現のボンド次元は増大してしまうことに注意してください。したがって、虚時間発展のシミュレーションを安定して継続するためには、ボンド次元をある一定値 D まで毎回打ち切る （ *truncate* ） 必要があります。

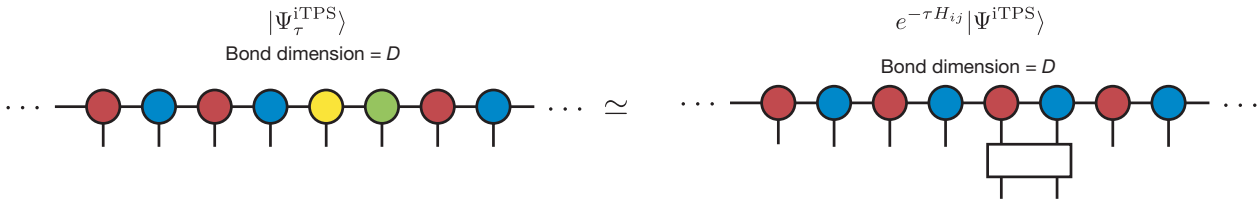
素朴には、効率的な打ち切りは、最小化問題

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

を解くことで行えます。しかし、この最小化問題を解く計算コストは、主に iTPS の並進対称性で問題が非線形問題になっているために、非常に膨大になってしまいます。そこで、通常は、代わりの問題として、局所的な一つの虚時間発展演算子だけを適用して、それを近似する iTPS $|\Psi_\tau^{\mathrm{iTPS}}\rangle$ を探す問題を考えます。ここで、新しい iTPS では、元の $|\Psi^{\mathrm{iTPS}}\rangle$ と比較して、数個のテンソルだけが変更されています。 この局所的な最小化問題は

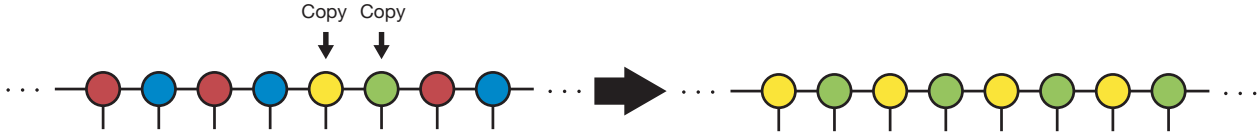
$$\min \left\| |\Psi_\tau^{\mathrm{iTPS}}\rangle - e^{-\tau H_{ij}}|\Psi^{\mathrm{iTPS}}\rangle \right\|^2$$

と書くことができます。一次元の最近接相互作用の場合、この最小化問題に対応するダイアグラムは、



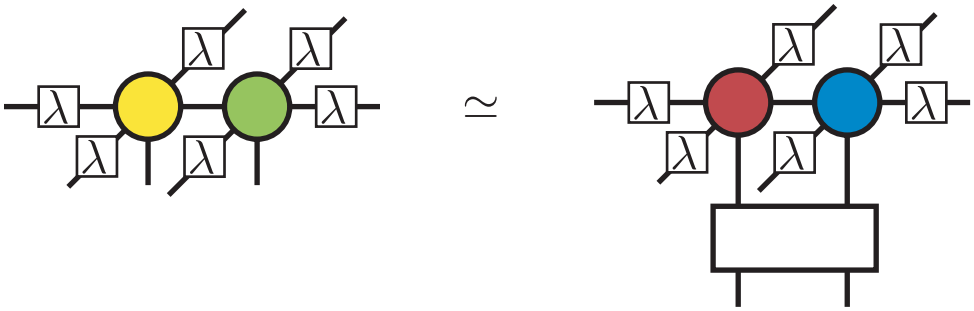
で与えられます。

差の二乗ノルム $\left\| |\Psi_\tau^{\mathrm{iTPS}}\rangle - e^{-\tau H_{ij}}|\Psi^{\mathrm{iTPS}}\rangle \right\|^2$ は、例えば CTMRG 等を使うことで効率的に計算できるため、この最適化問題は簡単に解くことができます *[ITE]* 。ここで新しく得られる iTPS は並進対称性を破っていますが、アップデートされたテンソルを他の場所に **コピー** することで、並進対称な iTPS を作ることができます。

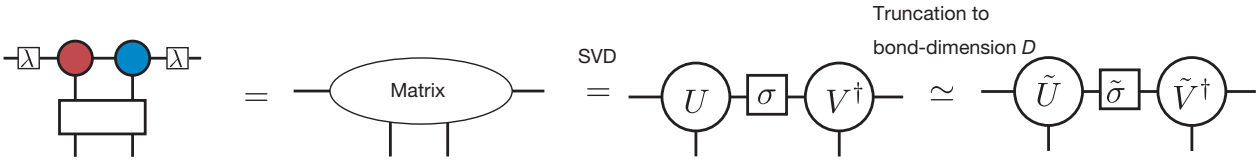


この iTPS は元の最小化問題の近似解だと考えることができます。このような虚時間発展の方法は、 *full update* 法と呼ばれます。 full update 法の計算の大部分は CTMRG であり、SVD の方法に応じて、計算コストは $O(D^{10})$ または $O(D^{12})$ でスケールします。

Simple update 法は虚時間発展を用いた、より計算コストの小さい最適化手法です。Simple update 法では、CTMRG による重い計算を避けるために, 波動関数全体ではなく、局所的なテンソルネットワークを考えます *[SimpleUpdate]* 。例えば 最近接相互作用の場合には、以下のような局所的な最適化問題を考えます。



このダイアグラムでは、 λ_i は非負の対角行列を表していて、これはボンド i の先にある無視した環境を表わす平均場だと考えることができます。 λ_i の具体的な定義は後で与えられます。 このダイアグラムが表わす最適化問題は、テンソル二つと虚時間発展演算子一つが一体となった行列の低ランク近似と見做すことができるため、SVD を用いて解くことができます。この手続きは、ダイアグラムを用いて、



と表すことができます。計算途中の SVD で出てきた行列の特異値は、次のステップでの平均場 λ として利用されます。Simple update 法の計算コストは、行列を構成する前に QR 分解を行うことで、 $O(D^5)$ になります *[QR]* 。したがって、simple update 法は full update 法よりもずっと計算コストが軽くなっています。

ただし、simple update 法は full update よりも計算コストが小さいですが、simple update 法は初期状態依存性が強

く、また、最終結果の局所磁化の大きさを過剰評価する問題が知られています。したがって、未知の問題に適用する場合には、得られた結果を慎重に検証する必要があります。

参考文献

[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; 西野友年、大久保毅 テンソルネットワーク形式の進展と応用, 日本物理学会誌 **72**, 702 (2017). link; 大久保毅 テンソルネットワークによる情報圧縮とフラストレート磁性体への応用, 物性研究・電子版 **7**, 072209 (2018) link.

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

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

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

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

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

第 7 章

謝辞

TeNeS の開発は，文部科学省ポスト「京」萌芽的課題 1 「基礎科学のフロンティア ー 極限への挑戦」及びポスト「京」重点課題 7 「次世代の産業を支える新機能デバイス・高性能材料の創成」の一環として実施されました．また, 東京大学物性研究所 ソフトウェア高度化プロジェクト (2019 年度) の支援も受け開発されました。この場を借りて感謝します。

第 8 章

お問い合わせ

TeNeS に関するお問い合わせはこちらにお寄せください。

- 質問・バグ報告

  TeNeS のバグ関連の報告は GitHub の Issues で受け付けています。バグを早期に解決するため、報告時には次のガイドラインに従ってください。

  - 使用している TeNeS、オペレーティングシステム、コンパイラの情報（名前およびバージョン）を記載してください。
  - インストールに問題がある場合には、cmake および make 実行時の入出力と CMakeCache.txt も一緒に記載してください。
  - 実行に問題が生じた場合は, 実行に使用した入力ファイルとその出力を記載してください。

- その他

  研究に関連するトピックなど、公開の場では相談しづらいことを問い合わせる際には, 以下の連絡先にコンタクトをしてください。

  E-mail: `tenes-dev__at__issp.u-tokyo.ac.jp` (_at_を@に変更してください)