block2

Huanchen Zhai

Apr 17, 2024

USER GUIDE

1 Contributors **3**

2 Features **5**

3 User Guide **9**
- 3.1 Installation . . . 9
- 3.2 Interfaces . . . 14
- 3.3 Input File: Basic Usage . . . 16
- 3.4 Input File: Advanced Usage . . . 37
- 3.5 Input File: Keywords . . . 62
- 3.6 DMRGSCF (pyscf) . . . 76
- 3.7 DMRGSCF (OpenMOLCAS) . . . 92
- 3.8 DMRGSCF (forte) . . . 100
- 3.9 MPS Import/Export . . . 106
- 3.10 References . . . 111

4 Python Interface Tutorial **117**
- 4.1 Quantum Chemistry Hamiltonians . . . 117
- 4.2 Energy Extrapolation . . . 188
- 4.3 Restarting DMRG . . . 195
- 4.4 DMRG with Spin-Orbit-Coupling . . . 239
- 4.5 Green's Function and TD-DMRG . . . 273
- 4.6 Custom Hamiltonians . . . 297
- 4.7 Hubbard Model . . . 310
- 4.8 Heisenberg Model . . . 330

5 Developer Guide **335**
- 5.1 DMRG Options . . . 335
- 5.2 MPS Orbital Rotation . . . 337
- 5.3 Point Group Mapping . . . 348
- 5.4 MPO Reloading . . . 354
- 5.5 Debugging Hints . . . 362
- 5.6 Notes . . . 372

6 API Reference **375**
- 6.1 Python Interface API . . . 375

6.2 Global Settings 430
6.3 Sparse Matrix 445
6.4 Tensor Functions 447
6.5 Tools 454

7 Theory **479**
7.1 DMRG Hamiltonian 479

Python Module Index **517**

Index **519**

block2

block2 is an efficient and highly scalable implementation of the Density Matrix Renormalization Group (DMRG) for quantum chemistry, based on Matrix Product Operator (MPO) formalism. The code is highly optimized for production level calculation of realistic systems. It also provides plenty of options for tuning performance and new algorithm development.

The block2 code is developed as an improved version of StackBlock, where the low-level structure of the code has been completely rewritten. The block2 code is developed and maintained in Garnet Chan group at Caltech.

Documentation: https://block2.readthedocs.io/en/latest/

Tutorial (python interface): https://block2.readthedocs.io/en/latest/tutorial/hubbard.html

Source code: https://github.com/block-hczhai/block2-preview

CHAPTER

ONE

CONTRIBUTORS

- Huanchen Zhai @hczhai: DMRG and parallelization
- Henrik R. Larsson @h-larsson: DMRG-MRCI/MRPT and big site
- Seunghoon Lee @seunghoonlee89: Stochastic perturbative DMRG
- Zhi-Hao Cui @zhcui: user interface

CHAPTER

TWO

FEATURES

- **State symmetry**
  - U(1) particle number symmetry
  - SU(2) or U(1) spin symmetry (spatial orbital)
  - No spin symmetry (general spin orbital)
  - Abelian point group symmetry
  - Translational (K point) / Lz symmetry
- **Sweep algorithms (1-site / 2-site / 2-site to 1-site transition)**
  - **Ground-State DMRG**
    - Decomposition types: density matrix / SVD
    - Noise types: wavefunction / density matrix / perturbative
  - **Multi-Target Excited-State DMRG**
    - State-averaged / state-specific
  - MPS compression / addition
  - Expectation
  - **Imaginary / real time evolution**
    - Hermitian / non-Hermitian Hamiltonian
    - Time-step targeting method
    - Time dependent variational principle method
  - Green's function
- Finite-Temperature DMRG (ancilla approach)
- Low-Temperature DMRG (partition function approach)
- **Particle Density Matrix (1-site / 2-site)**
  - 1PDM / 2PDM / 3PDM / 4PDM
  - Transition 1PDM / 2PDM / 3PDM / 4PDM

  - Spin / charge correlation
- **Quantum Chemistry MPO**
  - Normal-Complementary (NC) partition
  - Complementary-Normal (CN) partition
  - Conventional scheme (switch between NC and CN near the middle site)
- Symbolic MPO simplification
- MPS initialization using occupation number
- **Supported matrix representation of site operators**
  - Block-sparse (outer) / dense (inner)
  - Block-sparse (outer) / elementwise-sparse (CSR, inner)
- Fermionic MPS algebra (non-spin-adapted only)
- **Determinant/CSF coefficients of MPS**
  - Extracting Determinant/CSF coefficients from MPS
  - Constructing MPS from Determinant/CSF coefficients
- **Multi-level parallel DMRG**
  - Parallelism over sites (2-site only)
  - Parallelism over sum of MPOs (distributed)
  - Parallelism over operators (distributed/shared memory)
  - Parallelism over symmetry sectors (shared memory)
  - Parallelism within dense matrix multiplications (MKL)
- **DMRG-CASSCF and contracted dynamic correlation**
  - DMRG-CASSCF (pyscf / openMOLCAS / forte interface)
  - DMRG-CASSCF nuclear gradients and geometry optimization (pyscf interface, RHF reference only)
  - DMRG-sc-NEVPT2 (pyscf interface, classical approach)
  - DMRG-sc-MPS-NEVPT2 (pyscf interface, MPS compression approximation)
  - DMRG-CASPT2 (openMOLCAS interface)
  - DMRG-cu-CASPT2 (openMOLCAS interface)
  - DMRG-MRDSRG (forte interface)
- **DMRG with Spin-Orbit Coupling (SOC)**
  - 1-step approach (full complex one-MPO and hybrid real/complex two-MPO schemes)
  - 2-step approach

- Stochastic perturbative DMRG
- **Uncontracted dynamic correlation**
  - DMRG Multi-Reference Configuration Interaction (MRCI) of arbitrary order
  - DMRG Multi-Reference Averaged Quadratic Coupled Cluster (AQCC)/ Coupled Pair Functional (ACPF)
  - DMRG NEVPT2/3/..., REPT2/3/..., MR-LCC, ...
- **Orbital Reordering**
  - Fiedler
  - Genetic algorithm
- **MPS Transformation**
  - SU2 to SZ mapping
  - Point group mapping
  - Orbital basis rotation

CHAPTER

THREE

USER GUIDE

3.1 Installation

3.1.1 Using `pip`

One can install `block2` using `pip`:

- OpenMP-only version (no MPI dependence)

```
pip install block2
```

- Hybrid openMP/MPI version (requiring openMPI 4.1.x)

```
pip install block2-mpi
```

- Binary format are prepared via `pip` for python 3.7, 3.8, 3.9, 3.10, and 3.11 with macOS (x86 and arm64, no-MPI) or Linux (no-MPI/openMPI). If these binaries have some problems, you can use the `--no-binary` option of `pip` to force building from source (for example, `pip install block2 --no-binary block2`).
- One should only install one of `block2` and `block2-mpi`. `block2-mpi` covers all features in `block2`, but its dependence on mpi library can sometimes be difficult to deal with. Some guidance for resolving environment problems can be found in github issue #7.
- To install the most recent development version, use:

```
pip install block2==<version> --extra-index-url=https://block-hczhai.github.io/
↪block2-preview/pypi/
pip install block2-mpi==<version> --extra-index-url=https://block-hczhai.github.
↪io/block2-preview/pypi/
```

  where `<version>` can be some development version number like `0.5.2rc13`. To force reinstalling an updated version, you may consider `pip` options `--upgrade --force-reinstall --no-deps --no-cache-dir`.

3.1.2 Manual Installation

Dependence: `pybind11`, `python3`, and `mkl` (or `blas + lapack`).

For distributed parallel calculation, `mpi` library is required.

`cmake` (version >= 3.0) can be used to compile C++ part of the code, as follows

```
mkdir build
cd build
cmake .. -DUSE_MKL=ON -DBUILD_LIB=ON -DLARGE_BOND=ON -DMPI=ON
make -j 10
```

Which will build the python extension library (using 10 CPU cores) (serial code).

You may need to add both the repo root directory and the `build` directory into `PYTHONPATH` so that `import block2` and `import pyblock2` will work

```
export PYTHONPATH=/path/to/block2/build:/path/to/block2:${PYTHONPATH}
```

3.1.3 Options

MKL

If `-DUSE_MKL=ON` is not given, `blas` and `lapack` are required. Sometimes, the `blas` and `lapack` function names can contain the extra underscore. Therefore, it is recommended to use `-DUSE_MKL=OFF` and `-DF77UNDERSCORE=ON` together to prevent this underscore problem. If this generates the undefined reference error, one should try `-DUSE_MKL=OFF -DF77UNDERSCORE=OFF` instead.

Use `-DUSE_MKL64=ON` instead of `-DUSE_MKL=ON` to enable using matrices with 64-bit integer type.

Serial compilation

By default, the C++ templates will be explicitly instantiated in different compilation units, so that parallel compilation is possible.

Alternatively, one can do single-file compilation using `-DEXP_TMPL=NONE`, then total compilation time can be saved by avoiding unnecessary template instantiation, as follows

```
cmake .. -DUSE_MKL=ON -DBUILD_LIB=ON -DEXP_TMPL=NONE
make -j 1
```

This may take 11 minutes, requiring 14 GB memory.

MPI version

Adding option `-DMPI=ON` will build MPI parallel version. The C++ compiler and MPI library must be matched. If necessary, environment variables `CC`, `CXX`, and `MPIHOME` can be used to explicitly set the path. For manual compilation, the MPI library can have arbitrary version (openMPI, mpich, intelmpi, etc.).

For mixed openMP/MPI, use `mpirun --bind-to none -n ...` or `mpirun --bind-to core --map-by ppr:$NPROC:node:pe=$NOMPT ...` to execute binary.

Binary build

To build unit tests and binary executable (instead of python extension), use the following

```
cmake .. -DUSE_MKL=ON -DBUILD_TEST=ON
```

TBB (Intel Threading Building Blocks)

Adding (optional) option `-DTBB=ON` will utilize `malloc` from `tbbmalloc`. This can improve multi-threading performance.

openMP

If gnu openMP library `libgomp` is not available, one can use intel openMP library.

The following will switch to intel openMP library (incompatible with `-fopenmp`)

```
cmake .. -DUSE_MKL=ON -DBUILD_LIB=ON -DOMP_LIB=INTEL
```

The following will use sequential mkl library

```
cmake .. -DUSE_MKL=ON -DBUILD_LIB=ON -DOMP_LIB=SEQ
```

The following will use tbb mkl library

```
cmake .. -DUSE_MKL=ON -DBUILD_LIB=ON -DOMP_LIB=TBB -DTBB=ON
```

Note: (For developers.) For `CSR sparse MKL + ThreadingTypes::Operator`, if `-DOMP_LIB=GNU`, it is not possible to set both `n_threads_mkl` not equal to 1 and `n_threads_op` not equal to 1. In other words, nested openMP is not possible for CSR sparse matrix (generating wrong result/non-convergence). For `-DOMP_LIB=SEQ`, CSR sparse matrix is okay (non-nested openMP). For `-DOMP_LIB=TBB`, nested openMP + TBB MKL is okay.

`-DTBB=ON` can be combined with any `-DOMP_LIB=...`.

Complex mode

For complex integrals / spin-orbit coupling (SOC), extra options `-DUSE_COMPLEX=ON` and `-DUSE_SG=ON` are required (and the compilation time will increase).

Maximal bond dimension

The default maximal allowed bond dimension per symmetry block is `65535`. Adding option `-DSMALL_BOND=ON` will change this value to `255`. Adding option `-DLARGE_BOND=ON` will change this value to `4294967295`.

Release build

The release mode is controlled by CMAKE_BUILD_TYPE.

The following option will use optimization flags such as -O3 (default)

```
cmake .. -DCMAKE_BUILD_TYPE=Release
```

The following enables debug flags

```
cmake .. -DCMAKE_BUILD_TYPE=Debug
```

Installation with anaconda

An incorrectly installed `mpi4py` may produce this error:

```
undefined symbol: ompi_mpi_logical8
```

when you execute `from mpi4py import MPI` in a `python` interpreter.

When using `anaconda`, please make sure that `mpi4py` is linked with the same `mpi` library as the one used for compiling `block2`. We can create an `anaconda` virtual environment (optional):

```
conda create -n block2 python=3.8 anaconda
conda activate block2
```

Then make sure that a working `mpi` library is in the environment, using, for example:

```
module load openmpi/4.1.6
module load gcc/9.2.0
```

Then we should install `mpi4py` using this `mpi` library via `--no-binary` option of `pip`:

```
python -m pip install --no-binary :all: mpi4py
```

Sometimes, the above procedure may still give the `undefined symbol: ompi_mpi_logical8` error. Then it is possible that the `mpi4py` is still linked to the `mpich` (version 3 or lower) library installed in anaconda. If this is the case, one should first `conda uninstall mpich` and then `python -m pip -v install --no-binary :all: mpi4py` and if the installation is successful, we can `ldd $(python -c 'from mpi4py import MPI;print(MPI.__file__)')` to check the linkage of the `libmpi.so`. Ideally it should points to the `openmpi/4.1.6` library or any other version 4.1 mpi library. Alternatively, if you do not want to uninstall the `mpich` in anaconda, you may install `block2` from source using the same `mpich` library.

BLIS

Optionally, we can use BLIS for dense matrix GEMM operations. One can install the BLIS as the following:

```
git clone https://github.com/flame/blis.git
cd blis/
mkdir install
./configure --prefix=$PWD/install --enable-threading=openmp auto
make -j 10
make install
export BLIS_PREFIX=$PWD/install
```

Then adding the option `-DUSE_BLIS=ON` (when compiling `block2`) will use BLIS for GEMM. Other BLAS operations will still be performed using the standard BLAS or MKL.

Supported operating systems and compilers

- Linux + gcc 9.2.0 + MKL 2021.4
- MacOS 10.15 + Apple clang 12.0 + MKL 2021
- MacOS 10.15 + icpc 2021.1 + MKL 2021
- Windows 10 + Visual Studio 2019 (MSVC 14.28) + MKL 2021

Using `block2` together with other python extensions

Sometimes, when you have to use `block2` together with other python modules (such as `pyscf` or `pyblock`), it may have some problem coexisting with each other. In general, change the import order may help. For `pyscf`, `import block2` at the very beginning of the script may help. For `pyblock`, recompiling `block2` use `cmake .. -DUSE_MKL=OFF -DBUILD_LIB=ON -OMP_LIB=SEQ -DLARGE_BOND=ON` may help.

Using C++ Interpreter cling

Since `block2` is designed as a header-only C++ library, it can be conveniently executed using C++ interpreter cling (which can be installed via anaconda) without any compilation. This can be useful for testing samll changes in the C++ code.

Example C++ code for `cling` can be found at `tests/cling/hubbard.cl`.

3.2 Interfaces

`block2` can be used via many different interfaces.

3.2.1 Input File

Like many quantum chemistry packages, `block2` can be used by reading parameters and instrutions from a formatted input file. This interface is `StackBlock` compatible. See *Input File: Basic Usage*, *Input File: Advanced Usage*, and *Input File: Keywords*.

3.2.2 Interfaces for DMRGSCF

To do DMRGSCF, we need to connect `block2` to some external softwares for the CASSCF part. See *DMRGSCF (pyscf)*, *DMRGSCF (OpenMOLCAS)*, and *DMRGSCF (forte)*.

3.2.3 Python Interface (high level)

See https://block2.readthedocs.io/en/latest/tutorial/qc-hamiltonians.html.

3.2.4 Python Interface (low level)

General examples:

1. GS-DMRG

Test Ground-State DMRG (need *pyscf* module):

```
python3 -m pyblock2.gsdmrg
```

2. FT-DMRG

Test Finite-Temperature (FT)-DMRG (need *pyscf* module):

```
python3 -m pyblock2.ftdmrg
```

3. LT-DMRG

Test Low-Temperature (LT)-DMRG (need *pyscf* module):

```
python3 -m pyblock2.ltdmrg
```

4. GF-DMRG

Test Green's-Function (GF)-DMRG (DDMRG++) (need *pyscf* module):

```
python3 -m pyblock2.gfdmrg
```

5. SI-DMRG

Test State-Interaction (SI)-DMRG (need *pyscf* module):

```
python3 -m pyblock2.sidmrg
```

For special topics, see *MPO Reloading*, *MPS Orbital Rotation*, *Point Group Mapping*.

3.2.5 Input File (C++ Executable)

Example input file for binary executable `build/block2`:

```
rand_seed = 1000
memory = 4E9
scratch = ./scratch

pg = c1
fcidump = data/HUBBARD-L16.FCIDUMP
n_threads = 4
qc_type = conventional

# print_mpo
print_mpo_dims
print_fci_dims
print_mps_dims

bond_dims = 500
noises = 1E-6 1E-6 0.0

center = 0
dot = 2

n_sweeps = 10
tol = 1E-7
forward = 1

noise_type = perturbative
trunc_type = physical
```

To run this example:

```
./build/block2 input.txt
```

3.2.6 C++ Interpreter

Since block2 is designed as a header-only C++ library, it can be conveniently executed using C++ interpreter [cling](https://github.com/root-project/cling) (which can be installed via [anaconda](https://anaconda.org/conda-forge/cling)) without any compilation. This can be useful for testing small changes in the C++ code.

Example C++ code for cling can be found at tests/cling/hubbard.cl.

3.3 Input File: Basic Usage

In this documentation, we explain how to use block2 as an "executable". The input parameters are provided in a formatted input file. The input file format used in block2 is highly compatible to the StackBlock format. For the most cases, the StackBlock input/configuration file ("dmrg.conf") can be directly understood by block2, but block2 also has some important extension for the keywords.

The information provided below is analogous to the corresponding StackBlock documentation, since the same input file format is used. However, the output format of block2 can be very different from that of StackBlock.

3.3.1 Preparation

If block2 is installed using pip install block2, one can run a DMRG calculation using the following command:

```
block2main dmrg.conf > dmrg.out
```

Otherwise, for manual installation, please first compile the code according to *Installation* with cmake option -DBUILD_LIB=ON (and other necessary options). The following python script is used as the "block2 executable":

```
${BLOCK2HOME}/pyblock2/driver/block2main
```

where ${BLOCK2HOME} is the block2 root directory. The build directory under block2 root directory should be in PYTHONPATH. You can add the following line in your environemnt (such as ~/.bashrc) or submission script:

```
export PYTHONPATH=${BLOCK2HOME}/build:${PYTHONPATH}
```

Then you can run a DMRG calculation using the following command:

```
${BLOCK2HOME}/pyblock2/driver/block2main dmrg.conf > dmrg.out
```

where `dmrg.conf` is the input file and `dmrg.out` is the output file.

To run a DMRG calculation with MPI parallelization, please use the following command:

```
mpirun --bind-to core --map-by ppr:${SLURM_TASKS_PER_NODE}:node:pe=${OMP_NUM_THREADS}
↪ \
    python -u ${BLOCK2HOME}/pyblock2/driver/block2main dmrg.conf > dmrg.out
```

where `${SLURM_TASKS_PER_NODE}` is the number of mpi processes in each node. `${OMP_NUM_THREADS}` is the number of threads (CPU cores) used by each mpi process. When executed in multiple nodes, a global scratch space (network file system) is required.

3.3.2 Integral Generation

In the following we will use the C_2 molecule to demonstrate the `block2` features. Integrals and orbitals should be supplied externally in the Molpro's FCIDUMP format. The integral file for C_2 can be found in `${BLOCK2HOME}/data/C2.CAS.PVDZ.FCIDUMP.ORIG` or generated using the following script (only the RHF case is required):

```
from pyscf import gto, scf, mcscf
from pyblock2._pyscf.ao2mo import integrals as itg
from pyblock2.driver.core import DMRGDriver, SymmetryTypes

mol = gto.M(atom='C 0 0 0; C 0 0 1.2425', basis='ccpvdz', symmetry='d2h')

# RHF case (for spin-adapted / non-spin-adapted DMRG)
mf = scf.RHF(mol).run()
mc = mcscf.CASCI(mf, 26, 8)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf, mc.ncore,␣
↪mc.ncas, g2e_symm=8)
driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SU2)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)
driver.write_fcidump(h1e, g2e, ecore=ecore, filename='./FCIDUMP', pg="d2h", h1e_
↪symm=True)

# UHF case (for non-spin-adapted DMRG only)
mf = scf.UHF(mol).run()
mc = mcscf.UCASCI(mf, 26, 8)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_uhf_integrals(mf, mc.ncore[0],
↪ mc.ncas, g2e_symm=8)
driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SZ)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)
driver.write_fcidump(h1e, g2e, ecore=ecore, filename='./FCIDUMP.UHF', pg="d2h", h1e_
↪symm=True)
```

Alternatively, the integral file can be generated using the `pyscf/dmrgscf` interface:

```
from pyscf import gto, scf, mcscf, dmrgscf
import os

dmrgscf.settings.BLOCKEXE = os.popen("which block2main").read().strip()
dmrgscf.settings.MPIPREFIX = ''

mol = gto.M(atom='C 0 0 0; C 0 0 1.2425', basis='ccpvdz', symmetry=1)
mf = scf.RHF(mol).run()
mc = mcscf.CASCI(mf, 26, 8)
mc.fcisolver = dmrgscf.DMRGCI(mol)
mc.canonicalization = False
dmrgscf.dryrun(mc)
```

Note: Please see *DMRGSCF (pyscf)* for the instruction for the installation of `pyscf/dmrgscf`.

3.3.3 Ground State Energy

The following input file can be used to compute the ground state energy:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30
num_thrds 16
```

Note: Note that the integral file `C2.CAS.PVDZ.FCIDUMP.ORIG` should be in the working direcotry. By default, the orbitals will be reordered using the `fiedler` method. One can optionally add the keyword `noreorder` to avoid orbital reordering.

`num_thrds` indicates the number of OpenMP threads (shared-memory parallelism) to use.

`hf_occ integral` has no effects in `block2`, but it is required in `StackBlock`. If this line appears, `block2main` will try to write some output files in a stackblock-compatible format.

By default, the calculation will be done in the spin-adapted mode, which is the most efficient. One can optionally add the keyword `nonspinadapted` to use the non-spin-adapted mode.

The keyword `prefix <scratch dir>` can be used to set a folder for storing scratch files. If running in a HPC supercomputer, it is highly recommended to use the high IO speed scratch space (instead

of the "home" storage) to achieve high performance.

Lines start with ! in the input file will be ignored.[1]

D_{2h} point group is enabled by `sym d2h`. The keywords `schedule default` and `maxM` sets the default sweep schedule and the maximum number of renormalized states kept during the sweep, respectively. `block2` will then automatically set a sweep schedule as well as the defaults for various convergence thresholds.

The mps bond dimensions, sweep energies and the associated maximum discarded weights can be extracted by grepping the output `dmrg.out`.

```
$ grep Bond dmrg.out
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-03 |␣
↪Dav threshold =  1.00e-04
Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-03 |␣
↪Dav threshold =  1.00e-04
Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-03 |␣
↪Dav threshold =  1.00e-04
Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-03 |␣
↪Dav threshold =  1.00e-04
... ...
Sweep =   16 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-06
Sweep =   17 | Direction = backward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-06
Sweep =    0 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-06
Sweep =    1 | Direction = backward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-06

$ grep DW dmrg.out
Time elapsed =      1.678 | E =     -75.4879935448 | DW = 1.39e-05
Time elapsed =      2.936 | E =     -75.6007921322 | DE = -1.13e-01 | DW = 9.88e-06
Time elapsed =      4.203 | E =     -75.6367659659 | DE = -3.60e-02 | DW = 9.25e-05
Time elapsed =      5.750 | E =     -75.6373954252 | DE = -6.29e-04 | DW = 3.91e-05
... ...
Time elapsed =     38.782 | E =     -75.7283521752 | DE = -3.48e-05 | DW = 5.24e-06
Time elapsed =     41.169 | E =     -75.7283676788 | DE = -1.55e-05 | DW = 5.28e-06
Time elapsed =      2.009 | E =     -75.7283421257 | DW = 4.18e-17
Time elapsed =      4.158 | E =     -75.7283421257 | DE = -2.84e-14 | DW = 2.47e-16
```

Note that in the last two sweeps (in default schedule) the 1-site algorithm is used. As a result, the discarded weights are nearly zero.

If you set `outputlevel 1` in the input file, only essential information will be printed and the `grep` step can be skipped.

[1] This is an extension implemented only in the `block2` code, which is not available in `StackBlock`.

3.3.4 Targeting States

You can target the states distinguished by the number of electrons `nelec`, the total spin `spin` and the point-group symmetry of the state `irrep`.

The following input file computes the energy for a single B_{1g} state in D_{2h} point group:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 4

hf_occ integral
schedule default
maxM 500
maxiter 30
```

Note: In D_{2h} point group, `irrep` can be A_{1g} (1), B_{3u} (2), B_{2u} (3), B_{1g} (4), B_{1u} (5), B_{2g} (6), B_{3g} (7), A_{1u} (8).

This will generate the following output:

```
$ grep DW dmrg.out
Time elapsed =      1.983 | E =     -75.5422510106 | DW = 1.08e-05
Time elapsed =      3.580 | E =     -75.6245880097 | DE = -8.23e-02 | DW = 9.97e-06
Time elapsed =      5.376 | E =     -75.6366528654 | DE = -1.21e-02 | DW = 9.13e-05
Time elapsed =      7.172 | E =     -75.6374064699 | DE = -7.54e-04 | DW = 4.03e-05
... ...
Time elapsed =     38.611 | E =     -75.6389586629 | DE = -2.48e-05 | DW = 2.01e-06
Time elapsed =     40.981 | E =     -75.6389699555 | DE = -1.13e-05 | DW = 2.05e-06
Time elapsed =      2.029 | E =     -75.6389630224 | DW = 5.58e-15
Time elapsed =      4.106 | E =     -75.6389632670 | DE = -2.45e-07 | DW = 2.40e-16
```

3.3.5 State-Averaged Calculation

In the state-averaged DMRG algorithm, more than one state can be targeted in one calculation. The states being calculated can have the same or different `nelec`, `spin` or `irrep`. Multiple values can be given for the above keywords.[Page 19, 1] The number of states (roots) and the weight of each state can be specified using keywords `nroots` and `weights`, respectively. `block2` will then try to find the low energy states within the space of targets formed by all combintaions of the given values of `nelec`, `spin` and `irrep`.

Note: In `StackBlock`, state-averaged calculation can only be done for states with the same `nelec`,

spin and irrep. In block2, targetting multiple nelec, spin or irrep may cause the calculation hard to converge to the lowest energy states. Typically, one needs larger nroots than the number of states actually needed, to make sure that the low energy states are converged.

For normal non-state-averaged calculation, namely, when nroots is 1, you can also target multiple nelec, spin or irrep.

The following input file performs state-averged DMRG for two A_{1g} states in D_{2h} point group:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1
nroots 2
weights 0.5 0.5

hf_occ integral
schedule default
maxM 500
maxiter 30
```

This will generate the following output:

```
$ grep DW dmrg.out
Time elapsed =      3.257 | E[  2] =     -75.5019604920    -75.4800275143 | DW = 1.
↪54e-05
Time elapsed =      5.109 | E[  2] =     -75.5980474127    -75.5776457885 | DE = -9.
↪76e-02 | DW = 1.98e-05
Time elapsed =      6.854 | E[  2] =     -75.6711500018    -75.6363593637 | DE = -5.
↪87e-02 | DW = 1.86e-04
Time elapsed =      8.635 | E[  2] =     -75.6717525884    -75.6368970346 | DE = -5.
↪38e-04 | DW = 1.35e-04
Time elapsed =     45.946 | E[  2] =     -75.7279558636    -75.6386525742 | DE = -3.
↪41e-05 | DW = 2.49e-05
Time elapsed =     48.491 | E[  2] =     -75.7279954715    -75.6386699048 | DE = -1.
↪73e-05 | DW = 1.67e-05
Time elapsed =      2.215 | E[  2] =     -75.7279403993    -75.6386251036 | DW = 1.
↪77e-05
Time elapsed =      4.338 | E[  2] =     -75.7279224367    -75.6386152528 | DE = 9.
↪85e-06 | DW = 8.35e-06
```

3.3.6 State-Specific Calculation

Orthogonalization Approach

The state-specific calculation can be done as a restart calculation which assumes that a previous state-averaged DMRG calculation has been converged. The state-specific DMRG calculation then reads the MPS from scratch folder and refines them for each root separately. The state-specific DMRG calculation can be done with any of `onedot`, `twodot` or `twodot_to_onedot` (default) keywords.[Page 19, 1]

Note: In `StackBlock`, state-specific calculation can only be done with `onedot`.

A state-specific DMRG calculation for two A_{1g} states in D_{2h} point group consists of two steps.

- First, using the input file given in the previous section to obtain the state-averaged MPSs (in the scratch folder).
- Second, the state-specific DMRG calculation can be performed by setting the keyword `statespecific`. The MPSs from the previous DMRG calculation will be read from the scratch folder. The following input file can be used for this step:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1
nroots 2
weights 0.5 0.5
statespecific

hf_occ integral
schedule default
maxM 500
maxiter 30
```

This will generate the following output:

```
$ grep Energy dmrg.out
DMRG Energy for root    0 =  -75.728342642601376
DMRG Energy for root    1 =  -75.638959372610813
```

Sometimes, the orthogonalization approach can be unstable and when computing the exciated state it may fall back to the ground state. Adding the keyword `onedot` for the second step can alleviate this problem.

Level Shift Approach

The second step of the above can also be done with the level shift approach, by changing Hamiltonian from $\hat{H}$ to $\hat{H} + \sum_i w_i |\phi_i\rangle\langle\phi_i|$. Normally, the weights w_i are positive and they should be larger than the energy gap.

The following input file can be used for the second step:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1
nroots 2
weights 0.5 0.5
statespecific
proj_weights 5 5

hf_occ integral
schedule default
maxM 500
maxiter 30
```

This will generate the following output:

```
$ grep Energy dmrg.out
DMRG Energy for root    0 =  -75.728341047222145
DMRG Energy for root    1 =  -75.638958637510370
```

Without State-Average

The excited MPS and energies can also be obtained without performing a state-averaged calculation as the first step. Instead, we can do several DMRG, and each time projecting out MPSs from all previous DMRG.

Note: It is recommended to use `noreorder` or fixed manual orbital reordering for this approach. Otherwise, one should carefully check that the orbital reordering in all DMRG calculations are the same.

We first get the ground state using the following input file `dmrg-1.conf`:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
```

(continues on next page)

(continued from previous page)

```
spin 0
irrep 1

schedule default
maxM 500
maxiter 30
mps_tags KET1
```

After this is finished, we compute the first excited state using the following input file `dmrg-2.conf`:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1

schedule default
maxM 500
maxiter 30
mps_tags KET2

proj_mps_tags KET1
proj_weights 5
```

Then we compute the second excited state using the following input file `dmrg-3.conf`:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1

schedule default
maxM 500
maxiter 30
mps_tags KET3

proj_mps_tags KET1 KET2
proj_weights 5 5
```

And so on.

This will generate the following output:

```
$ grep Energy dmrg-*.out
dmrg-1.out:DMRG Energy =  -75.728342508616663
dmrg-2.out:DMRG Energy =  -75.638961566176221
dmrg-3.out:DMRG Energy =  -75.629597871820607
dmrg-4.out:DMRG Energy =  -75.467766576734363
dmrg-5.out:DMRG Energy =  -75.350470798772307
dmrg-6.out:DMRG Energy =  -75.312672909521751
```

Mixed with State-Average

The above approach can also be used together with the state-average approach. Namely, we can first compute the two lowest states, then we compute the next three lowest states, by projecting out the two lowest states. The MPS to be projected must not be in state-averaged format, so we need to use the `split_states` keyword to break state-averaged MPS into individual MPSs, so that they can be used for projection in the subsequent calculations.

Currently, this type of state-average calulcation cannot be used together with multiple targets.

We first get the two lowest states using the following input file `dmrg-1.conf`:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1
nroots 2
weights 0.5 0.5

schedule default
maxM 500
maxiter 30
mps_tags KET

copy_mps
split_states
```

After this is finished, we compute the next three states using the following input file `dmrg-2.conf`:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1
nroots 3
weights 0.5 0.5 0.5
```

(continues on next page)

(continued from previous page)

```
schedule default
maxM 500
maxiter 30
mps_tags EXKET

proj_mps_tags KET-0 KET-1
proj_weights 5 5

copy_mps
split_states
```

After this is finished, we compute the next one state using the following input file `dmrg-3.conf`:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1

schedule default
maxM 500
maxiter 30
mps_tags EXXKET

proj_mps_tags KET-0 KET-1 EXKET-0 EXKET-1 EXKET-2
proj_weights 5 5 5 5 5
```

This will generate the following output:

```
$ grep DW dmrg-1.out | tail -1
Time elapsed =      5.461 | E[  2] =     -75.7279224622    -75.6386156808 | DE = 9.
↪32e-06 | DW = 8.33e-06
$ grep DW dmrg-2.out | tail -1
Time elapsed =     13.165 | E[  3] =     -75.6290377907    -75.4669665917    -75.
↪3494878435 | DE = 8.63e-07 | DW = 8.45e-05
$ grep DW dmrg-3.out | tail -1
Time elapsed =      8.651 | E =     -75.3126745298 | DE = -6.24e-07 | DW = 3.79e-15
```

3.3.7 n-Particle Reduced Density Matrix

The 1-, 2-, 3-, and 4-particle DMRG reduced density matrix for a particular state can be calculated using the keywords `onepdm`, `twopdm`, `threepdm` and `fourpdm`. The reduced density matrix calculation can be done with either `onedot` or `twodot` keywords.[Page 19, 1]

Note: Most of the time, only `onedot` density matrix calculation makes sense, since the MPS should not change during the sweep.

Density matrices of the n-th state are calculated and stored in a `numpy` binary file named `1pdm-n-n.npy`, `2pdm-n-n.npy`, `3pdm-n-n.npy`, etc. (in the scratch folder), respectively, starting with `n = 0`. If there is only one root, the files are named `1pdm.npy`, `2pdm.npy`, `3pdm.npy`, etc. respectively.

The following input file computes the energy and 2-particle density matrix for the ground state:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1

schedule default
maxM 500
maxiter 30

twopdm
num_thrds 16
```

The 2-particle density matrix file can be loaded using the following python script:

```
>>> import numpy as np
>>> _2pdm = np.load('./nodex/2pdm.npy')
>>> print(_2pdm.shape)
(3, 26, 26, 26, 26)
```

The following input file computes the energy and 2-particle density matrix for two state-averaged A_{1g} states:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1
nroots 2
weights 0.5 0.5
```

(continues on next page)

(continued from previous page)

```
schedule default
maxM 500
maxiter 30

twopdm
num_thrds 16
```

The 2-particle density matrix file for the first state can be loaded using the following python script:

```
>>> import numpy as np
>>> n = 0
>>> _2pdm = np.load('./nodex/2pdm-%d-%d.npy' % (n, n))
>>> print(_2pdm.shape)
(3, 26, 26, 26, 26)
```

The 1-particle density matrix (in both the non-spin-adapted and spin-adapted mode) is stored as an array with the shape $[2, n, n]$, where `n` is the number of spatial orbitals, and the two components with indicies $[:, a, b]$ are for $\langle a^\dagger_{a\alpha} a_{b\alpha} \rangle$, and $\langle a^\dagger_{a\beta} a_{b\beta} \rangle$, respectively.

The 2-particle density matrix (in both the non-spin-adapted and spin-adapted mode) is stored as an array with the shape $[3, n, n, n, n]$, where the three components with indicies $[:, a, b, c, d]$ are for $\langle a^\dagger_{a\alpha} a^\dagger_{b\alpha} a_{c\alpha} a_{d\alpha} \rangle$, $\langle a^\dagger_{a\alpha} a^\dagger_{b\beta} a_{c\beta} a_{d\alpha} \rangle$, and $\langle a^\dagger_{a\beta} a^\dagger_{b\beta} a_{c\beta} a_{d\beta} \rangle$, respectively.

The 3-particle density matrix in the spin-adapted mode is stored as the spin-traced format with the shape $[n, n, n, n, n, n]$, defined as

$$\text{3pdm}[a, b, c, d, e, f] := \sum_{\sigma\tau\lambda} a^\dagger_{a\sigma} a^\dagger_{b\tau} a^\dagger_{c\lambda} a_{d\lambda} a_{e\tau} a_{f\sigma}$$

The 3-particle density matrix in the non-spin-adapted mode is stored as an array with the shape $[4, n, n, n, n, n, n]$, where the four components with indicies $[:, a, b, c, d, e, f]$ are for $\langle a^\dagger_{a\alpha} a^\dagger_{b\alpha} a^\dagger_{c\alpha} a_{d\alpha} a_{e\alpha} a_{f\alpha} \rangle$, $\langle a^\dagger_{a\alpha} a^\dagger_{b\alpha} a^\dagger_{c\beta} a_{d\beta} a_{e\alpha} a_{f\alpha} \rangle$, $\langle a^\dagger_{a\alpha} a^\dagger_{b\beta} a^\dagger_{c\beta} a_{d\beta} a_{e\beta} a_{f\alpha} \rangle$, and $\langle a^\dagger_{a\beta} a^\dagger_{b\beta} a^\dagger_{c\beta} a_{d\beta} a_{e\beta} a_{f\beta} \rangle$, respectively.

The 4-particle density matrix in the spin-adapted mode is stored as the spin-traced format with the shape $[n, n, n, n, n, n, n, n]$, defined as

$$\text{4pdm}[a, b, c, d, e, f, g, h] := \sum_{\sigma\tau\lambda\mu} a^\dagger_{a\sigma} a^\dagger_{b\tau} a^\dagger_{c\lambda} a^\dagger_{d\mu} a_{e\mu} a_{f\lambda} a_{g\tau} a_{h\sigma}$$

The 4-particle density matrix in the non-spin-adapted mode is stored as an array with the shape $[5, n, n, n, n, n, n, n, n]$, where the five components with indicies $[:, a, b, c, d, e, f, g, h]$ are for $\langle a^\dagger_{a\alpha} a^\dagger_{b\alpha} a^\dagger_{c\alpha} a^\dagger_{d\alpha} a_{e\alpha} a_{f\alpha} a_{g\alpha} a_{h\alpha} \rangle$, $\langle a^\dagger_{a\alpha} a^\dagger_{b\alpha} a^\dagger_{c\alpha} a^\dagger_{d\beta} a_{e\beta} a_{f\alpha} a_{g\alpha} a_{h\alpha} \rangle$, $\langle a^\dagger_{a\alpha} a^\dagger_{b\alpha} a^\dagger_{c\beta} a^\dagger_{d\beta} a_{e\beta} a_{f\beta} a_{g\alpha} a_{h\alpha} \rangle$, $\langle a^\dagger_{a\alpha} a^\dagger_{b\beta} a^\dagger_{c\beta} a^\dagger_{d\beta} a_{e\beta} a_{f\beta} a_{g\beta} a_{h\alpha} \rangle$, and $\langle a^\dagger_{a\beta} a^\dagger_{b\beta} a^\dagger_{c\beta} a^\dagger_{d\beta} a_{e\beta} a_{f\beta} a_{g\beta} a_{h\beta} \rangle$, respectively.

In the general spin orbital mode (with the keyword `use_general_spin`), the 1-, 2-, 3-, and 4-particle density matrices are stored with the shape $[1, n, n]$, $[1, n, n, n, n]$, $[1, n, n, n, n, n, n]$, and $[1, n, n, n, n, n, n, n, n]$ respectively, where `n` is the number of spin orbitals. The content is the expectation value for $\langle a^\dagger_a a_b \rangle$, $\langle a^\dagger_a a^\dagger_b a_c a_d \rangle$, $\langle a^\dagger_a a^\dagger_b a^\dagger_c a_d a_e a_f \rangle$, and $\langle a^\dagger_a a^\dagger_b a^\dagger_c a^\dagger_d a_e a_f a_g a_h \rangle$, respectively.

3.3.8 n-Particle Transition Reduced Density Matrix

The 1-, 2-, 3- and 4-particle DMRG transition density matrix can be calculated using the keywords `tran_onepdm`, `tran_twopdm`, `tran_threepdm` and `tran_fourpdm`.

Transition density matrices between the m-th (bra) and n-th (ket) states are calculated and stored in a `numpy` binary file named `1pdm-m-n.npy`, `2pdm-m-n.npy`, etc. (in the scratch folder), respectively, starting with `m = n = 0`.

The following input file computes the 2-particle transition density matrix for two state-averaged A_{1g} states:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1
nroots 2
weights 0.5 0.5

schedule default
maxM 500
maxiter 30

tran_twopdm
num_thrds 16
```

Note: There can be a overall undetermined +1/-1 factor in Transition density matrices due to the relative phase in two MPSs.

The following input file computes the state-specific 2-particle transition density matrix for two refined A_{1g} states:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1
nroots 2
weights 0.5 0.5
statespecific

schedule default
maxM 500
maxiter 30
```

(continues on next page)

(continued from previous page)

```
tran_twopdm
num_thrds 16
```

The transition density matrices between states with different point group irreducible representations are also available by simply adding the keyword `tran_twopdm` after the corresponding multi-target state-averaged calculation.[Page 19, 1]

3.3.9 Restart DMRG Energy Calculation

DMRG energy calculations can be restarted, using the MPS (stored in scratch folder) generated in the previous calculation, by specifying the keyword `fullrestart`. If the previous calulcation stopped during the middle of a sweep, it will be restarted from the middle of a sweep.

Alternatively, the user can also set a directory for storing MPS after each sweep using the keyword `restart_dir`.[Page 19, 1] When restarting, the MPS data and `mps_info.bin` in the scratch folder should be copied from the `restart_dir` to the scartch folder of the restarting calculation.

The keyword `restart_dir_per_sweep` can be used to save a copy of MPS for each sweep. The MPS from different sweeps will be put into different folders (by adding suffix to the given direcotry).

You may need to change the (custom) scheudle in the input file so that the sweeps (with smaller bond dimension) finished in previous calculations will not be repeated, when you are restarting an interrupted calculation.

The following input file restarts an interrupted calculation:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

fullrestart
```

3.3.10 Load MPS for Density Matrix Calculation

The density matrix and transition density matrix calculation can be carried out separately, by restarting from an existing MPS, state-averged MPSs or state-specific MPSs (stored in scartch folder from a previous DMRG energy calculation).

Assuming a previous ground-state energy calculation has been finished, the following input file computes the 2-particle density matrix for the ground-state (loaded from scratch folder):

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

restart_twopdm
```

Assuming a previous state-averaged energy calculation has been finished, the following input file computes the 2-particle transition density matrix for two state-averaged A_{1g} states (loaded from scratch folder):

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1
nroots 2
weights 0.5 0.5

hf_occ integral
schedule default
maxM 500
maxiter 30

restart_tran_twopdm
```

Now we explain how to compute 2-particle transition density matrix for bra and ket states belonging to different point group irreducible representations. We consider the A_{1g} (bra) and B_{3u} (ket) states.

The following input file computes the energy for a single B_{3u} state in D_{2h} point group. The keyword `mps_tags` can be used to assign a tag to the mps for later reference:[Page 19, 1]

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 2

hf_occ integral
schedule default
maxM 500
maxiter 30

mps_tags KET
num_thrds 16
```

The following input file computes the energy for a single A_{1g} state in D_{2h} point group:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

mps_tags BRA
num_thrds 16
```

The output looks like the following:

```
$ grep Energy dmrg-1.out
DMRG Energy =  -75.675393353797631
$ grep Energy dmrg-2.out
DMRG Energy =  -75.728342388135175
```

The following input file computes the 2-particle transition density matrix for the two states:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1
```

(continues on next page)

(continued from previous page)

```
mps_tags BRA KET

hf_occ integral
schedule default
maxM 500
maxiter 30
restart_tran_twopdm
num_thrds 16
```

Note that in the above input file, keywords such as `nelec`, `spin`, `irrep`, and `nroots` will be unimportant. The keyword `mps_tags` lists the tags for the MPSs that should be loaded.[Page 19, 1]

3.3.11 Diagonal 2-Particle Density Matrix

Since the full two-particle density matrix calculation can be expensive for some systems, it is possible to calculate only the diagonal parts, which is much cheaper, using the keywords `restart_diag_twopdm` or `diag_twopdm`.[Page 19, 1] The time cost for diagonal 2pdm is roughly 2 times of the cost of 1pdm.

Note that `diag_twopdm` implies `onepdm` and `correlation`. The diagonal 2pdm is defined as:

$$
\begin{aligned}
e_{pqqp} &\equiv \sum_{\sigma\tau} \langle a^\dagger_{p\sigma} a^\dagger_{q\tau} a_{q\tau} a_{p\sigma} \rangle = -\sum_{\sigma\tau} \langle a^\dagger_{p\sigma} a^\dagger_{q\tau} a_{p\sigma} a_{q\tau} \rangle = \sum_{\sigma\tau} \langle a^\dagger_{p\sigma} a_{p\sigma} a^\dagger_{q\tau} a_{q\tau} \rangle - \delta_{pq} \sum_{\sigma} \langle a^\dagger_{p\sigma} a_{q\sigma} \rangle \\
&= \sum_{\sigma\tau} \langle n_{p\sigma} n_{q\tau} \rangle - \delta_{pq} \sum_{\sigma} \langle a^\dagger_{p\sigma} a_{q\sigma} \rangle \\
e_{pqpq} &\equiv \sum_{\sigma\tau} \langle a^\dagger_{p\sigma} a^\dagger_{q\tau} a_{p\tau} a_{q\sigma} \rangle = -\sum_{\sigma\tau} \langle a^\dagger_{p\sigma} a_{p\tau} a^\dagger_{q\tau} a_{q\sigma} \rangle + \delta_{pq} \sum_{\sigma\tau} \langle a^\dagger_{p\sigma} a_{q\sigma} \rangle \\
&= -\sum_{\sigma\tau} \langle a^\dagger_{p\sigma} a_{p\tau} a^\dagger_{q\tau} a_{q\sigma} \rangle + 2\delta_{pq} \sum_{\sigma} \langle a^\dagger_{p\sigma} a_{q\sigma} \rangle
\end{aligned}
$$

The computed diagonal 2pdm will be stored as `e_pqqp.npy` and `e_pqpq.npy` in scratch folder.

If one also computed the full 2pdm using the keyword `twopdm` or `restart_twopdm`, we can verify that its diagonal part matches the `e_pqqp.npy` and `e_pqpq.npy` obtained here:

```
>>> import numpy as np
>>> _2pdm = np.load('./nodex/2pdm.npy')
>>> print(_2pdm.shape)
(3, 26, 26, 26, 26)
>>> _e_pqqp = np.load('./nodex/e_pqqp.npy')
>>> _e_pqpq = np.load('./nodex/e_pqpq.npy')
>>> _2pdm_spat = _2pdm[0] + 2 * _2pdm[1] + _2pdm[2]
>>> _2pdm_spat_pqqp = np.einsum('pqqp->pq', _2pdm_spat)
>>> _2pdm_spat_pqpq = np.einsum('pqpq->pq', _2pdm_spat)
>>> print(np.linalg.norm(_e_pqqp - _2pdm_spat_pqqp))
3.28666776770176e-14
```

(continues on next page)

(continued from previous page)

```
>>> print(np.linalg.norm(_e_pqpq - _2pdm_spat_pqpq))
1.6947732597975102e-14
```

3.3.12 Custom Sweep Schedule

The sweep schedule defines number of the renormalized states M kept , the convergence threshold for Davidson algorithm (in the unit of norm^2), and the noise (in the unit of norm^2) in successive DMRG sweeps. For finer control over the sweeps, customized sweep schedule should be used.

The following input file computes the ground state energy using a custom sweep schedule:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1

hf_occ integral
schedule
0  100  1E-4  1E-3
4  250  1E-4  1E-3
8  400  1E-5  1E-4
10 600  1E-6  1E-5
12 800  1E-7  1E-6
14 1000 1E-8  1E-7
16 1000 1E-8  0E+0
end
twodot_to_onedot 18
maxiter 100
sweep_tol 1E-9
```

In the above input file, `twodot_to_onedot` specifies the sweep at which the switch is made from a 2-site to a 1-site DMRG algorithm (counting from 0). `maxiter` gives the maximum number of sweep iterations to be performed. `sweep_tol` gives the final tolerance on the DMRG energy, and is analogous to an energy convergence threshold in other quantum chemistry methods.

In the above input file, between `schedule` and `end` each line has four values. They are corresponding to starting sweep iteration (counting from zero), MPS bond dimension, tolerance for the Davidson iteration, and noise, respectively. Starting sweep iteration is the sweep iteration in which the given parameters in the line should take effect.

This will generate the following output:

```
$ grep DW dmrg.out
Time elapsed =      1.686 | E =     -74.1599100997 | DW = 4.86e-05
```

(continues on next page)

(continued from previous page)

```
Time elapsed =      3.332 | E =     -74.6555553068 | DE = -4.96e-01 | DW = 7.28e-05
Time elapsed =      4.461 | E =     -75.6224601188 | DE = -9.67e-01 | DW = 1.55e-04
Time elapsed =      5.648 | E =     -75.6302268887 | DE = -7.77e-03 | DW = 1.26e-04
Time elapsed =      7.491 | E =     -75.6347292246 | DE = -4.50e-03 | DW = 6.46e-05
Time elapsed =     10.732 | E =     -75.6367873793 | DE = -2.06e-03 | DW = 2.96e-05
Time elapsed =     13.383 | E =     -75.6372588510 | DE = -4.71e-04 | DW = 1.01e-04
Time elapsed =     16.138 | E =     -75.6375874124 | DE = -3.29e-04 | DW = 3.83e-05
Time elapsed =     20.541 | E =     -75.6687725683 | DE = -3.12e-02 | DW = 8.76e-06
Time elapsed =     26.404 | E =     -75.7265879915 | DE = -5.78e-02 | DW = 9.21e-06
Time elapsed =     36.001 | E =     -75.7282887562 | DE = -1.70e-03 | DW = 3.43e-06
Time elapsed =     47.351 | E =     -75.7283943399 | DE = -1.06e-04 | DW = 3.04e-06
Time elapsed =     64.673 | E =     -75.7284858001 | DE = -9.15e-05 | DW = 1.24e-06
Time elapsed =     86.412 | E =     -75.7285031554 | DE = -1.74e-05 | DW = 1.21e-06
Time elapsed =    118.443 | E =     -75.7285302492 | DE = -2.71e-05 | DW = 4.82e-07
Time elapsed =    158.894 | E =     -75.7285335786 | DE = -3.33e-06 | DW = 5.44e-07
Time elapsed =    176.071 | E =     -75.7285376489 | DE = -4.07e-06 | DW = 5.73e-07
Time elapsed =    191.672 | E =     -75.7285377336 | DE = -8.46e-08 | DW = 5.76e-07
Time elapsed =     10.790 | E =     -75.7285342605 | DW = 1.47e-16
Time elapsed =     21.186 | E =     -75.7285342992 | DE = -3.87e-08 | DW = 3.21e-14
Time elapsed =     31.924 | E =     -75.7285343224 | DE = -2.32e-08 | DW = 3.07e-17
Time elapsed =     42.348 | E =     -75.7285343375 | DE = -1.51e-08 | DW = 8.17e-15
Time elapsed =     53.073 | E =     -75.7285343475 | DE = -9.98e-09 | DW = 4.35e-17
Time elapsed =     63.362 | E =     -75.7285343571 | DE = -9.58e-09 | DW = 6.64e-16
Time elapsed =     73.965 | E =     -75.7285343630 | DE = -5.87e-09 | DW = 3.96e-17
Time elapsed =     84.094 | E =     -75.7285343661 | DE = -3.17e-09 | DW = 1.14e-16
Time elapsed =     94.525 | E =     -75.7285343678 | DE = -1.71e-09 | DW = 1.34e-16
Time elapsed =    104.658 | E =     -75.7285343721 | DE = -4.29e-09 | DW = 2.45e-16
Time elapsed =    114.925 | E =     -75.7285343746 | DE = -2.44e-09 | DW = 1.38e-16
Time elapsed =    124.710 | E =     -75.7285343763 | DE = -1.76e-09 | DW = 3.03e-16
Time elapsed =    135.115 | E =     -75.7285343763 | DE = 5.68e-14 | DW = 2.24e-17
```

3.3.13 Sweep Energy Extrapolation

In practice the sweep energy converges almost linearly as a function of the "maximum discarded weight". Therefore, it is convenient to use the "maximum discarded weight" quantity as an estimate of the error of the DMRG calculation. It is recommended to use the 2-site algorithm for energy extrapolation since the 2-site DMRG wavefunction provides additional variational freedom over the 1-site DMRG wavefunction. A strong deviation from a linear function (e.g. a plateau behavior followed by a sudden drop of the energy as a function of discarded weight) indicates that the DMRG was stuck in a local minimum.

One can use restart a converged DMRG calculation with a "reverse schedule" to generate data for energy extrapolation. This can guarentee that the energy for each different MPS bond dimension is fully converged and not representing any local minima.

The following input file restarts the previous calculation using a custom reverse sweep schedule:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1

hf_occ integral
twodot
schedule
0  800  1E-8  0E+0
4  600  1E-8  0E+0
8  400  1E-8  0E+0
12 200  1E-8  0E+0
end
maxiter 16
sweep_tol 0.0
fullrestart
```

This will generate the following output (`dmrg-2.out`):

```
$ grep DW dmrg-2.out
Time elapsed =     12.597 | E =     -75.7285358881 | DW = 1.75e-06
Time elapsed =     23.720 | E =     -75.7285188420 | DE = 1.70e-05 | DW = 1.42e-06
Time elapsed =     33.955 | E =     -75.7285186195 | DE = 2.23e-07 | DW = 1.35e-06
Time elapsed =     44.842 | E =     -75.7285186529 | DE = -3.34e-08 | DW = 1.34e-06
Time elapsed =     52.432 | E =     -75.7285113908 | DE = 7.26e-06 | DW = 4.98e-06
Time elapsed =     59.530 | E =     -75.7284626837 | DE = 4.87e-05 | DW = 3.66e-06
Time elapsed =     66.036 | E =     -75.7284622858 | DE = 3.98e-07 | DW = 3.49e-06
Time elapsed =     73.045 | E =     -75.7284623697 | DE = -8.39e-08 | DW = 3.47e-06
Time elapsed =     77.523 | E =     -75.7284421278 | DE = 2.02e-05 | DW = 1.71e-05
Time elapsed =     81.396 | E =     -75.7282631341 | DE = 1.79e-04 | DW = 1.11e-05
Time elapsed =     85.001 | E =     -75.7282618298 | DE = 1.30e-06 | DW = 1.02e-05
Time elapsed =     88.824 | E =     -75.7282620286 | DE = -1.99e-07 | DW = 1.02e-05
Time elapsed =     91.267 | E =     -75.7282077342 | DE = 5.43e-05 | DW = 1.04e-04
Time elapsed =     93.148 | E =     -75.7270840401 | DE = 1.12e-03 | DW = 5.65e-05
Time elapsed =     95.144 | E =     -75.7270844505 | DE = -4.10e-07 | DW = 5.01e-05
Time elapsed =     96.921 | E =     -75.7270854757 | DE = -1.03e-06 | DW = 4.85e-05
```

Sweep energy extrapolation can be plotted using the following python script:

```
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats

fname = 'dmrg-2.out'
out = open(fname, 'r').readlines()
```

(continues on next page)

(continued from previous page)

```
eners, dws = [], []
for l in out:
    if "DW" in l:
        eners.append(float(l.split()[7]))
        dws.append(float(l.split()[-1]))

eners, dws = eners[3::4], dws[3::4]
reg = scipy.stats.linregress(dws, eners)
x_reg = np.array([0, 1E-4])

emin, emax = min(eners), max(eners)
de = emax - emin
plt.plot(x_reg, reg.intercept + reg.slope * x_reg, '--', linewidth=1, color='#5FA8AB
↪')
plt.plot(dws, eners, 'o', color='#38686A', markerfacecolor='white', markersize=5)
plt.text(2E-6, emax, "$E(M=\\infty) = %.6f \pm %.6f \\mathrm{\\ Hartree}$" %
    (reg.intercept, abs(reg.intercept - emin) / 5), color='#38686A', fontsize=12)
plt.text(2E-6, emax - de * 0.1, "$R^2 = %.6f$" % (reg.rvalue ** 2),
    color='#38686A', fontsize=12)
plt.xlim((0, 5E-5))
plt.ylim((emin - de * 0.1, emax + de * 0.1))
plt.xlabel("Largest Discarded Weight")
plt.ylabel("Sweep Energy (Hartree)")
plt.subplots_adjust(left=0.16, bottom=0.1, right=0.95, top=0.95)
plt.savefig("extra.png", dpi=600)
```

Alternatively, the keyword `extrapolation` can be added to the previous script, so that the extrapolation energy will be printed and the figure named `extrapolation.png` will be saved in the scartch folder.

The script will generate the following figure:

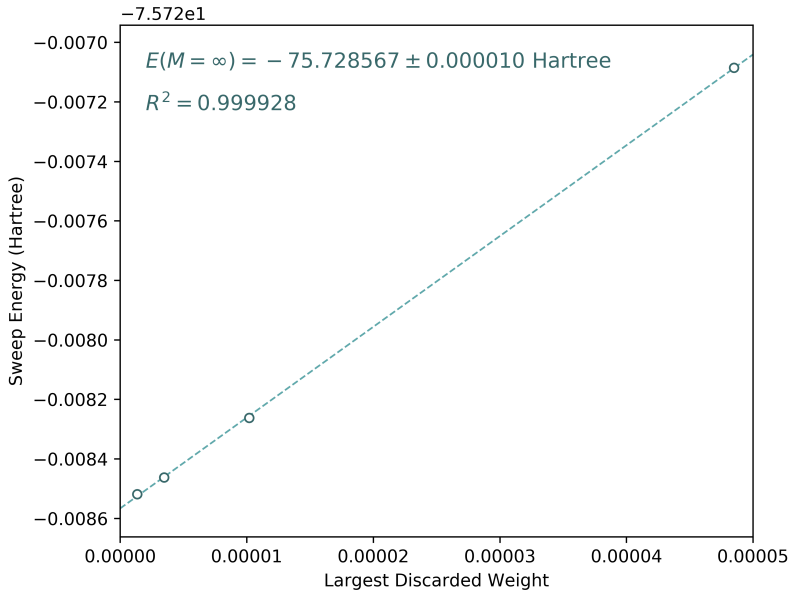
In the above script, we have used the largest discarded weights and associated sweep energies in the last sweep iteration of each bond dimension ($M = 800, 600, 400, 200$) to make linear regression. The extrapolated DMRG sweep energy is -75.728567 Hartree.

3.4 Input File: Advanced Usage

3.4.1 Orbital Rotation

In this calculation we illustrate how to compute the ground state MPS in the given set of orbitals, find the (new) DMRG natural orbitals, transform integrals to new orbitals, transform the ground state MPS to new orbitals, and finally evaluate the energy of the transformed MPS in the new orbitals to verify the quality of the transformed MPS.

First, we compute the energy and 1-particle density matrix for the ground state using the following



input file:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

onepdm
irrep_reorder
```

Note that we use the keyword `irrep_reorder` to reorder the orbitals so that orbitals belonging to the same point group irrep are grouped together. This can make the orbital rotation more local.

The DMRG occupation number (in original ordering) will be printed at the end of the calculation:

```
$ grep OCC dmrg-1.out
DMRG OCC =   1.957 1.625 1.870 1.870 0.361 0.098 0.098 0.006 0.008 0.008 0.008 0.013␣
↪0.014 0.014 0.011 0.006 0.006 0.006 0.005 0.005 0.002 0.002 0.002 0.001 0.001 0.001
$ grep Energy dmrg-1.out
DMRG Energy =  -75.728467269121097
```

Second, we use the keyword `nat_orbs` to compute the natural orbitals. The value of the keyword `nat_orbs` specifies the filename for storing the rotated integrals (FCIDUMP). If no value is associated with the keyword `nat_orbs`, the rotated integrals will not be computed. The keyword

nat_orbs can only be used together with restart_onepdm or onepdm, since natural orbitals are found by diagonalizing 1-particle density matrix.

The following input file is used for this step (it can also be combined with the previous calculation):

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

restart_onepdm
nat_orbs C2.NAT.FCIDUMP
nat_km_reorder
nat_positive_def
irrep_reorder
```

Where the optional keyword nat_km_reorder can be used to remove the artificial reordering in the natural orbitals using Kuhn-Munkres algorithm. The optional keyword nat_positive_def can be used to avoid artificial rotation in the logarithm of the rotation matrix, by make the rotation matrix quasi-positive-definite, with "quasi" in the sense that the rotation matrix is not Hermitian. The two options may be good for weakly correlated systems, but have limited effects for highly correlated systems (but for highly correlated systems it is also recommended to be used).

The occupation number in natural orbitals will be printed at the end of the calculation:

```
$ grep OCC dmrg-2.out
DMRG OCC =   1.957 1.625 1.870 1.870 0.361 0.098 0.098 0.006 0.008 0.008 0.008 0.013␣
↪0.014 0.014 0.011 0.006 0.006 0.006 0.005 0.005 0.002 0.002 0.002 0.001 0.001 0.001
REORDERED OCC =   1.957 0.002 0.361 0.006 0.013 0.008 0.002 0.006 0.011 0.001 0.006␣
↪1.625 0.008 1.870 0.005 0.098 0.001 0.014 0.005 1.870 0.008 0.001 0.014 0.098 0.
↪006 0.002
NAT OCC =   0.000465 0.003017 0.006424 0.007848 0.360936 1.968407 0.000081 0.000916␣
↪0.001991 0.004082 0.015623 1.628182 0.003669 0.008706 1.870680 0.000424 0.002862 0.
↪110463 0.003667 0.008705 1.870678 0.000424 0.002862 0.110480 0.006422 0.001989
```

With the optional keyword nat_km_reorder there will be an extra line:

```
REORDERED NAT OCC =   1.968407 0.000465 0.360936 0.006424 0.007848 0.003017 0.001991␣
↪0.000081 0.004082 0.000916 0.015623 1.628182 0.008706 1.870680 0.003669 0.110463 0.
↪000424 0.002862 0.003667 1.870678 0.008705 0.000424 0.002862 0.110480 0.006422 0.
↪001989
```

The rotation matrix for natural orbitals, the logarithm of the rotation matrix, and the occupation number in natural orbitals are stored as `nat_rotation.npy`, `nat_kappa.npy`, `nat_occs.npy` in scartch folder, respectively. In this example, the rotated integral is stored as `C2.NAT.FCIDUMP` in the working directory.

Third, we load the MPS in the old orbitals and transform it into the new orbitals. This is done using time evolution. The keyword `delta_t` is used to set a time step and indicate that this is a time evolution calculation. The keyword `orbital_rotation` is used to indicate that the operator (exponentiated) applied into the MPS should be the orbital rotation operator (constructed from `nat_kappa.npy` saved in the previous step).

Typically, a large bond dimension should be used depending how non-local the orbital rotation operator is. The `target_t` for orbital rotation is automatically set to 1.

The following input file is used for this step:

```
sym d2h

nelec 8
spin 0
irrep 1

schedule
    0 1000 0 0
end

mps_tags BRA
orbital_rotation
delta_t 0.05
outputlevel 1
noreorder
```

Note that `noreorder` must be used for orbital rotation. The orbital reordering in previous step has already been taken into account.

The keyword `te_type` can be used to set the time-evolution algorithm. The default is `rk4`, which is the original time-step-targeting (TST) method. Another possible choice is `tdvp`, which is the time dependent variational principle with the projector-splitting (TDVP-PS) algorithm.

The output looks like the following:

```
$ grep DW dmrg-3.out
Time elapsed =      2.263 | E =       0.0000000000 | Norm^2 =       0.9999999999 |␣
↪DW = 1.76e-10
Time elapsed =      4.910 | E =      -0.0000000000 | Norm^2 =       0.9999999997 |␣
↪DW = 1.43e-10
Time elapsed =      1.663 | E =      -0.0000000000 | Norm^2 =       0.9999999988 |␣
↪DW = 4.46e-10
Time elapsed =      3.475 | E =       0.0000000000 | Norm^2 =       0.9999999983 |␣
↪DW = 2.50e-10
```

(continues on next page)

(continued from previous page)

```
... ...
Time elapsed =      3.011 | E =       0.0000000000 | Norm^2 =       0.9999999315 |␣
↪DW = 1.04e-09
Time elapsed =      4.753 | E =       0.0000000000 | Norm^2 =       0.9999999284 |␣
↪DW = 8.68e-10
Time elapsed =      1.786 | E =       0.0000000000 | Norm^2 =       0.9999999245 |␣
↪DW = 1.07e-09
Time elapsed =      3.835 | E =       0.0000000000 | Norm^2 =       0.9999999213 |␣
↪DW = 9.09e-10
```

Since in every time step an orthogonal transformation is applied on the MPS, the expectation value of the orthogonal transformation (printed as the energy expectation) calculated on the MPS should always be zero.

Note that largest discarded weight is `1.07e-09`, and the norm of MPS is not far away from 1. So the transormation should be relatively accurate.

Finally, we calculate the energy expectation value using the transformed integral (`C2.NAT.FCIDUMP`) and the transformed MPS (stored in the scratch folder), using the following input file:

```
sym d2h
orbitals C2.NAT.FCIDUMP

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

mps_tags BRA
restart_oh
restart_onepdm
noreorder
```

Note that `noreorder` must be used, since the MPS generated in the previous step is in unreordered natural orbitals. The keyword `restart_oh` will calculate the expectation value of the given Hamiltonian loaded from integrals on the MPS loaded from scartch folder.

We have the following output:

```
$ grep Energy dmrg-4.out
OH Energy =  -75.728457535820155
```

The difference compared to the energy generated in the first step `DMRG Energy = -75.728467269121097` is only 9.7E-6. One can increase the bond dimension in the evolution to make

this closer to the value printed in the first step.

3.4.2 MPS Transform

The MPS can be copied and saved using another tag. For SU2 (spin-adapted) MPS, it can also be transformed to SZ (non-spin-adapted) MPS and saved using another tag.

Limitations:

- Total spin zero spin-adapted MPS can be transformed directly.
- For non-zero total spin, the spin-adapted MPS must be in singlet embedding format. See next section.

First, we compute the energy for the spin-adapted ground state using the following input file:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
mps_tags KET
```

The following script will read the spin-adapted MPS and tranform it to a non-spin-adapted MPS:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
mps_tags KET
restart_copy_mps ZKET
trans_mps_to_sz
```

Here the keyword `restart_copy_mps` indicates that the MPS will be copied, associated with a value indicating the new tag for saving the copied MPS. If the keyword `trans_mps_to_sz` is present, the MPS will be transformed to non-spin-adapted before being saved.

Finally, we calculate the energy expectation value using non-spin-adapted formalism and the transformed MPS (stored in the scratch folder), using the following input file:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
mps_tags ZKET
restart_oh
nonspinadapted
```

Some reference outputs for this example:

```
$ grep Energy dmrg-1.out
DMRG Energy =  -75.728467269121083
$ grep MPS dmrg-2.out
MPS =  KRRRRRRRRRRRRRRRRRRRRRRRR 0 2
GS INIT MPS BOND DIMS =       1     3    10    35   120   263   326   500   500  ␣
↪500   500   500   500   500   500   500   500   500   498   500   407   219    94 ␣
↪  32    10     3     1
$ grep 'MPS\|Energy' dmrg-3.out
MPS =  KRRRRRRRRRRRRRRRRRRRRRRRR 0 2
GS INIT MPS BOND DIMS =       1     4    16    64   246   578   712  1114  1097 ␣
↪1102  1110  1121  1126  1130  1116  1111  1111  1107  1074  1103   895   444   186␣
↪   59    16     4     1
OH Energy =  -75.728467269120898
```

We can see that the transformation from SU2 to SZ is nearly exact, and the required bond dimension for the SZ MPS is roughly two times of the SU2 bond dimension.

3.4.3 Singlet Embedding

For spin-adapted calculation with total spin not equal to zero, there can be some convergence problem even if in one-site algorithm. One way to solve this problem is to use singlet embedding. In `StackBlock` singlet embedding is used by default. In `block2`, by default singlet embedding is not used. If one adds the keyword `singlet_embedding` to the input file, the singlet embedding scheme will be used. For most total spin not equal to zero calculation, singlet embedding may be more stable. One cannot calculate transition density matrix between states with different total spins using singlet embedding. To do that one can translate the MPS between singlet embedding format and non-singlet-embedding format.

When total spin is equal to zero, the keyword `singlet_embedding` will not have any effect. If restarting a calculation, normally, the keyword `singlet_embedding` is not required since the format of the MPS can be automatically recognized.

For translating SU2 MPS to SZ MPS with total spin not equal to zero, the SU2 MPS must be in singlet embedding format.

First, we compute the energy for the spin-adapted with non-zero total spin using the following input file:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 2
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
mps_tags KET
```

The above input file indicates that singlet embedding is not used. The output is:

```
$ grep 'MPS = ' dmrg-1.out
MPS =  CCRRRRRRRRRRRRRRRRRRRRRRR 0 2 < N=8 S=1 PG=0 >
$ grep Energy dmrg-1.out
DMRG Energy =  -75.423916647509742
```

Here the printed target quantum number of the MPS indicates that it is a triplet.

We can add the keyword `singlet_embedding` to do a singlet embedding calculation:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG
```

(continues on next page)

(continued from previous page)

```
nelec 8
spin 2
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
mps_tags SEKET
singlet_embedding
```

When singlet embedding is used, the output is:

```
$ grep 'MPS = ' dmrg-2.out
MPS =  CCRRRRRRRRRRRRRRRRRRRRRRR 0 2 < N=10 S=0 PG=0 >
$ grep Energy dmrg-2.out
DMRG Energy =  -75.423879916245895
```

Here the printed target quantum number of the MPS indicates that it is a singlet (including some ghost particles).

One can use the keywords `trans_mps_to_singlet_embedding` and `trans_mps_from_singlet_embedding` combined with `restart_copy_mps` or `copy_mps` to translate between singlet embedding and normal formats.

The following script transforms the MPS from singlet embedding to normal format:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 2
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
mps_tags SEKET
restart_copy_mps TKET
trans_mps_from_singlet_embedding
```

We can verify that the transformed non-singlet-embedding MPS has the same energy as the singlet

embedding MPS:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 2
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
mps_tags TKET
restart_oh
```

With the outputs:

```
$ grep 'MPS = ' dmrg-4.out
MPS =  KRRRRRRRRRRRRRRRRRRRRRRRR 0 2 < N=8 S=1 PG=0 >
$ grep Energy dmrg-4.out
OH Energy =  -75.423879916245824
```

The following script will read the spin-adapted singlet embedding MPS and tranform it to a non-spin-adapted MPS:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 2
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
mps_tags SEKET
restart_copy_mps ZKETM2
trans_mps_to_sz
resolve_twosz -2
normalize_mps
```

Here the keyword `resolve_twosz` indicates that the transformed SZ MPS will have projected spin 2

`* SZ = -2`. For this case since `2 * S = 2`, the possible values for `resolve_twosz` are `-2, 0, 2`. If the keyword `resolve_twosz` is not given, an MPS with ensemble of all possible projected spins will be produced (which is often not very useful). Getting one component of the SU2 MPS means that the SZ MPS will not have the same norm as the SU2 MPS. If the keyword `normalize_mps` is added, the transformed SZ MPS will be normalized. The keyword `normalize_mps` can only have effect when `trans_mps_to_sz` is present.

Finally, we calculate the energy expectation value using non-spin-adapted formalism and the transformed MPS (stored in the scratch folder), using the following input file:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin -2
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
mps_tags ZKETM2
restart_oh
nonspinadapted
```

Some reference outputs for this example:

```
$ grep MPS dmrg-6.out
MPS =  KRRRRRRRRRRRRRRRRRRRRRRRR 0 2 < N=8 SZ=-1 PG=0 >
GS INIT MPS BOND DIMS =       1    12    48   192   601  1145  1398  1474  1476 ␣
↪1468  1466  1441  1356  1316  1255  1240  1217  1206  1198  1176   904   422   183␣
↪    59    16     4     1
$ grep Energy dmrg-6.out
OH Energy =  -75.423879916245909
```

We can see that the transformation from SU2 to SZ is nearly exact. The other two components of the SU2 MPS will also have the same energy as this one.

3.4.4 CSF or Determinant Sampling

The overlap between the spin-adapted MPS and Configuration State Functions (CSFs), or between the non-spin-adapted MPS and determinants can be calculated. Since there are exponentially many CSFs or determinants (when the number of electrons is close to the number of orbitals), normally it only makes sense to sample CSFs or determinants with (absolute value of) the overlap larger than a threshold. The sampling is deterministic, meaning that all overlap above the given threshold will be printed.

The keyword `sample` or `restart_sample` can be used to sample CSFs or determinants after DMRG or from an MPS loaded from disk. The value associated with the keyword `sample` or `restart_sample` is the threshold for sampling.

Setting the threshold to zero is allowed, but this may only be useful for some very small systems.

Limitations: For non-zero total spin CSF sampling, the spin-adapted MPS must be in singlet embedding format. See the previous section.

The following is an example of the input file:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
mps_tags KET
sample 0.05
```

Some reference outputs for this example:

```
$ grep CSF dmrg-1.out
Number of CSF =         17 (cutoff =      0.05)
Sum of weights of sampled CSF =    0.909360149891891
CSF          0 20000000000202000002000000  =    0.828657540546610
CSF          1 20200000000002000002000000  =   -0.330323898091116
CSF          2 20+00000000+0200000-000-00  =   -0.140063445607095
CSF          3 20+00000000+0-0-0002000000  =   -0.140041987646036
... ...
CSF         16 200000000002000+0-02000000  =    0.050020205617060
```

When there are more than 50 determinants, only the first 50 with largest weights will be printed. The complete list of determinants and coefficients are stored in `sample-dets.npy` and `sample-vals`.

npy in the scratch folder, respectively.

So the restricted Hartree-Fock determinant/CSF has a very large coefficient (0.83).

To verify this, we can also directly compress the ground-state MPS to bond dimension 1, to get the CSF with the largest coefficient. Note that the compression method may converge to some other CSFs if there are many determinants with similar coefficients.

3.4.5 MPS Compression

MPS compression can be used to compress or fit a given MPS to a different (larger or smaller) bond dimension.

The following is an example of the input file for the compression (which will load the MPS obtailed from the previous ground-state DMRG):

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

nelec 8
spin 0
irrep 1

hf_occ integral
schedule
0  250  0 0
2  125  0 0
4   62  0 0
6   31  0 0
8   15  0 0
10   7  0 0
12   3  0 0
14   1  0 0
end
maxiter 16

compression
overlap
read_mps_tags KET
mps_tags BRA

irrep_reorder
```

Here the keyword `compression` indicates that this is a compression calculation. When the keyword `overlap` is given, the loaded MPS will be compressed, otherwise, the result of H|MPS> will be compressed. The tag of the input MPS is given by `read_mps_tags`, and the tag of the output MPS is given by `mps_tags`.

Some reference outputs for this example:

```
$ grep 'Compression overlap' dmrg-2.out
Compression overlap =    0.828657540546619
```

We can see that the value obtained from compression is very close to the sampled value. But when a lower bound of the overlap is known, the sampling method should be more reliable and efficient for obtaining the CSF with the largest weight.

If the CSF or determinat pattern is required, one can do a quick sampling on the compressed MPS using the keyword `restart_sample 0`.

If the given MPS has a very small bond dimension, or the target (output) MPS has a very large bond dimension (namely, "decompression"), one should use the keyword `random_mps_init` to allow a better random initial guess for the target MPS. Otherwise, the generated output MPS may be inaccurate.

3.4.6 LZ Symmetry

For diatomic molecules or model Hamiltonian with translational symmetry (such as 1D Hubbard model in momentum space), it is possible to utilize additional K space symmetry. To support the K space symmetry, the code must be compiled with the option `-DUSE_KSYMM=ON` (default).

One can add the keyword `k_symmetry` in the input file to use this additional symmetry. Point group symmetry can be used together with k symmetry. Therefore, even for system without K space symmetry, the calculation can still run as normal when the keyword `k_symmetry` is added. Note, however, the MPS or MPO generated from an input file with/without the keyword `k_symmetry`, cannot be reloaded with an input file without/with the keyword `k_symmetry`.

For molecules, the integral file (FCIDUMP file) must be generated in a special way so that the K/LZ symmetry can be used. the following python script can be used to generate the integral with $C_2 \otimes L_z$ symmetry:

```
import numpy as np
from functools import reduce
from pyscf import gto, scf, ao2mo, symm, tools, lib
from block2 import FCIDUMP, VectorUInt8, VectorInt

# adapted from https://github.com/hczhai/pyscf/blob/1.6/examples/symm/33-lz_adaption.
↪py
# with the sign of lz
def lz_symm_adaptation(mol):
    z_irrep_map = {} # map from dooh to lz
    g_irrep_map = {} # map from dooh to c2
    symm_orb_map = {} # orbital rotation
    for ix in mol.irrep_id:
        rx, qx = ix % 10, ix // 10
        g_irrep_map[ix] = rx & 4
        z_irrep_map[ix] = (-1) ** ((rx & 1) == ((rx & 4) >> 2)) * ((qx << 1) + ((rx &
↪ 2) >> 1))
```

(continues on next page)

(continued from previous page)

```
        if z_irrep_map[ix] == 0:
            symm_orb_map[(ix, ix)] = 1
        else:
            if (rx & 1) == ((rx & 4) >> 2):
                symm_orb_map[(ix, ix)] = -np.sqrt(0.5) * ((rx & 2) - 1)
            else:
                symm_orb_map[(ix, ix)] = -np.sqrt(0.5) * 1j
            symm_orb_map[(ix, ix ^ 1)] = symm_orb_map[(ix, ix)] * 1j

    z_irrep_map = [z_irrep_map[ix] for ix in mol.irrep_id]
    g_irrep_map = [g_irrep_map[ix] for ix in mol.irrep_id]
    rev_symm_orb = [np.zeros_like(x) for x in mol.symm_orb]
    for iix, ix in enumerate(mol.irrep_id):
        for iiy, iy in enumerate(mol.irrep_id):
            if (ix, iy) in symm_orb_map:
                rev_symm_orb[iix] = rev_symm_orb[iix] + symm_orb_map[(ix, iy)] * mol.
↪symm_orb[iiy]
    return rev_symm_orb, z_irrep_map, g_irrep_map

# copied from https://github.com/hczhai/pyscf/blob/1.6/pyscf/symm/addons.py#L29
# with the support for complex orbitals
def label_orb_symm(mol, irrep_name, symm_orb, mo, s=None, check=True, tol=1e-9):
    nmo = mo.shape[1]
    if s is None:
        s = mol.intor_symmetric('int1e_ovlp')
    s_mo = np.dot(s, mo)
    norm = np.zeros((len(irrep_name), nmo))
    for i, csym in enumerate(symm_orb):
        moso = np.dot(csym.conj().T, s_mo)
        ovlpso = reduce(np.dot, (csym.conj().T, s, csym))
        try:
            s_moso = lib.cho_solve(ovlpso, moso)
        except:
            ovlpso[np.diag_indices(csym.shape[1])] += 1e-12
            s_moso = lib.cho_solve(ovlpso, moso)
        norm[i] = np.einsum('ki,ki->i', moso.conj(), s_moso).real
    norm /= np.sum(norm, axis=0)  # for orbitals which are not normalized
    iridx = np.argmax(norm, axis=0)
    orbsym = np.asarray([irrep_name[i] for i in iridx])

    if check:
        largest_norm = norm[iridx,np.arange(nmo)]
        orbidx = np.where(largest_norm < 1-tol)[0]
        if orbidx.size > 0:
            idx = np.where(largest_norm < 1-tol*1e2)[0]
```

(continues on next page)

(continued from previous page)

```
            if idx.size > 0:
                raise ValueError('orbitals %s not symmetrized, norm = %s' %
                                 (idx, largest_norm[idx]))
            else:
                raise ValueError('orbitals %s not strictly symmetrized.',
                                 np.unique(orbidx))
    return orbsym

mol = gto.M(
    atom=[["C", (0, 0, 0)],
          ["C", (0, 0, 1.2425)]],
    basis='ccpvdz',
    symmetry='dooh')

mol.symm_orb, z_irrep, g_irrep = lz_symm_adaptation(mol)
mf = scf.RHF(mol)
mf.run()

h1e = mf.mo_coeff.conj().T @ mf.get_hcore() @ mf.mo_coeff
print('h1e imag = ', np.linalg.norm(h1e.imag))
assert np.linalg.norm(h1e.imag) < 1E-14
e_core = mol.energy_nuc()
h1e = h1e.real.ravel()
_eri = ao2mo.restore(1, mf._eri, mol.nao)
g2e = np.einsum('pqrs,pi,qj,rk,sl->ijkl', _eri,
    mf.mo_coeff.conj(), mf.mo_coeff, mf.mo_coeff.conj(), mf.mo_coeff, optimize=True)
print('g2e imag = ', np.linalg.norm(g2e.imag))
assert np.linalg.norm(g2e.imag) < 1E-14
print('g2e symm = ', np.linalg.norm(g2e - g2e.transpose((1, 0, 3, 2))))
print('g2e symm = ', np.linalg.norm(g2e - g2e.transpose((2, 3, 0, 1))))
print('g2e symm = ', np.linalg.norm(g2e - g2e.transpose((3, 2, 1, 0))))
g2e = g2e.real.ravel()

fcidump_tol = 1E-13
na = nb = mol.nelectron // 2
n_mo = mol.nao
h1e[np.abs(h1e) < fcidump_tol] = 0
g2e[np.abs(g2e) < fcidump_tol] = 0

orb_sym_z = label_orb_symm(mol, z_irrep, mol.symm_orb, mf.mo_coeff, check=True)
orb_sym_g = label_orb_symm(mol, g_irrep, mol.symm_orb, mf.mo_coeff, check=True)
print(orb_sym_z)

fcidump = FCIDUMP()
fcidump.initialize_su2(n_mo, na + nb, na - nb, 1, e_core, h1e, g2e)
```

(continues on next page)

(continued from previous page)

```
orb_sym_mp = VectorUInt8([tools.fcidump.ORBSYM_MAP['D2h'][i] for i in orb_sym_g])
fcidump.orb_sym = VectorUInt8(orb_sym_mp)
print('g symm error = ', fcidump.symmetrize(VectorUInt8(orb_sym_g)))

fcidump.k_sym = VectorInt(orb_sym_z)
fcidump.k_mod = 0
print('z symm error = ', fcidump.symmetrize(fcidump.k_sym, fcidump.k_mod))

fcidump.write('FCIDUMP')
```

Note that, if only the LZ symmetry is required, one can simply set `orb_sym_g[:] = 0`.

The following input file can be used to perform the calculation with $C_2 \otimes L_z$ symmetry:

```
sym d2h
orbitals FCIDUMP
k_symmetry
k_irrep 0

nelec 12
spin 0
irrep 1

hf_occ integral
schedule
0  500 1E-8 1E-3
4  500 1E-8 1E-4
8  500 1E-9 1E-5
12 500 1E-9 0
end
maxiter 30
```

Where the `k_irrep` can be used to set the eigenvalue of LZ in the target state. Note that it can be easier for the Davidson procedure to get stuck in local minima with high symmetry. It is therefore recommended to use a custom schedule with larger noise and smaller Davidson threshold.

Some reference outputs for this input file:

```
$ grep 'Time elapsed' dmrg-1.out | tail -1
Time elapsed =     73.529 | E =     -75.7291544157 | DE = -6.31e-07 | DW = 1.28e-05
$ grep 'DMRG Energy' dmrg-1.out
DMRG Energy =  -75.729154415733063
```

When there are too many orbitals, and the default `warmup fci` initial guess is used, the initial MPS can have very large bond dimension (especially when the LZ symmetry is used, since LZ is not a finite group) and the first sweep will take very long time.

One way to solve this is to limit the LZ to a finite group, using modular arithmetic. We can limit LZ to Z4 or Z2. The efficiency gain will be smaller, but the convergence may be more stable. The keyword `k_mod` can be used to set the modulus. When `k_mod = 0`, it is the original infinite LZ group.

The following input file can be used to perform the calculation with $C_2 \otimes Z_4$ symmetry:

```
sym d2h
orbitals FCIDUMP
k_symmetry
k_irrep 0
k_mod 4

nelec 12
spin 0
irrep 1

hf_occ integral
schedule
0  500 1E-8 1E-3
4  500 1E-8 1E-4
8  500 1E-9 1E-5
12 500 1E-9 0
end
maxiter 30
```

Some reference outputs for this input file:

```
$ grep 'Time elapsed' dmrg-2.out | tail -1
Time elapsed =    111.491 | E =     -75.7292222457 | DE = -8.17e-08 | DW = 1.28e-05
$ grep 'DMRG Energy' dmrg-2.out
DMRG Energy =  -75.729222245693876
```

Similarly, setting `k_mod 2` gives the following output:

```
$ grep 'Time elapsed' dmrg-3.out | tail -1
Time elapsed =    135.394 | E =     -75.7314583188 | DE = -3.97e-07 | DW = 1.49e-05
$ grep 'DMRG Energy' dmrg-3.out
DMRG Energy =  -75.731458318751280
```

3.4.7 Initial Guess with Occupation Numbers

Once can use `warmup occ` initial guess to solve the initial guess problem, where another keywrod `occ` should be used, followed by a list of (fractional) occupation numbers separated by the space character, to set the occupation numbers. The occupation numbers can be obtained from a DMRG calculation using the same integral with/without K symmetry (or some other methods like CCSD and MP2). If `onepdm` is in the input file, the occupation numbers will be printed at the end of the output.

The following input file will perform the DMRG calculation using the same integral without the K symmetry (but with C2 symmetry):

```
sym d2h
orbitals FCIDUMP

nelec 12
spin 0
irrep 1

hf_occ integral
schedule
0  500 1E-8 1E-3
4  500 1E-8 1E-4
8  500 1E-9 1E-5
12 500 1E-9 0
end
maxiter 30
onepdm
```

Some reference outputs for this input file:

```
$ grep 'Time elapsed' dmrg-1.out | tail -2 | head -1
Time elapsed =    190.549 | E =     -75.7314655815 | DE = -1.88e-07 | DW = 1.53e-05
$ grep 'DMRG Energy' dmrg-1.out
DMRG Energy =  -75.731465581478815
$ grep 'DMRG OCC' dmrg-1.out
DMRG OCC =   2.000 2.000 1.957 1.626 1.870 1.870 0.360 0.098 0.098 0.006 0.008 0.008␣
↪0.008 0.013 0.014 0.014 0.011 0.006 0.006 0.006 0.005 0.005 0.002 0.002 0.002 0.
↪001 0.001 0.001
```

The following input file will perform the DMRG calculation using the K symmetry, but with initial guess generated from occupation numbers:

```
sym d2h
orbitals FCIDUMP
k_symmetry
k_irrep 0
warmup occ
occ 2.000 2.000 1.957 1.626 1.870 1.870 0.360 0.098 0.098 0.006 0.008 0.008 0.008 0.
↪013 0.014 0.014 0.011 0.006 0.006 0.006 0.005 0.005 0.002 0.002 0.002 0.001 0.001␣
↪0.001
cbias 0.2

nelec 12
spin 0
irrep 1
```

(continues on next page)

(continued from previous page)

```
hf_occ integral
schedule
0  500 1E-8 1E-3
4  500 1E-8 1E-4
8  500 1E-9 1E-5
12 500 1E-9 0
end
maxiter 30
```

Here `cbias` is the keyword to add a constant bias to the occ, so that 2.0 becomes 2.0 - cbias, and 0.098 becomes 0.098 + cbias. Without the bias it is also easy to converge to a local minima.

Some reference outputs for this input file:

```
$ grep 'Time elapsed' dmrg-3.out | tail -1
Time elapsed =     55.938 | E =     -75.7244716369 | DE = -5.25e-07 | DW = 7.45e-06
$ grep 'DMRG Energy' dmrg-3.out
DMRG Energy =  -75.724471636942383
```

Here the calculation runs faster because the better initial guess, but the energy becomes worse.

3.4.8 Time Evolution

Now we give an example on how to do time evolution. The computation will apply $|MPS_{out}\rangle = \exp(-tH)|MPS_{in}\rangle$ (with multiple steps). When t is a real floating point value, we will do imaginary time evolution of the MPS (namely, optimizing to ground state or finite-temperature state). When t is a pure imaginary value, we will do real time evolution of the MPS (namely, solving the time dependent Schrodinger equation).

To get accurate results, the time step has to be sufficiently small. The keyword `delta_t` is used to set a time step Δt and indicate that this is a time evolution calculation. The keyword `target_t` is used to set a target "stopping" time, namely, the t. The "starting" time is considered as zero. Therefore, the number of time steps is computed as $nsteps = t/\Delta t$ and printed.

If `delta_t` is too big, the time step error will be large. If `delta_t` is small, for fixed target time we have to do more time steps, with MPS bond dimension truncation happening after each sweep. So if `delta_t` is too small, the accumulated bond dimension truncation error will be large. Some meaningful time steps may be 0.01 to 0.1.

Real Time Evolution

First, we do a state-averaged calculation for the lowest two states using the following input file:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG
nroots 2

hf_occ integral
schedule default
maxM 500
maxiter 30

noreorder
```

Note that the orbital reordering is disabled. The output:

```
$ grep elapsed dmrg-1.out | tail -1
Time elapsed =      5.762 | E[  2] =     -75.7268133875    -75.6376794953 | DE = -8.
↪89e-08 | DW = 6.38e-05
$ grep Final dmrg-1.out
Final canonical form =  LLLLLLLLLLLLLLLLLLLLLLLLJ 25
```

The energy of the MPS at the last site is actually -75.72629673 and -75.63717415, which are slightly different from the above values.

Second, we can use the following input file to load the state-averaged MPS and then split it into individual MPSs:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG
nroots 2

hf_occ integral
schedule default
maxM 500
maxiter 30

restart_copy_mps
split_states
trans_mps_to_complex
noreorder
```

Note that here `nroots` must be the same as the previous case (or smaller, but larger than one), otherwise the state-averaged MPS cannot be correctly loaded. The state-averaged MPS has the default tag KET. We use calculation type keyword `restart_copy_mps` to do this transformation. The new keyword `split_states` indicates that we want to split the MPS, this keyword should only be used together with `restart_copy_mps`. The extra keyword `trans_mps_to_complex` will further

make the MPS a complex MPS. This is required for real time evolution, where `delta_t` can be imaginary.

For imaginary time evolution and real `delta_t` and real `target_t`, everything will be real during the time evolution, so normally we do not need this extra keyword `trans_mps_to_complex` (but if you add it it is also okay).

The output looks like :

```
$ tail -7 dmrg-2.out
----- root =   0 /   2 -----
    final tag = KET-CPX-0
    final canonical form = LLLLLLLLLLLLLLLLLLLLLLLLT
----- root =   1 /   2 -----
    final tag = KET-CPX-1
    final canonical form = LLLLLLLLLLLLLLLLLLLLLLLLT
MPI FINALIZE: rank 0 of 1
```

By default, the tranformed MPS will have tags `KET-0`, `KET-1` etc, if it is real, or `KET-CPX-0`, `KET-CPX-1` etc if it is complex. If you set a custom tag, for example, when the input is like `restart_copy_mps SKET`, the tranformed MPS will have tags `SKET-0`, `SKET-1`, etc, no matter it is real or complex.

Third, we use the following script to do real time evolution:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

hf_occ integral
schedule
0 500 0 0
end
maxiter 10

read_mps_tags KET-CPX-0
mps_tags BRA
delta_t 0.05i
target_t 0.20i
complex_mps
noreorder
```

Note that a custom sweep schedule has to be used, to set the bond dimension to `500` (for example). The keyword `maxiter` and `noise` in the sweep schedule are ignored.

For every time step, there can be multiple sweeps, called “sub sweeps”. The total number of sweeps is `n_sweeps = nsteps * n_sub_sweeps`. The keyword `n_sub_sweeps` can be used to set the number of sub sweeps. Default value is 2.

For real time evolution, `delta_t` and `target_t` should be pure imaginary values. But they can also be general complex values. When doing imaginary time evolution, `delta_t` and `target_t` should be all real.

The tag of the input MPS (old MPS) is given by `read_mps_tags`. The tag of the output MPS (new MPS) is given by `mps_tags`. The two tags cannot be the same. They should (better) not have common prefix. For example, `KET` and `KET-1` may not be used together, as `-1` may be used by the code internally which will lead to confusion.

For this example, `target_t` is four times `delta_t`, so we will have 4 steps. Each time step has 2 sweeps. In total there will be 8 sweeps. The output is the result of applying `\exp(-0.2i H)` to the input.

Whenever a complex MPS is used, the keyword `complex_mps` should be used, otherwise the code will load the MPS incorrectly.

The output :

```
$ grep 'final' dmrg-3.out
    mps final tag = BRA
    mps final canonical form = MRRRRRRRRRRRRRRRRRRRRRRRR
$ grep '<E>' dmrg-3.out
T = RE    0.00000 + IM    0.05000 <E> =  -75.726309692728165 <Norm^2> =    0.
↪999999608946318
T = RE    0.00000 + IM    0.10000 <E> =  -75.726336818185246 <Norm^2> =    0.
↪999994467614067
T = RE    0.00000 + IM    0.15000 <E> =  -75.726364807114123 <Norm^2> =    0.
↪999990200387707
T = RE    0.00000 + IM    0.20000 <E> =  -75.726389514836484 <Norm^2> =    0.
↪999986418355937
```

Here we see that the expectation value is printed after each time step. The energy is roughly conserved (similar to the DMRG output -75.72629673), and the norm is roughly one. Decreasing the time step may give more accurate results.

We can do the same for the excited state:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

hf_occ integral
schedule
0 500 0 0
end
maxiter 10

read_mps_tags KET-CPX-1
mps_tags BRAEX
delta_t 0.05i
target_t 0.20i
complex_mps
noreorder
```

The output :

```
$ grep 'final' dmrg-4.out
    mps final tag = BRAEX
    mps final canonical form = MRRRRRRRRRRRRRRRRRRRRRRRR
$ grep '<E>' dmrg-4.out
T = RE    0.00000 + IM    0.05000 <E> =  -75.637185795841717 <Norm^2> =    0.
↪999999661398567
T = RE    0.00000 + IM    0.10000 <E> =  -75.637212093724074 <Norm^2> =    0.
↪999995415040728
T = RE    0.00000 + IM    0.15000 <E> =  -75.637238086798163 <Norm^2> =    0.
↪999991630799571
T = RE    0.00000 + IM    0.20000 <E> =  -75.637260508028248 <Norm^2> =    0.
↪999988252849994
```

The energy is close to the DMRG value -75.63717415.

For imaginary time evolution, since the propagator is not unitary, the norm will increase exponentially. You may use the extra keyword `normalize_mps` to normalize MPS after each time step. The norm will still be computed and printed, but it will not be accumulated.

Finally, we can verify the energy at `T = 0.0` and `T = 0.2` and compute the overlap for these states. The overlap between the all four states can be computed using the following input :

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP.ORIG

hf_occ integral
schedule
0 500 0 0
end
maxiter 10

mps_tags KET-CPX-0 BRA KET-CPX-1 BRAEX
restart_tran_oh
complex_mps
overlap
noreorder
```

The output is:

```
$ grep 'OH' dmrg-5.out
OH Energy    0 -    0 = RE    1.000000000000002 + IM    0.000000000000000
OH Energy    1 -    0 = RE   -0.845792004408687 + IM   -0.533433527528264
OH Energy    1 -    1 = RE    0.999986418355938 + IM    0.000000000000000
OH Energy    2 -    0 = RE   -0.000000000000000 + IM    0.000000000000000
OH Energy    2 -    1 = RE   -0.000000827506956 + IM   -0.000000742303613
OH Energy    2 -    2 = RE    1.000000000000004 + IM    0.000000000000000
OH Energy    3 -    0 = RE    0.000001731091412 + IM   -0.000000316659748
OH Energy    3 -    1 = RE   -0.000001122421894 + IM    0.000002348984005
```

(continues on next page)

(continued from previous page)

```
OH Energy    3 -    2 = RE   -0.836158473098047 + IM   -0.548435696470209
OH Energy    3 -    3 = RE    0.999988252849993 + IM    0.000000000000000
```

Here in the output each MPS gets a number, according to the order of tags in `mps_tags`. We have `0 (KET-CPX-0)`, `1 (BRA)`, `2 (KET-CPX-1)` and `3 (BRAEX)`.

Note that state 1 (not normalized) is time evolved from state 0 (normalized). We see that the overlap `<1|1>` is exactly 1. To get the overlap between the normalized states, we have:

```
< normlized(0) | normlized(1) >
= <0|1> / sqrt(<0|0> * <1|1>)
= (-0.845792004408687 -0.533433527528264j) / sqrt( 0.999986418355938 * 1.
↪000000000000002)
= -0.8457977480901698 -0.5334371500173138j
```

The absolute value and the angle of this complex overlap is :

```
   np.abs( -0.8457977480901698 -0.5334371500173138j ) =  0.9999645112167714
np.angle ( -0.8457977480901698 -0.5334371500173138j ) = -2.578911293480138
```

The absolute value is close to one. So the time evolution simply introduced a complex phase factor for the state, as expected. The complex phase factor can be computed as the remainder of `E t` divided by `2 pi`:

```
-75.72638951483646 * 0.2 % (2 * np.pi) - 2 * np.pi = -2.5789072886081197
```

Which is close to the printed value.

Also note that the overlap between the ground state and the excited state `<2|0>` is exactly zero. The corresponding overlap between the time evolved states `<3|1>` is slightly different from zero, mainly due to the time step error and truncation error.

We can also get the energy expetation, by removing the keyword `overlap`:

```
$ grep 'OH' dmrg-6.out
OH Energy    0 -    0 = RE  -75.726296730204453 + IM    0.000000000000000
OH Energy    1 -    0 = RE   64.049088006450049 + IM   40.394772180607831
OH Energy    1 -    1 = RE  -75.725361025967970 + IM   -0.000000000000007
OH Energy    2 -    0 = RE    0.000000000000008 + IM    0.000000000000000
OH Energy    2 -    1 = RE    0.000061050951670 + IM    0.000056012958492
OH Energy    2 -    2 = RE  -75.637174152353893 + IM    0.000000000000000
OH Energy    3 -    0 = RE   -0.000132735557064 + IM    0.000024638559206
OH Energy    3 -    1 = RE    0.000086585167013 + IM   -0.000178008928209
OH Energy    3 -    2 = RE   63.244928578558032 + IM   41.482021915322555
OH Energy    3 -    3 = RE  -75.636371985782972 + IM    0.000000000000000
```

Note that here not all states are normalized, the printed value is not directly the energy. The printed value is `<A|H|B>`, but the energy is `<A|H|B>/<A|B>`. So the printed value should be divided by the square of the norm of the MPS (see previous output). For example, for state 1 we have :

```
-75.725361025967970 / 0.999986418355938 = -75.72638951483646
```

Which is the same as the number <E> printed by the time evolution (-75.726389514836484).

3.5 Input File: Keywords

In this section we provide a complete list of allowed keywords for the input file used in `block2main` with a short description for each keyword.

3.5.1 Global Settings

 # / !
: If a line starts with '!' or '#', the line will be ignored.

outputlevel
: Optional. Followed by one integer. 0 = Silent. 1 = Print information for each sweep. 2 = Print information for iteration at each site (default). 3 = Print information for each Davidson/CG iteration.

orbitals
: Required for most normal cases. Not required if reloading MPO or when `orbital_rotation` is the calculation type, or when `model` is given. Followed by the file name for the orbital definition and integrals, in FCIDUMP format or `hdf5` format (used only in `libdmet`). Only `nonspinadapted` is supported for `orbitals` with `hdf5` format .

integral_tol
: Optional. The integral values smaller than `integral_tol` will be discarded. Default is 1E-12 (for integral with `hdf5` format) or 0 (for integral with FCIDUMP format).

model
: Optional. Can be used to perform calculations for some simple model Hamiltonian and the `orbitals` keyword can be skipped. For example, `model hubbard 16 1 2` will calculate ground state for 1-dimensional non-periodic Hubbard model with 16 sites and nearest-neighbor interaction, t = 1 and U = 2. `model hubbard_periodic 16 1 2` will do the calculation for the periodic Hubbard model. `model hubbard_kspace 16 1 2` will do the calculation for the periodic Hubbard model in the momentum space. One can then use this together with `k_symmetry` to utilize the translational symmetry or not use it if the keyword `k_symmetry` is not given. `model hubbard 16 1 2 per-site` will print the energy for each site.

prefix
: Optional. Path to scratch folder. Default is `./nodex/`.

num_thrds
: Optional. Followed by an integer for the number of OpenMP threads to use. Default is 28 (if there is no `hf_occ integral` in the input file) or 1 (to be compatible with `StackBlock` when there is `hf_occ integral` in the input file). Note that the environment variable `OMP_NUM_THREADS` is ignored.

mkl_thrds
Optional. Followed by an integer for the number of OpenMP threads to use for the MKL library. Default is 1.

mem
Optional. Followed by an integer and a letter as the unit (g or G). Stack memory for doubles. Default is 2 GB. Note that the code may use a large amount of memory via dynamic allocation, which is not controlled by this number.

intmem
Optional. Followed by an integer and a letter as the unit (g or G). Stack memory for integers. Default is 10% of `mem`.

mem_ratio
Optional. Followed by a float number ($0.0 \sim 1.0$). The ratio of main stack memory. Default is 0.4.

min_mpo_mem
Optional. Followed by auto, True, or False. If True, MPO building and simplification will cost much less memory. But the computational cost will be higher due to IO cost. Default is auto, which is True if number of orbitals is $>= 120$.

qc_mpo_type
Optional. Followed by auto (default), conventional, nc, or cn. The Hamiltonian MPO formalism type. The default is to use Conventional for non-big-site, and NC for big-site. Conventional DMRG is overall 50% faster than NC, but the cost of the middle site is 2 times higher than NC. If the memory is limited and `min_mpo_mem` is used, one should set NC MPO type to make memory cost more uniform.

cached_contraction
Optional. Followed by an integer 0 or 1 (default). If 1, cached contraction is used for improving performance.

nonspinadapted
Optional. If given, the code will work in the non-spin-adapted `SZ` mode. Otherwise, it will work in the spin-adapted `SU2` mode.

k_symmetry
Optional. If given, the code will work in the non-spin-adapted or spin-adapted mode with additionally the K symmetry. Requiring the code to be built with `-DUSE_KSYMM`.

use_complex
Optional. If given, the code will work in the complex number mode, where the integral, MPO and MPS contain all complex numbers. FCIDUMP with real or complex integral can be accepted in this mode. Requiring the code to be built with `-DUSE_COMPLEX`. Conflict with `use_hybrid_complex` (checked).

use_hybrid_complex
Optional. If given, the code will work in the hybrid complex number mode, where the MPO is split into real and complex sub-MPOs. MPS rotation matrix are real matrices but center site tensor is complex. FCIDUMP with real or complex integral can be accepted in this mode. Requiring the code to be built with `-DUSE_COMPLEX`. Conflict with `use_complex` (checked).

use_general_spin
Optional. If given, the code will work in (fermionic) spin orbital (rather than spatial orbital). FCIDUMP will be intepreted as integrals between spin orbitals. If the FCIDUMP is actually the normal FCIDUMP for spatial orbitals, the extra keyword `trans_integral_to_spin_orbital` is required to make it work with general spin. Requiring the code to be built with `-DUSE_SG`. Currently cannot be used together with `k_symmetry`.

single_prec
Optional. If given, the code will work in single precision (float) rather than double precision (double).

integral_rescale
Optional. `auto` (default) or `none` or floating point number. If `auto` and the calculation is done with single precision, the average diagonal of the one-electron integral will be moved to the energy constant. Ideally, with single precision, we want the energy constant to be close to the final dmrg energy. If `auto` and the calculation is done with double precision, nothing will happen. If `none`, nothing will happen. If the value of `integral_rescale` is a number, the energy constant will be adjust to the given number by shifting the average diagonal of the one-electron integral. This should only be used when the particle number of the calculation is a constant (namely, `nelec` contains only one number).

check_dav_tol
Optional. `auto` (default) or `1` or `0`. If `auto` or `1` and the calculation is done with single precision, the davidson tolerance will be set to be no lower than 5E-6.

trans_integral_to_spin_orbital
Optional. If given, the FCIDUMP (in spatial orbitals) will be reinterpretted to work with general spin. Only makes sense together with `use_general_spin`.

singlet_embedding
Optional. If given, the code will use the singlet embedding formalism. Only have effects in the spin-adapted `SU2` mode. No effects if it is a restart calculation.

conn_centers
Optional. Followed by a list of indices of connection sites or by `auto` and the number of processor groups. If `conn_centers` is given, the parallelism over sites will be used (MPI required, `twodot` only). For example, `conn_centers auto 5` will divide the processors into 5 groups. Only supports the standard DMRG calculation.

restart_dir
Optional. Followed by directory name. If `restart_dir` is given, after each sweep, the MPS will be backed up in the given directory.

restart_dir_per_sweep
Optional. Followed by directory name. If `restart_dir_per_sweep` is given, after each sweep, the MPS will be backed up in the given directory name followed by the sweep index as the name suffix. This will save MPSs generated from all sweeps.

fp_cps_cutoff
Optional. Followed by a small fractional number. Sets the float-point number cutoff for saving disk storage. Default is `1E-16`.

release_integral
Optional. If given, memory used by stroring the full integral will be release after building MPO (but before DMRG).

3.5.2 Calculation Types

The default calculation type is DMRG (without the need to write any keywords).

fullrestart
Optional. If given, the initial MPS will be read from disk. Normally this keyword will be automatically added if any of the `restart_*` keywords are used.

oh / restart_oh
Expectation value calculation on the DMRG optimized MPS or reloaded MPS.

onepdm / restart_onepdm
One-particle density matrix calculation on the DMRG optimized MPS or reloaded MPS. `onepdm` can run with either `twodot_to_onedot`, `onedot` or `twodot`.

twopdm / restart_twopdm
Two-particle density matrix calculation on the DMRG optimized MPS or reloaded MPS.

threepdm / restart_threepdm
Three-particle density matrix calculation on the DMRG optimized MPS or reloaded MPS. Cannot be used together with `conventional_npdm`.

fourpdm / restart_fourpdm
Four-particle density matrix calculation on the DMRG optimized MPS or reloaded MPS. Cannot be used together with `conventional_npdm`.

tran_onepdm / restart_tran_onepdm
One-particle transition density matrix among a set of MPSs.

tran_twopdm / restart_tran_twopdm
Two-particle transition density matrix among a set of MPSs.

tran_threepdm / restart_tran_threepdm
Three-particle transition density matrix among a set of MPSs. Cannot be used together with `conventional_npdm`.

tran_fourpdm / restart_tran_fourpdm
Four-particle transition density matrix among a set of MPSs. Cannot be used together with `conventional_npdm`.

tran_oh / restart_tran_oh
Operator overlap between each pair in a set of MPSs.

diag_twopdm / restart_diag_twopdm
Diagonal two-particle density matrix calculation.

correlation / restart_correlation
Spin and charge correlation function.

copy_mps / restart_copy_mps
Copy MPS with one tag to another tag. Followed by the tag name for the output MPS. The input MPS tag is given by `mps_tags`. The MPS transformation is also handled with this calculation type.

sample / restart_sample
Printing configuration state function (CSF) or determinant coefficients.

orbital_rotation
Orbital rotation of an MPS to generate another MPS.

compression
MPS compression.

delta_t
Followed by a single float value or complex value as the time step for the time evolution. The computation will apply $\exp(-\Delta t H)|\psi\rangle$ (with multiple steps). So when it is a real float value, we will do imaginary time evolution of the MPS (namely, optimizing to ground state or finite-temperature state). When it is a pure imaginary value, we will do real time evolution of the MPS (namely, solving the time dependent Schrodinger equation). General complex value can also be supported, but may not be useful.

stopt_dmrg
First step of stochastic perturbative DMRG, which is the normal DMRG with a small bond dimension.

stopt_compression
Second step of stochastic perturbative DMRG, which is the compression of $QV|\Psi_0\rangle$. In general a bond diemension that is much larger than the first step should be used.

stopt_sampling
Third step of stochastic perturbative DMRG. Followed by an integer as the number of CSF / determinants to be sampled. If any of the first and second step is done in the non-spin-adapted mode, the determinants will be sampled and this step must also be in the non-spin-adapted mode. Otherwise, CSF will be sampled if the keyword `nonspinadapted` is given, and determinants will be sampled if the keyword `nonspinadapted` is not given.

restart_nevpt2_npdm
Compute 1-4 PDM for DMRG-SC-NEVPT2. If there are multiple roots, the calculation will be performed for all roots. The 1-4PDM will be used to compute the SC-NEVPT2 intermediate Eqs. (A16) and (A22) in the spin-free NEVPT2 paper. Only the two SC-NEVPT2 intermediates will be written into the disk.

restart_mps_nevpt
Followed by three integers, representing the number of active, inactive, and external orbitals. Compute the `V_i` and `V_a` correlation energy in DMRG-SC-NEVPT2 using MPS compression. Only the spin-adapted version is implemented. If there are multiple roots, the keyword `nevpt_state_num` is required to set which root should be used to compute the correlation energy.

3.5.3 Calculation Modifiers

target_t
Optional. Followed by a single float value as the total time for time evolution. This keyword should be used only together with `delta_t`. Default is 1.

te_type
Optional. Followed by `rk4` or `tangent_space`. This keyword sets the time evolution algorithm. This keyword should be used only together with `delta_t`. Default is `rk4`.

statespecific
If `statespecific` keyword is in the input (with no associated value). This option implies that a previous state-averaged dmrg calculation has already been performed. This calculation will refine each individual state. This keyword should be used only with DMRG calculation type.

soc
If `soc` keyword is in the input (with no associated value), the (normal or transition) one pdm for triplet excitation operators will be calculated (which can be used for spin-orbit coupling calculation). This keyword should be used only together with `onepdm`, `tran_onepdm`, `restart_onepdm`, or `restart_tran_onepdm`. Not supported for `nonspinadapted`.

overlap
If `overlap` keyword is in the input (with no associated value), the expectation of identity operator will be calculated (which can be used for the overlap matrix between states). Otherwise, when the *overlap* keyword is not given, the full Hamiltonian is used. For compression, if this keyword is in the input, it directly compresses the given MPS. Otherwise, the contration of full Hamiltonian MPO and MPS is compressed. This keyword should only be used together with `oh`, `tran_oh`, `restart_oh`, `restart_tran_oh`, `compression`, and `stopt_compression`.

nat_orbs
If given, the natural orbitals will be computed. Optionally followed by the filename for storing the rotated integrals (FCIDUMP). If no value is associated with the keyword `nat_orbs`, the rotated integrals will not be computed. This keyword can only be used together with `restart_onepdm` or `onepdm`.

nat_km_reorder
Optional keyword with no associated value. If given, the artificial reordering in the natural orbitals will be removed using Kuhn-Munkres algorithm. This keyword can only be used together with `restart_onepdm` or `onepdm`. And the keyword `nat_orbs` must also exist.

nat_positive_def
Optional keyword with no associated value. If given, artificial rotation in the logarithm of the rotation matrix can be avoid, by make the rotation matrix quasi-positive-definite, with "quasi" in the sense that the rotation matrix is not Hermitian. This keyword can only be used together with `restart_onepdm` or `onepdm`. And the keyword `nat_orbs` must also exist.

trans_mps_to_sz
Optional keyword with no associated value. If given, the MPS will be transformed to non-spin-adapted before being saved. This keyword can only be used together with `restart_copy_mps` or `copy_mps`.

trans_mps_to_singlet_embedding

Optional keyword with no associated value. If given, the MPS will be transformed to singlet-embedding format before being saved. This keyword can only be used together with `restart_copy_mps` or `copy_mps`.

trans_mps_from_singlet_embedding
Optional keyword with no associated value. If given, the MPS will be transformed to non-singlet-embedding format before being saved. This keyword can only be used together with `restart_copy_mps` or `copy_mps`.

trans_mps_to_complex
Optional keyword with no associated value. If given, the MPS will be transformed to complex wavefunction with real rotation matrix before being saved. This keyword can only be used together with `restart_copy_mps` or `copy_mps`, and optionally with `split_states`. This keyword is conflict with other `trans\_mps\_*` keywords. To load this MPS in the subsequent calculations, the keyword `complex_mps` must be used.

split_states
Optional keyword with no associated value. If given, the state averaged MPS will be split into individual MPSs. This keyword can only be used together with `restart_copy_mps` or `copy_mps`, and optionally with `trans_mps_to_complex`. This keyword is conflict with other `trans\_mps\_*` keywords. The individual MPS will be the tag given by the keyword `restart_copy_mps` or `copy_mps` with `-<n>` appended, where `n` is the root index counting from zero.

resolve_twosz
Optional. Followed by an integer, which is two times the projected spin. The transformed SZ MPS will have the specified projected spin. If the keyword `resolve_twosz` is not given, an MPS with ensemble of all possible projected spins will be produced (which is often not very useful). This keyword can only be used together with `restart_copy_mps` or `copy_mps`. And the keyword `trans_mps_to_sz` must also exist.

normalize_mps
Optional keyword with no associated value. If given, the transformed SZ MPS will be normalized. This keyword can only be used together with `restart_copy_mps` or `copy_mps`. And the keyword `trans_mps_to_sz` must also exist.

big_site
Optional. Followed by a string for the implementation of the big site. Possible implementations are `folding`, `fock` (only with `nonspinadapted`), `csf` (only without `nonspinadapted`). This keyword can only be used in dynamic correlation calculations. If this keyword is not given, the dynamic correlation calculation will be performed with normal MPS with no big sites.

expt_algo_type
Optional. Followed by a string `auto`, `fast`, `normal`, `symbolfree`, or `lowmem`. Default is `auto`. This keyword can only be used with density matrix or transition density matrix calculations. `auto` is `fast` if `conventional_npdm` is given, or `symbolfree` if `conventional_npdm` is not given. `normal` uses less memory compared to `fast`, but the complexity can be higher. `lowmem` uses less memory compared to `symbolfree`, but the complexity can be higher. `symbolfree` is in general more efficient than `fast` and `normal`, but it is only available if `conventional_npdm` is not given. For 3- and 4-particle density matrices, when this keyword is not `auto` or `symbolfree`, it may

consume a significant large amount of memory to store the symbols.

conventional_npdm
Optional, mainly for backward compatibility. If given, will use the conventional manual npdm code. This is only available for 1- and 2- particle density matrices. For most cases, the conventional manual code is slower. For soc 1-particle density matrix, only the conventional manual code is available.

simple_parallel
Optional. Followed by an empty string (same as `ij`) or `ij` or `kl`. When this keyword is not given, the conventional parallel rule for QC-DMRG will be used. Otherwise, the simple parallel scheme based on distributing integral according to `ij` or `kl` indices is used. When `qc_mpo_type` is auto, this simple scheme will also change the center for middle transformation to reduce the MPO bond dimension. The simple parallel scheme may be good for saving per-processor MPO memory cost for large scale parallelized DMRG.

condense_mpo
Optional. Followed by an integer (must be a power of 2, default is 1). When `condense_mpo` is not 1, `block2` will merge every two adjacent MPO sites into a larger site (after the MPO is created), repeating `log(condense_mpo)` times, until the total number of sites is `n_sites / condense_mpo`. Not working with SU2 symmetry. When `condense_mpo 2` is used with general spin, the calculation will be done with two spin orbitals as a site rather than one spin orbital. Not working with `twopdm` related keywords. Require the keyword `simple_parallel` for the parallelization of the condensed MPO.

one_body_parallel_rule
Optional keyword with no associated value. If given, the more efficient parallelization rule will be used to distribute the MPO. This rule only works when the two-body term is zero or purely local. Real space Huabbard model is one of the case. For such Hamiltonian, the default (quantum chemistry) parallelization rule can still work, but may have no improvements with multiple processors. If this keyword is used with non-trivial two-body term, runtime error may happen.

complex_mps
Optional keyword with no associated value. If given, complex expectation values will be computed for MPS with complex wavefunction tensor and real rotation matrices (in non-complex mode). Should be used together with `pdm`, `oh`, or (complex) `delta_t` type calculations. In complex mode, this should not be used as everything is complex.

tran_bra_range
Optional. Followed by the range parameter of bra state indices for computing transition density matrices. Normally two numbers are given, which is the starting index and endding index (not included).

tran_ket_range
Optional. Followed by the range parameter of ket state indices for computing transition density matrices. Normally two numbers are given, which is the starting index and endding index (not included).

tran_triangular
Optional keyword with no associated value. If given, only the transition density matrices with bra state index equal to or greater than the ket state index will be computed.

skip_inact_ext_sites
Optional keyword with no associated value. If given, for uncontracted dynamic correlation calculations, the sweeps will skip inactive and external sites, so that the efficiency can be higher and the accuracy is not affected. This should only be used with uncontracted dynamic correlation keywords (checked) without any big sites. Normally it is useful only for dynamic correlation with singles (such as `mrcis`).

full_integral
Optional keyword with no associated value. If **not** given, and it is a dynamic correlation with singles (namely, with keywords `nevpt2s`, `mrcis`, `mrrept2s`, `nevpt2-i`, `nevpt2-r`, `mrrept2-i`, or `mrrept2-r`), the two-electron integral elements with more than two virtual indices will be set to zero. This should save some MPO contruction time, without affecting the sweep time cost and accuracy. If this keyword is given, the full integral elements will be used for constructing MPO.

nevpt_state_num
Followed by a single integer, the index of the root (counting from zero) used for SC-NEVPT2. Only useful for the calculation type `restart_mps_nevpt`.

3.5.4 Uncontracted Dynamic Correlation

There can only be at most one dynamic correlation keyword (checked). Any of the following keyword must be followed by 2 integers (representing number of orbitals in the active space and number of electrons in the active space), or 3 integers (representing number of orbitals in the inactive, active, and external space, respectively).

dmrgfci
Not useful for general purpose. Treating the inactive and external space using full Configuration Interaction (FCI).

casci
Treating the inactive space as a single CSF (all occupied) and the external space as a single CSF (all empty).

mrci
Same as `mrcisd`.

mrcis
Multi-configuration CI with singles. The inactive / virtual space can have at most one hole / electron.

mrcisd
Multi-configuration CI with singles and doubles. The inactive / virtual space can have at most two holes / electrons.

mrcisdt
Multi-configuration CI with singles and doubles and triples. The inactive / virtual space can have at most three holes / electrons.

nevpt2
Same as `nevpt2sd`.

nevpt2s
Second order N-Electron Valence States for Multireference Perturbation Theory with singles. The inactive / virtual space can have at most one hole / electron.

nevpt2sd
Second order N-Electron Valence States for Multireference Perturbation Theory with singles and doubles. The inactive / virtual space can have at most two holes / electrons. The zeroth-order Hamiltonian is Dyall's Hamiltonian.

mrrept2
Same as `mrrept2sd`.

mrrept2s
Second order Restraining the Excitation degree Multireference Perturbation Theory (MR-REPT) with singles. The inactive / virtual space can have at most one hole / electron.

mrrept2sd
Second order Restraining the Excitation degree Multireference Perturbation Theory (MR-REPT) with singles and doubles. The inactive / virtual space can have at most two holes / electrons. The zeroth-order Hamiltonian is Fink's Hamiltonian.

3.5.5 Schedule

onedot
Using the one-site DMRG algorithm. `onedot` will be implicitly used if you restart from a `onedot` mps (can be obtained from previous run with `twodot_to_onedot`).

twodot
Default. Using the two-site DMRG algorithm.

twodot_to_onedot
Followed by a single number to indicate the sweep iteration when to switch from the two-site DMRG algorithm to the one-site DMRG algorithm. The sweep iteration is counted from zero.

schedule
Optional. Followed by the word `default` or a multi-line DMRG schedule with the last line being `end`. If not given, the defualt schedule will be used. Between the keyword `schedule` and `end` each line needs to have four values. They are corresponding to starting sweep iteration (counting from zero), MPS bond dimension, tolerance for the Davidson iteration, and noise, respectively. Starting sweep iteration is the sweep iteration in which the given parameters in the line should take effect. For each line, alternatively, one can provide `n_sites - 1` values for the MPS bond dimension, where the ith number represents the right virtual bond dimension for the MPS tensor at site i. If this is the case, the site-dependent MPS bond dimension truncation will be used.

store_wfn_spectra
Optional with no associated value. If given, the singular values at each left-right partition during the last DMRG sweep will be stored as `sweep_wfn_spectra.npy` after convergence. Only works with DMRG type calculation. The stored array is a numpy array of 1 dimensional numpy array. The inner arrays normally do not have all the same length. For spin-adapted, each singular values correspond to a multiplet. So for non-singlet, the wavefunction spectra

have different interpretation between SU2 and SZ. Additionally, when this keyword is given, the bipartite entanglement of the MPS will be computed, as $S_k = -\sum_i \Lambda_i^2 \log \Lambda_i^2$ where Λ_i are all singular values found at site k. The bipartite entanglement will be printed and stored as `sweep_wfn_entropy.npy` as a 1 dimensional numpy array.

extrapolation
Optional. Should only be used for standard DMRG calculation with the reverse schedule. Will print the extrapolated energy and generate the energy extrapolation plot (saved as a figure).

maxiter
Optional. Followed by an integer. Maximum number of sweep iterations. Default is 1.

sweep_tol
Optional. Followed by a small float number. Convergence for the sweep. Default is 1E-6.

startM
Optional. Followed by an integer. Starting bond dimension in the default schedule. Default is 250.

maxM
Required for default schedule. Followed by an integer. Maximum bond dimension in the default schedule.

lowmem_noise
Optional. If given, the noise step will require less memory but potentially worse openmp load-balancing.

dm_noise
Optional. If given, the density matrix noise will be used instead of the default perturbative noise. Density matrix noise is much cheaper but not very effective.

cutoff
Optional. Followed by a small float number. States with eigenvalue below this number will be discarded, even when the bond dimension is large enough to keep this state. Default is 1E-14.

svd_cutoff
Optional. Followed by a small float number. Cutoff of singular values used in parallel over sites. Default is 1E-12.

svd_eps
Optional. Followed by a small float number. Accuracy of SVD for connection sites used in parallel over sites. Default is 1E-4.

trunc_type
Optional. Can be `physical` (default) or `reduced`, where `reduced` re-weight eigenvalues by their multiplicities (only useful in the `SU2` mode).

decomp_type
Optional. Can be `density_matrix` (default) or `svd`, where *svd* may be less numerical stable and not working with `nroots > 1`.

real_density_matrix
Optional. Only have effects in the complex mode and when `decomp_type` is `density_matrix`.

If given, the imaginary part of the density matrix will be discarded before diagonalization. This means that all rotation matrices will be orthogonal rather than unitary, although they will be stored as complex matrices. For complex mode DMRG with more than one roots, this keyword has to be used (not checked).

davidson_max_iter
Optional. Maximal number of iterations in Davidson. Default is 5000. If this number is reached but convergence is not achieved, the calculation will abort. If this numebr is larger than `davidson_soft_max_iter`, this keyword has no effect.

davidson_soft_max_iter
Optional. Maximal number of iterations in Davidson. Default is 4000. If this number is reached but convergence is not achieved, the calculation will continue as if the convergence is achieved. If this numebr is -1, or larger than or equal to `davidson_max_iter`, this keyword has no effect and `davidson_max_iter` is used instead.

davidson_def_max_size
Optional. Max size of the Krylov subspace in Davidson. Default is 50. The Davidson working memory is two times this number times `Wmem` printed during sweeps. One may reduce this number to save memory when the MPS bond dimension is very large.

n_sub_sweeps
Optional. Number of sweeps for each time step. Defualt is 2. This keyword only has effect when used with `delta_t` and when `te_type` is `rk4`.

3.5.6 System Definition

nelec
Optional. Followed by one or more integrers. Number of electrons in the target wavefunction. If not given, the value from FCIDUMP is used (and the keyword `orbtials` must be given).

spin
Optional. Followed by one or more integrers. Two times the total spin of the target wavefunction in spin-adapted calculation. Or Two times the projected spin (number of alpha electrons minus number of beta electrons) of the target wavefunction in non-spin-adapted calculation. If not given, the value from FCIDUMP is used. If FCIDUMP is not given, 0 is used.

irrep
Optional. Followed by one or more integrers. Point group irreducible representation of the target wavefunction. If not given, the value from FCIDUMP is used. If FCIDUMP is not given, 1 is used. MOLPRO notation is used, where 1 always means the trivial irreducible representation.

sym
Optional. Followed by a lowercase string for the (Abelian) point group name. Default is `d2h`. If the real point group is `c1` or `c2`, setting `sym d2h` will also work.

k_irrep
Optional. Followed by one or more integrers. LZ / K irreducible representation number of the target wavefunction. If not given, the value from FCIDUMP is used. If FCIDUMP is not given, 0 is used.

k_mod
Optional. Followed by one integer. Modulus for the K symmetry. Zero means LZ symmetry. If not given, the value from FCIDUMP is used. If FCIDUMP is not given, 0 is used.

nroots
Optional. Followed by one integer. Number of roots. Default is 1. For `nroots > 1`, `oh` or `restart_oh` will calculate the expectation of Hamiltonian on every state. `tran_oh` or `restart_tran_oh` will calculate the expectation of Hamiltonian on every possible pair of states as bra and ket states. The parameters for the quantum number of the MPS, namely `spin`, `isym` and `nelec` can also take multiple numbers. This can also be combined with `nroots > 1`, which will then enable transition density matrix between MPS with different quantum numbers to be calculated (in a single run). This kind of calulation usually needs a larger `nroots` than the `nroots` actually needed, otherwise, some excited states with different quantum number from the ground-state may be missing. To save time, one may first do a calculation with larger `nroots` and small bond dimensions, and then do `fullrestart` and change `nroots` to a smaller value. Then only the lowest `nroots` MPSs will be restarted.

weights
Optional. Followed by a list of fractional numbers. The weights of each state for the state average calculation. If not given, equal weight will be used for all states.

mps_tags
Optional. Followed by a single string or a list of strings. The MPS in scratch directory with the specific tag/tags will be loaded for restart (for `statespecific`, `restart_onepdm`, etc.). The default MPS tag for input/output is `KET`.

read_mps_tags
Optional. Followed by a string. The tag for the constant (right hand side) MPS for compression. The tag of the output MPS in compression is set using `mps_tags`.

proj_mps_tags
Optional. Followed by a single string or a list of strings. The tag for the MPSs to be projected out during DMRG. Must be used together with `proj_weights`. The projection will be done by changing Hamiltonian from $\hat{H}$ to $\hat{H} + \sum_i w_i |\phi_i\rangle\langle\phi_i|$ (the level shift approach), where $|\phi_i\rangle$ are the MPSs to be projected out. w_i are the weights.

proj_weights
Optional. Followed by a single float number or a list of float numbers. Can be used together with `proj_mps_tags`. The number of float numbers in this keyword must be equal to the length of `proj_mps_tags`. Normally, the weights are positive and they should be larger than the energy gap. If the weight is too small, you will get unphyiscal eigenvalues as `Ei + wi`, where `Ei` is the energy of the MPSs to be projected out. If `statespecific` keyword is in the input, it will change the projection method from the orthogonalization method $\hat{H} - \sum_i |\phi_i\rangle\langle\phi_i|$ to the level shift approach $\hat{H} + \sum_i w_i |\phi_i\rangle\langle\phi_i|$.

symmetrize_ints
Optional. Followed by a small float number. Setting the largest allowed value for the integral element that violates the point group or K symmetry. Default is 1E-10. The symmetry-breaking integral elements will be discarded in the calculation anyway. Setting this keyword will only control whether the calculation can be performed or an error will be generated.

occ
Optional. Followed by a list of float numbers between 0 and 2 for spatial orbital occupation numbers, or a list of float numbers between 0 and 1 for spin orbital occupation numbers, or a list of float numbers between 0 and 1 for the probability for each of four states at each site (experimental). This keyword should only be used together with `warmup occ`.

bias
Optional. Followed by a non-negative float number. If not 1.0, sets an power based bias to occ.

cbias
Optional. Followed by a non-negative float number. If not 0.0, sets a constant shift towards the equal-possibility occ. `cbias` is normally useful for shifting integral occ, while `bias` only shifts fractional occ.

init_mps_center
Optional. Followed by a site index (counting from zero). Default is zero. This is the canonical center for the initial guess MPS.

full_fci_space
Optional, not useful for general user. If `full_fci_space` keyword is in the input (with no associated value), the full fci space is used (including block quantum numbers outside the space of the wavefunction target quantum number).

trans_mps_info
Optional, experimental. If `trans_mps_info` keyword is in the input (with no associated value), the `MPSInfo` will be initialized using `SZ` quantum numbers if in `SU2` mode, or using `SU2` quantum numbers if in `SZ` mode. A transformation of `MPSInfo` is then performed between `SZ` and `SU2` quantum numbers. `MultiMPSInfo` cannot be supported with this keyword.

random_mps_init
Optional. If given, the initial guess for the output MPS in compression will be random initialized in the way set by the `warmup` keyword. Otherwise, the constant right hand side MPS will be copied as the the initial guess for the output MPS.

warmup
Optional. If `wamup occ` then the initial guess will be generated using occupation numbers. Otherwise, the initial guess will be generated assuming every quantum number has the same probability (default).

3.5.7 Orbital Reordering

There can only be at most one orbital reordering keyword (checked).

noreorder
The order of orbitals is not changed.

nofiedler
Same as `noreorder`.

gaopt
Genetic algorithm for orbital ordering. Followed by (optionally) the configuration file for the

gaopt subroutine. Default parameters for the genetic algorithm will be used if no configuration file is given.

fiedler
Default. Fiedler orbital reordering.

irrep_reorder
Group orbitals with the same irrep together.

reorder
Followed by the name of a file including the space-sparated orbital reordering indices (counting from one).

3.5.8 Unused Keywords

hf_occ integral
Optional. For StackBlock compatibility only.

3.6 DMRGSCF (pyscf)

In this section we explain how to use `block2` (and optionally `StackBlock`) and `pyscf` for `DMRGSCF` (CASSCF with DMRG as the active space solver).

3.6.1 Preparation

`pyscf` can be installed using `pip install pyscf`. One also needs to install the pyscf extension called `dmrgscf`, which can be obtained from https://github.com/pyscf/dmrgscf. If it is installed using `pip`, one also needs to create a file named `settings.py` under the `dmrgscf` folder, as follows:

```
$ pip install git+https://github.com/pyscf/dmrgscf
$ PYSCFHOME=$(pip show pyscf-dmrgscf | grep 'Location' | tr ' ' '\n' | tail -n 1)
$ wget https://raw.githubusercontent.com/pyscf/dmrgscf/master/pyscf/dmrgscf/settings.
↪py.example
$ mv settings.py.example ${PYSCFHOME}/pyscf/dmrgscf/settings.py
$ chmod +x ${PYSCFHOME}/pyscf/dmrgscf/nevpt_mpi.py
```

Here we also assume that you have installed `block2` either using `pip` or manually.

3.6.2 DMRGSCF (serial)

The following is an example python script for DMRGSCF using `block2` running in a single node without MPI parallelism:

```
from pyscf import gto, scf, lib, dmrgscf
import os

dmrgscf.settings.BLOCKEXE = os.popen("which block2main").read().strip()
dmrgscf.settings.MPIPREFIX = ''

mol = gto.M(atom='C 0 0 0; C 0 0 1.2425', basis='ccpvdz',
    symmetry='d2h', verbose=4, max_memory=10000) # mem in MB
mf = scf.RHF(mol)
mf.kernel()

from pyscf.mcscf import avas
nactorb, nactelec, coeff = avas.avas(mf, ["C 2p", "C 3p", "C 2s", "C 3s"])
print('CAS = ', nactorb, nactelec)

mc = dmrgscf.DMRGSCF(mf, nactorb, nactelec, maxM=1000, tol=1E-10)
mc.fcisolver.runtimeDir = lib.param.TMPDIR
mc.fcisolver.scratchDirectory = lib.param.TMPDIR
mc.fcisolver.threads = int(os.environ.get("OMP_NUM_THREADS", 4))
mc.fcisolver.memory = int(mol.max_memory / 1000) # mem in GB

mc.canonicalization = True
mc.natorb = True
mc.kernel(coeff)
```

Note: Alternatively, to use `StackBlock` instead of `block2` as the DMRG solver, one can change the line involving `dmrgscf.settings.BLOCKEXE` to:

```
dmrgscf.settings.BLOCKEXE = os.popen("which block.spin_adapted").read().strip()
```

Please see *MPS Import/Export* for the instruction for the installation of `StackBlock`.

Note: It is important to set a suitable `mc.fcisolver.threads` if you have multiple CPU cores in the node, to get high efficiency.

This will generate the following output:

```
$ grep 'CASSCF energy' cas1.out
CASSCF energy = -75.6231442712648
```

3.6.3 DMRGSCF (distributed parallel)

The following example is DMRGSCF in hybrid MPI (distributed) and openMP (shared memory) parallelism. For example, we can use 7 MPI processors and each processor uses 4 threads (so in total the calculation will be done with 28 CPU cores):

```
from pyscf import gto, scf, lib, dmrgscf
import os

dmrgscf.settings.BLOCKEXE = os.popen("which block2main").read().strip()
dmrgscf.settings.MPIPREFIX = 'mpirun -n 7 --bind-to none'

mol = gto.M(atom='C 0 0 0; C 0 0 1.2425', basis='ccpvdz',
    symmetry='d2h', verbose=4, max_memory=10000) # mem in MB
mf = scf.RHF(mol)
mf.kernel()

from pyscf.mcscf import avas
nactorb, nactelec, coeff = avas.avas(mf, ["C 2p", "C 3p", "C 2s", "C 3s"])
print('CAS = ', nactorb, nactelec)

mc = dmrgscf.DMRGSCF(mf, nactorb, nactelec, maxM=1000, tol=1E-10)
mc.fcisolver.runtimeDir = lib.param.TMPDIR
mc.fcisolver.scratchDirectory = lib.param.TMPDIR
mc.fcisolver.threads = 4
mc.fcisolver.memory = int(mol.max_memory / 1000) # mem in GB

mc.canonicalization = True
mc.natorb = True
mc.kernel(coeff)
```

Note: To use MPI with `block2`, the block2 must be either (a) installed using `pip install block2-mpi` or (b) manually built with `-DMPI=ON`. Note that the `block2` installed using `pip install block2` cannot be used together with `mpirun` if there are more than one processors (if this happens, it will generate wrong results and undefined behavior).

If you have already `pip install block2`, you must first `pip uninstall block2` then `pip install block2-mpi`.

Note: If you do not have the `--bind-to` option in the `mpirun` command, sometimes every processor will only be able to use one thread (even if you set a larger number in the script), which will decrease the CPU usage and efficiency.

This will generate the following output:

```
$ grep 'CASSCF energy' cas2.out
CASSCF energy = -75.6231442712753
```

3.6.4 CASSCF Reference

For this small (8, 8) active space, we can also compare the above DMRG results with the CASSCF result:

```
from pyscf import gto, scf, lib, mcscf
import os

mol = gto.M(atom='C 0 0 0; C 0 0 1.2425', basis='ccpvdz',
    symmetry='d2h', verbose=4, max_memory=10000) # mem in MB
mf = scf.RHF(mol)
mf.kernel()

from pyscf.mcscf import avas
nactorb, nactelec, coeff = avas.avas(mf, ["C 2p", "C 3p", "C 2s", "C 3s"])
print('CAS = ', nactorb, nactelec)

mc = mcscf.CASSCF(mf, nactorb, nactelec)
mc.fcisolver.conv_tol = 1E-10
mc.canonicalization = True
mc.natorb = True
mc.kernel(coeff)
```

This will generate the following output:

```
$ grep 'CASSCF energy' cas3.out
CASSCF energy = -75.6231442712446
```

3.6.5 State-Average with Different Spins

The following is an example python script for state-averaged DMRGSCF with singlet and triplet:

```
from pyscf import gto, scf, lib, dmrgscf, mcscf
import os

dmrgscf.settings.BLOCKEXE = os.popen("which block2main").read().strip()
dmrgscf.settings.MPIPREFIX = ''

mol = gto.M(atom='C 0 0 0; C 0 0 1.2425', basis='ccpvdz',
    symmetry='d2h', verbose=4, max_memory=10000) # mem in MB
mf = scf.RHF(mol)
mf.kernel()
```

(continues on next page)

(continued from previous page)

```
from pyscf.mcscf import avas
nactorb, nactelec, coeff = avas.avas(mf, ["C 2p", "C 3p", "C 2s", "C 3s"])
print('CAS = ', nactorb, nactelec)

lib.param.TMPDIR = os.path.abspath(lib.param.TMPDIR)

solvers = [dmrgscf.DMRGCI(mol, maxM=1000, tol=1E-10) for _ in range(2)]
weights = [1.0 / len(solvers)] * len(solvers)

solvers[0].spin = 0
solvers[1].spin = 2

for i, mcf in enumerate(solvers):
    mcf.runtimeDir = lib.param.TMPDIR + "/%d" % i
    mcf.scratchDirectory = lib.param.TMPDIR + "/%d" % i
    mcf.threads = 8
    mcf.memory = int(mol.max_memory / 1000) # mem in GB

mc = mcscf.CASSCF(mf, nactorb, nactelec)
mcscf.state_average_mix_(mc, solvers, weights)

mc.canonicalization = True
mc.natorb = True
mc.kernel(coeff)
```

Note: The `mc` parameter in the function `state_average_mix_` must be a `CASSCF` object. It cannot be a `DMRGSCF` object (will produce a runtime error).

This will generate the following output:

```
$ grep 'State ' cas4.out
State 0 weight 0.5  E = -75.6175232350073 S^2 = 0.0000000
State 1 weight 0.5  E = -75.298522666384  S^2 = 2.0000000
```

3.6.6 Unrestricted DMRGSCF

One can also perform Unrestricted CASSCF (UCASSCF) with `block2` using a UHF reference. Currently this is not directly supported by the `pyscf/dmrgscf` package, but here we can add some small modifications. The following is an example:

```
from pyscf import gto, scf, lib, dmrgscf, mcscf, fci
import os
```

(continues on next page)

(continued from previous page)

```
dmrgscf.settings.BLOCKEXE = os.popen("which block2main").read().strip()
dmrgscf.settings.MPIPREFIX = ''

mol = gto.M(atom='C 0 0 0; C 0 0 1.2425', basis='ccpvdz',
    symmetry=False, verbose=4, max_memory=10000) # mem in MB
mf = scf.UHF(mol)
mf.kernel()

def write_uhf_fcidump(DMRGCI, h1e, g2e, n_sites, nelec, ecore=0, tol=1E-15):

    import numpy as np
    from pyscf import ao2mo
    from subprocess import check_call
    from block2 import FCIDUMP, VectorUInt8

    if isinstance(nelec, (int, np.integer)):
        na = nelec // 2 + nelec % 2
        nb = nelec - na
    else:
        na, nb = nelec

    assert isinstance(h1e, tuple) and len(h1e) == 2
    assert isinstance(g2e, tuple) and len(g2e) == 3

    mh1e_a = h1e[0][np.tril_indices(n_sites)]
    mh1e_b = h1e[1][np.tril_indices(n_sites)]
    mh1e_a[np.abs(mh1e_a) < tol] = 0.0
    mh1e_b[np.abs(mh1e_b) < tol] = 0.0

    g2e_aa = ao2mo.restore(8, g2e[0], n_sites)
    g2e_bb = ao2mo.restore(8, g2e[2], n_sites)
    g2e_ab = ao2mo.restore(4, g2e[1], n_sites)
    g2e_aa[np.abs(g2e_aa) < tol] = 0.0
    g2e_bb[np.abs(g2e_bb) < tol] = 0.0
    g2e_ab[np.abs(g2e_ab) < tol] = 0.0

    mh1e = (mh1e_a, mh1e_b)
    mg2e = (g2e_aa, g2e_bb, g2e_ab)

    cmd = ' '.join((DMRGCI.mpiprefix, "mkdir -p", DMRGCI.scratchDirectory))
    check_call(cmd, shell=True)
    if not os.path.exists(DMRGCI.runtimeDir):
        os.makedirs(DMRGCI.runtimeDir)
```

(continues on next page)

(continued from previous page)

```
    fd = FCIDUMP()
    fd.initialize_sz(n_sites, na + nb, na - nb, 1, ecore, mh1e, mg2e)
    fd.orb_sym = VectorUInt8([1] * n_sites)
    integral_file = os.path.join(DMRGCI.runtimeDir, DMRGCI.integralFile)
    fd.write(integral_file)
    DMRGCI.groupname = None
    DMRGCI.nonspinAdapted = True
    return integral_file

def make_rdm12s(DMRGCI, state, norb, nelec, **kwargs):

    import numpy as np

    if isinstance(nelec, (int, np.integer)):
        na = nelec // 2 + nelec % 2
        nb = nelec - na
    else:
        na, nb = nelec

    file2pdm = "2pdm-%d-%d.npy" % (state, state) if DMRGCI.nroots > 1 else "2pdm.npy"
    dm2 = np.load(os.path.join(DMRGCI.scratchDirectory, "node0", file2pdm))
    dm2 = dm2.transpose(0, 1, 4, 2, 3)
    dm1a = np.einsum('ikjj->ki', dm2[0]) / (na - 1)
    dm1b = np.einsum('ikjj->ki', dm2[2]) / (nb - 1)

    return (dm1a, dm1b), dm2

dmrgscf.dmrgci.writeIntegralFile = write_uhf_fcidump
dmrgscf.DMRGCI.make_rdm12s = make_rdm12s

mc = mcscf.UCASSCF(mf, 8, 8)
mc.fcisolver = dmrgscf.DMRGCI(mol, maxM=1000, tol=1E-7)
mc.fcisolver.runtimeDir = lib.param.TMPDIR
mc.fcisolver.scratchDirectory = lib.param.TMPDIR
mc.fcisolver.threads = int(os.environ["OMP_NUM_THREADS"])
mc.fcisolver.memory = int(mol.max_memory / 1000) # mem in GB

mc.canonicalization = True
mc.natorb = True
mc.kernel()
```

Note: In the above example, `mf` is the `UHF` object and `mc` is the `UCASSCF` object. It is important to ensure that both of them are with unrestricted orbitals. Otherwise the calculation may be done with only restricted orbitals. `DMRGSCF` wrapper cannot be used for this example.

Note: Due to limitations in pyscf/UCASCI, currently the point group symmetry is not supported in UCASSCF/UCASCI with DMRG solver. pyscf/avas does not support creating active space with unrestricted orbtials so here we did not use avas. The above example will not work with StackBlock (the compatibility with StackBlock will be considered in future).

This will generate the following output:

```
$ grep 'UCASSCF energy' cas5.out
UCASSCF energy = -75.6231442541606
```

3.6.7 UCASSCF Reference

We compare the above DMRG results with the UCASSCF result using the FCI solver:

```
mc = mcscf.UCASSCF(mf, 8, 8)
mc.fcisolver.conv_tol = 1E-10
mc.canonicalization = True
mc.natorb = True
mc.kernel(coeff)
```

This will generate the following output:

```
$ grep 'UCASSCF energy' cas6.out
UCASSCF energy = -75.6231442706386
```

3.6.8 DMRGSCF Nuclear Gradients and Geometry Optimization

The following is an example python script for computing DMRGSCF nuclear gradients and geometry optimization using block2:

```
from pyscf import gto, scf, lib, dmrgscf
import os

dmrgscf.settings.BLOCKEXE = os.popen("which block2main").read().strip()
dmrgscf.settings.MPIPREFIX = ''

mol = gto.M(atom='C 0 0 0; C 0 0 1.2425', basis='ccpvdz',
    symmetry='d2h', verbose=4, max_memory=10000) # mem in MB
mf = scf.RHF(mol)
mf.kernel()

from pyscf.mcscf import avas
nactorb, nactelec, coeff = avas.avas(mf, ["C 2p", "C 3p", "C 2s", "C 3s"])
print('CAS = ', nactorb, nactelec)
```

(continues on next page)

(continued from previous page)

```
mc = mcscf.CASSCF(mf, nactorb, nactelec)
mc.fcisolver = dmrgscf.DMRGCI(mol, maxM=1000, tol=1E-10)
mc.fcisolver.runtimeDir = lib.param.TMPDIR
mc.fcisolver.scratchDirectory = lib.param.TMPDIR
mc.fcisolver.threads = int(os.environ.get("OMP_NUM_THREADS", 4))
mc.fcisolver.memory = int(mol.max_memory / 1000) # mem in GB

mc.canonicalization = True
mc.natorb = True
mc.kernel(coeff)

grad = mc.nuc_grad_method().kernel()

mol_eq = mc.nuc_grad_method().optimizer(solver='geomeTRIC').kernel()
print(mol_eq.atom_coords())
```

This will generate the following output (the nuclear gradient at the initial geometry and the optimized geometry):

```
$ grep -A 4 'SymAdaptedCASSCF gradients' cas7.out
--------------- SymAdaptedCASSCF gradients ---------------
        x                y                z
0 C     0.0000000000     0.0000000000     0.0388202961
1 C     0.0000000000     0.0000000000    -0.0388202961
----------------------------------------------
$ tail -n 3 cas7.out
cycle 3: E = -75.6240204052  dE = -5.51573e-07  norm(grad) = 9.37108e-05
[[ 0.          0.         -1.19709701]
[ 0.          0.          1.19709701]]
```

Note: Currently, gradients for UCASSCF is not supported in `pyscf`. The geometry optimization part requires an additional module called `geomeTRIC`, which can be installed via `pip install geometric`.

3.6.9 DMRG-SC-NEVPT2

The following is an example python script for a DMRG-SC-NEVPT2 calculation (with explicit 4pdm) using `block2`:

```
from pyscf import gto, scf, mcscf, mrpt, dmrgscf, lib
import os

dmrgscf.settings.BLOCKEXE = os.popen("which block2main").read().strip()
```

(continues on next page)

(continued from previous page)

```
dmrgscf.settings.MPIPREFIX = ''

mol = gto.M(atom='O 0 0 0; O 0 0 1.207', basis='cc-pvdz', spin=2, verbose=4)
mf = scf.RHF(mol).run(conv_tol=1E-20)

mc = mcscf.CASSCF(mf, 6, 8)

mc.fcisolver = dmrgscf.DMRGCI(mol, maxM=500, tol=1E-10)
mc.fcisolver.runtimeDir = os.path.abspath(lib.param.TMPDIR)
mc.fcisolver.scratchDirectory = os.path.abspath(lib.param.TMPDIR)
mc.fcisolver.threads = 8
mc.fcisolver.memory = int(mol.max_memory / 1000) # mem in GB

mc.fcisolver.conv_tol = 1e-14
mc.canonicalization = True
mc.natorb = True
mc.run()

sc = mrpt.NEVPT(mc).run()
```

The alternative faster compress_approx approach using MPS compression is also supported:

```
from pyscf import gto, scf, mcscf, mrpt, dmrgscf, lib
import os

dmrgscf.settings.BLOCKEXE = os.popen("which block2main").read().strip()
dmrgscf.settings.BLOCKEXE_COMPRESS_NEVPT = os.popen("which block2main").read().
↪strip()
dmrgscf.settings.MPIPREFIX = ''

mol = gto.M(atom='O 0 0 0; O 0 0 1.207', basis='cc-pvdz', spin=2, verbose=4)
mf = scf.RHF(mol).run(conv_tol=1E-20)

mc = mcscf.CASSCF(mf, 6, 8)

mc.fcisolver = dmrgscf.DMRGCI(mol, maxM=500, tol=1E-10)
mc.fcisolver.runtimeDir = os.path.abspath(lib.param.TMPDIR)
mc.fcisolver.scratchDirectory = os.path.abspath(lib.param.TMPDIR)
mc.fcisolver.threads = 8
mc.fcisolver.memory = int(mol.max_memory / 1000) # mem in GB

mc.fcisolver.conv_tol = 1e-14
mc.canonicalization = True
mc.natorb = True
mc.run()
```

(continues on next page)

(continued from previous page)

```
sc = mrpt.NEVPT(mc).compress_approx(maxM=200).run()
```

This will generate the following output (for `compress_approx` approach):

```
$ grep 'CASSCF energy' sc-nevpt2.out
CASSCF energy = -149.708657771219
$ grep 'Nevpt2 Energy' sc-nevpt2.out
Nevpt2 Energy = -0.249182302692906
```

So the total NEVPT2 energy using the `compress_approx` approach is `-149.708657771219 + -0.249182302692906 = -149.9578400739119`.

Note: The first "4pdm" approach is not supported by `StackBlock`, but it is supported in the old `Block` code. The second "compression" approach is supported by `StackBlock`. `Block2` supports both approaches.

When using the second approach, it will generate a warning saying that `WARN: DMRG executable file for nevptsolver is the same to the executable file for DMRG solver. If they are both compiled by MPI compilers, they may cause error or random results in DMRG-NEVPT calculation.`. Please ignore this warning for `block2`. For `block2`, it is okay to set `BLOCKEXE` and `BLOCKEXE_COMPRESS_NEVPT` to the same file. `BLOCKEXE_COMPRESS_NEVPT` can be compiled with or without MPI. So only a single version of `block2main` is required. If you want to use MPI, please set both `BLOCKEXE` and `BLOCKEXE_COMPRESS_NEVPT` to the same `block2main` and compile `block2` with MPI, or use `pip install block2-mpi`, and then set an appropriate `MPIPREFIX`.

The second "compression" approach requires the `mpi4py` python package. Make sure `import mpi4py` works in python before trying this example. Also, make sure that the file `${PYSCFHOME}/pyscf/dmrgscf/nevpt_mpi.py` has the `execute` permission. You can do `chmod +x ${PYSCFHOME}/pyscf/dmrgscf/nevpt_mpi.py` to fix the permission.

Note that for the second "compression" approach, if you need to add any extra keywords for the DMRG solver, such as `singlet_embedding`, you need to add it using `mc.fcisolver.block_extra_keyword` instead of `mc.fcisolver.extraline`.

3.6.10 DMRG-SC-NEVPT2 (Multi-State)

The following is an example input file for state-averaged DMRGSCF for three states, and then the SC-NEVPT2 treatment of each of the three states.

```
import numpy as np
from pyscf import gto, scf, mcscf, mrpt, dmrgscf, lib
import os

dmrgscf.settings.BLOCKEXE = os.popen("which block2main").read().strip()
dmrgscf.settings.BLOCKEXE_COMPRESS_NEVPT = os.popen("which block2main").read().
```

(continues on next page)

(continued from previous page)

```
↪strip()
dmrgscf.settings.MPIPREFIX = ''

mol = gto.M(atom='O 0 0 0; O 0 0 1.207', basis='cc-pvdz', spin=2, verbose=4)
mf = scf.RHF(mol).run(conv_tol=1E-20)

# state average casscf
mc = mcscf.CASSCF(mf, 6, 8)
mc.fcisolver = dmrgscf.DMRGCI(mol, maxM=500, tol=1E-10)
mc.fcisolver.runtimeDir = os.path.abspath(lib.param.TMPDIR)
mc.fcisolver.scratchDirectory = os.path.abspath(lib.param.TMPDIR)
mc.fcisolver.threads = 8
mc.fcisolver.memory = int(mol.max_memory / 1000) # mem in GB
mc.fcisolver.conv_tol = 1e-14
mc.fcisolver.nroots = 3
mc = mcscf.state_average_(mc, [1.0 / 3] * 3)
mc.kernel()
mf.mo_coeff = mc.mo_coeff

# need an extra casci before calling mrpt
mc = mcscf.CASCI(mf, 6, 8)
mc.fcisolver = dmrgscf.DMRGCI(mol, maxM=500, tol=1E-10)
mc.fcisolver.runtimeDir = os.path.abspath(lib.param.TMPDIR)
mc.fcisolver.scratchDirectory = os.path.abspath(lib.param.TMPDIR)
mc.fcisolver.threads = 8
mc.fcisolver.memory = int(mol.max_memory / 1000) # mem in GB
mc.fcisolver.conv_tol = 1e-14
mc.fcisolver.nroots = 3
mc.natorb = True
mc.kernel()

# canonicalization for each state
ms = [None] * mc.fcisolver.nroots
cs = [None] * mc.fcisolver.nroots
es = [None] * mc.fcisolver.nroots
for ir in range(mc.fcisolver.nroots):
    ms[ir], cs[ir], es[ir] = mc.canonicalize(mc.mo_coeff, ci=mc.ci[ir], cas_
↪natorb=False)

refs = [-149.956650684550, -149.725338427894, -149.725338427894]

# mrpt
for ir in range(mc.fcisolver.nroots):
    mc.mo_coeff, mc.ci, mc.mo_energy = ms[ir], cs, es[ir]
    mr = mrpt.nevpt2.NEVPT(mc).set(canonicalized=True).compress_approx(maxM=200).
```

(continues on next page)

(continued from previous page)

```
↪run(root=ir)
    print('root =', ir, 'E =', mc.e_tot[ir] + mr.e_corr, 'diff =', mc.e_tot[ir] + mr.
↪e_corr - refs[ir])
```

This will generate the following output:

```
$ grep 'diff' multi.out
root = 0 E = -149.95664910937998 diff = 1.5751700175314909e-06
root = 1 E = -149.72529848179465 diff = 3.994609934920845e-05
root = 2 E = -149.7252985999243 diff = 3.9827969715133804e-05
```

Note: The above script should generate the same result if the explicit 4PDM approach is used, by removing `.compress_approx(maxM=200)`.

Changing `mc.fcisolver` to the default FCI active space solver should also generate the same result (note that `.compress_approx(maxM=200)` is not supported by the FCI active space solver).

When the FCI active space solver is used, explicit canonicalization is also optional, namely, one can also remove `.set(canonicalized=True)` and `mc.mo_coeff, mc.ci, mc.mo_energy = ms[ir], cs, es[ir]` and the result will still be the same.

3.6.11 DMRG-IC-NEVPT2

The following is an example python script for SC-NEVPT2 / IC-NEVPT2 with equations derived on the fly (using the FCI solver):

```
import numpy
from pyscf import gto, scf, mcscf

mol = gto.M(atom='O 0 0 0; O 0 0 1.207', basis='cc-pvdz', spin=2, verbose=4)
mf = scf.RHF(mol).run(conv_tol=1E-20)

mc = mcscf.CASSCF(mf, 6, 8)
mc.fcisolver.conv_tol = 1e-14
mc.conv_tol = 1e-11
mc.canonicalization = True
mc.run()

from pyblock2.icmr.scnevpt2 import WickSCNEVPT2
wsc = WickSCNEVPT2(mc).run()

from pyblock2.icmr.icnevpt2_full import WickICNEVPT2
wic = WickICNEVPT2(mc).run()
```

This will generate the following output:

```
$ grep 'E(WickSCNEVPT2)' nevpt2.out
E(WickSCNEVPT2) = -149.9578403403482  E_corr_pt = -0.2491825691128931
$ grep 'E(WickICNEVPT2)' nevpt2.out
E(WickICNEVPT2) = -149.9601376470851  E_corr_pt = -0.2514798758497859
```

The above example can also run with the block2 DMRG solver:

```
import numpy
from pyscf import gto, scf, mcscf, dmrgscf, lib
import os

if not os.path.exists(lib.param.TMPDIR):
    os.mkdir(lib.param.TMPDIR)

dmrgscf.settings.BLOCKEXE = os.popen("which block2main").read().strip()
dmrgscf.settings.MPIPREFIX = ''

mol = gto.M(atom='O 0 0 0; O 0 0 1.207', basis='cc-pvdz', spin=2, verbose=4)
mf = scf.RHF(mol).run(conv_tol=1E-20)

mc = mcscf.CASSCF(mf, 6, 8)

mc.fcisolver = dmrgscf.DMRGCI(mol, maxM=500, tol=1E-14)
mc.fcisolver.runtimeDir = os.path.abspath(lib.param.TMPDIR)
mc.fcisolver.scratchDirectory = os.path.abspath(lib.param.TMPDIR)
mc.fcisolver.threads = 28
mc.fcisolver.memory = int(mol.max_memory / 1000) # mem in GB

# set very tight thresholds for small system
mc.fcisolver.scheduleSweeps = [0, 4, 8, 12, 16]
mc.fcisolver.scheduleMaxMs = [250, 500, 500, 500, 500]
mc.fcisolver.scheduleTols = [1e-08, 1e-10, 1e-12, 1e-12, 1e-12]
mc.fcisolver.scheduleNoises = [0.0001, 0.0001, 5e-05, 5e-05, 0.0]
mc.fcisolver.maxIter = 30
mc.fcisolver.twodot_to_onedot = 20
mc.fcisolver.block_extra_keyword = ['singlet_embedding', 'full_fci_space', 'fp_cps_
↪cutoff 0', 'cutoff 0']

mc.fcisolver.conv_tol = 1e-14
mc.conv_tol = 1e-11
mc.canonicalization = True
mc.run()

from pyblock2.icmr.scnevpt2 import WickSCNEVPT2
wsc = WickSCNEVPT2(mc).run()
```

(continues on next page)

(continued from previous page)

```
from pyblock2.icmr.icnevpt2_full import WickICNEVPT2
wic = WickICNEVPT2(mc).run()
```

This will generate the following output:

```
$ grep 'E(WickSCNEVPT2)' dmrg-nevpt2.out
E(WickSCNEVPT2) = -149.9578400627551  E_corr_pt = -0.2491822915198339
$ grep 'E(WickICNEVPT2)' dmrg-nevpt2.out
E(WickICNEVPT2) = -149.9601376425396  E_corr_pt = -0.2514798713043632
```

3.6.12 DMRG-FIC-MRCISD

The following is an example python script for fully internally contracted MRCISD with equations derived on the fly (using the FCI solver):

```
# need first import numpy (before pyblock2)
# otherwise the numpy multi-threading may not work
import numpy

from pyscf import gto, scf, mcscf
from pyblock2.icmr.icmrcisd_full import WickICMRCISD

mol = gto.M(atom='O 0 0 0; O 0 0 1.207', basis='6-31g', spin=2, verbose=4)
mf = scf.RHF(mol).run(conv_tol=1E-20)

mc = mcscf.CASSCF(mf, 6, 8)
mc.fcisolver.conv_tol = 1e-14
mc.conv_tol = 1e-11
mc.run()

mol.verbose = 5
wsc = WickICMRCISD(mc).run()
```

This will generate the following output:

```
$ grep 'CASSCF energy' mrci.out
CASSCF energy = -149.636563280267
$ grep 'WickICMRCISD' mrci.out
E(WickICMRCISD)   = -149.7792742741091  E_corr_ci = -0.1427109938418027
E(WickICMRCISD+Q) = -149.7858102349944  E_corr_ci = -0.1492469547270254
```

Similarly, we can do DMRG-FIC-MRCISD:

```
# need first import numpy (before pyblock2)
# otherwise the numpy multi-threading may not work
```

(continues on next page)

(continued from previous page)

```
import numpy

from pyscf import gto, scf, mcscf, dmrgscf, lib
from pyblock2.icmr.icmrcisd_full import WickICMRCISD
import os

if not os.path.exists(lib.param.TMPDIR):
    os.mkdir(lib.param.TMPDIR)

dmrgscf.settings.BLOCKEXE = os.popen("which block2main").read().strip()
dmrgscf.settings.MPIPREFIX = ''

mol = gto.M(atom='O 0 0 0; O 0 0 1.207', basis='6-31g', spin=2, verbose=4)
mf = scf.RHF(mol).run(conv_tol=1E-20)

mc = mcscf.CASSCF(mf, 6, 8)

mc.fcisolver = dmrgscf.DMRGCI(mol, maxM=500, tol=1E-14)
mc.fcisolver.runtimeDir = os.path.abspath(lib.param.TMPDIR)
mc.fcisolver.scratchDirectory = os.path.abspath(lib.param.TMPDIR)
mc.fcisolver.threads = 28
mc.fcisolver.memory = int(mol.max_memory / 1000) # mem in GB

# set very tight thresholds for small system
mc.fcisolver.scheduleSweeps = [0, 4, 8, 12, 16]
mc.fcisolver.scheduleMaxMs = [250, 500, 500, 500, 500]
mc.fcisolver.scheduleTols = [1e-08, 1e-10, 1e-12, 1e-12, 1e-12]
mc.fcisolver.scheduleNoises = [0.0001, 0.0001, 5e-05, 5e-05, 0.0]
mc.fcisolver.maxIter = 30
mc.fcisolver.twodot_to_onedot = 20
mc.fcisolver.block_extra_keyword = ['singlet_embedding', 'full_fci_space', 'fp_cps_
↪cutoff 0', 'cutoff 0']

mc.fcisolver.conv_tol = 1e-14
mc.conv_tol = 1e-11
mc.run()

mol.verbose = 5
wsc = WickICMRCISD(mc).run()
```

This will generate the following output:

```
$ grep 'CASSCF energy' dmrg-mrci.out
CASSCF energy = -149.636563280264
$ grep 'WickICMRCISD' dmrg-mrci.out
E(WickICMRCISD)   = -149.7792742857885  E_corr_ci = -0.1427110055241769
```

(continues on next page)

(continued from previous page)

```
E(WickICMRCISD+Q) = -149.785810250064  E_corr_ci = -0.1492469697996863
```

Note: The current FIC-MRCI / DMRG-FIC-MRCI implementation requires the explicit construction of the MRCI Hamiltonian, which is not practical for production runs.

3.7 DMRGSCF (OpenMOLCAS)

In this section we explain how to use `block2` and `OpenMOLCAS` for CASSCF and CASPT2 with DMRG as the active space solver.

3.7.1 Preparation

First, make sure `block2` is installed correctly (either compiled manually or installed using `pip`, and for `pip` the version of block2 should be `>=0.5.1rc17`), so that the command `which block2main` can print a valid file path to `block2main`.

For example, the required `block2` can be installed using `pip` as:

```
pip install block2>=0.5.1rc17 --extra-index-url=https://block-hczhai.github.io/
↪block2-preview/pypi/
```

Then we need to compile an OpenMOLCAS with the `block2` interface. The source code of the required OpenMOLCAS code can be found in https://github.com/hczhai/OpenMolcas, which is a slightly modified version of https://github.com/quanp/OpenMolcas (the OpenMOLCAS interface for the `block 1.5` and `StackBlock`). To activate the `blcok2` interface, run `cmake` for this OpenMOLCAS with the option `-DBLOCK2=ON`. The detailed procedure is as follows:

```
git clone https://github.com/hczhai/OpenMolcas
export MOLCASHOME=$PWD/OpenMolcas
cd OpenMolcas
mkdir build
cd build
CC=gcc CXX=g++ FC=gfortran MKLROOT=/usr/local cmake .. -DCMAKE_INSTALL_PREFIX=../
↪install -DLINALG=MKL -DOPENMP=ON -DBLOCK2=ON
make -j 10
make install
```

Remember to change the `MKLROOT` variable in the above example for your case.

Then one can run OpenMolcas using the following command:

```
MOLCAS=$MOLCASHOME/install MOLCAS_WORKDIR=/content/tmp pymolcas test.in
```

Where `test.in` is an OpenMolcas input file. Sometimes you may need to add the `--not-here` option to `pymolcas` if it cannot find the `molcas` executable.

3.7.2 DMRGSCF

The following is an example input file for DMRGSCF for a O2 triplet state (see *DMRGSCF (pyscf)* for the similar calculation using `pyscf`):

```
&GATEWAY
    Title
    O2 Molecule
    Coord
    2

    O 0 0 -0.6035
    O 0 0 0.6035
    Basis set
    CC-PVDZ

&SEWARD

&SCF
    Spin = 1

&RASSCF
    Spin
    3
    Symmetry
    4
    nActEl
    2 0 0
    Inactive
    3 1 1 0 2 0 0 0
    Ras2
    0 0 0 0 0 1 1 0
    CIROOT = 1 1 ; 1

&RASSCF
    Spin
    3
    Symmetry
    4
    nActEl
    8 0 0
    Inactive
    2 0 0 0 2 0 0 0
```

(continues on next page)

(continued from previous page)

```
    Ras2
    1 1 1 0 1 1 1 0
    CIROOT = 1 1 ; 1
    CISOlver = BLOCK
    DMRG = 1000
```

Note that the first RASSCF is actually a ROHF mean-field calculation.

The same calculation in pyscf is:

```
from pyscf import gto, scf, mcscf, mrpt, dmrgscf, lib, symm
from pyblock2._pyscf.ao2mo import integrals as itg
import os

mol = gto.M(atom='O 0 0 0; O 0 0 1.207', basis='cc-pvdz', spin=2, symmetry='d2h',␣
↪cart=False, verbose=4)
mf = scf.RHF(mol).run(conv_tol=1E-20)

ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf, g2e_symm=8)

print(orb_sym)
print(mf.mo_occ)
orb_sym_name = [symm.irrep_id2name(mol.groupname, ir) for ir in orb_sym]
print(orb_sym_name)

mc = mcscf.CASSCF(mf, 6, 8)

mc.fcisolver.conv_tol = 1e-14
mc.canonicalization = True
mc.natorb = True
mc.run()
```

From the pyscf output we can see the occupation number and orbtial irreps are :

```
[0, 5, 0, 5, 0, 6, 7, 2, 3, 5, 5, 6, 7, 0, 2, 3, 0, 5, 6, 7, 0, 1, 4, 5, 0, 2, 3, 5]
↪# XOR irreps
[2. 2. 2. 2. 2. 2. 2. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
↪] # occ
['Ag', 'B1u', 'Ag', 'B1u', 'Ag', 'B2u', 'B3u', 'B2g', 'B3g', 'B1u', 'B1u', 'B2u',
↪'B3u', 'Ag', 'B2g', 'B3g', 'Ag', 'B1u', 'B2u', 'B3u', 'Ag', 'B1g', 'Au', 'B1u', 'Ag
↪', 'B2g', 'B3g', 'B1u']
```

The MOLCAS ordering of irreps of D2h is:

```
ag b3u b2u b1g b1u b2g b3g au
```

This information can help us setting the Inactive and Ras2 in the MOLCAS inputfile.

From the pyscf output we have:

```
$ grep 'converged SCF energy' pyscf.out
converged SCF energy = -149.608181589162
$ grep 'CASSCF energy' pyscf.out
CASSCF energy = -149.708657770064
```

From the openMOLCAS output we have:

```
$ grep '::    RASSCF' o2.out
::    RASSCF root number  1 Total energy:   -149.60818159
::    RASSCF root number  1 Total energy:   -149.70865773
```

Note that in the openMOLCAS output, the first line is actually the SCF (ROHF) energy, and the second line is the CASSCF energy. So they are consistent.

3.7.3 DMRG-cu-CASPT2

The following is an example input file for CASPT2 calculation after DMRGSCF for a O2 triplet state. In this example, the cumulant approximation of 4PDM is used for CASPT2. Note that the IPEA shift $= 0.25$ is used by default.

```
&GATEWAY
    Title
    O2 Molecule
    Coord
    2

    O 0 0 -0.6035
    O 0 0 0.6035
    Basis set
    CC-PVDZ

&SEWARD

&SCF
    Spin = 1

&RASSCF
    Spin
    3
    Symmetry
    4
    nActEl
    2 0 0
    Inactive
    3 1 1 0 2 0 0 0
```

(continues on next page)

(continued from previous page)

```
    Ras2
    0 0 0 0 0 1 1 0
    CIROOT = 1 1 ; 1

&RASSCF
    Spin
    3
    Symmetry
    4
    nActEl
    8 0 0
    Inactive
    2 0 0 0 2 0 0 0
    Ras2
    1 1 1 0 1 1 1 0
    CIROOT = 1 1 ; 1
    CISOlver = BLOCK
    DMRG = 1000
    3RDM
    NO4R

&CASPT2
    MULT = 1 1
    CUMU
```

The keyword `NO4R` is required in the `RASSCF` section to avoid spending time on computing 4pdms.

This will generate the following output:

```
$ grep '::    CASPT2' o2.out
::    CASPT2 Root  1     Total energy:   -149.97055932
```

3.7.4 DMRG-CASPT2

The following is an example input file for CASPT2 calculation after DMRGSCF for a O2 triplet state. In this example, the exact 4PDM is computed and used.

```
&GATEWAY
    Title
    O2 Molecule
    Coord
    2

    O 0 0 -0.6035
    O 0 0 0.6035
```

(continues on next page)

(continued from previous page)

```
    Basis set
    CC-PVDZ

&SEWARD

&SCF
    Spin = 1

&RASSCF
    Spin
    3
    Symmetry
    4
    nActEl
    2 0 0
    Inactive
    3 1 1 0 2 0 0 0
    Ras2
    0 0 0 0 0 1 1 0
    CIROOT = 1 1 ; 1

&RASSCF
    Spin
    3
    Symmetry
    4
    nActEl
    8 0 0
    Inactive
    2 0 0 0 2 0 0 0
    Ras2
    1 1 1 0 1 1 1 0
    CIROOT = 1 1 ; 1
    CISOlver = BLOCK
    DMRG = 1000
    3RDM

&CASPT2
    BLOCK
    MULT = 1 1
```

In the above example, we use the keyword `BLOCK` to replace the old keyword `CUMU` so that the cumulant approximation is not used.

Note: By default there will be frozen orbitals in the CASPT2 treatment. One can add

```
FROZEN
0 0 0 0 0 0 0 0
```

in the CASPT2 section in the above example to avoid frozen orbitals.

This will generate the following output:

```
$ grep '::    CASPT2' o2.out
::    CASPT2 Root  1     Total energy:   -149.96959847
```

3.7.5 State-Average

The following is an example input file for state-averaged DMRGSCF for three states, and then the CASPT2 treatment of each of the three states. In this example, the exact 4PDM is computed and used.

```
&GATEWAY
    Title
    O2 Molecule
    Coord
    2

    O 0 0 -0.6035
    O 0 0 0.6035
    Basis set
    CC-PVDZ

&SEWARD

&SCF
    Spin = 1

&RASSCF
    Spin
    3
    Symmetry
    4
    nActEl
    2 0 0
    Inactive
    3 1 1 0 2 0 0 0
    Ras2
    0 0 0 0 0 1 1 0
    CIROOT = 1 1 ; 1
```

(continues on next page)

(continued from previous page)

```
&RASSCF
    Spin
    3
    Symmetry
    4
    nActEl
    8 0 0
    Inactive
    2 0 0 0 2 0 0 0
    Ras2
    1 1 1 0 1 1 1 0
    CIROOT = 3 3 1
    CISOlver = BLOCK
    DMRG = 1000
    3RDM

&CASPT2
    BLOCK
    MULT = 1 1

&CASPT2
    BLOCK
    MULT = 1 2

&CASPT2
    BLOCK
    MULT = 1 3
```

From the output we have:

```
$ grep '::    RASSCF' o2.out
::    RASSCF root number  1 Total energy:   -149.60818159
::    RASSCF root number  1 Total energy:   -149.69063345
::    RASSCF root number  2 Total energy:   -149.09370540
::    RASSCF root number  3 Total energy:   -148.86158577
$ grep '::    CASPT2' o2.out
::    CASPT2 Root  1     Total energy:   -149.96175902
::    CASPT2 Root  1     Total energy:   -149.39685470
::    CASPT2 Root  1     Total energy:   -149.13012648
```

3.8 DMRGSCF (forte)

In this section we explain how to use `block2` and `forte` for CASSCF and DSRG calculations with DMRG as the active space solver.

`forte` is an open-source package for strongly correlated methods, developed by Evangelista group. The detailed instruction for the installation and the usage of the code can be found in https://forte.readthedocs.io/.

3.8.1 Preparation

First, we need to build and install the C++ library of `block2`. This can be done using the `-DBUILD_CLIB=ON` option:

```
git clone -b p0.5.3rc7 https://github.com/block-hczhai/block2-preview
cd block2-preview
mkdir build
cd build
cmake .. -DUSE_MKL=ON -DBUILD_CLIB=ON -DLARGE_BOND=ON -DMPI=OFF -DCMAKE_INSTALL_
↪PREFIX=../install
make -j 10
make install
export BLOCK2_DIR=$PWD/../install
cd ../..
```

After this, you will be able to find the block2 include files in `${BLOCK2_DIR}/include/` and `libblock2.so` in `${BLOCK2_DIR}/lib64/`. The `block2Config.cmake` file can be found in `${BLOCK2_DIR}/share/cmake/block2/`.

Second, we need to build psi4. Make sure an `eigen3` library is available in the system, which can be installed using `apt install libeigen3-dev` or `conda install -c omnia eigen3`. If you use the conda package, you may need to add the `cmake` option `-DCMAKE_PREFIX_PATH=${CONDA_PREFIX}` so that `cmake` can find it.

To save compiling time, one may install `libint2` using `conda install conda-forge::libint==2.8.1` before the following.

Then we can build `psi4` as follows:

```
git clone -b v1.9 https://github.com/psi4/psi4
git pull origin master --tags
cd psi4
mkdir build
cd build
export MATH_ROOT=${CONDA_PREFIX}
cmake .. -DCMAKE_PREFIX_PATH=${CONDA_PREFIX}
make -j 10
export PSI4_DIR=$PWD/stage
cd ../..
```

Then we can add the following environment variables:

```
export PATH=${PSI4_DIR}/bin:$PATH
export PYTHONPATH=${PSI4_DIR}/lib:$PYTHONPATH
export PSI_SCRATCH=/scratch/.../psi4      # use a valid scratch folder here
```

Then the command `psi4` should be available in the terminal. And `python -c 'import psi4'` should work. If there is the error `TypeError: issubclass() arg 1 must be a class`, try `pip install --force-reinstall typing-extensions==4.5.0`.

Third, we need to build abmit as follows:

```
git clone -b v0.7.1 https://github.com/jturney/ambit
cd ambit
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=../install -DENABLE_TESTS=OFF
make -j 10
make install
export AMBIT_DIR=$PWD/../install
cd ../..
```

Finally, we can build forte. Here we use a revised version with the `block2` interface, which can be found in the `block2_dmrg` branch of the forked repo https://github.com/hczhai/forte. To build it, we can:

```
git clone -b block2_dmrg https://github.com/hczhai/forte
cd forte
$(psi4 --plugin-compile) -Dambit_DIR=${AMBIT_DIR}/share/cmake/ambit \
    -DENABLE_block2=ON \
    -Dblock2_DIR=${BLOCK2_DIR}/share/cmake/block2 \
    -DCMAKE_PREFIX_PATH=${CONDA_PREFIX}
make -j 10
export FORTE_DIR=$PWD
cd ..
```

Then we can add the following environment variables:

```
export PYTHONPATH=${FORTE_DIR}:${AMBIT_DIR}/lib:$PYTHONPATH
```

Then `python -c 'import forte'` should work.

3.8.2 DMRG

The following is an example python script for DMRG for N2 in the minimal basis set:

```
import psi4
import forte

psi4.geometry("""
0 1
N 0.0 0.0 0.0
N 0.0 0.0 1.1
""")

psi4.set_options(
    {
        'basis': 'sto-3g',
        'scf_type': 'pk',
        'e_convergence': 14,
        'reference': 'rhf',
        'forte__active_space_solver': 'block2',
        'forte__block2_sweep_davidson_tols': [1E-15],
    }
)

psi4.energy('forte')
```

This will generate the following output:

```
$ grep 'Energy Summary' -A 4 dmrg.out | tail -1
1  (  0)    Ag     0     -107.654122447812   0.000000
```

3.8.3 DMRGSCF

The following is an example python script for DMRGSCF for an O2 triplet state (see *DMRGSCF (pyscf)* for the similar calculation using `pyscf`):

```
import psi4
import forte

psi4.geometry("""
0 3
O 0.0 0.0 -0.6035
O 0.0 0.0 0.6035
""")

psi4.set_options(
```

(continues on next page)

(continued from previous page)

```
    {
        'basis': 'cc-pvdz',
        'scf_type': 'direct',
        'e_convergence': 20,
        'reference': 'rohf',
        'forte__job_type': 'casscf',
        'forte__casscf_ci_solver': 'block2',
        'forte__block2_sweep_davidson_tols': [1E-15],
        'forte__restricted_docc': [2, 0, 0, 0, 0, 2, 0, 0],
        'forte__active': [1, 0, 1, 1, 0, 1, 1, 1],
        'forte__root_sym': 1, # B1g
    }
)

psi4.energy('forte')
```

This will generate the following output:

```
$ grep 'Energy Summary' -A 4 dmrg.out | grep B1g
3  (  0)   B1g     0     -149.671533509344   2.000000
3  (  0)   B1g     0     -149.689293451723   2.000000
3  (  0)   B1g     0     -149.703603100002   2.000000
3  (  0)   B1g     0     -149.708080545113   2.000000
3  (  0)   B1g     0     -149.708521258412   2.000000
3  (  0)   B1g     0     -149.708617815460   2.000000
3  (  0)   B1g     0     -149.708645817441   2.000000
3  (  0)   B1g     0     -149.708654215054   2.000000
3  (  0)   B1g     0     -149.708656716926   2.000000
3  (  0)   B1g     0     -149.708657458784   2.000000
3  (  0)   B1g     0     -149.708657678545   2.000000
3  (  0)   B1g     0     -149.708657743713   2.000000
3  (  0)   B1g     0     -149.708657763065   2.000000
3  (  0)   B1g     0     -149.708657768818   2.000000
3  (  0)   B1g     0     -149.708657770529   2.000000
3  (  0)   B1g     0     -149.708657771038   2.000000
3  (  0)   B1g     0     -149.708657771254   2.000000
```

3.8.4 DMRG-DSRG

The following is an example python script for DMRG-DSRG for an O2 triplet state, using the DMRGSCF state as the reference state:

```
import psi4
import forte

psi4.geometry("""
0 3
O 0.0 0.0 -0.6035
O 0.0 0.0 0.6035
""")

psi4.set_options(
    {
        'basis': 'cc-pvdz',
        'scf_type': 'direct',
        'e_convergence': 20,
        'reference': 'rohf',
        'forte__job_type': 'casscf',
        'forte__casscf_ci_solver': 'block2',
        'forte__block2_sweep_davidson_tols': [1E-15],
        'forte__restricted_docc': [2, 0, 0, 0, 0, 2, 0, 0],
        'forte__active': [1, 0, 1, 1, 0, 1, 1, 1],
        'forte__root_sym': 1, # B1g
    }
)

e, wfn = psi4.energy('forte', return_wfn=True)

psi4.set_options(
    {
        'forte__job_type': 'newdriver',
        'forte__active_space_solver': 'block2',
        'forte__correlation_solver': 'sa-mrdsrg',
        'forte__dsrg_s': 0.5,
    }
)

psi4.energy('forte', ref_wfn=wfn)
```

This will generate the following output:

```
$ grep 'E0 (reference)' dsrg.out
E0 (reference)                 =   -149.708657771253996
$ grep 'DSRG-MRPT2 correlation' -A 1 dsrg.out
```

(continues on next page)

(continued from previous page)

```
DSRG-MRPT2 correlation energy  =     -0.263404857500777
DSRG-MRPT2 total energy        =   -149.972062628754770
```

3.8.5 State-Average

The following is an example python script for state-averaged DMRGSCF for three states:

```
import psi4
import forte

psi4.geometry("""
0 3
O 0.0 0.0 -0.6035
O 0.0 0.0 0.6035
""")

psi4.set_options(
    {
        'basis': 'cc-pvdz',
        'scf_type': 'direct',
        'e_convergence': 20,
        'reference': 'rohf',
        'forte__job_type': 'casscf',
        'forte__casscf_ci_solver': 'block2',
        'forte__block2_sweep_davidson_tols': [1E-15],
        'forte__restricted_docc': [2, 0, 0, 0, 0, 2, 0, 0],
        'forte__active': [1, 0, 1, 1, 0, 1, 1, 1],
        'forte__avg_state': [[1, 3, 3]], # (B1g, triplet, 3 states)
    }
)

psi4.energy('forte')
```

This will generate the following output:

```
$ grep '==> Energy Summary <==' -A 6 03.out | tail -3
3  (  0)   B1g     0     -149.690635774964   2.000000
3  (  0)   B1g     1     -149.093708503131   2.000000
3  (  0)   B1g     2     -148.861580599165   2.000000
```

Note: For realistic calculations one should not rely on the default settings for the DMRG schedule. Customized schedule can be set using for example:

```
'forte__block2_sweep_n_sweeps': [4, 4, 4, 6],
'forte__block2_sweep_bond_dims': [250, 500, 1000, 1000],
'forte__block2_sweep_noises': [1E-4, 1E-5, 1E-5, 0],
'forte__block2_sweep_davidson_tols': [1E-5, 1E-7, 1E-7, 1E-9],
'forte__block2_energy_convergence': 1E-8,
'forte__block2_n_total_sweeps': 18,
'forte__block2_verbose': 2
```

3.9 MPS Import/Export

The block2 MPS can be translated into StackBlock format for restarting the calculation in StackBlock. Alternatively, the StackBlock rotation matrices and wavefunction can be translated into block2 MPS. Since different initial guess for MPS is generated in StackBlock and block2, this feature can be useful for sharing MPS initial guess among different codes, debugging, or performing some DMRG methods not implemented in one of the code.

The translation itself should be exact, with the support for both spin-adapted and non-spin-adapted case. If the canonical form is not LLL...KR, some small error may occur during the canonical form translation.

The script ${BLOCK2HOME}/pyblock2/driver/readwfn.py can be used to translate from StackBlock to block2. The script ${BLOCK2HOME}/pyblock2/driver/writewfn.py can be used to translate from block2 to StackBlock. These two scripts depend on block2, pyblock and StackBlock. To install pyblock and StackBlock, the boost package is required. We will first explain the installation of these extra dependencies.

3.9.1 Boost Installation

One can download the most recent version of boost in https://www.boost.org/users/download/ Assuming the downloaded file is named boost_1_76_0.tar.gz, stored in ~/program/boost-1.76 (you can choose any other directory). One can then install boost in the following way. Please make sure the correct version of C++ compiler is set in the environment. The same C++ compiler should be used for compiling boost and block2, pyblock and StackBlock.

```
$ mkdir ~/program/boost-1.76
$ cd ~/program/boost-1.76
$ PREFIX=$PWD
$ wget https://boostorg.jfrog.io/artifactory/main/release/1.76.0/source/boost_1_76_0.
↪tar.gz
$ tar zxf boost_1_76_0.tar.gz
$ cd boost_1_76_0
$ gcc --version
gcc (GCC) 9.2.0
$ bash bootstrap.sh
```

(continues on next page)

(continued from previous page)

```
$ echo 'using mpi ;' >> project-config.jam
$ ./b2 install --prefix=$PREFIX
$ echo $PREFIX
/home/.../program/boost-1.76
```

Now an environment variable `BOOSTROOT` should be added. This will ensure that this `boost` installation can be found by `cmake` for compiling `StackBlock` and `pyblock`. For example, one can add the following line into `~/.bashrc`.

```
export BOOSTROOT=~/program/boost-1.76
```

Note: The `--prefix` parameter cannot be set to a path beginning with `~`. If you need such a path, please use an absolute path instead, namely, setting `--prefix=/home/<user>/...`.

3.9.2 StackBlock Installation

The default vresion of `StackBlock` has some bugs for some non-popular features. It is recommended to use this fork, which can be compiled using `cmake`. First, make sure a working MPI library such as `openmpi 4.0` can be found in the system, a C++ compiler with the correct version can be found, and `BOOSTROOT` is set. Then `StackBlock` can be compiled in the following way (starting from the cloned `StackBlock` repo directory):

```
export STACKBLOCK=$PWD
mkdir build
cd build
cmake ..
make -j 10
rm CMakeCache.txt
cmake .. -DBUILD_OH=ON
make -j 10
rm CMakeCache.txt
cmake .. -DBUILD_READROT=ON
make -j 10
rm CMakeCache.txt
cmake .. -DBUILD_GAOPT=ON
make -j 10
```

This will generate four executables in the build direcotry.

`block.spin_adapted` is the main `StackBlock` program. One can optionally add the following to `~/.bashrc`:

```
export PATH=${STACKBLOCK}/build:$PATH
```

`OH` is the program to compute the expectation value on an MPS (or between two MPSs), of Hamiltonian and/or the identity operator.

`read_rot` is the program to translate the intermediate rotation matrix format (from the spin-projected `zmpo_dmrg` code) to the `StackBlock` rotation matrix and wavefunction format.

`gaopt` is the program for orbital reordering using genetic algorithm. Note that for this purpose, the `block2` driver `${BLOCK2HOME}/pyblock2/driver/gaopt.py` should provide much better performance.

3.9.3 pyblock Installation

`pyblock` is a python3 wrapper for the StackBlock code. `pyblock` contains a slightly revised version of StackBlock, which must be compiled. The code can be obtained here. First, make sure that a C++ compiler with the correct version can be found, and `BOOSTROOT` is set. Then `pyblock` can be compiled in the following way (starting from the cloned `pyblock` repo directory):

```
export PYBLOCKHOME=$PWD
mkdir build
cd build
cmake .. -DBUILD_LIB=ON -DUSE_MKL=ON
make -j 10
```

3.9.4 Import MPS to block2

Now we are ready to show how to translate a `StackBlock` MPS to `block2` MPS.

First, make sure a testing integral file `C2.CAS.PVDZ.FCIDUMP` is in the working directory. The integral file can be found in `${BLOCK2HOME}/data/C2.CAS.PVDZ.FCIDUMP`.

Note: Normally, orbital reordering can create some unnecessary complexities. It is recommended to use a already reordered FCIDUMP file across different codes. If the MPS has to be adjusted for orbital reordering, see *MPS Orbital Rotation*.

We will first perform a DMRG ground-state calculation using the following input file `dmrg.conf`:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
```

(continues on next page)

(continued from previous page)

```
maxiter 30
prefix ./tmp
noreorder
```

The following command can be used to run StackBlock with this input file:

```
mkdir ./tmp
${STACKBLOCK}/build/block.spin_adapted dmrg.conf > dmrg.out
```

The DMRG ground-state energy can be obtained from the output file:

```
$ grep 'Sweep Energy' dmrg.out | tail -1
M = 500     state = 0     Largest Discarded Weight = 0.000000000000  Sweep Energy = -
↪75.728442606745
```

The energy for the MPS that will be translated is the energy at the last site of the last sweep:

```
$ grep 'sweep energy' dmrg.out | tail -1
Finished Sweep with 500 states and sweep energy for State [ 0 ] with Spin [ 0 ] :: -
↪75.728442606745
```

Since in the default schedule the one-site algorithm is used for the last sweep. This two energies are identical.

Now the MPS in StackBlock format is stored in the scratch folder `./tmp/node0`. We will only need files in this folder with file names `Rotation-*`, `StateInfo-*`, `wave-*`. The other files `Block-b-*` and `Block-f-*` (with renormalized operators stored) are not part of the MPS, which can be deleted.

The folowing commands can be used to translate the MPS. Please make sure that the environment variables `${STACKBLOCK}`, `${PYBLOCKHOME}`, and `${BLOCK2HOME}` are correctly set.

```
$ PYTHONPATH=${BLOCK2HOME}/build:$PYTHONPATH
$ PYTHONPATH=${PYBLOCKHOME}:$PYTHONPATH
$ PYTHONPATH=${PYBLOCKHOME}/build:$PYTHONPATH
$ READWFN=${BLOCK2HOME}/pyblock2/driver/readwfn.py
$ python3 $READWFN dmrg.conf -expect
-75.72844260674495
```

Note: Here we use a special build of `block2` python extension, which was built using the `cmake` option `-DTBB=OFF` (the default is `OFF`). On some systems `-DUSE_MKL=OFF -OMP_LIB=SEQ` may be required. This is to solve the conflicts for importing `pyblock` and `block2` in the same script.

Note that `-expect` option is optional. With this option, the energy of the translated MPS will be evaluted in `block2` and printed. We can see that the printed `block2` energy is almost exactly the same as the one obtained from StackBlock. By default, the translated `block2` MPS will be put in the output directory named `./out` with the tag `KET`.

3.9.5 Export MPS from block2

Now we show how to translate a block2 MPS to StackBlock MPS.

We will first perform a DMRG ground-state calculation using the following input file dmrg2.conf:

```
sym d2h
orbitals C2.CAS.PVDZ.FCIDUMP

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30
prefix ./tmp2
noreorder
```

Note that the only difference between dmrg.conf and dmrg2.conf is the prefix. The following command can be used to run block2 with this input file:

```
${BLOCK2HOME}/pyblock2/driver/block2main dmrg2.conf > dmrg2.out
```

The energy for the MPS that will be translated is the energy at the last site of the last sweep:

```
$ grep 'DW' dmrg2.out | tail -1
Time elapsed =      3.883 | E =     -75.7284436933 | DE = -3.85e-07 | DW = 3.76e-16
```

The folowing commands can be used to translate the MPS. Please make sure that the environment variables ${STACKBLOCK}, ${PYBLOCKHOME}, and ${BLOCK2HOME} are correctly set.

```
$ PYTHONPATH=${BLOCK2HOME}/build:$PYTHONPATH
$ PYTHONPATH=${PYBLOCKHOME}:$PYTHONPATH
$ PYTHONPATH=${PYBLOCKHOME}/build:$PYTHONPATH
$ WRITEWFN=${BLOCK2HOME}/pyblock2/driver/writewfn.py
$ python3 $WRITEWFN dmrg2.conf -out out2
load MPSInfo from ./../tmp2/KET-mps_info.bin
SRRRRRRRRRRRRRRRRRRRRRRRR -> LLLLLLLLLLLLLLLLLLLLLLLR 24
```

From the print we can see that the canonical form of MPS has been changed, which may cause some small error in the translated MPS. The translated MPS in StackBlock format is now stored in the out2 directory. We can now evaluate the energy of the translated MPS using the OH program in StackBlock:

```
$ sed -i "s|^prefix.*|prefix ./out2|" dmrg2.conf
$ ${STACKBLOCK}/build/OH dmrg2.conf | grep -A 1 'printing hamiltonian' | tail -1
-75.7284436933
```

We can see that the printed StackBlock energy is exactly the same as the one obtained from block2.

Note: The OH program in StackBlock can only evalute the onedot MPS (namely, MPS used in 1-site DMRG algorithm). The MPS can be spin-adapted or non-spin-adapted. If you use the OH in the default standard version of StackBlock, the non-spin-adapted MPS is not supported and you need an extra argument for a file including the MPS ids. For example, you should use /path/to/default/StackBlock/OH dmrg2.conf wavenum where a file named wavenum should be set with contents 0 (or any space-separated list of integers, if you have multiple MPSs).

Alternatively, we can also translate back to block2 and evaluate the energy:

```
$ sed -i "s|^prefix.*|prefix ./out2|" dmrg2.conf
$ READWFN=${BLOCK2HOME}/pyblock2/driver/readwfn.py
$ python3 $READWFN dmrg2.conf -dot 1 -expect -out out3
-75.72844369332921
```

Which also prints the same energy.

3.10 References

If you find **block2** useful in your research, please cite the paper

- Zhai, H.; Larsson, H. R.; Lee, S.; Cui, Z.; Zhu, T.; Sun, C.; Peng, L.; Peng, R.; Liao, K.; Tölle, J.; Yang, J.; Li, S.; Chan, G. K.-L. Block2: a comprehensive open source framework to develop and apply state-of-the-art DMRG algorithms in electronic structure and beyond. *The Journal of Chemical Physics* 2023, **159**, 234801. doi: 10.1063/5.0180424

A detailed description of the parallel DMRG algorithm implemented in **block2** can be found in the following paper

- Zhai, H.; Chan, G. K.-L. Low communication high performance ab initio density matrix renormalization group algorithms. *The Journal of Chemical Physics* 2021, **154**, 224116. doi: 10.1063/5.0050902

For large site DMRG-MRCI/MRPT, please cite

- Larsson, H. R.; Zhai, H.; Gunst, K.; Chan, G. K.-L. Matrix product states with large sites. *Journal of Chemical Theory and Computation* 2022, **18**, 749-762. doi: 10.1021/acs.jctc.1c00957

For DMRG with spin-orbit-coupling, please cite

- Zhai, H.; Chan, G. K.-L. A comparison between the one- and two-step spin-orbit coupling approaches based on the ab initio Density Matrix Renormalization Group. *The Journal of Chemical Physics* 2022, **157**, 164108. doi: 10.1063/5.0107805

You can find a bibtex file in CITATIONS.bib.

The other algorithms implemented in **block2** are based on the following papers.

3.10.1 Qauntum Chemisty DMRG

- Chan, G. K.-L.; Head-Gordon, M. Highly correlated calculations with a polynomial cost algorithm: A study of the density matrix renormalization group. *The Journal of Chemical Physics* 2002, **116**, 4462–4476. doi: 10.1063/1.1449459
- Sharma, S.; Chan, G. K.-L. Spin-adapted density matrix renormalization group algorithms for quantum chemistry. *The Journal of Chemical Physics* 2012, **136**, 124121. doi: 10.1063/1.3695642
- Wouters, S.; Van Neck, D. The density matrix renormalization group for ab initio quantum chemistry. *The European Physical Journal D* 2014, **68**, 272. doi: 10.1140/epjd/e2014-50500-1

3.10.2 Parallelization

- Chan, G. K.-L. An algorithm for large scale density matrix renormalization group calculations. *The Journal of Chemical Physics* 2004, **120**, 3172–3178. doi: 10.1063/1.1638734
- Chan, G. K.-L.; Keselman, A.; Nakatani, N.; Li, Z.; White, S. R. Matrix product operators, matrix product states, and ab initio density matrix renormalization group algorithms. *The Journal of Chemical Physics* 2016, **145**, 014102. doi: 10.1063/1.4955108
- Stoudenmire, E.; White, S. R. Real-space parallel density matrix renormalization group. *Physical Review B* 2013, **87**, 155137. doi: 10.1103/PhysRevB.87.155137
- Zhai, H., Chan, G. K. L. Low communication high performance ab initio density matrix renormalization group algorithms. *The Journal of Chemical Physics* 2021, **154**, 224116. doi: 10.1063/5.0050902

3.10.3 Spin-Orbit Coupling

- Sayfutyarova, E. R., Chan, G. K. L. A state interaction spin-orbit coupling density matrix renormalization group method. *The Journal of Chemical Physics* 2016, **144**, 234301. doi: 10.1063/1.4953445
- Sayfutyarova, E. R., Chan, G. K. L. Electron paramagnetic resonance g-tensors from state interaction spin-orbit coupling density matrix renormalization group. *The Journal of Chemical Physics* 2018, **148**, 184103. doi: 10.1063/1.5020079
- Zhai, H., Chan, G. K. A comparison between the one- and two-step spin-orbit coupling approaches based on the ab initio Density Matrix Renormalization Group. *The Journal of Chemical Physics* 2022, **157**, 164108. doi: 10.1063/5.0107805

3.10.4 Green's Function

- Ronca, E., Li, Z., Jimenez-Hoyos, C. A., Chan, G. K. L. Time-step targeting time-dependent and dynamical density matrix renormalization group algorithms with ab initio Hamiltonians. *Journal of Chemical Theory and Computation* 2017, **13**, 5560-5571. doi: 10.1021/acs.jctc.7b00682

3.10.5 Finite-Temperature DMRG

- Feiguin, A. E., White, S. R. Finite-temperature density matrix renormalization using an enlarged Hilbert space. *Physical Review B* 2005, **72**, 220401. doi: 10.1103/PhysRevB.72.220401

3.10.6 Time-Dependent DMRG

- Feiguin, A. E., White, S. R. Time-step targeting methods for real-time dynamics using the density matrix renormalization group. *Physical Review B* 2005, **72**, 020404. doi: 10.1103/PhysRevB.72.020404
- Haegeman, J., Lubich, C., Oseledets, I., Vandereycken, B., Verstraete, F. Unifying time evolution and optimization with matrix product states. *Physical Review B* 2016, **94**, 165116. doi: 10.1103/PhysRevB.94.165116

3.10.7 Linear Response

- Sharma, S., Chan, G. K. Communication: A flexible multi-reference perturbation theory by minimizing the Hylleraas functional with matrix product states. *Journal of Chemical Physics* 2014, **141**, 111101. doi: 10.1063/1.4895977

3.10.8 Perturbative Noise

- White, S. R. Density matrix renormalization group algorithms with a single center site. *Physical Review B* 2005, **72**, 180403. doi: 10.1103/PhysRevB.72.180403
- Hubig, C., McCulloch, I. P., Schollwöck, U., Wolf, F. A. Strictly single-site DMRG algorithm with subspace expansion. *Physical Review B* 2015, **91**, 155115. doi: 10.1103/PhysRevB.91.155115

3.10.9 Particle Density Matrix

- Ghosh, D., Hachmann, J., Yanai, T., Chan, G. K. L. Orbital optimization in the density matrix renormalization group, with applications to polyenes and -carotene. *The Journal of Chemical Physics* 2008, **128**, 144117. doi: 10.1063/1.2883976
- Guo, S., Watson, M. A., Hu, W., Sun, Q., Chan, G. K. L. N-electron valence state perturbation theory based on a density matrix renormalization group reference function, with applications to the chromium dimer and a trimer model of poly (p-phenylenevinylene). *Journal of Chemical Theory and Computation* 2016, **12**, 1583-1591. doi: 10.1021/acs.jctc.5b01225

3.10.10 DMRG-SC-NEVPT2

- Roemelt, M., Guo, S., Chan, G. K. L. A projected approximation to strongly contracted N-electron valence perturbation theory for DMRG wavefunctions. *The Journal of Chemical Physics* 2016, **144**, 204113. doi: 10.1063/1.4950757
- Sokolov, A. Y., Guo, S., Ronca, E., Chan, G. K. L. Time-dependent N-electron valence perturbation theory with matrix product state reference wavefunctions for large active spaces and basis sets: Applications to the chromium dimer and all-trans polyenes. *The Journal of Chemical Physics* 2017, **146**, 244102. doi: 10.1063/1.4986975

3.10.11 DMRG-CASPT2

- Kurashige, Y., Yanai, T. Second-order perturbation theory with a density matrix renormalization group self-consistent field reference function: Theory and application to the study of chromium dimer. *The Journal of Chemical Physics* 2011, **135**, 094104. doi: 10.1063/1.3629454
- Wouters, S., Van Speybroeck, V., Van Neck, D. DMRG-CASPT2 study of the longitudinal static second hyperpolarizability of all-trans polyenes. *The Journal of Chemical Physics* 2016, **145**, 054120. doi: 10.1063/1.4959817
- Nakatani, N., Guo, S. Density matrix renormalization group (DMRG) method as a common tool for large active-space CASSCF/CASPT2 calculations. *The Journal of Chemical Physics* 2017, **146**, 094102. doi: 10.1063/1.4976644

3.10.12 Multi-Reference Correlation Theories

- Szalay, P. G.; Müller, T.; Gidofalvi, G.; Lischka, H.; Shepard, R. Multiconfiguration Self-Consistent Field and Multireference Configuration Interaction Methods and Applications. *Chemical Reviews* 2012, **112**, 108-181. doi: 10.1021/cr200137a
- Gdanitz, R. J., Ahlrichs, R. The Averaged Coupled-Pair Functional (ACPF): A Size-Extensive Modification of MR CI(SD). *Chemical Physics Letters* 1988, **143**, 413-420. doi: 10.1016/0009-2614(88)87388-3

- Szalay, P. G., Bartlett, R. J. Multi-Reference Averaged Quadratic Coupled-Cluster Method: A Size-Extensive Modification of Multi-Reference CI. *Chemical Physics Letters* 1993, **214**, 481-488. doi: 10.1016/0009-2614(93)85670-J
- Laidig, W. D.; Bartlett, R. J. A Multi-Reference Coupled-Cluster Method for Molecular Applications. *Chemical Physics Letters* 1984, **104**, 424-430. doi: 10.1016/0009-2614(84)85617-1
- Laidig, W. D., Saxe, P., Bartlett, R. J. The Description of N 2 and F 2 Potential Energy Surfaces Using Multireference Coupled Cluster Theory. *The Journal of Chemical Physics* 1987, **86**, 887-907. doi: 10.1063/1.452291
- Angeli, C., Cimiraglia, R., Evangelisti, S., Leininger, T., Malrieu, J.-P. Introduction of N-Electron Valence States for Multireference Perturbation Theory. *J. Chem. Phys.* 2001, **114**, 10252–10264. doi: 10.1063/1.1361246
- Angeli, C., Cimiraglia, R., Malrieu J.-P. N-electron valence state perturbation theory: A spinless formulation and an efficient implementation of the strongly contracted and of the partially contracted variants. *The Journal of chemical physics* 2002, **117**, 9138-9153. doi: 10.1063/1.1515317
- Angeli, C., Pastore, M., Cimiraglia, R. New Perspectives in Multireference Perturbation Theory: The n-Electron Valence State Approach. *Theor Chem Account* 2007, **117**, 743–754. doi: 10.1007/s00214-006-0207-0
- Fink, R. F. The Multi-Reference Retaining the Excitation Degree Perturbation Theory: A Size-Consistent, Unitary Invariant, and Rapidly Convergent Wavefunction Based Ab Initio Approach. *Chemical Physics* 2009, **356**, 39-46. doi: 10.1016/j.chemphys.2008.10.004
- Fink, R. F. Two New Unitary-Invariant and Size-Consistent Perturbation Theoretical Approaches to the Electron Correlation Energy. *Chemical Physics Letters* 2006, **428**, 461–466. doi: 10.1016/j.cplett.2006.07.081
- Sharma, S., Chan, G. K.-L. Communication: A Flexible Multi-Reference Perturbation Theory by Minimizing the Hylleraas Functional with Matrix Product States. *The Journal of Chemical Physics* 2014, **141**, 111101. doi: 10.1063/1.4895977
- Sharma, S., Alavi, A. Multireference Linearized Coupled Cluster Theory for Strongly Correlated Systems Using Matrix Product States. *The Journal of Chemical Physics* 2015, **143**, 102815. doi: 10.1063/1.4928643
- Sharma, S., Jeanmairet, G., Alavi, A. Quasi-Degenerate Perturbation Theory Using Matrix Product States. *The Journal of Chemical Physics* 2016, **144**, 034103. doi: 10.1063/1.4939752
- Larsson, H. R., Zhai, H., Gunst, K., Chan, G. K. L. Matrix product states with large sites. *Journal of Chemical Theory and Computation* 2022, **18**, 749-762. doi: 10.1021/acs.jctc.1c00957

3.10.13 Determinant Coefficients

- Lee, S., Zhai, H., Sharma, S., Umrigar, C. J., Chan, G. K. Externally Corrected CCSD with Renormalized Perturbative Triples (R-ecCCSD (T)) and the Density Matrix Renormalization Group and Selected Configuration Interaction External Sources. *Journal of Chemical Theory and Computation* 2021, **17**, 3414-3425. doi: 10.1021/acs.jctc.1c00205

3.10.14 Perturbative DMRG

- Guo, S., Li, Z., Chan, G. K. L. Communication: An efficient stochastic algorithm for the perturbative density matrix renormalization group in large active spaces. *The Journal of chemical physics* 2018, **148**, 221104. doi: 10.1063/1.5031140
- Guo, S., Li, Z., Chan, G. K. L. A perturbative density matrix renormalization group algorithm for large active spaces. *Journal of chemical theory and computation* 2018, **14**, 4063-4071. doi: 10.1021/acs.jctc.8b00273

3.10.15 Orbital Reordering

- Olivares-Amaya, R.; Hu, W.; Nakatani, N.; Sharma, S.; Yang, J.;Chan, G. K.-L. The ab-initio density matrix renormalization group in practice. *The Journal of Chemical Physics* 2015, **142**, 034102. doi: 10.1063/1.4905329

CHAPTER

FOUR

PYTHON INTERFACE TUTORIAL

4.1 Quantum Chemistry Hamiltonians

[1]:
```
!pip install block2==0.5.3rc5 -qq --progress-bar off --extra-index-url=https://block-
↪hczhai.github.io/block2-preview/pypi/
!pip install pyscf==2.3.0 -qq --progress-bar off
```

4.1.1 Introduction

In this tutorial we explain how to perform quantum chemistry DMRG using the python interface of `block2`.

The quantum chemistry Hamiltonian in its second quantized form has to be defined in a set of orbitals, such as the Hartree-Fock (or Density Functional Theory) orbitals. The symmetries that can be used in the DMRG calculation thus have a dependence on the symmetry of the Hartree-Fock orbitals.

1. For spin-restricted Hartree-Fock (RHF) orbitals, we can perform spin-adapted DMRG (`SU2` mode in `block2`) or non-spin-adapted DMRG with any lower symmetries (`SZ` or `SGF`).
2. For spin-unrestricted Hartree-Fock (UHF) orbitals, we can perform non-spin-adapted DMRG (`SZ` mode in `block2`) or DMRG with lower symmetries (such as `SGF`).
3. For general Hartree-Fock (GHF) orbitals, we can perform DMRG in spin-orbitals (`SGF` mode in `block2`) or first translate the Hamiltonian into the qubit Hamiltonian then do DMRG (`SGB` mode in `block2`).
4. For relativistic Dirac Hartree-Fock (DHF) orbitals, we can perform DMRG in complex spin-orbitals (`SGFCPX` mode in `block2`).
5. For atom and diatomic molecules, we can perform spin-adapted/non-spin-adapted/spin-orbital DMRG (`SAnySU2LZ/SAnySZLZ/SAnySGFLZ` modes in `block2`) with the L_z symmetry.

Next, we will explain how to set up the integrals and perform DMRG in each of the modes (1) (2) (3) (4) and (5). The quantum chemistry integrals will be generated using `pyscf` and transformed using funtions defined in `pyblock2._pyscf.ao2mo`.

[2]:
```
import numpy as np
from pyblock2._pyscf.ao2mo import integrals as itg
from pyblock2.driver.core import DMRGDriver, SymmetryTypes

bond_dims = [250] * 4 + [500] * 4
noises = [1e-4] * 4 + [1e-5] * 4 + [0]
thrds = [1e-10] * 8
```

4.1.2 Spin-Restricted Integrals

Here we use `get_rhf_integrals` function to get the integrals. Note that in order to do DMRG in a CASCI space, one can set the `ncore` (number of core orbitals) and `ncas` (number of active orbitals) parameters in `get_*_integrals`. `ncas=None` will include all orbitals in DMRG.

For medium to large scale DMRG calculations, it is highly recommended to use a scratch space with high IO speed rather than the `./tmp` used in the following example. One also needs to set a suitable `stack_mem` in the `DMRGDriver` constructor to set the memory used for storing renormalized operators (in bytes). The default is `stack_mem=int(1024**3)` (1 GB). For medium scale calculations 10 to 30 GB might be required.

For the meaning of DMRG parameters, please have a look at the Hubbard - Run DMRG page.

[3]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.RHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SU2, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=1)
ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

pdm1 = driver.get_1pdm(ket)
pdm2 = driver.get_2pdm(ket).transpose(0, 3, 1, 2)
print('Energy from pdms = %20.15f' % (np.einsum('ij,ij->', pdm1, h1e)
    + 0.5 * np.einsum('ijkl,ijkl->', pdm2, driver.unpack_g2e(g2e)) + ecore))

impo = driver.get_identity_mpo()
expt = driver.expectation(ket, mpo, ket) / driver.expectation(ket, impo, ket)
print('Energy from expectation = %20.15f' % expt)
```

```
integral symmetrize error =  5.5986859646912075e-14
integral cutoff error =  0.0
mpo terms =       1030

Build MPO | Nsites =    10 | Nterms =       1030 | Algorithm = FastBIP | Cutoff = 1.
↪00e-20
 Site =     0 /    10 .. Mmpo =    13 DW = 0.00e+00 NNZ =       13 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.008
 Site =     1 /    10 .. Mmpo =    34 DW = 0.00e+00 NNZ =       63 SPT = 0.8575 Tmvc␣
↪= 0.000 T = 0.004
 Site =     2 /    10 .. Mmpo =    56 DW = 0.00e+00 NNZ =      121 SPT = 0.9364 Tmvc␣
↪= 0.001 T = 0.004
 Site =     3 /    10 .. Mmpo =    74 DW = 0.00e+00 NNZ =      373 SPT = 0.9100 Tmvc␣
↪= 0.000 T = 0.004
 Site =     4 /    10 .. Mmpo =    80 DW = 0.00e+00 NNZ =      269 SPT = 0.9546 Tmvc␣
↪= 0.000 T = 0.004
 Site =     5 /    10 .. Mmpo =    94 DW = 0.00e+00 NNZ =      169 SPT = 0.9775 Tmvc␣
↪= 0.000 T = 0.004
 Site =     6 /    10 .. Mmpo =    54 DW = 0.00e+00 NNZ =      181 SPT = 0.9643 Tmvc␣
↪= 0.000 T = 0.003
 Site =     7 /    10 .. Mmpo =    30 DW = 0.00e+00 NNZ =       73 SPT = 0.9549 Tmvc␣
↪= 0.000 T = 0.014
 Site =     8 /    10 .. Mmpo =    14 DW = 0.00e+00 NNZ =       41 SPT = 0.9024 Tmvc␣
↪= 0.000 T = 0.005
 Site =     9 /    10 .. Mmpo =     1 DW = 0.00e+00 NNZ =       14 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.005
Ttotal =      0.055 Tmvc-total = 0.003 MPO bond dimension =    94 MaxDW = 0.00e+00
NNZ =         1317 SIZE =        27073 SPT = 0.9514

Rank =     0 Ttotal =      0.125 MPO method = FastBipartite bond dimension =      94␣
↪NNZ =         1317 SIZE =        27073 SPT = 0.9514

Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.856 | E =    -107.6541224475 | DW = 1.87e-10

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.252 | E =    -107.6541224475 | DE = -5.37e-12 | DW = 4.75e-20

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.706 | E =    -107.6541224475 | DE = 2.84e-14 | DW = 1.87e-10

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
```

(continues on next page)

(continued from previous page)

```
Time elapsed =      2.132 | E =    -107.6541224475 | DE = -9.09e-13 | DW = 5.74e-20

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.609 | E =    -107.6541224475 | DE = 2.84e-14 | DW = 1.65e-20

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.111 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 5.79e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.623 | E =    -107.6541224475 | DE = -5.68e-14 | DW = 2.13e-20

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.076 | E =    -107.6541224475 | DE = 2.84e-14 | DW = 6.59e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      4.452 | E =    -107.6541224475 | DE = 2.84e-14 | DW = 2.76e-20

DMRG energy = -107.654122447524415
Energy from pdms = -107.654122447524443
Energy from expectation = -107.654122447524287
```

We can also run non-spin-adapted DMRG (SZ mode) using the restricted integrals.

[4]:
```
driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SZ, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=1)
ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)
```

```
integral symmetrize error =  7.225136772727681e-14
integral cutoff error =  0.0
mpo terms =       2778

Build MPO | Nsites =    10 | Nterms =       2778 | Algorithm = FastBIP | Cutoff = 1.
↪00e-20
 Site =     0 /    10 .. Mmpo =    26 DW = 0.00e+00 NNZ =       26 SPT = 0.0000 Tmvc␣
↪= 0.001 T = 0.034
 Site =     1 /    10 .. Mmpo =    66 DW = 0.00e+00 NNZ =      143 SPT = 0.9167 Tmvc␣
```

(continues on next page)

(continued from previous page)

```
↪= 0.001 T = 0.034
 Site =     2 /    10 .. Mmpo =   110 DW = 0.00e+00 NNZ =      283 SPT = 0.9610 Tmvc␣
↪= 0.011 T = 0.042
 Site =     3 /    10 .. Mmpo =   138 DW = 0.00e+00 NNZ =     1023 SPT = 0.9326 Tmvc␣
↪= 0.004 T = 0.018
 Site =     4 /    10 .. Mmpo =   158 DW = 0.00e+00 NNZ =      535 SPT = 0.9755 Tmvc␣
↪= 0.001 T = 0.046
 Site =     5 /    10 .. Mmpo =   186 DW = 0.00e+00 NNZ =      463 SPT = 0.9842 Tmvc␣
↪= 0.001 T = 0.015
 Site =     6 /    10 .. Mmpo =   106 DW = 0.00e+00 NNZ =      415 SPT = 0.9790 Tmvc␣
↪= 0.001 T = 0.021
 Site =     7 /    10 .. Mmpo =    58 DW = 0.00e+00 NNZ =      163 SPT = 0.9735 Tmvc␣
↪= 0.000 T = 0.015
 Site =     8 /    10 .. Mmpo =    26 DW = 0.00e+00 NNZ =       87 SPT = 0.9423 Tmvc␣
↪= 0.000 T = 0.022
 Site =     9 /    10 .. Mmpo =     1 DW = 0.00e+00 NNZ =       26 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.017
Ttotal =      0.263 Tmvc-total = 0.021 MPO bond dimension =   186 MaxDW = 0.00e+00
NNZ =         3164 SIZE =       102772 SPT = 0.9692

Rank =     0 Ttotal =      0.504 MPO method = FastBipartite bond dimension =     186␣
↪NNZ =         3164 SIZE =       102772 SPT = 0.9692

Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.658 | E =    -107.6541224475 | DW = 4.14e-08

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.407 | E =    -107.6541224475 | DE = -1.32e-11 | DW = 5.09e-09

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.150 | E =    -107.6541224475 | DE = -1.31e-12 | DW = 4.14e-08

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.998 | E =    -107.6541224475 | DE = 1.42e-12 | DW = 5.21e-09

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.947 | E =    -107.6541224475 | DE = -1.19e-12 | DW = 3.60e-11

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
```

(continues on next page)

(continued from previous page)

```
Time elapsed =      5.944 | E =    -107.6541224475 | DE = 9.09e-13 | DW = 1.83e-19

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      6.888 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 3.60e-11

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      7.889 | E =    -107.6541224475 | DE = -1.25e-12 | DW = 1.43e-19

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      8.448 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 5.39e-20

DMRG energy = -107.654122447524529
```

We can also run DMRG in spin orbitals (SGF mode) using the restricted integrals, which will be much slower (for more realistic systems).

[5]:
```
driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SGF, n_threads=4)

driver.n_sites = ncas
g2e = driver.unpack_g2e(g2e)
orb_sym = [orb_sym[i // 2] for i in range(len(orb_sym) * 2)]
n_sites = ncas * 2

driver.initialize_system(n_sites=n_sites, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=1)
ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)
```

```
integral symmetrize error =  7.225136772727683e-14
integral cutoff error =  0.0
mpo terms =       2438

Build MPO | Nsites =    20 | Nterms =       2438 | Algorithm = FastBIP | Cutoff = 1.
↪00e-20
 Site =     0 /    20 .. Mmpo =     7 DW = 0.00e+00 NNZ =        7 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.005
 Site =     1 /    20 .. Mmpo =    20 DW = 0.00e+00 NNZ =       19 SPT = 0.8643 Tmvc␣
↪= 0.001 T = 0.010
 Site =     2 /    20 .. Mmpo =    45 DW = 0.00e+00 NNZ =       45 SPT = 0.9500 Tmvc␣
↪= 0.001 T = 0.012
```

(continues on next page)

(continued from previous page)

```
 Site =     3 /    20 .. Mmpo =    62 DW = 0.00e+00 NNZ =      131 SPT = 0.9530 Tmvc␣
↪= 0.001 T = 0.020
 Site =     4 /    20 .. Mmpo =    81 DW = 0.00e+00 NNZ =      159 SPT = 0.9683 Tmvc␣
↪= 0.002 T = 0.008
 Site =     5 /    20 .. Mmpo =   104 DW = 0.00e+00 NNZ =      203 SPT = 0.9759 Tmvc␣
↪= 0.001 T = 0.009
 Site =     6 /    20 .. Mmpo =   125 DW = 0.00e+00 NNZ =      265 SPT = 0.9796 Tmvc␣
↪= 0.002 T = 0.019
 Site =     7 /    20 .. Mmpo =   126 DW = 0.00e+00 NNZ =      974 SPT = 0.9382 Tmvc␣
↪= 0.001 T = 0.029
 Site =     8 /    20 .. Mmpo =   151 DW = 0.00e+00 NNZ =      188 SPT = 0.9901 Tmvc␣
↪= 0.001 T = 0.011
 Site =     9 /    20 .. Mmpo =   148 DW = 0.00e+00 NNZ =      344 SPT = 0.9846 Tmvc␣
↪= 0.001 T = 0.015
 Site =    10 /    20 .. Mmpo =   177 DW = 0.00e+00 NNZ =      246 SPT = 0.9906 Tmvc␣
↪= 0.001 T = 0.010
 Site =    11 /    20 .. Mmpo =   178 DW = 0.00e+00 NNZ =      402 SPT = 0.9872 Tmvc␣
↪= 0.001 T = 0.009
 Site =    12 /    20 .. Mmpo =   147 DW = 0.00e+00 NNZ =      306 SPT = 0.9883 Tmvc␣
↪= 0.000 T = 0.012
 Site =    13 /    20 .. Mmpo =   100 DW = 0.00e+00 NNZ =      242 SPT = 0.9835 Tmvc␣
↪= 0.000 T = 0.006
 Site =    14 /    20 .. Mmpo =    77 DW = 0.00e+00 NNZ =      150 SPT = 0.9805 Tmvc␣
↪= 0.000 T = 0.006
 Site =    15 /    20 .. Mmpo =    54 DW = 0.00e+00 NNZ =       94 SPT = 0.9774 Tmvc␣
↪= 0.000 T = 0.010
 Site =    16 /    20 .. Mmpo =    39 DW = 0.00e+00 NNZ =       82 SPT = 0.9611 Tmvc␣
↪= 0.000 T = 0.003
 Site =    17 /    20 .. Mmpo =    20 DW = 0.00e+00 NNZ =       50 SPT = 0.9359 Tmvc␣
↪= 0.000 T = 0.003
 Site =    18 /    20 .. Mmpo =     7 DW = 0.00e+00 NNZ =       20 SPT = 0.8571 Tmvc␣
↪= 0.000 T = 0.003
 Site =    19 /    20 .. Mmpo =     1 DW = 0.00e+00 NNZ =        7 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.004
Ttotal =      0.204 Tmvc-total = 0.014 MPO bond dimension =   178 MaxDW = 0.00e+00
NNZ =         3934 SIZE =       200866 SPT = 0.9804

Rank =     0 Ttotal =      0.341 MPO method = FastBipartite bond dimension =     178␣
↪NNZ =         3934 SIZE =       200866 SPT = 0.9804

Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.578 | E =    -107.6541216205 | DW = 7.62e-08

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
```

(continues on next page)

(continued from previous page)

```
↪Dav threshold =  1.00e-10
Time elapsed =      2.716 | E =    -107.6541223420 | DE = -7.21e-07 | DW = 6.45e-08

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.877 | E =    -107.6541224347 | DE = -9.27e-08 | DW = 7.64e-08

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.516 | E =    -107.6541224347 | DE = 9.38e-13 | DW = 6.15e-08

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      5.397 | E =    -107.6541224379 | DE = -3.17e-09 | DW = 6.46e-11

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      6.224 | E =    -107.6541224379 | DE = -3.38e-11 | DW = 7.42e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      7.123 | E =    -107.6541224379 | DE = 0.00e+00 | DW = 6.46e-11

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      7.919 | E =    -107.6541224379 | DE = 6.82e-13 | DW = 7.25e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      8.532 | E =    -107.6541224379 | DE = 0.00e+00 | DW = 6.05e-20

DMRG energy = -107.654122437941069
```

4.1.3 Read and Write FCIDUMP Files

Instead of generating integrals (`h1e` and `g2e`) using `pyscf`, we can also read these integrals from a FCIDUMP file (which can be generated using any of many other quantum chemistry packages) then perform DMRG. Additionally, we also provide methods to write the FCIDUMP file using the data in the `h1e` and `g2e` arrays.

After invoking `driver.read_fcidump`, the integrals and target state infomation can be obtained from `driver.h1e`, `driver.g2e`, `driver.n_sites`, etc.

[6]:
```
from pyscf import gto, scf
```

(continues on next page)

(continued from previous page)

```
mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.RHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SU2, n_threads=4)

# write integrals to file
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)
driver.write_fcidump(h1e, g2e, ecore, filename='N2.STO3G.FCIDUMP', h1e_symm=True, pg=
↪'d2h')

# read integrals from file
driver.read_fcidump(filename='N2.STO3G.FCIDUMP', pg='d2h')
driver.initialize_system(n_sites=driver.n_sites, n_elec=driver.n_elec,
                         spin=driver.spin, orb_sym=driver.orb_sym)

mpo = driver.get_qc_mpo(h1e=driver.h1e, g2e=driver.g2e, ecore=driver.ecore, iprint=1)
ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)
```

```
symmetrize error =  2.4200000000000016e-14
integral symmetrize error =  0.0
integral cutoff error =  0.0
mpo terms =       1030

Build MPO | Nsites =    10 | Nterms =       1030 | Algorithm = FastBIP | Cutoff = 1.
↪00e-20
 Site =     0 /    10 .. Mmpo =    13 DW = 0.00e+00 NNZ =       13 SPT = 0.0000 Tmvc␣
↪= 0.001 T = 0.015
 Site =     1 /    10 .. Mmpo =    34 DW = 0.00e+00 NNZ =       63 SPT = 0.8575 Tmvc␣
↪= 0.002 T = 0.008
 Site =     2 /    10 .. Mmpo =    56 DW = 0.00e+00 NNZ =      121 SPT = 0.9364 Tmvc␣
↪= 0.001 T = 0.008
 Site =     3 /    10 .. Mmpo =    74 DW = 0.00e+00 NNZ =      373 SPT = 0.9100 Tmvc␣
↪= 0.001 T = 0.008
 Site =     4 /    10 .. Mmpo =    80 DW = 0.00e+00 NNZ =      269 SPT = 0.9546 Tmvc␣
↪= 0.004 T = 0.019
 Site =     5 /    10 .. Mmpo =    94 DW = 0.00e+00 NNZ =      169 SPT = 0.9775 Tmvc␣
↪= 0.000 T = 0.005
 Site =     6 /    10 .. Mmpo =    54 DW = 0.00e+00 NNZ =      181 SPT = 0.9643 Tmvc␣
↪= 0.000 T = 0.004
 Site =     7 /    10 .. Mmpo =    30 DW = 0.00e+00 NNZ =       73 SPT = 0.9549 Tmvc␣
↪= 0.000 T = 0.003
```

(continues on next page)

(continued from previous page)

```
 Site =     8 /    10 .. Mmpo =    14 DW = 0.00e+00 NNZ =       41 SPT = 0.9024 Tmvc␣
↪= 0.000 T = 0.007
 Site =     9 /    10 .. Mmpo =     1 DW = 0.00e+00 NNZ =       14 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.002
Ttotal =      0.079 Tmvc-total = 0.009 MPO bond dimension =    94 MaxDW = 0.00e+00
NNZ =         1317 SIZE =        27073 SPT = 0.9514

Rank =     0 Ttotal =      0.140 MPO method = FastBipartite bond dimension =      94␣
↪NNZ =         1317 SIZE =        27073 SPT = 0.9514

Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.513 | E =    -107.6541224475 | DW = 1.87e-10

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.765 | E =    -107.6541224475 | DE = -3.13e-12 | DW = 4.68e-20

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.010 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 1.87e-10

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.246 | E =    -107.6541224475 | DE = -5.97e-13 | DW = 6.02e-20

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.513 | E =    -107.6541224475 | DE = -5.68e-14 | DW = 1.68e-20

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.770 | E =    -107.6541224475 | DE = 8.53e-14 | DW = 4.69e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.045 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 2.14e-20

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.294 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 4.87e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      2.472 | E =    -107.6541224475 | DE = 2.84e-14 | DW = 4.03e-20
```

(continues on next page)

(continued from previous page)

```
DMRG energy = -107.654122447524557
```

4.1.4 The SZ Mode

Here we use `get_uhf_integrals` function to get the integrals.

[7]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.UHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_uhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SZ, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=1)
ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)
```

```
integral symmetrize error =  1.5510746027850855e-13
integral cutoff error =  0.0
mpo terms =       2778

Build MPO | Nsites =    10 | Nterms =       2778 | Algorithm = FastBIP | Cutoff = 1.
↪00e-20
 Site =     0 /    10 .. Mmpo =    26 DW = 0.00e+00 NNZ =       26 SPT = 0.0000 Tmvc␣
↪= 0.001 T = 0.014
 Site =     1 /    10 .. Mmpo =    66 DW = 0.00e+00 NNZ =      143 SPT = 0.9167 Tmvc␣
↪= 0.001 T = 0.015
 Site =     2 /    10 .. Mmpo =   110 DW = 0.00e+00 NNZ =      283 SPT = 0.9610 Tmvc␣
↪= 0.002 T = 0.013
 Site =     3 /    10 .. Mmpo =   138 DW = 0.00e+00 NNZ =     1023 SPT = 0.9326 Tmvc␣
↪= 0.001 T = 0.011
 Site =     4 /    10 .. Mmpo =   158 DW = 0.00e+00 NNZ =      535 SPT = 0.9755 Tmvc␣
↪= 0.001 T = 0.010
 Site =     5 /    10 .. Mmpo =   186 DW = 0.00e+00 NNZ =      463 SPT = 0.9842 Tmvc␣
↪= 0.001 T = 0.008
 Site =     6 /    10 .. Mmpo =   106 DW = 0.00e+00 NNZ =      415 SPT = 0.9790 Tmvc␣
↪= 0.000 T = 0.008
 Site =     7 /    10 .. Mmpo =    58 DW = 0.00e+00 NNZ =      163 SPT = 0.9735 Tmvc␣
↪= 0.000 T = 0.006
```

(continues on next page)

(continued from previous page)

```
 Site =     8 /    10 .. Mmpo =    26 DW = 0.00e+00 NNZ =       87 SPT = 0.9423 Tmvc␣
↪= 0.000 T = 0.005
 Site =     9 /    10 .. Mmpo =     1 DW = 0.00e+00 NNZ =       26 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.005
Ttotal =      0.095 Tmvc-total = 0.007 MPO bond dimension =   186 MaxDW = 0.00e+00
NNZ =         3164 SIZE =       102772 SPT = 0.9692

Rank =     0 Ttotal =      0.170 MPO method = FastBipartite bond dimension =     186␣
↪NNZ =         3164 SIZE =       102772 SPT = 0.9692

Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.401 | E =    -107.6541224475 | DW = 4.14e-08

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.190 | E =    -107.6541224475 | DE = -1.80e-11 | DW = 5.07e-09

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.846 | E =    -107.6541224475 | DE = -1.11e-12 | DW = 4.14e-08

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.333 | E =    -107.6541224475 | DE = 1.11e-12 | DW = 5.10e-09

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.881 | E =    -107.6541224475 | DE = -9.38e-13 | DW = 3.60e-11

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.435 | E =    -107.6541224475 | DE = 8.81e-13 | DW = 1.28e-19

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.997 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 3.60e-11

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      5.553 | E =    -107.6541224475 | DE = -1.34e-12 | DW = 1.64e-19

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      5.919 | E =    -107.6541224475 | DE = -5.68e-14 | DW = 9.83e-20
```

(continues on next page)

(continued from previous page)

```
DMRG energy = -107.654122447524585
```

4.1.5 The SGF Mode

Here we use `get_ghf_integrals` function to get the integrals.

[8]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.GHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_ghf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SGF, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=1)
ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)
```

```
integral symmetrize error =  2.2056297680002223e-13
integral cutoff error =  0.0
mpo terms =       5992

Build MPO | Nsites =    20 | Nterms =       5992 | Algorithm = FastBIP | Cutoff = 1.
↪00e-20
 Site =     0 /    20 .. Mmpo =     7 DW = 0.00e+00 NNZ =        7 SPT = 0.0000 Tmvc␣
↪= 0.001 T = 0.017
 Site =     1 /    20 .. Mmpo =    20 DW = 0.00e+00 NNZ =       19 SPT = 0.8643 Tmvc␣
↪= 0.001 T = 0.014
 Site =     2 /    20 .. Mmpo =    47 DW = 0.00e+00 NNZ =       49 SPT = 0.9479 Tmvc␣
↪= 0.002 T = 0.015
 Site =     3 /    20 .. Mmpo =    62 DW = 0.00e+00 NNZ =      251 SPT = 0.9139 Tmvc␣
↪= 0.002 T = 0.011
 Site =     4 /    20 .. Mmpo =    81 DW = 0.00e+00 NNZ =      273 SPT = 0.9456 Tmvc␣
↪= 0.003 T = 0.020
 Site =     5 /    20 .. Mmpo =   104 DW = 0.00e+00 NNZ =      357 SPT = 0.9576 Tmvc␣
↪= 0.001 T = 0.013
 Site =     6 /    20 .. Mmpo =   129 DW = 0.00e+00 NNZ =      563 SPT = 0.9580 Tmvc␣
↪= 0.002 T = 0.011
 Site =     7 /    20 .. Mmpo =   126 DW = 0.00e+00 NNZ =     2318 SPT = 0.8574 Tmvc␣
↪= 0.003 T = 0.021
```

(continues on next page)

(continued from previous page)

```
 Site =     8 /    20 .. Mmpo =   155 DW = 0.00e+00 NNZ =      212 SPT = 0.9891 Tmvc␣
↪= 0.001 T = 0.008
 Site =     9 /    20 .. Mmpo =   148 DW = 0.00e+00 NNZ =      702 SPT = 0.9694 Tmvc␣
↪= 0.001 T = 0.010
 Site =    10 /    20 .. Mmpo =   181 DW = 0.00e+00 NNZ =      396 SPT = 0.9852 Tmvc␣
↪= 0.001 T = 0.009
 Site =    11 /    20 .. Mmpo =   178 DW = 0.00e+00 NNZ =      728 SPT = 0.9774 Tmvc␣
↪= 0.001 T = 0.010
 Site =    12 /    20 .. Mmpo =   147 DW = 0.00e+00 NNZ =      496 SPT = 0.9810 Tmvc␣
↪= 0.001 T = 0.008
 Site =    13 /    20 .. Mmpo =   100 DW = 0.00e+00 NNZ =      412 SPT = 0.9720 Tmvc␣
↪= 0.000 T = 0.011
 Site =    14 /    20 .. Mmpo =    77 DW = 0.00e+00 NNZ =      254 SPT = 0.9670 Tmvc␣
↪= 0.000 T = 0.005
 Site =    15 /    20 .. Mmpo =    54 DW = 0.00e+00 NNZ =      120 SPT = 0.9711 Tmvc␣
↪= 0.000 T = 0.004
 Site =    16 /    20 .. Mmpo =    39 DW = 0.00e+00 NNZ =      124 SPT = 0.9411 Tmvc␣
↪= 0.000 T = 0.005
 Site =    17 /    20 .. Mmpo =    20 DW = 0.00e+00 NNZ =       76 SPT = 0.9026 Tmvc␣
↪= 0.000 T = 0.003
 Site =    18 /    20 .. Mmpo =     7 DW = 0.00e+00 NNZ =       22 SPT = 0.8429 Tmvc␣
↪= 0.000 T = 0.002
 Site =    19 /    20 .. Mmpo =     1 DW = 0.00e+00 NNZ =        7 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.002
Ttotal =      0.199 Tmvc-total = 0.021 MPO bond dimension =   181 MaxDW = 0.00e+00
NNZ =         7386 SIZE =       204350 SPT = 0.9639

Rank =     0 Ttotal =      0.296 MPO method = FastBipartite bond dimension =     181␣
↪NNZ =         7386 SIZE =       204350 SPT = 0.9639

Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.091 | E =    -107.6541220194 | DW = 8.01e-08

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.230 | E =    -107.6541223360 | DE = -3.17e-07 | DW = 7.44e-08

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.557 | E =    -107.6541224309 | DE = -9.49e-08 | DW = 8.01e-08

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.886 | E =    -107.6541224309 | DE = 1.79e-12 | DW = 6.88e-08
```

(continues on next page)

(continued from previous page)

```
Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      6.128 | E =    -107.6541224313 | DE = -3.64e-10 | DW = 8.01e-11

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      7.178 | E =    -107.6541224313 | DE = 1.31e-12 | DW = 5.93e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      8.296 | E =    -107.6541224313 | DE = -5.68e-14 | DW = 8.01e-11

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      9.355 | E =    -107.6541224313 | DE = -2.30e-12 | DW = 8.16e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =     10.016 | E =    -107.6541224313 | DE = 2.84e-14 | DW = 8.22e-20

DMRG energy = -107.654122431266515
```

4.1.6 The SGB Mode

In this section, we try to solve the problem by first transfroming the model into a qubit (spin) model. The code will automatically use Jordan-Wigner transform to change the fermionic operators in the Hamiltonian into spin operators, before constructing the MPO.

To use the SGB mode for ab initio systems, remember to add the `heis_twos=1` parameter (indicating the 1/2 spin at each site) in `driver.initialize_system`.

[9]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.GHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_ghf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SGB, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym,␣
↪heis_twos=1)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=1)
ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
```

(continues on next page)

(continued from previous page)

```
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)
```

```
integral symmetrize error =  2.03655637126935e-13
integral cutoff error =  0.0
mpo terms =       5906

Build MPO | Nsites =    20 | Nterms =       5906 | Algorithm = FastBIP | Cutoff = 1.
↪00e-20
 Site =     0 /    20 .. Mmpo =     7 DW = 0.00e+00 NNZ =        7 SPT = 0.0000 Tmvc␣
↪= 0.001 T = 0.009
 Site =     1 /    20 .. Mmpo =    20 DW = 0.00e+00 NNZ =       19 SPT = 0.8643 Tmvc␣
↪= 0.001 T = 0.011
 Site =     2 /    20 .. Mmpo =    47 DW = 0.00e+00 NNZ =       49 SPT = 0.9479 Tmvc␣
↪= 0.002 T = 0.012
 Site =     3 /    20 .. Mmpo =    62 DW = 0.00e+00 NNZ =      251 SPT = 0.9139 Tmvc␣
↪= 0.002 T = 0.017
 Site =     4 /    20 .. Mmpo =    81 DW = 0.00e+00 NNZ =      273 SPT = 0.9456 Tmvc␣
↪= 0.003 T = 0.018
 Site =     5 /    20 .. Mmpo =   104 DW = 0.00e+00 NNZ =      357 SPT = 0.9576 Tmvc␣
↪= 0.002 T = 0.014
 Site =     6 /    20 .. Mmpo =   129 DW = 0.00e+00 NNZ =      563 SPT = 0.9580 Tmvc␣
↪= 0.003 T = 0.017
 Site =     7 /    20 .. Mmpo =   126 DW = 0.00e+00 NNZ =     2318 SPT = 0.8574 Tmvc␣
↪= 0.004 T = 0.019
 Site =     8 /    20 .. Mmpo =   155 DW = 0.00e+00 NNZ =      208 SPT = 0.9893 Tmvc␣
↪= 0.001 T = 0.011
 Site =     9 /    20 .. Mmpo =   148 DW = 0.00e+00 NNZ =      684 SPT = 0.9702 Tmvc␣
↪= 0.001 T = 0.012
 Site =    10 /    20 .. Mmpo =   181 DW = 0.00e+00 NNZ =      384 SPT = 0.9857 Tmvc␣
↪= 0.001 T = 0.011
 Site =    11 /    20 .. Mmpo =   178 DW = 0.00e+00 NNZ =      718 SPT = 0.9777 Tmvc␣
↪= 0.001 T = 0.009
 Site =    12 /    20 .. Mmpo =   147 DW = 0.00e+00 NNZ =      496 SPT = 0.9810 Tmvc␣
↪= 0.001 T = 0.014
 Site =    13 /    20 .. Mmpo =   100 DW = 0.00e+00 NNZ =      412 SPT = 0.9720 Tmvc␣
↪= 0.001 T = 0.010
 Site =    14 /    20 .. Mmpo =    77 DW = 0.00e+00 NNZ =      234 SPT = 0.9696 Tmvc␣
↪= 0.000 T = 0.005
 Site =    15 /    20 .. Mmpo =    54 DW = 0.00e+00 NNZ =      118 SPT = 0.9716 Tmvc␣
↪= 0.000 T = 0.006
 Site =    16 /    20 .. Mmpo =    39 DW = 0.00e+00 NNZ =      124 SPT = 0.9411 Tmvc␣
↪= 0.000 T = 0.005
 Site =    17 /    20 .. Mmpo =    20 DW = 0.00e+00 NNZ =       76 SPT = 0.9026 Tmvc␣
↪= 0.000 T = 0.006
```

(continues on next page)

(continued from previous page)

```
 Site =    18 /    20 .. Mmpo =     7 DW = 0.00e+00 NNZ =       22 SPT = 0.8429 Tmvc␣
↪= 0.000 T = 0.005
 Site =    19 /    20 .. Mmpo =     1 DW = 0.00e+00 NNZ =        7 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.005
Ttotal =      0.214 Tmvc-total = 0.024 MPO bond dimension =   181 MaxDW = 0.00e+00
NNZ =         7320 SIZE =       204350 SPT = 0.9642

Rank =     0 Ttotal =      0.317 MPO method = FastBipartite bond dimension =     181␣
↪NNZ =         7320 SIZE =       204350 SPT = 0.9642

Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.172 | E =    -107.6541214496 | DW = 5.29e-08

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.994 | E =    -107.6541223419 | DE = -8.92e-07 | DW = 6.70e-08

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.754 | E =    -107.6541224354 | DE = -9.35e-08 | DW = 5.28e-08

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.536 | E =    -107.6541224354 | DE = -1.14e-13 | DW = 6.50e-08

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.606 | E =    -107.6541224379 | DE = -2.55e-09 | DW = 6.45e-11

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      5.653 | E =    -107.6541224379 | DE = 2.27e-13 | DW = 5.96e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      6.786 | E =    -107.6541224379 | DE = 5.68e-14 | DW = 6.45e-11

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      7.843 | E =    -107.6541224379 | DE = 1.39e-12 | DW = 6.44e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      8.501 | E =    -107.6541224379 | DE = -2.84e-14 | DW = 6.36e-20
```

(continues on next page)

(continued from previous page)

```
DMRG energy = -107.654122437938938
```

4.1.7 Relativistic DMRG

For relativistic DMRG, we use `get_dhf_integrals` function to get the integrals. We use the `SGFCPX` Mode in `block2` to execute DMRG. Note that the integrals, MPO, and MPS will all contain complex numbers in this mode.

The `symm_type` parameter `SymmetryTypes.SGFCPX` can also be written as `SymmetryTypes.SGF | SymmetryTypes.CPX`.

[10]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.DHF(mol).set(with_gaunt=True, with_breit=True).run(conv_tol=1E-12)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_dhf_integrals(mf,
    ncore=0, ncas=None, pg_symm=False)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SGFCPX, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=1)
ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)
```

```
integral symmetrize error =  0.0
integral cutoff error =  0.0
mpo terms =      44348

Build MPO | Nsites =    20 | Nterms =      44348 | Algorithm = FastBIP | Cutoff = 1.
↪00e-20
 Site =     0 /    20 .. Mmpo =     9 DW = 0.00e+00 NNZ =        9 SPT = 0.0000 Tmvc␣
↪= 0.004 T = 0.017
 Site =     1 /    20 .. Mmpo =    28 DW = 0.00e+00 NNZ =       23 SPT = 0.9087 Tmvc␣
↪= 0.005 T = 0.020
 Site =     2 /    20 .. Mmpo =    63 DW = 0.00e+00 NNZ =      404 SPT = 0.7710 Tmvc␣
↪= 0.006 T = 0.029
 Site =     3 /    20 .. Mmpo =    78 DW = 0.00e+00 NNZ =      589 SPT = 0.8801 Tmvc␣
↪= 0.007 T = 0.029
 Site =     4 /    20 .. Mmpo =    97 DW = 0.00e+00 NNZ =      933 SPT = 0.8767 Tmvc␣
↪= 0.007 T = 0.030
 Site =     5 /    20 .. Mmpo =   120 DW = 0.00e+00 NNZ =     1315 SPT = 0.8870 Tmvc␣
↪= 0.008 T = 0.032
```

(continues on next page)

(continued from previous page)

```
 Site =     6 /    20 .. Mmpo =   147 DW = 0.00e+00 NNZ =     1722 SPT = 0.9024 Tmvc␣
↪= 0.010 T = 0.038
 Site =     7 /    20 .. Mmpo =   178 DW = 0.00e+00 NNZ =     2141 SPT = 0.9182 Tmvc␣
↪= 0.009 T = 0.034
 Site =     8 /    20 .. Mmpo =   213 DW = 0.00e+00 NNZ =     2577 SPT = 0.9320 Tmvc␣
↪= 0.009 T = 0.037
 Site =     9 /    20 .. Mmpo =   252 DW = 0.00e+00 NNZ =     2967 SPT = 0.9447 Tmvc␣
↪= 0.010 T = 0.040
 Site =    10 /    20 .. Mmpo =   213 DW = 0.00e+00 NNZ =    17974 SPT = 0.6651 Tmvc␣
↪= 0.010 T = 0.056
 Site =    11 /    20 .. Mmpo =   178 DW = 0.00e+00 NNZ =     2599 SPT = 0.9315 Tmvc␣
↪= 0.004 T = 0.023
 Site =    12 /    20 .. Mmpo =   147 DW = 0.00e+00 NNZ =     2195 SPT = 0.9161 Tmvc␣
↪= 0.003 T = 0.021
 Site =    13 /    20 .. Mmpo =   120 DW = 0.00e+00 NNZ =     1787 SPT = 0.8987 Tmvc␣
↪= 0.002 T = 0.017
 Site =    14 /    20 .. Mmpo =    97 DW = 0.00e+00 NNZ =     1403 SPT = 0.8795 Tmvc␣
↪= 0.005 T = 0.021
 Site =    15 /    20 .. Mmpo =    78 DW = 0.00e+00 NNZ =     1013 SPT = 0.8661 Tmvc␣
↪= 0.002 T = 0.014
 Site =    16 /    20 .. Mmpo =    63 DW = 0.00e+00 NNZ =      652 SPT = 0.8673 Tmvc␣
↪= 0.001 T = 0.011
 Site =    17 /    20 .. Mmpo =    28 DW = 0.00e+00 NNZ =      496 SPT = 0.7188 Tmvc␣
↪= 0.001 T = 0.012
 Site =    18 /    20 .. Mmpo =     9 DW = 0.00e+00 NNZ =       26 SPT = 0.8968 Tmvc␣
↪= 0.000 T = 0.005
 Site =    19 /    20 .. Mmpo =     1 DW = 0.00e+00 NNZ =        9 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.003
Ttotal =      0.488 Tmvc-total = 0.103 MPO bond dimension =   252 MaxDW = 0.00e+00
NNZ =        40834 SIZE =       323082 SPT = 0.8736

Rank =     0 Ttotal =      0.589 MPO method = FastBipartite bond dimension =     252␣
↪NNZ =        40834 SIZE =       323082 SPT = 0.8736

Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     27.182 | E =    -107.6929206708 | DW = 6.28e-10

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     40.137 | E =    -107.6929209452 | DE = -2.74e-07 | DW = 7.49e-10

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     51.567 | E =    -107.6929209492 | DE = -3.94e-09 | DW = 2.15e-10
```

(continues on next page)

(continued from previous page)

```
Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     60.941 | E =    -107.6929209492 | DE = -5.40e-13 | DW = 7.52e-10

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     80.487 | E =    -107.6929209492 | DE = -4.55e-13 | DW = 7.91e-20

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     98.052 | E =    -107.6929209492 | DE = -5.68e-14 | DW = 1.15e-19

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    121.383 | E =    -107.6929209492 | DE = 0.00e+00 | DW = 6.27e-20

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    138.658 | E =    -107.6929209492 | DE = -2.84e-14 | DW = 1.15e-19

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =    147.981 | E =    -107.6929209492 | DE = 2.84e-14 | DW = 7.89e-20

DMRG energy = -107.692920949170187
```

4.1.8 The LZ Mode

For diatomic molecules, we can set symmetry dooh in pyscf, and then use `itg.lz_symm_adaptation` to adapt the atomic orbitals for the LZ symmetry before running Hartree-Fock. Then we can use the LZ modes in block2 to perform DMRG.

The LZ mode can be combined with SU2, SZ or SGF spin symmetries, and the SAny prefix in SymmetryTypes. To activate the SAny prefix, the block2 code needs to be compiled with the -DUSE_SANY=ON option (this option is ON by default in the pip precompiled binaries). Optionally, when -DUSE_SANY=ON, one can also set -DUSE_SG=OFF -DUSE_SU2SZ=OFF to disable the normal SU2/SZ/SGF modes. One can use SymmetryTypes.SAnySU2/SymmetryTypes.SAnySZ/SymmetryTypes.SAnySGF instead for normal symmetries (with some limitations) when -DUSE_SG=OFF -DUSE_SU2SZ=OFF is used.

With SU2 (spin-adapted DMRG):

[11]:
```
mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="dooh", verbose=0)
mol.symm_orb, z_irrep, g_irrep = itg.lz_symm_adaptation(mol)
mf = scf.RHF(mol).run(conv_tol=1E-14)
```

(continues on next page)

(continued from previous page)

```
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym_z = itg.get_rhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=1, irrep_id=z_irrep)
print(orb_sym_z)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SAnySU2LZ, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym_z,␣
↪pg_irrep=0)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=1)
ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)
```

```
[ 0  0  0  0 -1  1  0  1 -1  0]
integral symmetrize error =  2.0562981879479644e-15
integral cutoff error =  6.776263578034403e-21
mpo terms =       1882

Build MPO | Nsites =    10 | Nterms =       1882 | Algorithm = FastBIP | Cutoff = 1.
↪00e-20
 Site =     0 /    10 .. Mmpo =    14 DW = 0.00e+00 NNZ =       14 SPT = 0.0000 Tmvc␣
↪= 0.001 T = 0.036
 Site =     1 /    10 .. Mmpo =    34 DW = 0.00e+00 NNZ =      106 SPT = 0.7773 Tmvc␣
↪= 0.001 T = 0.049
 Site =     2 /    10 .. Mmpo =    56 DW = 0.00e+00 NNZ =      189 SPT = 0.9007 Tmvc␣
↪= 0.001 T = 0.054
 Site =     3 /    10 .. Mmpo =    66 DW = 0.00e+00 NNZ =      824 SPT = 0.7771 Tmvc␣
↪= 0.001 T = 0.058
 Site =     4 /    10 .. Mmpo =    88 DW = 0.00e+00 NNZ =      154 SPT = 0.9735 Tmvc␣
↪= 0.001 T = 0.035
 Site =     5 /    10 .. Mmpo =    94 DW = 0.00e+00 NNZ =      318 SPT = 0.9616 Tmvc␣
↪= 0.000 T = 0.029
 Site =     6 /    10 .. Mmpo =    64 DW = 0.00e+00 NNZ =      237 SPT = 0.9606 Tmvc␣
↪= 0.000 T = 0.024
 Site =     7 /    10 .. Mmpo =    40 DW = 0.00e+00 NNZ =      145 SPT = 0.9434 Tmvc␣
↪= 0.000 T = 0.020
 Site =     8 /    10 .. Mmpo =    14 DW = 0.00e+00 NNZ =       49 SPT = 0.9125 Tmvc␣
↪= 0.000 T = 0.010
 Site =     9 /    10 .. Mmpo =     1 DW = 0.00e+00 NNZ =       14 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.007
Ttotal =      0.321 Tmvc-total = 0.005 MPO bond dimension =    94 MaxDW = 0.00e+00
NNZ =         2050 SIZE =        29320 SPT = 0.9301

Rank =     0 Ttotal =      0.379 MPO method = FastBipartite bond dimension =      94␣
↪NNZ =         2050 SIZE =        29320 SPT = 0.9301
```

(continues on next page)

(continued from previous page)

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.704 | E =    -107.6541224475 | DW = 2.06e-10

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.071 | E =    -107.6541224475 | DE = 1.19e-12 | DW = 3.19e-20

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.438 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 2.06e-10

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.808 | E =    -107.6541224475 | DE = -8.64e-12 | DW = 3.50e-20

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.291 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 5.14e-20

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.713 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 3.17e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.196 | E =    -107.6541224475 | DE = 2.84e-14 | DW = 5.84e-20

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.622 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 1.86e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      3.889 | E =    -107.6541224475 | DE = 2.84e-14 | DW = 1.94e-20

DMRG energy = -107.654122447523761
```

With SZ (non-spin-adapted DMRG):

[12]:
```
mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="dooh", verbose=0)
mol.symm_orb, z_irrep, g_irrep = itg.lz_symm_adaptation(mol)
mf = scf.UHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym_z = itg.get_uhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=1, irrep_id=z_irrep)
```

(continues on next page)

(continued from previous page)

```
print(orb_sym_z)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SAnySZLZ, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym_z,␣
↪pg_irrep=0)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=1)
ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)
```

```
[ 0  0  0  0 -1  1  0  1 -1  0]
integral symmetrize error =  8.030340066584701e-15
integral cutoff error =  1.249373597200093e-20
mpo terms =       5237

Build MPO | Nsites =    10 | Nterms =       5237 | Algorithm = FastBIP | Cutoff = 1.
↪00e-20
 Site =     0 /    10 .. Mmpo =    30 DW = 0.00e+00 NNZ =       30 SPT = 0.0000 Tmvc␣
↪= 0.002 T = 0.077
 Site =     1 /    10 .. Mmpo =    66 DW = 0.00e+00 NNZ =      278 SPT = 0.8596 Tmvc␣
↪= 0.001 T = 0.117
 Site =     2 /    10 .. Mmpo =   110 DW = 0.00e+00 NNZ =      478 SPT = 0.9342 Tmvc␣
↪= 0.002 T = 0.126
 Site =     3 /    10 .. Mmpo =   123 DW = 0.00e+00 NNZ =     2285 SPT = 0.8311 Tmvc␣
↪= 0.001 T = 0.131
 Site =     4 /    10 .. Mmpo =   170 DW = 0.00e+00 NNZ =      360 SPT = 0.9828 Tmvc␣
↪= 0.001 T = 0.062
 Site =     5 /    10 .. Mmpo =   186 DW = 0.00e+00 NNZ =      814 SPT = 0.9743 Tmvc␣
↪= 0.001 T = 0.065
 Site =     6 /    10 .. Mmpo =   126 DW = 0.00e+00 NNZ =      579 SPT = 0.9753 Tmvc␣
↪= 0.001 T = 0.046
 Site =     7 /    10 .. Mmpo =    82 DW = 0.00e+00 NNZ =      264 SPT = 0.9744 Tmvc␣
↪= 0.000 T = 0.025
 Site =     8 /    10 .. Mmpo =    30 DW = 0.00e+00 NNZ =      187 SPT = 0.9240 Tmvc␣
↪= 0.000 T = 0.017
 Site =     9 /    10 .. Mmpo =     1 DW = 0.00e+00 NNZ =       30 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.011
Ttotal =      0.678 Tmvc-total = 0.009 MPO bond dimension =   186 MaxDW = 0.00e+00
NNZ =         5305 SIZE =       111588 SPT = 0.9525

Rank =     0 Ttotal =      0.740 MPO method = FastBipartite bond dimension =     186␣
↪NNZ =         5305 SIZE =       111588 SPT = 0.9525

Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
```

(continues on next page)

(continued from previous page)

```
↪Dav threshold =  1.00e-10
Time elapsed =      1.233 | E =    -107.6541224475 | DW = 3.06e-08

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.979 | E =    -107.6541224475 | DE = -1.18e-11 | DW = 3.32e-08

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.687 | E =    -107.6541224455 | DE = 2.05e-09 | DW = 3.06e-08

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.429 | E =    -107.6541224455 | DE = -3.41e-13 | DW = 3.27e-08

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.297 | E =    -107.6541224455 | DE = -1.14e-13 | DW = 3.09e-11

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      5.229 | E =    -107.6541224455 | DE = 1.14e-13 | DW = 1.40e-19

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      6.354 | E =    -107.6541224455 | DE = -2.84e-14 | DW = 3.09e-11

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      7.963 | E =    -107.6541224455 | DE = 1.71e-13 | DW = 1.58e-19

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      9.102 | E =    -107.6541224455 | DE = -2.84e-14 | DW = 1.23e-19

DMRG energy = -107.654122445468445
```

With SGF (spin-orbital DMRG):

[13]:
```
from pyscf.scf.ghf_symm import GHF
from pyscf import symm, lib
from pyscf.scf import hf_symm
import scipy.linalg
import numpy as np

# fix pyscf 2.3.0 bug in ghf_symm for complex orbtials
```

(continues on next page)

(continued from previous page)

```
def ghf_eig(self, h, s, symm_orb=None, irrep_id=None):
    if symm_orb is None or irrep_id is None:
        mol = self.mol
        symm_orb = mol.symm_orb
        irrep_id = mol.irrep_id
    nirrep = len(symm_orb)
    symm_orb = [scipy.linalg.block_diag(c, c) for c in symm_orb]
    h = symm.symmetrize_matrix(h, symm_orb)
    s = symm.symmetrize_matrix(s, symm_orb)
    cs = []
    es = []
    orbsym = []
    for ir in range(nirrep):
        e, c = self._eigh(h[ir], s[ir])
        cs.append(c)
        es.append(e)
        orbsym.append([irrep_id[ir]] * e.size)
    e = np.hstack(es)
    c = hf_symm.so2ao_mo_coeff(symm_orb, cs)
    c = lib.tag_array(c, orbsym=np.hstack(orbsym))
    return e, c

GHF.eig = ghf_eig

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="dooh", verbose=0)
mol.symm_orb, z_irrep, g_irrep = itg.lz_symm_adaptation(mol)
mf = scf.GHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym_z = itg.get_ghf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=1, irrep_id=z_irrep)
print(orb_sym_z)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SAnySGFLZ, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym_z,␣
↪pg_irrep=0)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=1)
ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)
```

```
[ 0  0  0  0  0  0  0  0 -1 -1  1  1  0  0  1  1 -1 -1  0  0]
integral symmetrize error =  1.457638075028426e-14
integral cutoff error =  6.527447649778452e-20
mpo terms =      13645
```

(continues on next page)

(continued from previous page)

```
Build MPO | Nsites =    20 | Nterms =      13645 | Algorithm = FastBIP | Cutoff = 1.
↪00e-20
 Site =     0 /    20 .. Mmpo =     9 DW = 0.00e+00 NNZ =        9 SPT = 0.0000 Tmvc␣
↪= 0.002 T = 0.100
 Site =     1 /    20 .. Mmpo =    28 DW = 0.00e+00 NNZ =       23 SPT = 0.9087 Tmvc␣
↪= 0.002 T = 0.171
 Site =     2 /    20 .. Mmpo =    47 DW = 0.00e+00 NNZ =      307 SPT = 0.7667 Tmvc␣
↪= 0.003 T = 0.226
 Site =     3 /    20 .. Mmpo =    62 DW = 0.00e+00 NNZ =      358 SPT = 0.8771 Tmvc␣
↪= 0.003 T = 0.276
 Site =     4 /    20 .. Mmpo =    81 DW = 0.00e+00 NNZ =      515 SPT = 0.8975 Tmvc␣
↪= 0.003 T = 0.292
 Site =     5 /    20 .. Mmpo =   104 DW = 0.00e+00 NNZ =      667 SPT = 0.9208 Tmvc␣
↪= 0.003 T = 0.305
 Site =     6 /    20 .. Mmpo =   131 DW = 0.00e+00 NNZ =      796 SPT = 0.9416 Tmvc␣
↪= 0.003 T = 0.309
 Site =     7 /    20 .. Mmpo =   126 DW = 0.00e+00 NNZ =     5583 SPT = 0.6618 Tmvc␣
↪= 0.003 T = 0.305
 Site =     8 /    20 .. Mmpo =   161 DW = 0.00e+00 NNZ =      266 SPT = 0.9869 Tmvc␣
↪= 0.002 T = 0.142
 Site =     9 /    20 .. Mmpo =   168 DW = 0.00e+00 NNZ =      732 SPT = 0.9729 Tmvc␣
↪= 0.002 T = 0.138
 Site =    10 /    20 .. Mmpo =   187 DW = 0.00e+00 NNZ =     1400 SPT = 0.9554 Tmvc␣
↪= 0.002 T = 0.130
 Site =    11 /    20 .. Mmpo =   178 DW = 0.00e+00 NNZ =      964 SPT = 0.9710 Tmvc␣
↪= 0.001 T = 0.110
 Site =    12 /    20 .. Mmpo =   147 DW = 0.00e+00 NNZ =      850 SPT = 0.9675 Tmvc␣
↪= 0.001 T = 0.080
 Site =    13 /    20 .. Mmpo =   120 DW = 0.00e+00 NNZ =      667 SPT = 0.9622 Tmvc␣
↪= 0.001 T = 0.066
 Site =    14 /    20 .. Mmpo =    97 DW = 0.00e+00 NNZ =      441 SPT = 0.9621 Tmvc␣
↪= 0.001 T = 0.041
 Site =    15 /    20 .. Mmpo =    78 DW = 0.00e+00 NNZ =      300 SPT = 0.9603 Tmvc␣
↪= 0.000 T = 0.031
 Site =    16 /    20 .. Mmpo =    57 DW = 0.00e+00 NNZ =      417 SPT = 0.9062 Tmvc␣
↪= 0.000 T = 0.025
 Site =    17 /    20 .. Mmpo =    28 DW = 0.00e+00 NNZ =       84 SPT = 0.9474 Tmvc␣
↪= 0.000 T = 0.009
 Site =    18 /    20 .. Mmpo =     9 DW = 0.00e+00 NNZ =       28 SPT = 0.8889 Tmvc␣
↪= 0.000 T = 0.007
 Site =    19 /    20 .. Mmpo =     1 DW = 0.00e+00 NNZ =        9 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.004
Ttotal =      2.767 Tmvc-total = 0.032 MPO bond dimension =   187 MaxDW = 0.00e+00
NNZ =        14416 SIZE =       229418 SPT = 0.9372
```

(continues on next page)

(continued from previous page)

```
Rank =     0 Ttotal =      2.864 MPO method = FastBipartite bond dimension =     187␣
↪NNZ =        14416 SIZE =       229418 SPT = 0.9372

Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.314 | E =    -107.6541218464 | DW = 1.76e-08

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.490 | E =    -107.6541222920 | DE = -4.46e-07 | DW = 5.23e-08

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      5.869 | E =    -107.6541224411 | DE = -1.49e-07 | DW = 1.75e-08

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      7.083 | E =    -107.6541224411 | DE = -5.34e-12 | DW = 5.24e-08

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      8.880 | E =    -107.6541224448 | DE = -3.71e-09 | DW = 9.90e-12

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     10.506 | E =    -107.6541224455 | DE = -6.31e-10 | DW = 5.25e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     12.313 | E =    -107.6541224455 | DE = -2.84e-14 | DW = 9.88e-12

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     13.931 | E =    -107.6541224455 | DE = 8.53e-14 | DW = 5.12e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =     14.954 | E =    -107.6541224455 | DE = 0.00e+00 | DW = 8.00e-20

DMRG energy = -107.654122445469611
```

4.1.9 Expectation and N-Particle Density Matrices

Once the optimized MPS is obtained, we can compute the expectation value on it, including its norm, the energy expectation, $\langle S^2 \rangle$, N-particle density matrix, or any operator that can be constructed as an MPO.

In this example, we compute the triplet state.

[14]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.RHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

spin = 2

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SU2, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)

ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

impo = driver.get_identity_mpo()

norm = driver.expectation(ket, impo, ket)
ener = driver.expectation(ket, mpo, ket)

print('Norm = %20.15f' % norm)
print('Energy expectation = %20.15f' % (ener / norm))

# <S^2> [ in spin-adapted mode this is always S(S+1) ]
ssq_mpo = driver.get_spin_square_mpo(iprint=0)
ssq = driver.expectation(ket, ssq_mpo, ket)
print('<S^2> expectation = %20.15f' % (ssq / norm))
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.580 | E =    -106.9391328597 | DW = 3.38e-10

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.880 | E =    -106.9391328597 | DE = -3.04e-12 | DW = 9.74e-19
```

(continues on next page)

(continued from previous page)

```
Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.193 | E =    -106.9391328597 | DE = -5.68e-14 | DW = 3.38e-10

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.481 | E =    -106.9391328597 | DE = 2.56e-13 | DW = 1.53e-18

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.833 | E =    -106.9391328597 | DE = 5.68e-14 | DW = 1.12e-16

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.168 | E =    -106.9391328597 | DE = -1.42e-13 | DW = 1.35e-19

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.507 | E =    -106.9391328597 | DE = 2.84e-14 | DW = 1.12e-16

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.825 | E =    -106.9391328597 | DE = 1.14e-13 | DW = 1.38e-19

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      3.045 | E =    -106.9391328597 | DE = 0.00e+00 | DW = 2.47e-20

DMRG energy = -106.939132859666600
Norm =    1.000000000000000
Energy expectation = -106.939132859666586
<S^2> expectation =    2.000000000000001
```

We can also evaluate expectation of arbitray operator such as the occupancy in the first orbital

$$\hat{N}_0 = a^\dagger_{0\alpha} a_{0\alpha} + a^\dagger_{0\beta} a_{0\beta} = \sqrt{2} \left(a^\dagger_0\right)^{[1/2]} \otimes_{[0]} \left(a_0\right)^{[1/2]}$$

[15]:
```
b = driver.expr_builder()
b.add_term("(C+D)0", [0, 0], np.sqrt(2))
n_mpo = driver.get_mpo(b.finalize(), iprint=0)

n_0 = driver.expectation(ket, n_mpo, ket)
print('N0 expectation = %20.15f' % (n_0 / norm))
```

```
N0 expectation =    1.999995824361892
```

We can then verify this number using 1PDM:

[16]:
```
pdm1 = driver.get_1pdm(ket)
print('N0 expectation from 1pdm = %20.15f' % pdm1[0, 0])
```

```
N0 expectation from 1pdm =    1.999995824361891
```

We can compute the 3PDM and compare the result with the FCI 3PDM. Note that in `pyscf` the 3PDM is defined as

$$\mathrm{DM}_{ijklmn} := \langle E_{ij} E_{kl} E_{mn} \rangle$$

So we have to use the same convention in `block2` by setting the `npdm_expr` parameter in `block2` to `((C+D)0+((C+D)0+(C+D)0)0)0`.

[17]:
```
pdm3_b2 = driver.get_3pdm(ket, iprint=0, npdm_expr="((C+D)0+((C+D)0+(C+D)0)0)0")

from pyscf import fci

mx = fci.addons.fix_spin_(fci.FCI(mf), ss=2)
mx.kernel(h1e, g2e, ncas, nelec=n_elec, nroots=3, tol=1E-12)
print(mx.e_tot)
pdm3_fci = fci.rdm.make_dm123('FCI3pdm_kern_sf', mx.ci[0], mx.ci[0], ncas, n_elec)[2]

print('diff = ', np.linalg.norm(pdm3_fci - pdm3_b2))
```

```
[-106.93913286 -106.85412245 -106.70055113]
diff =  5.3418848100122e-06
```

4.1.10 Extract CSF and Determinant Coefficients

We can extract CSF (or determinant) coefficients from the spin-adapted MPS (or the non-spin-adapted MPS). The algorithm can compute all CSF or determinant with the absolute value of the coefficient above a threshold (called `cutoff`). The square of coefficient is the probability (weight) of the CSF or determinant.

Extracting CSF coefficients from spin-adapted MPS in the `SU2` mode:

[18]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.RHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SU2, n_threads=4)
```

(continues on next page)

(continued from previous page)

```
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)

ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

csfs, coeffs = driver.get_csf_coefficients(ket, cutoff=0.05, iprint=1)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.497 | E =    -107.6541224475 | DW = 1.87e-10

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.747 | E =    -107.6541224475 | DE = -3.33e-12 | DW = 4.05e-20

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.990 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 1.87e-10

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.224 | E =    -107.6541224475 | DE = -4.83e-13 | DW = 5.95e-20

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.498 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 2.70e-20

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.755 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 6.84e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.121 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 2.50e-20

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.617 | E =    -107.6541224475 | DE = 2.84e-14 | DW = 6.27e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
```

(continues on next page)

(continued from previous page)

```
Time elapsed =      2.911 | E =    -107.6541224475 | DE = 2.84e-14 | DW = 3.40e-20

DMRG energy = -107.654122447524642
mps center changed (temporarily)
dtrie finished 0.004684378000092693
Number of CSF =          9 (cutoff =      0.05)
Sum of weights of included CSF =    0.984368811359562

CSF          0 2222222000  =    0.957506528669410
CSF          1 2222202200  =   -0.131287867807398
CSF          2 2222022020  =   -0.131287794595548
CSF          3 2222+-2+-0  =    0.118594301088579
CSF          4 2222++2--0  =   -0.084968224982026
CSF          5 2220222020  =   -0.054710635444609
CSF          6 2220222200  =   -0.054710465272209
CSF          7 2222+2+0--  =    0.053881241407867
CSF          8 22222++-0-  =    0.053881215166800
```

Extracting determinant coefficients from non-spin-adapted MPS in the SZ mode:

[19]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.UHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_uhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SZ, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)

ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

csfs, coeffs = driver.get_csf_coefficients(ket, cutoff=0.05, iprint=1)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.826 | E =    -107.6541224475 | DW = 4.14e-08

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
```

(continues on next page)

(continued from previous page)

```
Time elapsed =      1.306 | E =    -107.6541224475 | DE = -1.90e-12 | DW = 5.67e-09

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.783 | E =    -107.6541224475 | DE = -1.42e-12 | DW = 4.14e-08

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.259 | E =    -107.6541224475 | DE = 1.42e-12 | DW = 5.13e-09

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.822 | E =    -107.6541224475 | DE = 2.07e-12 | DW = 3.60e-11

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.380 | E =    -107.6541224475 | DE = -3.24e-12 | DW = 1.71e-19

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.960 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 3.60e-11

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.519 | E =    -107.6541224475 | DE = 8.81e-13 | DW = 1.76e-19

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      4.910 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 7.05e-20

DMRG energy = -107.654122447523420
mps center changed (temporarily)
dtrie finished 0.0024573000000600587
Number of DET =          7 (cutoff =      0.05)
Sum of weights of included DET =    0.971331592192659

DET          0 2222222000  =   -0.957506477722952
DET          1 2222202200  =    0.131287949363695
DET          2 2222022020  =    0.131287934434239
DET          3 2222ab2ab0  =   -0.083825435733035
DET          4 2222ba2ba0  =   -0.083825435117335
DET          5 2220222020  =    0.054710529942414
DET          6 2220222200  =    0.054710516357351
```

4.1.11 Construct MPS from CSFs or Determinants

If we know important CSFs or determinants in the state and their coefficients, we can also use this information to construct MPS, and this can be used as an initial guess and further optimized. Note that this initial guess can generate very good initial energies, but if the given CSFs have many common doubly occupied and empty orbitals, the MPS will very likely optimize to a local minima.

In the SU2 mode:

[20]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.RHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SU2, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)

ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

csfs, coeffs = driver.get_csf_coefficients(ket, cutoff=0.05, iprint=1)

mps = driver.get_mps_from_csf_coefficients(csfs, coeffs, tag="CMPS", dot=2)
impo = driver.get_identity_mpo()
print(driver.expectation(mps, impo, mps))
print(driver.expectation(mps, mpo, mps) / driver.expectation(mps, impo, mps))

energy = driver.dmrg(mpo, mps, n_sweeps=5, bond_dims=[500] * 5, noises=[1E-5],
    thrds=[1E-10] * 5, iprint=1)
print('Ground state energy = %20.15f' % energy)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.517 | E =    -107.6541224475 | DW = 1.87e-10

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.760 | E =    -107.6541224475 | DE = -1.25e-11 | DW = 5.82e-20

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
```

(continues on next page)

(continued from previous page)

```
Time elapsed =      1.007 | E =    -107.6541224475 | DE = 2.84e-14 | DW = 1.87e-10

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.256 | E =    -107.6541224475 | DE = -1.39e-12 | DW = 4.80e-20

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.533 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 1.06e-20

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.779 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 5.63e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.052 | E =    -107.6541224475 | DE = 5.68e-14 | DW = 4.42e-20

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.328 | E =    -107.6541224475 | DE = -5.68e-14 | DW = 5.11e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      2.508 | E =    -107.6541224475 | DE = 2.84e-14 | DW = 5.29e-20

DMRG energy = -107.654122447524387
mps center changed (temporarily)
dtrie finished 0.0019679109999515276
Number of CSF =          9 (cutoff =      0.05)
Sum of weights of included CSF =    0.984368809657108

CSF          0 2222222000  =    0.957506521875156
CSF          1 2222202200  =   -0.131287813613838
CSF          2 2222022020  =   -0.131287676854108
CSF          3 2222+-2+-0  =    0.118594392287611
CSF          4 2222++2--0  =   -0.084968279487238
CSF          5 2220222020  =   -0.054710673444325
CSF          6 2220222200  =   -0.054710596031564
CSF          7 2222+2+0--  =    0.053881289057961
CSF          8 22222++-0-  =    0.053881233355426
0.9843688096571082
-107.61553663531718

Sweep =    0 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
```

(continues on next page)

(continued from previous page)

```
↪Dav threshold =  1.00e-10
Time elapsed =      0.052 | E =    -107.6285064669 | DW = 6.01e-21

Sweep =    1 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.102 | E =    -107.6288048128 | DE = -2.98e-04 | DW = 1.43e-21

Sweep =    2 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.155 | E =    -107.6289019710 | DE = -9.72e-05 | DW = 1.63e-21

Sweep =    3 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.206 | E =    -107.6289019713 | DE = -3.35e-10 | DW = 1.14e-20

Ground state energy = -107.628901971339630
```

In the SZ mode:

[21]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.UHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_uhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SZ, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)

ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

dets, coeffs = driver.get_csf_coefficients(ket, cutoff=0.05, iprint=1)

mps = driver.get_mps_from_csf_coefficients(dets, coeffs, tag="CMPS", dot=2)
impo = driver.get_identity_mpo()
print(driver.expectation(mps, impo, mps))
print(driver.expectation(mps, mpo, mps) / driver.expectation(mps, impo, mps))

energy = driver.dmrg(mpo, mps, n_sweeps=5, bond_dims=[500] * 5, noises=[1E-5],
    thrds=[1E-10] * 5, iprint=1)
print('Ground state energy = %20.15f' % energy)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.819 | E =    -107.6541224475 | DW = 4.14e-08

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.305 | E =    -107.6541224475 | DE = -5.66e-12 | DW = 5.12e-09

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.770 | E =    -107.6541224475 | DE = -6.25e-13 | DW = 4.14e-08

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.241 | E =    -107.6541224475 | DE = 1.14e-12 | DW = 4.98e-09

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.812 | E =    -107.6541224475 | DE = -1.14e-12 | DW = 3.60e-11

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.601 | E =    -107.6541224475 | DE = 8.53e-13 | DW = 1.77e-19

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.558 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 3.60e-11

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      5.488 | E =    -107.6541224475 | DE = -1.14e-12 | DW = 1.67e-19

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      6.143 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 4.96e-20

DMRG energy = -107.654122447524500
mps center changed (temporarily)
dtrie finished 0.004142805999890697
Number of DET =          7 (cutoff =      0.05)
Sum of weights of included DET =    0.971331639061928

DET          0 2222222000  =   -0.957506569121758
DET          1 2222022020  =    0.131287834673405
DET          2 2222202200  =    0.131287815095743
```

(continues on next page)

(continued from previous page)

```
DET          3 2222ba2ba0  =   -0.083825277849152
DET          4 2222ab2ab0  =   -0.083825277218030
DET          5 2220222200  =    0.054710476468495
DET          6 2220222020  =    0.054710443995518
0.9713316390619284
-107.5944732555996

Sweep =    0 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.090 | E =    -107.6101011814 | DW = 8.24e-21

Sweep =    1 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.185 | E =    -107.6102811404 | DE = -1.80e-04 | DW = 1.03e-20

Sweep =    2 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.282 | E =    -107.6104759466 | DE = -1.95e-04 | DW = 1.28e-20

Sweep =    3 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.375 | E =    -107.6104759470 | DE = -4.56e-10 | DW = 1.14e-20

Ground state energy = -107.610475947023915
```

4.1.12 MPS Initial Guess from Occupancies

We can also construct the MPS using an estimate of the occupancy information at each site from a cheaper method (such as CCSD). We can use this information to construct the initial quantum number distribution in the MPS.

Note that this initial guess can generate very good energies in the first few sweeps, but if the given occupancies have many doubly occupied and empty orbitals, the MPS will very likely optimize to a local minima. One can shift the occupancies into the equal probability occupancies (for example, setting `bias=0.4` in the example below) to randomize the initial guess.

In the `SU2` mode:

[22]:
```
from pyscf import gto, scf, cc

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.RHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

bias = 0.1 # make it more random
```

(continues on next page)

(continued from previous page)

```
occs = np.diag(cc.CCSD(mf).run().make_rdm1())
occs = occs + bias * (occs < 1) - bias * (occs > 1)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SU2, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)
ket = driver.get_random_mps(tag="GS", bond_dim=250, occs=occs, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.053 | E =    -107.5033999317 | DW = 2.74e-21

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.107 | E =    -107.5830578672 | DE = -7.97e-02 | DW = 5.53e-21

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.175 | E =    -107.5836402840 | DE = -5.82e-04 | DW = 1.44e-20

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.238 | E =    -107.5867932582 | DE = -3.15e-03 | DW = 3.21e-21

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.292 | E =    -107.5868061724 | DE = -1.29e-05 | DW = 1.17e-20

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.355 | E =    -107.5868152464 | DE = -9.07e-06 | DW = 9.11e-21

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.412 | E =    -107.5868152464 | DE = 0.00e+00 | DW = 1.51e-20

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.466 | E =    -107.5868152464 | DE = 0.00e+00 | DW = 1.40e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
```

(continues on next page)

(continued from previous page)

```
↪Dav threshold =  1.00e-09
Time elapsed =      0.521 | E =    -107.5868152464 | DE = 0.00e+00 | DW = 2.11e-20

DMRG energy = -107.586815246376730
```

In the SZ mode:

[23]:
```
from pyscf import gto, scf, cc

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.UHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_uhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

bias = 0.1 # make it more random
occs = np.diag(np.sum(cc.UCCSD(mf).run().make_rdm1(), axis=0))
occs = occs + bias * (occs < 1) - bias * (occs > 1)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SZ, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)
ket = driver.get_random_mps(tag="GS", bond_dim=250, occs=occs, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.060 | E =    -107.5033999316 | DW = 1.96e-21

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.131 | E =    -107.5830508082 | DE = -7.97e-02 | DW = 5.41e-21

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.204 | E =    -107.5836402367 | DE = -5.89e-04 | DW = 3.59e-21

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.294 | E =    -107.5867932578 | DE = -3.15e-03 | DW = 1.80e-20

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
```

(continues on next page)

(continued from previous page)

```
Time elapsed =      0.382 | E =    -107.5868061676 | DE = -1.29e-05 | DW = 1.53e-20

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.469 | E =    -107.5868152464 | DE = -9.08e-06 | DW = 1.02e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.557 | E =    -107.5868152464 | DE = 0.00e+00 | DW = 1.19e-20

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.641 | E =    -107.5868152464 | DE = 0.00e+00 | DW = 1.39e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      0.717 | E =    -107.5868152464 | DE = 2.84e-14 | DW = 1.67e-20

DMRG energy = -107.586815246376702
```

In the SGF mode:

[24]:
```
from pyscf import gto, scf, cc

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.GHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_ghf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

bias = 0.25 # make it more random
occs = np.sum(cc.GCCSD(mf).run().make_rdm1(), axis=0)
occs = occs + bias * (occs < 0.5) - bias * (occs > 0.5)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SGF, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)
ket = driver.get_random_mps(tag="GS", bond_dim=250, occs=occs, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.097 | E =    -107.6538892686 | DW = 2.59e-08
```

(continues on next page)

(continued from previous page)

```
Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.444 | E =    -107.6539226748 | DE = -3.34e-05 | DW = 9.28e-09

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.623 | E =    -107.6539461908 | DE = -2.35e-05 | DW = 5.62e-08

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.728 | E =    -107.6539506829 | DE = -4.49e-06 | DW = 1.96e-08

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      5.627 | E =    -107.6539506829 | DE = 3.07e-12 | DW = 2.73e-11

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      6.502 | E =    -107.6539506829 | DE = 7.39e-13 | DW = 4.70e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      7.419 | E =    -107.6539506829 | DE = 0.00e+00 | DW = 2.73e-11

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      8.281 | E =    -107.6539506829 | DE = -3.84e-12 | DW = 4.67e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      8.825 | E =    -107.6539506829 | DE = -2.84e-14 | DW = 5.57e-20

DMRG energy = -107.653950682886418
```

4.1.13 Change from SU2 MPS to SZ MPS

We can also transform the spin-adapted MPS generated in the `SU2` mode to the non-spin-adapted MPS, which can be used in the `SZ` mode. After obtaining the non-spin-adapted MPS, one need to redo `driver.initialize_system`, `driver.get_qc_mpo`, etc. to make sure every object is now represented in the `SZ` mode, then you can operate on the `SZ` non-spin-adapted MPS.

In the following example, we first compute the spin-adapted MPS, then translate it into the non-spin-adapted MPS to extract the determinant coefficients.

[25]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.RHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SU2, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)

ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

csfs, coeffs = driver.get_csf_coefficients(ket, cutoff=0.05, iprint=1)
zket = driver.mps_change_to_sz(ket, "ZKET")

driver.symm_type = SymmetryTypes.SZ
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)
mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)
impo = driver.get_identity_mpo()
print(driver.expectation(zket, mpo, zket) / driver.expectation(zket, impo, zket))
csfs, vals = driver.get_csf_coefficients(zket, cutoff=0.05, iprint=1)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.499 | E =    -107.6541224475 | DW = 1.87e-10

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.747 | E =    -107.6541224475 | DE = -4.09e-12 | DW = 5.36e-20

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.058 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 1.87e-10

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.439 | E =    -107.6541224475 | DE = -5.97e-13 | DW = 6.52e-20

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.891 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 1.75e-20
```

(continues on next page)

(continued from previous page)

```
Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.294 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 4.50e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.744 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 3.05e-20

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.162 | E =    -107.6541224475 | DE = 2.84e-14 | DW = 4.93e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      3.455 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 2.44e-20

DMRG energy = -107.654122447524585
mps center changed (temporarily)
dtrie finished 0.002923151000004509
Number of CSF =          9 (cutoff =      0.05)
Sum of weights of included CSF =    0.984368801370672

CSF          0 2222222000  =   -0.957506527302991
CSF          1 2222202200  =    0.131287890948123
CSF          2 2222022020  =    0.131287890439278
CSF          3 2222+-2+-0  =   -0.118594401232681
CSF          4 2222++2--0  =    0.084968039984560
CSF          5 2220222020  =    0.054710469983632
CSF          6 2220222200  =    0.054710326229694
CSF          7 2222+2+0--  =   -0.053881243065345
CSF          8 22222++-0-  =   -0.053881235680181
-107.65412244752449
mps center changed (temporarily)
dtrie finished 0.004040642000063599
Number of DET =          7 (cutoff =      0.05)
Sum of weights of included DET =    0.971331607927357

DET          0 2222222000  =   -0.957506527302991
DET          1 2222202200  =    0.131287890948123
DET          2 2222022020  =    0.131287890439278
DET          3 2222ba2ba0  =   -0.083825360995141
DET          4 2222ab2ab0  =   -0.083825360995141
DET          5 2220222020  =    0.054710469983632
DET          6 2220222200  =    0.054710326229694
```

4.1.14 Change between Real and Complex MPS

We can also change between the MPS with complex numbers and the MPS with real numbers. For complex MPS to real MPS, the imaginary part will be discarded (and the norm of the transformed MPS may decrease). This may be useful when you do a ground state calculation in the real domain and then do the real time evolution in the complex domain.

From real to complex:

[26]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.RHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SU2, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

impo = driver.get_identity_mpo()
mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)

ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

print(driver.expectation(ket, impo, ket))
print(driver.expectation(ket, mpo, ket) / driver.expectation(ket, impo, ket))
zket = driver.mps_change_complex(ket, "ZKET")

driver.symm_type = driver.symm_type ^ SymmetryTypes.CPX
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)
impo = driver.get_identity_mpo()
mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, integral_cutoff=1E-8,␣
↪iprint=1)
print(driver.expectation(zket, impo, zket))
print(driver.expectation(zket, mpo, zket) / driver.expectation(zket, impo, zket))
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.521 | E =    -107.6541224475 | DW = 1.87e-10

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.758 | E =    -107.6541224475 | DE = -1.76e-12 | DW = 6.23e-20
```

(continues on next page)

(continued from previous page)

```
Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.002 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 1.87e-10

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.260 | E =    -107.6541224475 | DE = -2.84e-13 | DW = 5.93e-20

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.552 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 1.65e-20

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.792 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 5.19e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.053 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 2.83e-20

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.301 | E =    -107.6541224475 | DE = 5.68e-14 | DW = 6.71e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      2.491 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 2.80e-20

DMRG energy = -107.654122447524671
1.0
-107.65412244752463
integral symmetrize error =  1.758383454541612e-14
integral cutoff error =  0.0
mpo terms =       1030

Build MPO | Nsites =    10 | Nterms =       1030 | Algorithm = FastBIP | Cutoff = 1.
↪00e-20
 Site =     0 /    10 .. Mmpo =    13 DW = 0.00e+00 NNZ =       13 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.006
 Site =     1 /    10 .. Mmpo =    34 DW = 0.00e+00 NNZ =       63 SPT = 0.8575 Tmvc␣
↪= 0.001 T = 0.006
 Site =     2 /    10 .. Mmpo =    56 DW = 0.00e+00 NNZ =      121 SPT = 0.9364 Tmvc␣
↪= 0.001 T = 0.005
 Site =     3 /    10 .. Mmpo =    74 DW = 0.00e+00 NNZ =      373 SPT = 0.9100 Tmvc␣
↪= 0.001 T = 0.006
```

(continues on next page)

(continued from previous page)

```
 Site =     4 /    10 .. Mmpo =    80 DW = 0.00e+00 NNZ =      269 SPT = 0.9546 Tmvc␣
↪= 0.000 T = 0.005
 Site =     5 /    10 .. Mmpo =    94 DW = 0.00e+00 NNZ =      169 SPT = 0.9775 Tmvc␣
↪= 0.000 T = 0.004
 Site =     6 /    10 .. Mmpo =    54 DW = 0.00e+00 NNZ =      181 SPT = 0.9643 Tmvc␣
↪= 0.000 T = 0.004
 Site =     7 /    10 .. Mmpo =    30 DW = 0.00e+00 NNZ =       73 SPT = 0.9549 Tmvc␣
↪= 0.000 T = 0.003
 Site =     8 /    10 .. Mmpo =    14 DW = 0.00e+00 NNZ =       41 SPT = 0.9024 Tmvc␣
↪= 0.000 T = 0.002
 Site =     9 /    10 .. Mmpo =     1 DW = 0.00e+00 NNZ =       14 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.003
Ttotal =      0.044 Tmvc-total = 0.003 MPO bond dimension =    94 MaxDW = 0.00e+00
NNZ =         1317 SIZE =        27073 SPT = 0.9514

Rank =     0 Ttotal =      0.081 MPO method = FastBipartite bond dimension =      94␣
↪NNZ =         1317 SIZE =        27073 SPT = 0.9514
(0.9999999999999999+0j)
(-107.65412244752464+0j)
```

From complex to real:

[27]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.RHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SU2 | SymmetryTypes.CPX,
↪ n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

impo = driver.get_identity_mpo()
mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)

ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

print(driver.expectation(ket, impo, ket))
print(driver.expectation(ket, mpo, ket) / driver.expectation(ket, impo, ket))
rket = driver.mps_change_complex(ket, "rket")

driver.symm_type = driver.symm_type ^ SymmetryTypes.CPX
```

(continues on next page)

(continued from previous page)

```
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)
impo = driver.get_identity_mpo()
mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, integral_cutoff=1E-8,␣
↪iprint=1)
print(driver.expectation(rket, impo, rket))
print(driver.expectation(rket, mpo, rket) / driver.expectation(rket, impo, rket))
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.593 | E =    -107.6541224475 | DW = 1.87e-10

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.095 | E =    -107.6541224475 | DE = -7.25e-12 | DW = 1.53e-19

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.625 | E =    -107.6541224475 | DE = -5.68e-14 | DW = 1.87e-10

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.922 | E =    -107.6541224475 | DE = -1.22e-12 | DW = 1.47e-19

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.467 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 1.49e-19

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.856 | E =    -107.6541224475 | DE = 5.68e-14 | DW = 5.30e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.193 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 1.30e-19

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.511 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 6.00e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      3.733 | E =    -107.6541224475 | DE = 2.84e-14 | DW = 3.78e-20

DMRG energy = -107.654122447524557
(1+3.326128828347501e-17j)
```

(continues on next page)

(continued from previous page)

```
(-107.65412244752451+5.487212746008987e-15j)
integral symmetrize error =  1.9716021141012156e-14
integral cutoff error =  0.0
mpo terms =       1030

Build MPO | Nsites =    10 | Nterms =       1030 | Algorithm = FastBIP | Cutoff = 1.
↪00e-20
 Site =     0 /    10 .. Mmpo =    13 DW = 0.00e+00 NNZ =       13 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.006
 Site =     1 /    10 .. Mmpo =    34 DW = 0.00e+00 NNZ =       63 SPT = 0.8575 Tmvc␣
↪= 0.000 T = 0.011
 Site =     2 /    10 .. Mmpo =    56 DW = 0.00e+00 NNZ =      121 SPT = 0.9364 Tmvc␣
↪= 0.001 T = 0.005
 Site =     3 /    10 .. Mmpo =    74 DW = 0.00e+00 NNZ =      373 SPT = 0.9100 Tmvc␣
↪= 0.000 T = 0.005
 Site =     4 /    10 .. Mmpo =    80 DW = 0.00e+00 NNZ =      269 SPT = 0.9546 Tmvc␣
↪= 0.000 T = 0.005
 Site =     5 /    10 .. Mmpo =    94 DW = 0.00e+00 NNZ =      169 SPT = 0.9775 Tmvc␣
↪= 0.000 T = 0.004
 Site =     6 /    10 .. Mmpo =    54 DW = 0.00e+00 NNZ =      181 SPT = 0.9643 Tmvc␣
↪= 0.000 T = 0.004
 Site =     7 /    10 .. Mmpo =    30 DW = 0.00e+00 NNZ =       73 SPT = 0.9549 Tmvc␣
↪= 0.000 T = 0.003
 Site =     8 /    10 .. Mmpo =    14 DW = 0.00e+00 NNZ =       41 SPT = 0.9024 Tmvc␣
↪= 0.000 T = 0.003
 Site =     9 /    10 .. Mmpo =     1 DW = 0.00e+00 NNZ =       14 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.002
Ttotal =      0.047 Tmvc-total = 0.003 MPO bond dimension =    94 MaxDW = 0.00e+00
NNZ =         1317 SIZE =        27073 SPT = 0.9514

Rank =     0 Ttotal =      0.079 MPO method = FastBipartite bond dimension =      94␣
↪NNZ =         1317 SIZE =        27073 SPT = 0.9514
0.8950611475659227
-107.6541224475247
```

4.1.15 MPS Bipartite Entanglement

We can get the the bipartite entanglement $S_k = -\sum_i \Lambda_k^2 \log \Lambda_k^2$ at each virtual bond (at site k) in MPS in the `SZ/SGF` mode, where Λ_k are singular values in the bond at site k.

In the `SZ` mode:

[28]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
```

(continues on next page)

(continued from previous page)

```
mf = scf.RHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SZ, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)
ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

bip_ent_sz = driver.get_bipartite_entanglement()
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.789 | E =    -107.6541224475 | DW = 4.15e-08

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.279 | E =    -107.6541224475 | DE = -4.43e-12 | DW = 5.20e-09

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.734 | E =    -107.6541224475 | DE = 1.51e-12 | DW = 4.14e-08

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.209 | E =    -107.6541224475 | DE = -1.62e-12 | DW = 5.14e-09

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.793 | E =    -107.6541224475 | DE = -1.25e-12 | DW = 3.60e-11

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.342 | E =    -107.6541224475 | DE = 9.95e-13 | DW = 2.04e-19

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.991 | E =    -107.6541224475 | DE = 2.84e-14 | DW = 3.60e-11

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
```

(continues on next page)

(continued from previous page)

```
Time elapsed =      4.931 | E =    -107.6541224475 | DE = -1.14e-12 | DW = 1.69e-19

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      5.595 | E =    -107.6541224475 | DE = 5.68e-14 | DW = 7.96e-20

DMRG energy = -107.654122447524585
```

In the SGF mode:

[29]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.GHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_ghf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SGF, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)
ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

bip_ent_sgf = driver.get_bipartite_entanglement()
```

```

Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.182 | E =    -107.6541211614 | DW = 8.43e-08

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.011 | E =    -107.6541223352 | DE = -1.17e-06 | DW = 7.39e-08

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.806 | E =    -107.6541224309 | DE = -9.56e-08 | DW = 8.43e-08

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.602 | E =    -107.6541224309 | DE = 3.58e-12 | DW = 6.84e-08

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
```

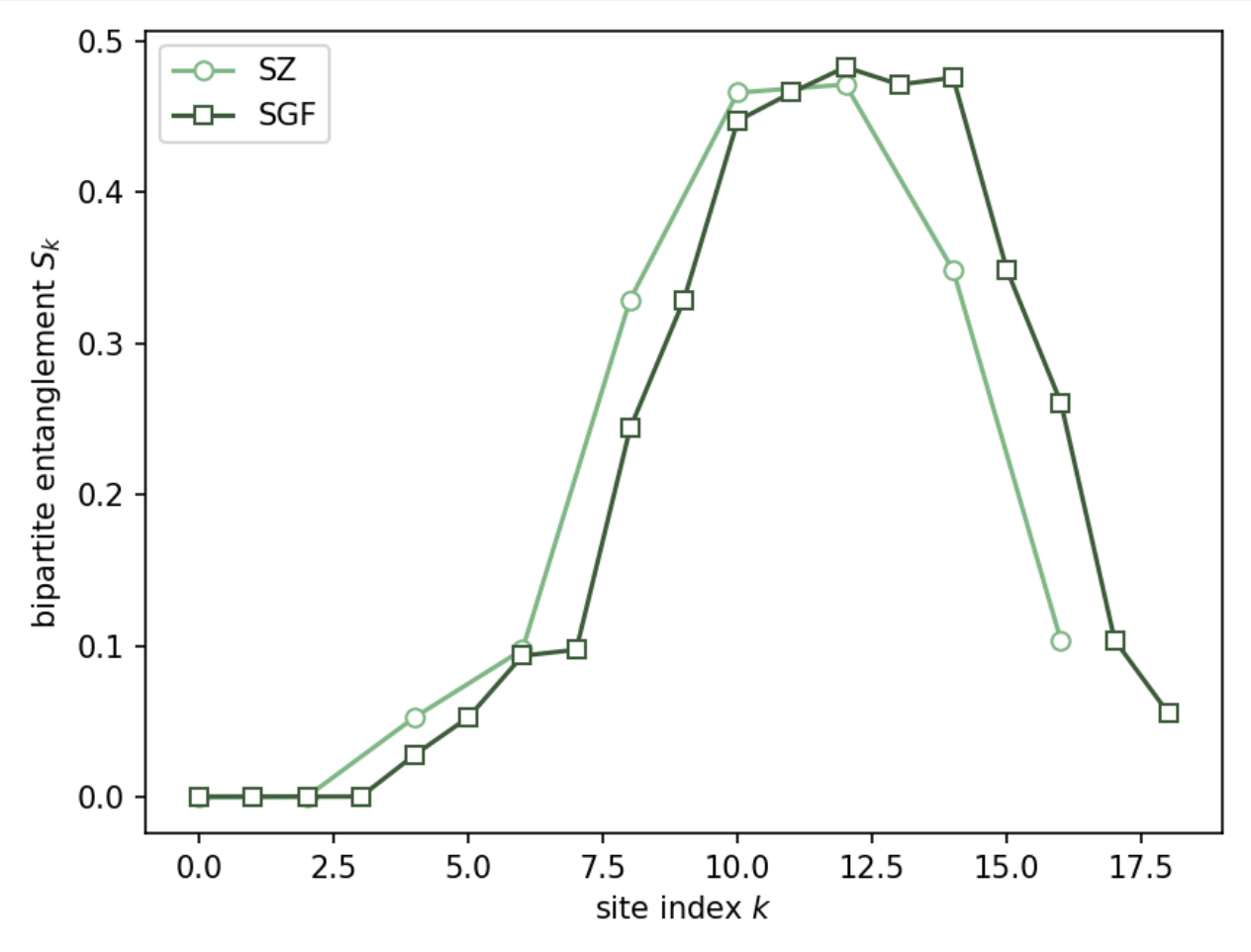
(continues on next page)

(continued from previous page)

```
↪Dav threshold =  1.00e-10
Time elapsed =      4.655 | E =    -107.6541224313 | DE = -4.10e-10 | DW = 8.63e-11

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      5.697 | E =    -107.6541224313 | DE = 1.90e-12 | DW = 8.46e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      6.823 | E =    -107.6541224313 | DE = 8.53e-14 | DW = 8.62e-11

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      8.152 | E =    -107.6541224313 | DE = -2.73e-12 | DW = 7.37e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      9.272 | E =    -107.6541224313 | DE = 0.00e+00 | DW = 7.39e-20

DMRG energy = -107.654122431266543
```

We can then plot the bipartite entanglement in MPS with spatial orbital (`SZ`) and spin orbital (`SGF`) sites:

[36]:
```
import matplotlib.pyplot as plt
plt.plot(np.arange(len(bip_ent_sz)) * 2, bip_ent_sz, linestyle='-', marker='o',
    mfc='white', mec="#7FB685", color="#7FB685", label='SZ')
plt.plot(np.arange(len(bip_ent_sgf)), bip_ent_sgf, linestyle='-', marker='s',
    mfc='white', mec="#3C5B39", color="#3C5B39", label='SGF')
plt.xlim((-1, ncas - 1))
plt.legend(loc='upper left')
plt.xlabel("site index $k$")
plt.ylabel("bipartite entanglement $S_k$")
plt.gcf().set_dpi(150)
plt.show()
```



4.1.16 Orbital Entropy and Mutual Information

For the optimized MPS in the SZ/SGF mode, we can compute the 1- and 2- orbital density matrices and mutual information for pairs of orbitals.

In the SZ mode:

[37]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.RHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SZ, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)
ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
```

(continues on next page)

(continued from previous page)

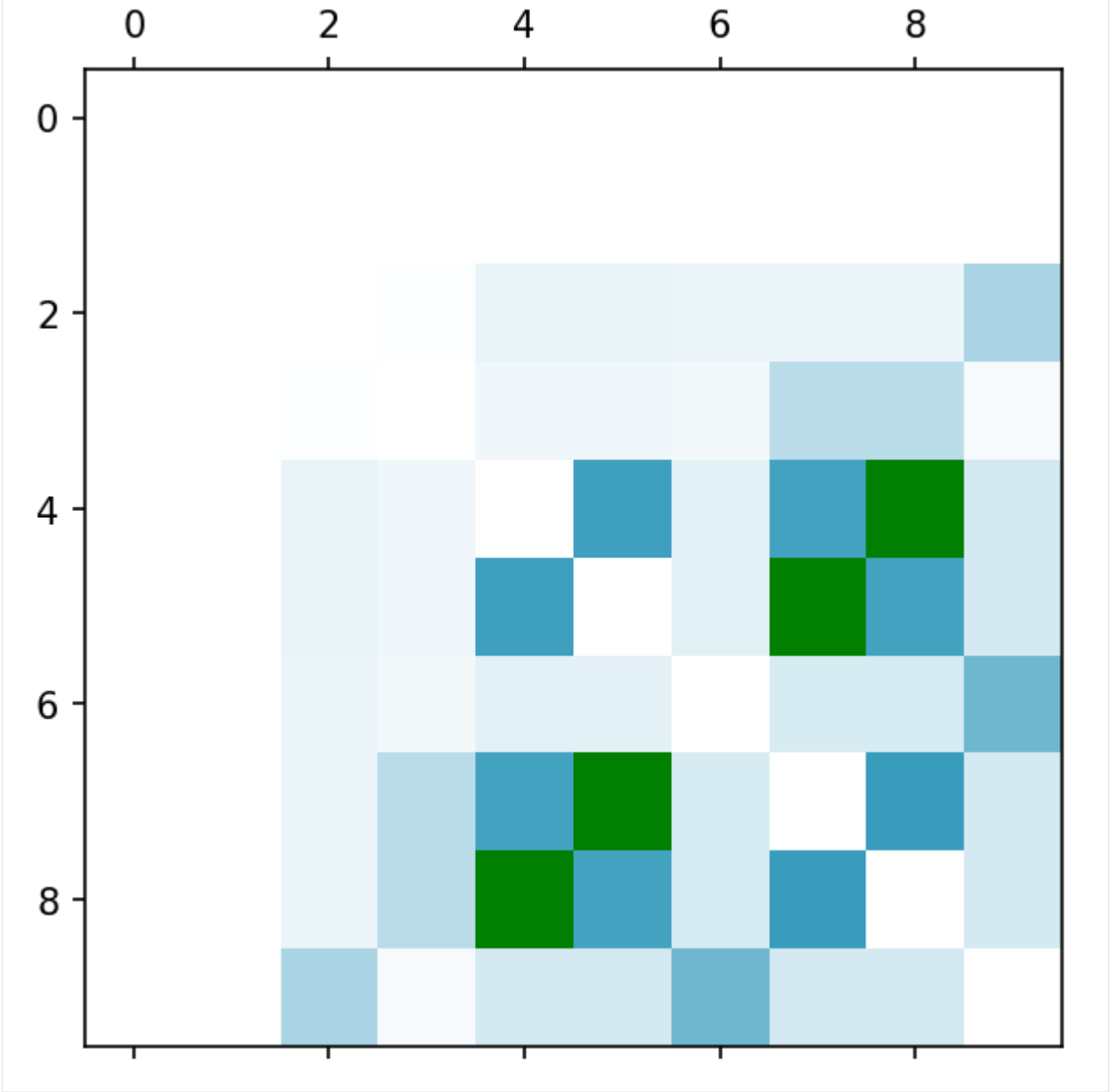
```
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

ordm1 = driver.get_orbital_entropies(ket, orb_type=1)
ordm2 = driver.get_orbital_entropies(ket, orb_type=2)
minfo = 0.5 * (ordm1[:, None] + ordm1[None, :] - ordm2) * (1 - np.
↪identity(len(ordm1)))

import matplotlib.pyplot as plt
plt.matshow(minfo, cmap='ocean_r')
plt.gcf().set_dpi(150)
plt.show()
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.404 | E =    -107.6541224475 | DW = 4.14e-08

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.188 | E =    -107.6541224475 | DE = -1.58e-11 | DW = 5.10e-09

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.906 | E =    -107.6541224475 | DE = -9.66e-13 | DW = 4.14e-08

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.666 | E =    -107.6541224475 | DE = 1.48e-12 | DW = 5.15e-09

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.563 | E =    -107.6541224475 | DE = -7.11e-13 | DW = 3.61e-11

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      5.463 | E =    -107.6541224475 | DE = 9.09e-13 | DW = 1.59e-19

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      6.332 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 3.60e-11

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      7.195 | E =    -107.6541224475 | DE = -1.68e-12 | DW = 1.90e-19
```

(continues on next page)

(continued from previous page)

```
Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      7.761 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 6.47e-20

DMRG energy = -107.654122447524557
```



In the SGF mode:

[38]:
```
from pyscf import gto, scf
```

(continues on next page)

(continued from previous page)

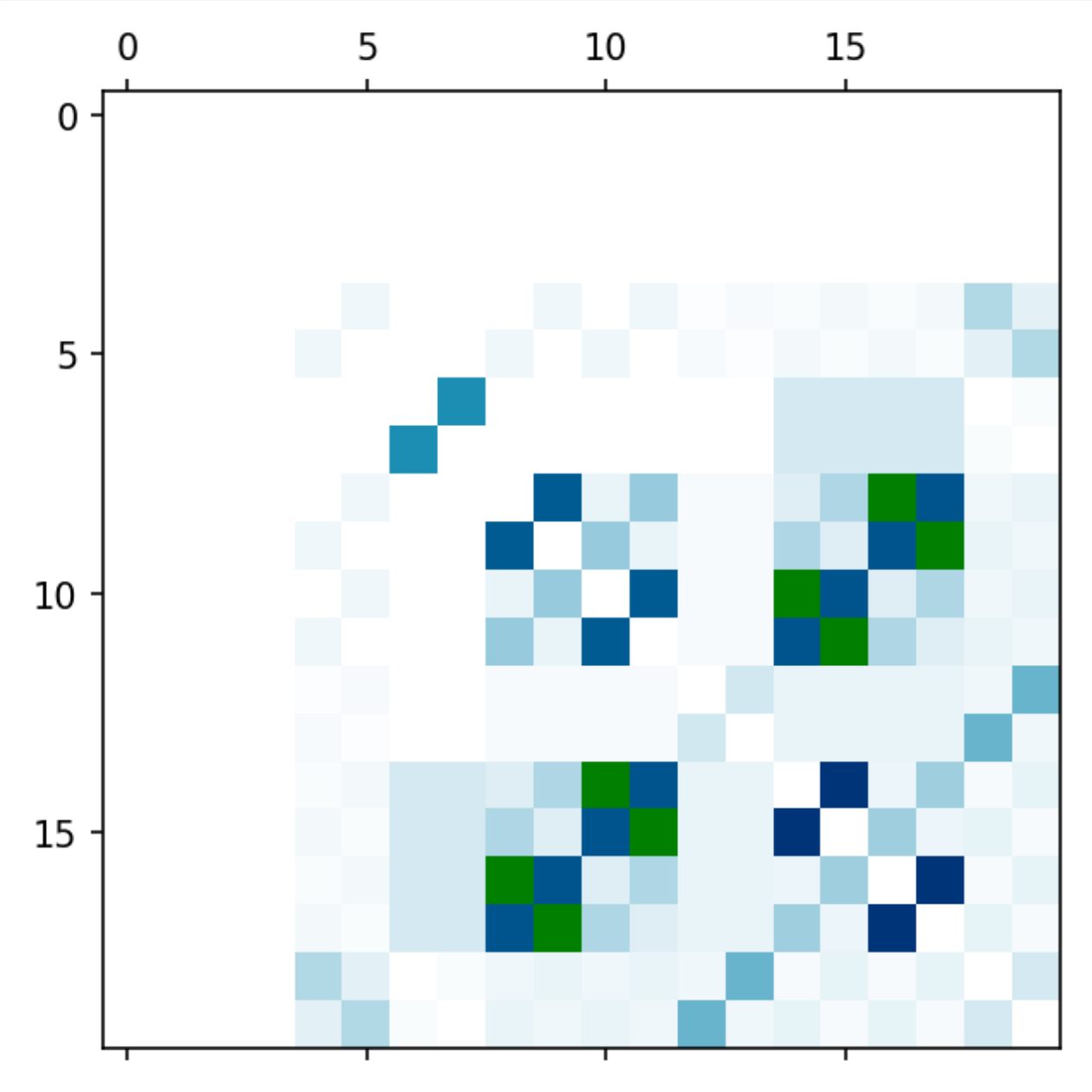
```
mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.GHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_ghf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SGF, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)
ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

ordm1 = driver.get_orbital_entropies(ket, orb_type=1)
ordm2 = driver.get_orbital_entropies(ket, orb_type=2)
minfo = 0.5 * (ordm1[:, None] + ordm1[None, :] - ordm2) * (1 - np.
↪identity(len(ordm1)))

import matplotlib.pyplot as plt
plt.matshow(minfo, cmap='ocean_r')
plt.gcf().set_dpi(150)
plt.show()
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.852 | E =    -107.6541217988 | DW = 7.63e-08

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.490 | E =    -107.6541223313 | DE = -5.33e-07 | DW = 7.48e-08

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      5.184 | E =    -107.6541224348 | DE = -1.04e-07 | DW = 7.65e-08

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      6.857 | E =    -107.6541224348 | DE = 6.82e-12 | DW = 7.21e-08

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      8.588 | E =    -107.6541224379 | DE = -3.11e-09 | DW = 8.57e-11
```

(continues on next page)

(continued from previous page)

```
Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     10.540 | E =    -107.6541224379 | DE = -9.01e-12 | DW = 6.84e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     12.280 | E =    -107.6541224379 | DE = -2.84e-14 | DW = 8.57e-11

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     13.677 | E =    -107.6541224379 | DE = 2.84e-12 | DW = 9.28e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =     14.361 | E =    -107.6541224379 | DE = -2.84e-14 | DW = 7.96e-20

DMRG energy = -107.654122437939137
```



4.1.17 Orbital Reordering

We support several algorithms (such as "fiedler" and "gaopt") to reorder the orbitals used in DMRG, so that the entanglement in the MPS can be decreased. With an optimal orbital reordering, one can expect a better energy expectation with a low bond dimension MPS.

The orbital reordering can be applied directly on the integrals `h1e`, `g2e` (unpacked), and `orb_sym` (if there are point group symmetries), before they are used by `pyblock2`. This is the recommended way, and the user is responsible for tracking the orbital ordering.

In the `SZ` mode (using the exchange integral as the cost function):

[39]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.RHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e_orig, g2e_orig, orb_sym_orig = itg.get_rhf_
↪integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SZ, n_threads=4)
g2e_orig = driver.unpack_g2e(g2e_orig, n_sites=ncas)

minfos = {}
for method in ["fiedler", "gaopt"]:
    idx = driver.orbital_reordering(h1e_orig, g2e_orig, method=method)
    h1e = h1e_orig[idx][:, idx]
    g2e = g2e_orig[idx][:, idx][:, :, idx][:, :, :, idx]
    orb_sym = np.array(orb_sym_orig)[idx]

    driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)
    mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)
    ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
    energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
        thrds=thrds, iprint=1)
    print('Method = %s DMRG energy = %20.15f' % (method, energy))
    minfos[method] = driver.get_orbital_interaction_matrix(ket)

import matplotlib.pyplot as plt
f, axs = plt.subplots(1, 2, sharey=True)
f.set_size_inches(w=14 / 2, h=7 / 2)
for ax, (method, minfo) in zip(axs, minfos.items()):
    ax.matshow(minfo, cmap='ocean_r')
    ax.set_title(method)
plt.gcf().set_dpi(150)
plt.show()
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.154 | E =    -107.6541224475 | DW = 3.94e-14

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.779 | E =    -107.6541224475 | DE = -1.71e-13 | DW = 5.09e-14

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.519 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 3.53e-14
```

(continues on next page)

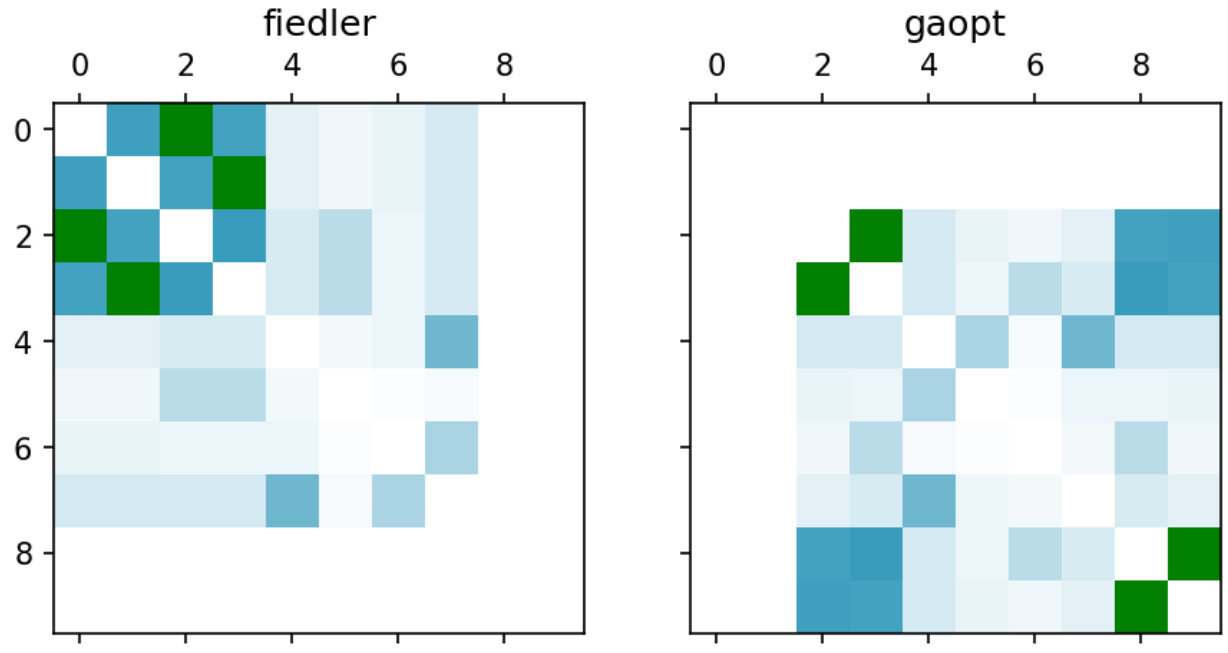
(continued from previous page)

```
Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.267 | E =    -107.6541224475 | DE = -1.71e-13 | DW = 5.99e-14

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.325 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 1.60e-19

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      5.298 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 9.10e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      6.249 | E =    -107.6541224475 | DE = 1.42e-13 | DW = 1.68e-19

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      7.029 | E =    -107.6541224475 | DE = 5.68e-14 | DW = 8.36e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      7.552 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 2.65e-20

Method = fiedler DMRG energy = -107.654122447502473

Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.165 | E =    -107.6541224475 | DW = 4.14e-10

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.551 | E =    -107.6541224475 | DE = -1.21e-11 | DW = 4.20e-11

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.929 | E =    -107.6541224475 | DE = -2.30e-12 | DW = 4.13e-10

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.314 | E =    -107.6541224475 | DE = -2.27e-13 | DW = 4.41e-11

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
```

(continues on next page)

(continued from previous page)

```
Time elapsed =      2.783 | E =    -107.6541224475 | DE = 5.68e-14 | DW = 8.96e-21

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.298 | E =    -107.6541224475 | DE = -1.14e-13 | DW = 1.20e-14

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.812 | E =    -107.6541224475 | DE = 5.68e-14 | DW = 1.14e-20

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.337 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 1.21e-14

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      4.638 | E =    -107.6541224475 | DE = -1.42e-13 | DW = 4.79e-20

Method = gaopt DMRG energy = -107.654122447524131
```



In the SGF mode (using the exchange integral as the cost function):

[40]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.GHF(mol).run(conv_tol=1E-14)
```

(continues on next page)

(continued from previous page)

```
ncas, n_elec, spin, ecore, h1e_orig, g2e_orig, orb_sym_orig = itg.get_ghf_
↪integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SGF, n_threads=4)
g2e_orig = driver.unpack_g2e(g2e_orig, n_sites=ncas)

minfos = {}
for method in ["fiedler", "gaopt"]:
    idx = driver.orbital_reordering(h1e_orig, g2e_orig, method=method)
    h1e = h1e_orig[idx][:, idx]
    g2e = g2e_orig[idx][:, idx][:, :, idx][:, :, :, idx]
    orb_sym = np.array(orb_sym_orig)[idx]

    driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)
    mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)
    ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
    energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
        thrds=thrds, iprint=1)
    print('Method = %s DMRG energy = %20.15f' % (method, energy))
    minfos[method] = driver.get_orbital_interaction_matrix(ket)

import matplotlib.pyplot as plt
f, axs = plt.subplots(1, 2, sharey=True)
f.set_size_inches(w=14 / 2, h=7 / 2)
for ax, (method, minfo) in zip(axs, minfos.items()):
    ax.matshow(minfo, cmap='ocean_r')
    ax.set_title(method)
plt.gcf().set_dpi(150)
plt.show()
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.089 | E =    -107.6421167128 | DW = 1.94e-08

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      6.269 | E =    -107.6541224404 | DE = -1.20e-02 | DW = 2.02e-08

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      7.843 | E =    -107.6541224449 | DE = -4.57e-09 | DW = 2.63e-08

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
```

(continues on next page)

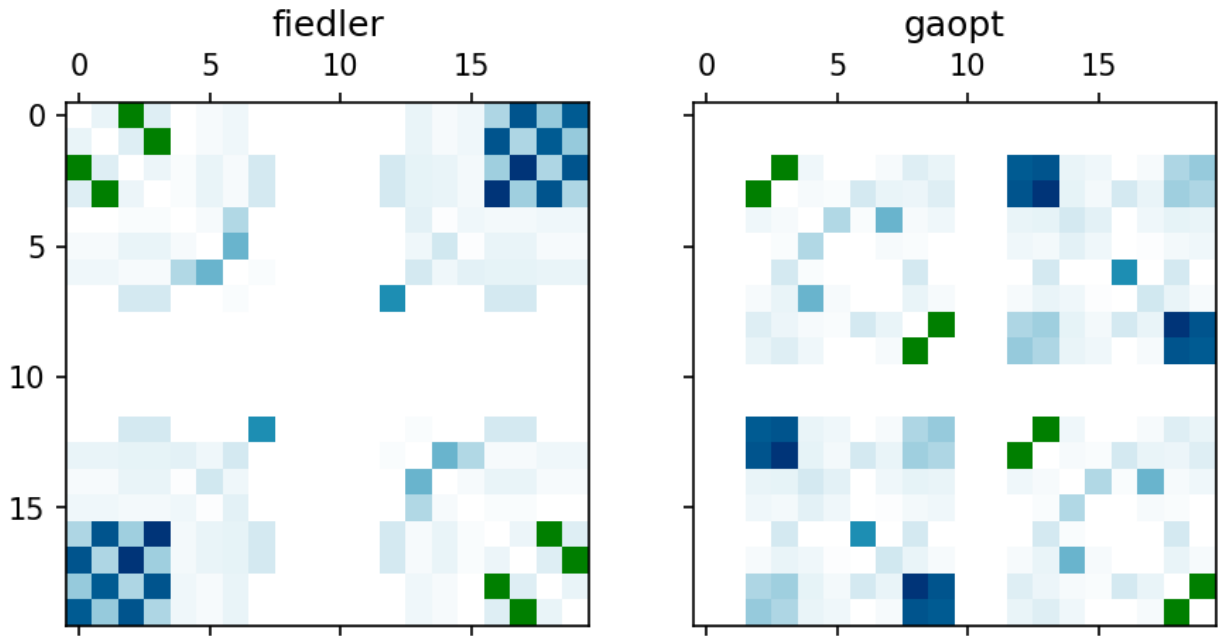
(continued from previous page)

```
Time elapsed =      9.347 | E =    -107.6541224419 | DE = 3.04e-09 | DW = 2.48e-08

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     11.393 | E =    -107.6541224475 | DE = -5.62e-09 | DW = 2.71e-11

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     13.951 | E =    -107.6541224475 | DE = -9.66e-13 | DW = 2.95e-11

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     16.841 | E =    -107.6541224475 | DE = -3.13e-13 | DW = 2.74e-11

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     18.893 | E =    -107.6541224475 | DE = 1.99e-13 | DW = 3.67e-11

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =     19.757 | E =    -107.6541224475 | DE = 7.11e-12 | DW = 3.64e-20

Method = fiedler DMRG energy = -107.654122447517537

Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.646 | E =    -107.6540666856 | DW = 5.18e-08

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.397 | E =    -107.6541222892 | DE = -5.56e-05 | DW = 4.78e-08

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      5.321 | E =    -107.6541224442 | DE = -1.55e-07 | DW = 5.35e-08

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      6.216 | E =    -107.6541224442 | DE = -8.87e-12 | DW = 4.55e-08

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      7.531 | E =    -107.6541224475 | DE = -3.35e-09 | DW = 1.02e-10

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
```

(continues on next page)

(continued from previous page)

```
↪Dav threshold =  1.00e-10
Time elapsed =      8.955 | E =    -107.6541224475 | DE = -6.82e-13 | DW = 2.97e-11

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     10.368 | E =    -107.6541224475 | DE = 4.55e-13 | DW = 1.01e-10

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     11.805 | E =    -107.6541224475 | DE = -7.67e-13 | DW = 2.97e-11

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =     12.680 | E =    -107.6541224475 | DE = 3.69e-13 | DW = 2.59e-20

Method = gaopt DMRG energy = -107.654122447524415
```



The orbital reordering can also be done implicitly using the `reorder` argument in `DMRGDriver.get_qc_mpo`. `pyblock2` will automatically recover the original ordering for observables, whenever possible, such as N-PDMs. But ordering for quantities like bipartite/orbital entropy are not recovered (kept as the computational ordering since they are indicators of the orbital topology). In order to avoid any confusion, this implicit reordering approach is not recommended unless only the ground state energy and N-PDMs are needed.

Implicit orbital reordering using `gaopt` in the `SZ` mode:

[41]:
```
from pyscf import gto, scf
```

(continues on next page)

(continued from previous page)

```
mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.RHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SZ, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)
mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, reorder="gaopt", iprint=0)
ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.869 | E =    -107.6541224475 | DW = 4.11e-10

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.480 | E =    -107.6541224475 | DE = -1.18e-11 | DW = 4.28e-11

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.116 | E =    -107.6541224475 | DE = -2.10e-12 | DW = 4.13e-10

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.707 | E =    -107.6541224475 | DE = -1.42e-13 | DW = 4.76e-11

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.456 | E =    -107.6541224475 | DE = -8.53e-14 | DW = 1.46e-20

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.258 | E =    -107.6541224475 | DE = 5.68e-14 | DW = 1.22e-14

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      5.046 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 2.52e-20

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      6.088 | E =    -107.6541224475 | DE = -5.68e-14 | DW = 1.21e-14
```

(continues on next page)

(continued from previous page)

```
Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      6.794 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 4.16e-20

DMRG energy = -107.654122447523960
```

One can also run a small bond dimension DMRG with the default orbital ordering to get the mutual information for orbital correlation, then use this matrix as the cost function to get the orbital ordering for doing more efficient DMRG (this is supported in the SZ and SGF modes). See the following example.

In the SGF mode (using the mutual information as the cost function):

[43]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.GHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e_orig, g2e_orig, orb_sym_orig = itg.get_ghf_
↪integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SGF, n_threads=4)
g2e_orig = driver.unpack_g2e(g2e_orig, n_sites=ncas)

# approx DMRG to get orbital_interaction_matrix
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym_
↪orig)
mpo = driver.get_qc_mpo(h1e=h1e_orig, g2e=g2e_orig, ecore=ecore, iprint=0)
ket = driver.get_random_mps(tag="GS", bond_dim=50, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=10, bond_dims=[50] * 8, noises=noises,
    thrds=thrds, iprint=1)
print('Approx DMRG energy = %20.15f' % energy)
minfo_orig = driver.get_orbital_interaction_matrix(ket)

minfos = {}
for method in ["fiedler", "gaopt"]:
    idx = driver.orbital_reordering_interaction_matrix(minfo_orig, method=method)
    h1e = h1e_orig[idx][:, idx]
    g2e = g2e_orig[idx][:, idx][:, :, idx][:, :, :, idx]
    orb_sym = np.array(orb_sym_orig)[idx]

    driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)
    mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)
    ket = driver.get_random_mps(tag="GS", bond_dim=250, nroots=1)
    energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
```

(continues on next page)

(continued from previous page)

```
        thrds=thrds, iprint=1)
    print('Method = %s DMRG energy = %20.15f' % (method, energy))
    minfos[method] = driver.get_orbital_interaction_matrix(ket)

import matplotlib.pyplot as plt
f, axs = plt.subplots(1, 2, sharey=True)
f.set_size_inches(w=14 / 2, h=7 / 2)
for ax, (method, minfo) in zip(axs, minfos.items()):
    ax.matshow(minfo, cmap='ocean_r')
    ax.set_title(method)
plt.gcf().set_dpi(150)
plt.show()
```

```
Sweep =    0 | Direction =  forward | Bond dimension =   50 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.790 | E =    -107.6461019596 | DW = 1.12e-04

Sweep =    1 | Direction = backward | Bond dimension =   50 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.199 | E =    -107.6518646319 | DE = -5.76e-03 | DW = 8.45e-05

Sweep =    2 | Direction =  forward | Bond dimension =   50 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.658 | E =    -107.6524131412 | DE = -5.49e-04 | DW = 4.76e-05

Sweep =    3 | Direction = backward | Bond dimension =   50 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.123 | E =    -107.6524673390 | DE = -5.42e-05 | DW = 4.58e-05

Sweep =    4 | Direction =  forward | Bond dimension =   50 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.589 | E =    -107.6525301747 | DE = -6.28e-05 | DW = 3.18e-05

Sweep =    5 | Direction = backward | Bond dimension =   50 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.076 | E =    -107.6525427750 | DE = -1.26e-05 | DW = 3.17e-05

Sweep =    6 | Direction =  forward | Bond dimension =   50 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.514 | E =    -107.6525457470 | DE = -2.97e-06 | DW = 3.25e-05

Sweep =    7 | Direction = backward | Bond dimension =   50 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.945 | E =    -107.6525492373 | DE = -3.49e-06 | DW = 3.25e-05
```

(continues on next page)

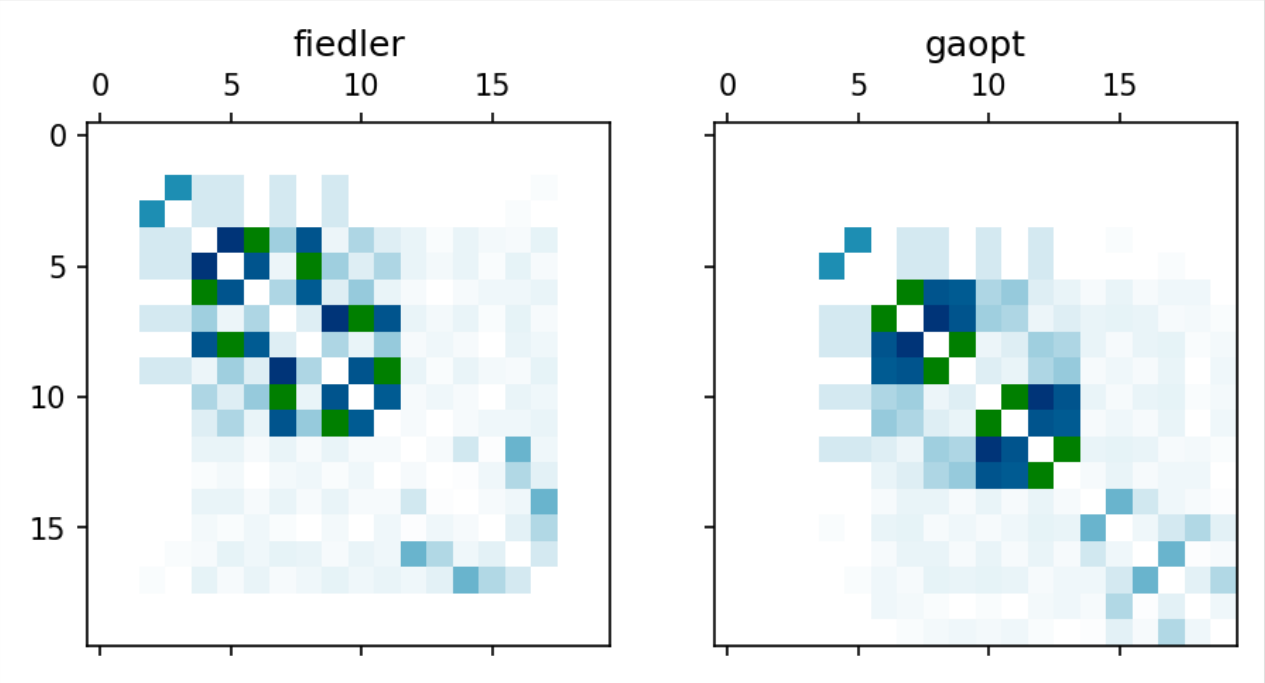
(continued from previous page)

```
Sweep =    8 | Direction =  forward | Bond dimension =   50 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      4.315 | E =    -107.6525424769 | DE = 6.76e-06 | DW = 2.77e-05

Sweep =    9 | Direction = backward | Bond dimension =   50 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      4.642 | E =    -107.6525448992 | DE = -2.42e-06 | DW = 2.79e-05

ATTENTION: DMRG is not converged to desired tolerance of 1.00e-08
Approx DMRG energy = -107.652544899236361

Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.878 | E =    -107.6541143612 | DW = 4.90e-10

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.658 | E =    -107.6541224474 | DE = -8.09e-06 | DW = 2.78e-08

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.499 | E =    -107.6541224475 | DE = -1.10e-10 | DW = 4.87e-10

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.253 | E =    -107.6541224475 | DE = 1.76e-12 | DW = 2.78e-08

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      5.260 | E =    -107.6541224475 | DE = -3.67e-12 | DW = 7.63e-20

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      6.300 | E =    -107.6541224475 | DE = -5.68e-14 | DW = 9.52e-12

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      7.397 | E =    -107.6541224475 | DE = 1.42e-12 | DW = 8.05e-20

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      8.455 | E =    -107.6541224475 | DE = 1.14e-13 | DW = 9.44e-12

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
```

(continues on next page)

(continued from previous page)

```
Time elapsed =      9.077 | E =    -107.6541224475 | DE = 3.27e-12 | DW = 2.36e-20

Method = fiedler DMRG energy = -107.654122447518702

Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.205 | E =    -107.6541223034 | DW = 1.54e-09

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.032 | E =    -107.6541224434 | DE = -1.40e-07 | DW = 2.01e-09

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.921 | E =    -107.6541224475 | DE = -4.10e-09 | DW = 1.56e-09

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.728 | E =    -107.6541224475 | DE = 2.64e-12 | DW = 2.01e-09

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.899 | E =    -107.6541224475 | DE = -5.54e-12 | DW = 1.05e-13

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      6.131 | E =    -107.6541224475 | DE = 2.84e-14 | DW = 3.63e-14

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      8.302 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 1.05e-13

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     11.248 | E =    -107.6541224475 | DE = -5.68e-14 | DW = 3.58e-14

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =     12.502 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 3.66e-20

Method = gaopt DMRG energy = -107.654122447523818
```



4.1.18 Excited States

To obtain the excited states and their energies, we can perform DMRG for a state-averged MPS, optionally followed by a state-specific refinement.

[44]:
```
from pyscf import gto, scf

mol = gto.M(atom="N 0 0 0; N 0 0 1.1", basis="sto3g", symmetry="d2h", verbose=0)
mf = scf.RHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf,
    ncore=0, ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SZ, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)

ket = driver.get_random_mps(tag="KET", bond_dim=100, nroots=3)
energies = driver.dmrg(mpo, ket, n_sweeps=10, bond_dims=[100], noises=[1e-5] * 4 +␣
↪[0],
    thrds=[1e-10] * 8, iprint=1)
print('State-averaged MPS energies = [%s]' % " ".join("%20.15f" % x for x in␣
↪energies))

kets = [driver.split_mps(ket, ir, tag="KET-%d" % ir) for ir in range(ket.nroots)]
for ir in range(ket.nroots):
```

(continues on next page)

(continued from previous page)

```
    energy = driver.dmrg(mpo, kets[ir], n_sweeps=10, bond_dims=[200], noises=[1e-5]␣
↪* 4 + [0],
        thrds=[1e-10] * 8, iprint=0, proj_weights=[5.0] * ir, proj_mpss=kets[:ir])
    print('State-specific MPS E[%d] = %20.15f' % (ir, energy))
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  100 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.873 | E[  3] =    -107.6541184424   -107.0314355209   -106.
↪9595839138 | DW = 8.49e-05

Sweep =    1 | Direction = backward | Bond dimension =  100 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.807 | E[  3] =    -107.6541184424   -107.0314355209   -106.
↪9595839138 | DE = -1.51e-11 | DW = 1.61e-04

Sweep =    2 | Direction =  forward | Bond dimension =  100 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.640 | E[  3] =    -107.6541138875   -107.0314486179   -106.
↪9594564029 | DE = 1.28e-04 | DW = 7.20e-05

Sweep =    3 | Direction = backward | Bond dimension =  100 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      4.520 | E[  3] =    -107.6541138875   -107.0314486178   -106.
↪9594564029 | DE = -9.18e-12 | DW = 1.47e-04

Sweep =    4 | Direction =  forward | Bond dimension =  100 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      5.353 | E[  3] =    -107.6540972369   -107.0314487272   -106.
↪9594391430 | DE = 1.73e-05 | DW = 6.81e-05

Sweep =    5 | Direction = backward | Bond dimension =  100 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      6.315 | E[  3] =    -107.6540972369   -107.0314487272   -106.
↪9594391430 | DE = -1.78e-11 | DW = 1.43e-04

State-averaged MPS energies = [-107.654097236858902 -107.031448727151187 -106.
↪959439143016382]
State-specific MPS E[0] = -107.654122447521317
State-specific MPS E[1] = -107.031449471624697
State-specific MPS E[2] = -106.959626154671795
```

4.2 Energy Extrapolation

[1]:
```
!pip install block2==0.5.2rc13 -qq --progress-bar off --extra-index-url=https://
↪block-hczhai.github.io/block2-preview/pypi/
!pip install pyscf==2.3.0 -qq --progress-bar off
```

4.2.1 Introduction

In this tutorial we explain how to do DMRG energy extrapolation and also get an estimate of the error in the extrapolated energy. First we need to do a normal DMRG calculation, and then a DMRG with a reserve schedule, and then we do the energy extrapolation using the data from the reserve schedule. The DMRG energy is extrapolated using a linear fit for discarded weight vs energies for different bond dimensions. The error is estimated as one fifth of the extrapolation distance, as a convention. It is not recommended to use the normal DMRG data for the extrapolation because the energy for each bond dimension may not be fully converged. For the same reason, the energy at the largest bond dimension may also be excluded from the extrapolation.

First, we prepare the integrals and the MPO and the initial guess for the MPS.

[2]:
```
import numpy as np
from pyblock2._pyscf.ao2mo import integrals as itg
from pyblock2.driver.core import DMRGDriver, SymmetryTypes

from pyscf import gto, scf

mol = gto.M(atom='C 0 0 0; C 0 0 1.2425', basis='ccpvdz', symmetry='d2h')
mf = scf.RHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf,
    ncore=2, ncas=26, g2e_symm=8)

print("NCAS = %d NCASELEC = %d" % (ncas, n_elec))
driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SU2,
                    stack_mem=4 << 30, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)
mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)
ket = driver.get_random_mps(tag="KET", bond_dim=250, nroots=1)
```

```
converged SCF energy = -75.3869023777059
NCAS = 26 NCASELEC = 8
```

4.2.2 The Normal Schedule

First, we do a DMRG with the normal (forward) schedule (namely, the bond dimension increses). We first do 20 2-site sweeps, and then 8 1-site sweeps when it is close to convergence, which is cheaper.

To prevent optimizing to local minima, it is highly recommended to set the first bond dimension in the forward to be at least 250. If the gap of the system is small or the truncation error is large (for example, for 2D Hubbard model), one may need to do more sweeps for each bond dimension. Sometimes state averaging over multiple states (for example, setting `nroots=4` in `driver.get_random_mps` above) can help converging the ground state.

[3]:
```
bond_dims = [250] * 4 + [500] * 4 + [750] * 4 + [1000] * 4
noises = [1e-4] * 4 + [1e-5] * 12 + [0]
thrds = [1e-8] * 20

energy = driver.dmrg(mpo, ket, n_sweeps=28, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1, twosite_to_onesite=20)
print('DMRG energy (variational) = %20.15f' % energy)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-08
Time elapsed =     48.206 | E =     -75.6988045936 | DW = 4.92e-07

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-08
Time elapsed =     61.355 | E =     -75.7234052111 | DE = -2.46e-02 | DW = 4.32e-05

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-08
Time elapsed =     74.078 | E =     -75.7249238917 | DE = -1.52e-03 | DW = 6.80e-05

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-08
Time elapsed =     87.346 | E =     -75.7254307257 | DE = -5.07e-04 | DW = 6.87e-05

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
Time elapsed =    110.212 | E =     -75.7273558056 | DE = -1.93e-03 | DW = 3.97e-06

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
Time elapsed =    142.710 | E =     -75.7278860763 | DE = -5.30e-04 | DW = 1.58e-05

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
Time elapsed =    173.278 | E =     -75.7278862359 | DE = -1.60e-07 | DW = 1.67e-05
```

(continues on next page)

(continued from previous page)

```
Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
Time elapsed =    203.646 | E =     -75.7279905032 | DE = -1.04e-04 | DW = 1.65e-05

Sweep =    8 | Direction =  forward | Bond dimension =  750 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
Time elapsed =    249.619 | E =     -75.7282187494 | DE = -2.28e-04 | DW = 1.40e-06

Sweep =    9 | Direction = backward | Bond dimension =  750 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
Time elapsed =    308.157 | E =     -75.7283283738 | DE = -1.10e-04 | DW = 2.39e-06

Sweep =   10 | Direction =  forward | Bond dimension =  750 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
Time elapsed =    368.993 | E =     -75.7283309384 | DE = -2.56e-06 | DW = 3.28e-06

Sweep =   11 | Direction = backward | Bond dimension =  750 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
Time elapsed =    426.670 | E =     -75.7283513281 | DE = -2.04e-05 | DW = 2.59e-06

Sweep =   12 | Direction =  forward | Bond dimension = 1000 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
Time elapsed =    502.719 | E =     -75.7284190450 | DE = -6.77e-05 | DW = 1.18e-06

Sweep =   13 | Direction = backward | Bond dimension = 1000 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
Time elapsed =    592.335 | E =     -75.7284581391 | DE = -3.91e-05 | DW = 1.11e-06

Sweep =   14 | Direction =  forward | Bond dimension = 1000 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
Time elapsed =    681.041 | E =     -75.7284587601 | DE = -6.21e-07 | DW = 1.80e-06

Sweep =   15 | Direction = backward | Bond dimension = 1000 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
Time elapsed =    769.707 | E =     -75.7284677459 | DE = -8.99e-06 | DW = 1.16e-06

Sweep =   16 | Direction =  forward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-08
Time elapsed =    854.210 | E =     -75.7284682379 | DE = -4.92e-07 | DW = 1.14e-06

Sweep =   17 | Direction = backward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-08
Time elapsed =    934.636 | E =     -75.7284724835 | DE = -4.25e-06 | DW = 1.16e-06
```

(continues on next page)

(continued from previous page)

```
Sweep =   18 | Direction =  forward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-08
Time elapsed =   1019.953 | E =     -75.7284724841 | DE = -6.41e-10 | DW = 1.16e-06

Sweep =   19 | Direction = backward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-08
Time elapsed =   1099.196 | E =     -75.7284733044 | DE = -8.20e-07 | DW = 1.17e-06


Sweep =   20 | Direction =  forward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =     44.519 | E =     -75.7284670047 | DE = 6.30e-06 | DW = 4.75e-17

Sweep =   21 | Direction = backward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =    113.701 | E =     -75.7284673359 | DE = -3.31e-07 | DW = 6.22e-19

Sweep =   22 | Direction =  forward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =    157.729 | E =     -75.7284677042 | DE = -3.68e-07 | DW = 4.78e-17

Sweep =   23 | Direction = backward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =    224.673 | E =     -75.7284679694 | DE = -2.65e-07 | DW = 5.68e-19

Sweep =   24 | Direction =  forward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =    268.982 | E =     -75.7284681140 | DE = -1.45e-07 | DW = 5.17e-17

Sweep =   25 | Direction = backward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =    334.809 | E =     -75.7284681905 | DE = -7.64e-08 | DW = 5.74e-19

Sweep =   26 | Direction =  forward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =    378.814 | E =     -75.7284682377 | DE = -4.72e-08 | DW = 1.32e-17

Sweep =   27 | Direction = backward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =    444.799 | E =     -75.7284682709 | DE = -3.32e-08 | DW = 6.42e-19

ATTENTION: DMRG is not converged to desired tolerance of 1.00e-08
DMRG energy (variational) =  -75.728468270883781
```

4.2.3 The Reverse Schedule

For the reverse schedule, we decrease the bond dimension to make the energy at each bond dimension fully converged. The noise should be zero. The MPS is first changed from the 1-site format to the 2-site format. And the reverse schedule has to be done fully using the 2-site DMRG (since 1-site DMRG with zero noise will generate close-to-zero discarded weights). The energy convergence tolerance is set to zero to prevent early stop.

Note that the reverse schedule will destroy the data in the optimized MPS. We can first backup the MPS by copying it to another tag `KET-ORIG`. If you want to later compute the properties on the optimized MPS, the `ket_orig` MPS should be used. If you want to restart a calculation from the MPS (automatically) stored in the scratch foloder, you can load it using its tag (for example, using `ket = driver.load_mps(tag='KET-ORIG')`).

In practice, it is recommended to perform an even number of sweeps for each reverse schedule bond dimension, to prevent DW difference for odd/even sweeps. The first reverse schedule bond dimension should be slightly smaller than the last forward schedule bond dimension, to make sure that the energy obatined in reverse schedule is fully converged. One can also discard the energy from the first bond dimension to ensure this. The last bond dimension in reverse schedule should not be too small, since it is hard to obtain a stable energy of a very low bond dimension MPS.

[4]:
```
bond_dims = [800] * 4 + [700] * 4 + [600] * 4 + [500] * 4
noises = [0] * 16
thrds = [1e-10] * 16

ket_orig = driver.copy_mps(ket, tag='KET-ORIG')
ket = driver.adjust_mps(ket, dot=2)[0]
energy = driver.dmrg(mpo, ket, n_sweeps=16, bond_dims=bond_dims, noises=noises,
                     tol=0, thrds=thrds, iprint=1)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  800 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     75.765 | E =     -75.7284682749 | DW = 6.89e-06

Sweep =    1 | Direction = backward | Bond dimension =  800 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    138.033 | E =     -75.7284033759 | DE = 6.49e-05 | DW = 2.47e-06

Sweep =    2 | Direction =  forward | Bond dimension =  800 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    205.532 | E =     -75.7284034664 | DE = -9.04e-08 | DW = 2.42e-06

Sweep =    3 | Direction = backward | Bond dimension =  800 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    310.035 | E =     -75.7284040381 | DE = -5.72e-07 | DW = 2.40e-06

Sweep =    4 | Direction =  forward | Bond dimension =  700 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
```

(continues on next page)

(continued from previous page)

```
Time elapsed =    388.303 | E =     -75.7283965259 | DE = 7.51e-06 | DW = 8.05e-06

Sweep =    5 | Direction = backward | Bond dimension =  700 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    435.992 | E =     -75.7283294371 | DE = 6.71e-05 | DW = 3.42e-06

Sweep =    6 | Direction =  forward | Bond dimension =  700 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    483.591 | E =     -75.7283294384 | DE = -1.30e-09 | DW = 3.42e-06

Sweep =    7 | Direction = backward | Bond dimension =  700 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    531.150 | E =     -75.7283301970 | DE = -7.59e-07 | DW = 3.40e-06

Sweep =    8 | Direction =  forward | Bond dimension =  600 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    574.284 | E =     -75.7283301973 | DE = -2.32e-10 | DW = 1.29e-05

Sweep =    9 | Direction = backward | Bond dimension =  600 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    610.012 | E =     -75.7282237067 | DE = 1.06e-04 | DW = 8.58e-06

Sweep =   10 | Direction =  forward | Bond dimension =  600 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    646.755 | E =     -75.7282237089 | DE = -2.27e-09 | DW = 8.58e-06

Sweep =   11 | Direction = backward | Bond dimension =  600 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    683.397 | E =     -75.7282244369 | DE = -7.28e-07 | DW = 8.41e-06

Sweep =   12 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    714.398 | E =     -75.7282244373 | DE = -3.28e-10 | DW = 2.34e-05

Sweep =   13 | Direction = backward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    741.278 | E =     -75.7280407981 | DE = 1.84e-04 | DW = 1.76e-05

Sweep =   14 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    768.488 | E =     -75.7280408085 | DE = -1.04e-08 | DW = 1.76e-05

Sweep =   15 | Direction = backward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    795.707 | E =     -75.7280421420 | DE = -1.33e-06 | DW = 1.74e-05
```

4.2.4 Energy Extrapolation

We can get the extrapolated energy using linear fitting.

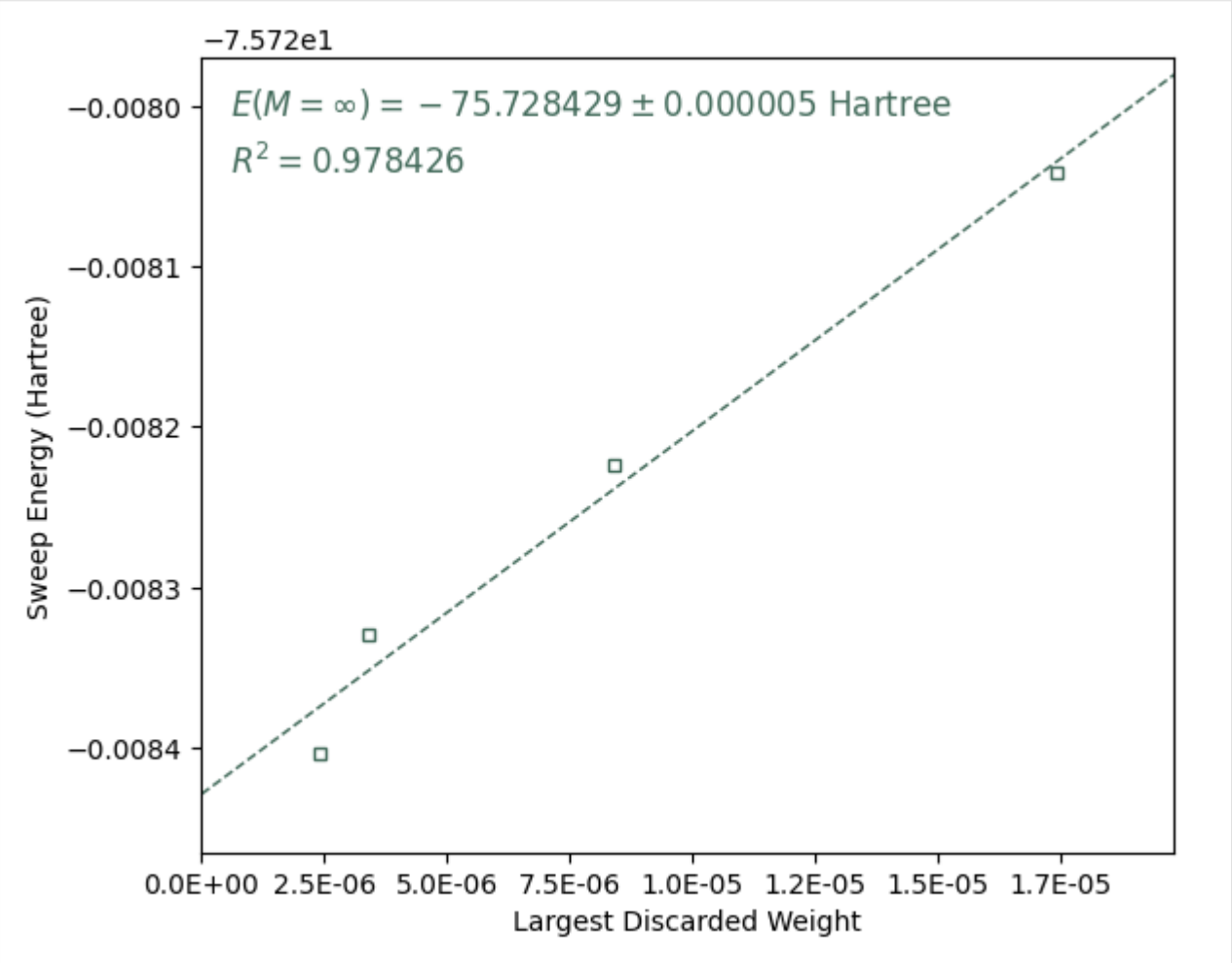
[5]:
```
import scipy.stats

ds, dws, eners = driver.get_dmrg_results()
print('BOND DIMS         = ', ds[3::4])
print('Discarded Weights = ', dws[3::4])
print('Energies          = ', eners[3::4, 0])
reg = scipy.stats.linregress(dws[3::4], eners[3::4, 0])
emin, emax = min(eners[3::4, 0]), max(eners[3::4, 0])
print('DMRG energy (extrapolated) = %20.15f +/- %15.10f' %
      (reg.intercept, abs(reg.intercept - emin) / 5))
```

```
BOND DIMS         =  [800 700 600 500]
Discarded Weights =  [2.39910258e-06 3.39687452e-06 8.41354132e-06 1.74228067e-05]
Energies          =  [-75.72840404 -75.7283302  -75.72822444 -75.72804214]
DMRG energy (extrapolated) =  -75.728429118584003 +/-    0.0000050161
```

Finally, we plot the energy extrapolation.

If you see a non-linear behavior, for example, the actual energy at larger bond dimension is higher than the linear fitting, it can be a signal of local minima. One may need to use increase the largest bond dimension used in the forward sweep, change the orbital ordering, or use a stricter Davidson convergence threshold (such as 1E-10).

[6]:
```
import matplotlib.pyplot as plt
from matplotlib import ticker
de = emax - emin
x_reg = np.array([0, dws[-1] + dws[3]])
ax = plt.gca()
ax.xaxis.set_major_formatter(ticker.StrMethodFormatter("{x:.1E}"))
plt.plot(x_reg, reg.intercept + reg.slope * x_reg, '--', linewidth=1, color='#426A5A
↪')
plt.plot(dws[3::4], eners[3::4, 0], ' ', marker='s', mfc='white', mec='#426A5A',␣
↪color='#426A5A', markersize=5)
plt.text(dws[3] * 0.25, emax + de * 0.1, "$E(M=\\infty) = %.6f \\pm %.6f \\mathrm{\\␣
↪Hartree}$" %
    (reg.intercept, abs(reg.intercept - emin) / 5), color='#426A5A', fontsize=12)
plt.text(dws[3] * 0.25, emax - de * 0.0, "$R^2 = %.6f$" % (reg.rvalue ** 2),
    color='#426A5A', fontsize=12)
plt.xlim((0, dws[-1] + dws[3]))
plt.ylim((reg.intercept - de * 0.1, emax + de * 0.2))
plt.xlabel("Largest Discarded Weight")
plt.ylabel("Sweep Energy (Hartree)")
plt.subplots_adjust(left=0.17, bottom=0.1, right=0.95, top=0.95)
```



4.3 Restarting DMRG

[1]:
```
!pip install block2==0.5.3rc5 -qq --progress-bar off --extra-index-url=https://block-
↪hczhai.github.io/block2-preview/pypi/
!pip install pyscf==2.3.0 -qq --progress-bar off
```

4.3.1 Introduction

In this tutorial we explain how to restart a DMRG calculation using MPS stored in disk. When you use the `pyblock2` interface, the MPS data is automatically stored into the scratch folder so one can recover the calculation as long as the scratch data is not removed.

4.3.2 Restarting from Interruption

One common case is that the DMRG calculation is interrupted by the `Ctrl+C` signal, or the job has reached the time limit. Let's say we have a normal DMRG, but interrupted in the middle of sweep 1 (counting from zero):

[2]:
```
import numpy as np
from pyblock2._pyscf.ao2mo import integrals as itg
from pyblock2.driver.core import DMRGDriver, SymmetryTypes

from pyscf import gto, scf

mol = gto.M(atom='C 0 0 0; C 0 0 1.2425', basis='ccpvdz', symmetry='d2h')
mf = scf.RHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf,
    ncore=2, ncas=26, g2e_symm=8)

print("NCAS = %d NCASELEC = %d" % (ncas, n_elec))
driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SU2,
                    stack_mem=4 << 30, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)
mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)
ket = driver.get_random_mps(tag="KET", bond_dim=250, nroots=1)

bond_dims = [250] * 4 + [500] * 4 + [750] * 4 + [1000] * 4
noises = [1e-4] * 4 + [1e-5] * 12 + [0]
thrds = [1e-8] * 20

energy = driver.dmrg(mpo, ket, n_sweeps=28, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=2, twosite_to_onesite=20)
print('DMRG energy (variational) = %20.15f' % energy)
```

```
converged SCF energy = -75.3869023777059
NCAS = 26 NCASELEC = 8
Environment initialization | Nsites =    26 | Center =     0
 INIT-R <-- Site =   23 ..  Bmem =   200 B Rmem =   200 B T = 0.01
 INIT-R <-- Site =   22 ..  Bmem = 2.55 KB Rmem = 2.55 KB T = 0.00
 INIT-R <-- Site =   21 ..  Bmem = 22.6 KB Rmem = 22.6 KB T = 0.00
 INIT-R <-- Site =   20 ..  Bmem =  237 KB Rmem =  237 KB T = 0.03
 INIT-R <-- Site =   19 ..  Bmem = 3.46 MB Rmem = 1.62 MB T = 0.21
 INIT-R <-- Site =   18 ..  Bmem = 25.7 MB Rmem = 1.91 MB T = 0.32
```

(continues on next page)

(continued from previous page)

```
 INIT-R <-- Site =   17 ..  Bmem = 31.7 MB Rmem = 2.39 MB T = 0.55
 INIT-R <-- Site =   16 ..  Bmem = 40.6 MB Rmem = 3.04 MB T = 0.86
 INIT-R <-- Site =   15 ..  Bmem = 51.7 MB Rmem = 3.84 MB T = 1.21
 INIT-R <-- Site =   14 ..  Bmem = 66.2 MB Rmem = 4.84 MB T = 1.77
 INIT-R <-- Site =   13 ..  Bmem = 83.1 MB Rmem = 6.77 MB T = 1.80
 INIT-R <-- Site =   12 ..  Bmem =  116 MB Rmem = 8.38 MB T = 4.38
 INIT-R <-- Site =   11 ..  Bmem =  127 MB Rmem = 10.0 MB T = 4.09
 INIT-R <-- Site =   10 ..  Bmem =  144 MB Rmem = 9.57 MB T = 10.25
 INIT-R <-- Site =    9 ..  Bmem =  122 MB Rmem = 9.15 MB T = 2.72
 INIT-R <-- Site =    8 ..  Bmem =  116 MB Rmem = 7.99 MB T = 2.62
 INIT-R <-- Site =    7 ..  Bmem =  101 MB Rmem = 7.88 MB T = 2.62
 INIT-R <-- Site =    6 ..  Bmem = 97.8 MB Rmem = 6.72 MB T = 3.23
 INIT-R <-- Site =    5 ..  Bmem = 82.3 MB Rmem = 6.12 MB T = 3.73
 INIT-R <-- Site =    4 ..  Bmem = 73.9 MB Rmem = 5.19 MB T = 2.41
 INIT-R <-- Site =    3 ..  Bmem = 61.7 MB Rmem = 4.42 MB T = 2.19
 INIT-R <-- Site =    2 ..  Bmem = 52.0 MB Rmem = 3.42 MB T = 2.07
 INIT-R <-- Site =    1 ..  Bmem = 32.7 MB Rmem = 2.33 MB T = 1.27
 INIT-R <-- Site =    0 ..  Bmem = 19.1 MB Rmem = 1.36 MB T = 0.93
Time init sweep =       49.511 | MaxBmem =  144 MB | MaxRmem = 9.57 MB
 | Tread = 0.000 | Twrite = 0.398 | Tfpread = 0.000 | Tfpwrite = 0.351 | data = 107␣
↪MB | cpsd = 85.9 MB | Tasync = 0.000


Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-08
 --> Site =    0-   1 .. Mmps =    3 Ndav =   6 E =    -74.2848920054 Error = 0.
↪00e+00 FLOPS = 1.09e+06 Tdav = 0.00 T = 0.44
 --> Site =    1-   2 .. Mmps =   10 Ndav =   6 E =    -74.8270759121 Error = 0.
↪00e+00 FLOPS = 1.18e+07 Tdav = 0.00 T = 1.49
 --> Site =    2-   3 .. Mmps =   35 Ndav =   8 E =    -75.4399264227 Error = 0.
↪00e+00 FLOPS = 2.87e+07 Tdav = 0.01 T = 1.82
 --> Site =    3-   4 .. Mmps =  126 Ndav =  13 E =    -75.5128300575 Error = 0.
↪00e+00 FLOPS = 6.82e+07 Tdav = 0.06 T = 1.86
 --> Site =    4-   5 .. Mmps =  250 Ndav =  14 E =    -75.5518177150 Error = 1.40e-
↪07 FLOPS = 2.09e+08 Tdav = 0.14 T = 2.09
 --> Site =    5-   6 .. Mmps =  250 Ndav =  11 E =    -75.5891500696 Error = 4.92e-
↪07 FLOPS = 4.79e+08 Tdav = 0.11 T = 2.28
 --> Site =    6-   7 .. Mmps =  250 Ndav =   9 E =    -75.5958682759 Error = 2.28e-
↪07 FLOPS = 4.37e+08 Tdav = 0.08 T = 2.51
 --> Site =    7-   8 .. Mmps =  250 Ndav =   8 E =    -75.6089186044 Error = 2.20e-
↪07 FLOPS = 2.12e+08 Tdav = 0.15 T = 3.67
 --> Site =    8-   9 .. Mmps =  250 Ndav =   7 E =    -75.6161570937 Error = 1.87e-
↪07 FLOPS = 4.00e+08 Tdav = 0.07 T = 2.98
 --> Site =    9-  10 .. Mmps =  250 Ndav =   7 E =    -75.6260053169 Error = 1.84e-
↪07 FLOPS = 3.22e+08 Tdav = 0.09 T = 2.46
```

(continues on next page)

(continued from previous page)

```
 --> Site =   10-  11 .. Mmps =  250 Ndav =   7 E =    -75.6343632235 Error = 1.40e-
↪07 FLOPS = 3.62e+08 Tdav = 0.09 T = 2.54
 --> Site =   11-  12 .. Mmps =  250 Ndav =   7 E =    -75.6474368387 Error = 8.99e-
↪08 FLOPS = 3.74e+08 Tdav = 0.10 T = 10.13
 --> Site =   12-  13 .. Mmps =  250 Ndav =   7 E =    -75.6594509402 Error = 1.08e-
↪07 FLOPS = 3.61e+08 Tdav = 0.10 T = 4.26
 --> Site =   13-  14 .. Mmps =  250 Ndav =   6 E =    -75.6733135783 Error = 9.68e-
↪08 FLOPS = 3.54e+08 Tdav = 0.09 T = 2.94
 --> Site =   14-  15 .. Mmps =  250 Ndav =   5 E =    -75.6796365116 Error = 8.91e-
↪08 FLOPS = 1.91e+08 Tdav = 0.11 T = 2.58
 --> Site =   15-  16 .. Mmps =  250 Ndav =   5 E =    -75.6823006277 Error = 8.14e-
↪08 FLOPS = 3.11e+08 Tdav = 0.05 T = 1.63
 --> Site =   16-  17 .. Mmps =  250 Ndav =   6 E =    -75.6919467723 Error = 2.93e-
↪08 FLOPS = 3.34e+08 Tdav = 0.05 T = 0.82
 --> Site =   17-  18 .. Mmps =  250 Ndav =   5 E =    -75.6941463201 Error = 5.04e-
↪08 FLOPS = 3.38e+08 Tdav = 0.03 T = 0.61
 --> Site =   18-  19 .. Mmps =  250 Ndav =   4 E =    -75.6948176472 Error = 1.15e-
↪08 FLOPS = 2.80e+08 Tdav = 0.02 T = 0.44
 --> Site =   19-  20 .. Mmps =  250 Ndav =   5 E =    -75.6983966348 Error = 1.60e-
↪08 FLOPS = 4.35e+08 Tdav = 0.02 T = 0.28
 --> Site =   20-  21 .. Mmps =  250 Ndav =   4 E =    -75.6988045936 Error = 3.52e-
↪09 FLOPS = 4.16e+08 Tdav = 0.01 T = 0.14
 --> Site =   21-  22 .. Mmps =  250 Ndav =   1 E =    -75.6988045919 Error = 1.20e-
↪09 FLOPS = 3.08e+08 Tdav = 0.00 T = 0.09
 --> Site =   22-  23 .. Mmps =  204 Ndav =   1 E =    -75.6988045918 Error = 6.36e-
↪21 FLOPS = 3.66e+08 Tdav = 0.00 T = 0.05
 --> Site =   23-  24 .. Mmps =   72 Ndav =   1 E =    -75.6988045918 Error = 7.51e-
↪21 FLOPS = 3.83e+08 Tdav = 0.00 T = 0.03
 --> Site =   24-  25 .. Mmps =   14 Ndav =   1 E =    -75.6988045918 Error = 8.88e-
↪23 FLOPS = 8.37e+07 Tdav = 0.00 T = 0.01
Time elapsed =     48.314 | E =     -75.6988045936 | DW = 4.92e-07
Time sweep =       48.314 | 404 MFLOP/SWP
 | Dmem = 10.1 MB (0%) | Imem = 61.8 MB (93%) | Hmem = 219 MB | Wmem = 11.6 KB |␣
↪Pmem = 1.12 MB
 | Tread = 0.364 | Twrite = 0.213 | Tfpread = 0.349 | Tfpwrite = 0.208 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 83.3 MB | cpsd = 66.7 MB
 | Trot = 0.320 | Tctr = 0.009 | Tint = 0.004 | Tmid = 0.000 | Tdctr = 0.164 | Tdiag␣
↪= 0.054 | Tinfo = 0.332
 | Teff = 45.367 | Tprt = 0.296 | Teig = 1.369 | Tblk = 47.254 | Tmve = 0.907 | Tdm␣
↪= 0.033 | Tsplt = 0.025 | Tsvd = 0.000 | Torth = 0.000

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-08
 <-- Site =   24-  25 .. Mmps =    3 Ndav =   1 E =    -75.6988045918 Error = 0.
```

(continues on next page)

(continued from previous page)

```
↪00e+00 FLOPS = 4.36e+07 Tdav = 0.00 T = 0.01
 <-- Site =   23-  24 .. Mmps =    9 Ndav =   1 E =    -75.6988045918 Error = 0.
↪00e+00 FLOPS = 3.97e+08 Tdav = 0.00 T = 0.02
 <-- Site =   22-  23 .. Mmps =   23 Ndav =   1 E =    -75.6988045918 Error = 0.
↪00e+00 FLOPS = 2.97e+08 Tdav = 0.00 T = 0.03
 <-- Site =   21-  22 .. Mmps =   46 Ndav =   1 E =    -75.6988045918 Error = 0.
↪00e+00 FLOPS = 3.05e+08 Tdav = 0.00 T = 0.06
 <-- Site =   20-  21 .. Mmps =   84 Ndav =   2 E =    -75.6988045944 Error = 0.
↪00e+00 FLOPS = 3.76e+08 Tdav = 0.01 T = 0.09
 <-- Site =   19-  20 .. Mmps =  143 Ndav =   2 E =    -75.6988046953 Error = 0.
↪00e+00 FLOPS = 4.09e+08 Tdav = 0.01 T = 0.13
 <-- Site =   18-  19 .. Mmps =  222 Ndav =   4 E =    -75.6992261762 Error = 0.
↪00e+00 FLOPS = 5.95e+08 Tdav = 0.02 T = 0.21
 <-- Site =   17-  18 .. Mmps =  250 Ndav =   4 E =    -75.6993353012 Error = 4.83e-
↪09 FLOPS = 9.58e+08 Tdav = 0.03 T = 0.28
 <-- Site =   16-  17 .. Mmps =  250 Ndav =   4 E =    -75.6993678454 Error = 2.76e-
↪08 FLOPS = 1.07e+09 Tdav = 0.04 T = 0.37
 <-- Site =   15-  16 .. Mmps =  250 Ndav =   5 E =    -75.6999200129 Error = 4.96e-
↪08 FLOPS = 1.31e+09 Tdav = 0.06 T = 0.47
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-2-37efcb0b4e9c> in <cell line: 23>()
     21 thrds = [1e-8] * 20
---> 23 energy = driver.dmrg(mpo, ket, n_sweeps=28, bond_dims=bond_dims,␣
↪noises=noises,
     22
     24     thrds=thrds, iprint=2, twosite_to_onesite=20)
     25 print('DMRG energy (variational) = %20.15f' % energy)

/usr/local/lib/python3.10/dist-packages/pyblock2/driver/core.py in dmrg(self, mpo,␣
↪ket, n_sweeps, tol, bond_dims, noises, thrds, iprint, dav_type, cutoff, twosite_to_
↪onesite, dav_max_iter, dav_def_max_size, proj_mpss, proj_weights, store_wfn_
↪spectra, spectra_with_multiplicity, lowmem_noise, sweep_start, forward)
   2903             assert twosite_to_onesite < n_sweeps
   2904             if sweep_start < twosite_to_onesite:
-> 2905                 ener = dmrg.solve(twosite_to_onesite, forward, 0, sweep_
↪start)
   2906                 dmrg.me.dot = 1
   2907                 for ext_me in dmrg.ext_mes:

KeyboardInterrupt:
```

Then we can restart the calculation using the following script. Note that we change `driver.get_random_mps` to `driver.load_mps` with the same tag, so that data in the scratch folder `./tmp` with that tag will loaded. If we want to use the previous sweep schedule, we can keep the sweep schedule unchanged, and when calling `driver.dmrg`, we use the `sweep_start` parameter to set the

sweep index of the interrupted sweep (counting from zero), and the `forward` parameter to indicate the direction of the first restarting sweep. Then the DMRG will restart from the middle of the interrupted sweep.

For the MPS loaded from the interrupted sweep, `driver.adjust_mps` should not be used after `driver.load_mps`.

[3]:
```
import numpy as np
from pyblock2._pyscf.ao2mo import integrals as itg
from pyblock2.driver.core import DMRGDriver, SymmetryTypes

from pyscf import gto, scf

mol = gto.M(atom='C 0 0 0; C 0 0 1.2425', basis='ccpvdz', symmetry='d2h')
mf = scf.RHF(mol).run(conv_tol=1E-14)
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf,
    ncore=2, ncas=26, g2e_symm=8)

print("NCAS = %d NCASELEC = %d" % (ncas, n_elec))
driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SU2,
                    stack_mem=4 << 30, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)
mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, iprint=0)
ket = driver.load_mps(tag="KET", nroots=1)

bond_dims = [250] * 4 + [500] * 4 + [750] * 4 + [1000] * 4
noises = [1e-4] * 4 + [1e-5] * 12 + [0]
thrds = [1e-8] * 20

sweep_start = 1
forward = sweep_start % 2 == 0

energy = driver.dmrg(mpo, ket, n_sweeps=28, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=2, twosite_to_onesite=20, forward=forward, sweep_start=sweep_
↪start)
print('DMRG energy (variational) = %20.15f' % energy)
```

```
converged SCF energy = -75.3869023777058
NCAS = 26 NCASELEC = 8
Environment initialization | Nsites =    26 | Center =    12
 INIT-L --> Site =    1 ..  Bmem =   200 B Rmem =   200 B T = 0.00
 INIT-L --> Site =    2 ..  Bmem = 2.86 KB Rmem = 2.86 KB T = 0.00
 INIT-L --> Site =    3 ..  Bmem = 26.6 KB Rmem = 26.6 KB T = 0.00
 INIT-L --> Site =    4 ..  Bmem =  237 KB Rmem =  237 KB T = 0.02
 INIT-L --> Site =    5 ..  Bmem = 3.46 MB Rmem = 1.38 MB T = 0.07
 INIT-L --> Site =    6 ..  Bmem = 20.4 MB Rmem = 1.91 MB T = 0.13
 INIT-L --> Site =    7 ..  Bmem = 27.4 MB Rmem = 2.63 MB T = 0.14
 INIT-L --> Site =    8 ..  Bmem = 35.1 MB Rmem = 3.56 MB T = 0.16
```

(continues on next page)

(continued from previous page)

```
 INIT-L --> Site =    9 ..  Bmem = 44.6 MB Rmem = 4.54 MB T = 0.18
 INIT-L --> Site =   10 ..  Bmem = 54.2 MB Rmem = 5.57 MB T = 0.19
 INIT-L --> Site =   11 ..  Bmem = 64.3 MB Rmem = 6.78 MB T = 0.21
 INIT-L --> Site =   12 ..  Bmem = 75.4 MB Rmem = 7.88 MB T = 0.26
 INIT-R <-- Site =   23 ..  Bmem =   200 B Rmem =   200 B T = 0.01
 INIT-R <-- Site =   22 ..  Bmem = 2.55 KB Rmem = 2.27 KB T = 0.00
 INIT-R <-- Site =   21 ..  Bmem = 20.0 KB Rmem = 12.6 KB T = 0.00
 INIT-R <-- Site =   20 ..  Bmem =  129 KB Rmem = 50.0 KB T = 0.01
 INIT-R <-- Site =   19 ..  Bmem =  697 KB Rmem =  228 KB T = 0.02
 INIT-R <-- Site =   18 ..  Bmem = 2.95 MB Rmem =  845 KB T = 0.03
 INIT-R <-- Site =   17 ..  Bmem = 10.6 MB Rmem = 2.47 MB T = 0.07
 INIT-R <-- Site =   16 ..  Bmem = 31.9 MB Rmem = 4.12 MB T = 0.11
 INIT-R <-- Site =   15 ..  Bmem = 48.8 MB Rmem = 4.60 MB T = 0.16
 INIT-R <-- Site =   14 ..  Bmem = 55.8 MB Rmem = 5.12 MB T = 0.17
 INIT-R <-- Site =   13 ..  Bmem = 64.1 MB Rmem = 5.60 MB T = 0.23
 INIT-R <-- Site =   12 ..  Bmem = 72.2 MB Rmem = 6.41 MB T = 0.36
Time init sweep =        2.657 | MaxBmem = 75.4 MB | MaxRmem = 7.88 MB
 | Tread = 0.000 | Twrite = 0.189 | Tfpread = 0.000 | Tfpwrite = 0.160 | data = 63.9␣
↪MB | cpsd = 53.1 MB | Tasync = 0.000


Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-08
 <-- Site =   12-  13 .. Mmps =  250 Ndav =   6 E =    -75.7019455569 Error = 2.43e-
↪07 FLOPS = 1.52e+09 Tdav = 0.14 T = 1.51
 <-- Site =   11-  12 .. Mmps =  250 Ndav =   6 E =    -75.7033367067 Error = 2.75e-
↪07 FLOPS = 1.56e+09 Tdav = 0.15 T = 1.39
 <-- Site =   10-  11 .. Mmps =  250 Ndav =   7 E =    -75.7054373205 Error = 5.29e-
↪07 FLOPS = 1.74e+09 Tdav = 0.15 T = 0.65
 <-- Site =    9-  10 .. Mmps =  250 Ndav =   7 E =    -75.7087798252 Error = 5.56e-
↪07 FLOPS = 1.91e+09 Tdav = 0.13 T = 0.57
 <-- Site =    8-   9 .. Mmps =  250 Ndav =   7 E =    -75.7103058282 Error = 7.69e-
↪07 FLOPS = 2.12e+09 Tdav = 0.12 T = 0.51
 <-- Site =    7-   8 .. Mmps =  250 Ndav =   7 E =    -75.7119354371 Error = 1.88e-
↪06 FLOPS = 2.26e+09 Tdav = 0.13 T = 0.51
 <-- Site =    6-   7 .. Mmps =  250 Ndav =   9 E =    -75.7148564117 Error = 6.02e-
↪06 FLOPS = 2.31e+09 Tdav = 0.16 T = 0.50
 <-- Site =    5-   6 .. Mmps =  250 Ndav =  11 E =    -75.7194018891 Error = 3.92e-
↪05 FLOPS = 1.25e+09 Tdav = 0.39 T = 0.82
 <-- Site =    4-   5 .. Mmps =  250 Ndav =  14 E =    -75.7234052111 Error = 4.32e-
↪05 FLOPS = 7.05e+08 Tdav = 0.42 T = 0.81
 <-- Site =    3-   4 .. Mmps =  250 Ndav =   5 E =    -75.7232592109 Error = 2.20e-
↪06 FLOPS = 2.64e+08 Tdav = 0.07 T = 0.29
 <-- Site =    2-   3 .. Mmps =  250 Ndav =   1 E =    -75.7232592101 Error = 3.64e-
↪08 FLOPS = 1.10e+08 Tdav = 0.01 T = 0.15
```

(continues on next page)

(continued from previous page)

```
 <-- Site =    1-   2 .. Mmps =  112 Ndav =   1 E =    -75.7232592101 Error = 1.33e-
↪20 FLOPS = 1.39e+08 Tdav = 0.00 T = 0.09
 <-- Site =    0-   1 .. Mmps =   18 Ndav =   1 E =    -75.7232592101 Error = 1.47e-
↪21 FLOPS = 1.70e+07 Tdav = 0.00 T = 0.02
Time elapsed =      7.886 | E =     -75.7234052111 | DE = -7.57e+01 | DW = 4.32e-05
Time sweep =        7.886 | 2.45 GFLOP/SWP
 | Dmem = 7.99 MB (1%) | Imem = 2.38 MB (96%) | Hmem = 150 MB | Wmem = 59.4 KB |␣
↪Pmem = 4.98 MB
 | Tread = 0.069 | Twrite = 0.111 | Tfpread = 0.068 | Tfpwrite = 0.108 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 41.0 MB | cpsd = 33.0 MB
 | Trot = 0.230 | Tctr = 0.005 | Tint = 0.006 | Tmid = 0.000 | Tdctr = 0.007 | Tdiag␣
↪= 0.051 | Tinfo = 0.136
 | Teff = 4.712 | Tprt = 0.661 | Teig = 1.844 | Tblk = 7.307 | Tmve = 0.496 | Tdm =␣
↪0.038 | Tsplt = 0.025 | Tsvd = 0.000 | Torth = 0.000

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-08
 --> Site =    0-   1 .. Mmps =    3 Ndav =   1 E =    -75.7232592101 Error = 0.
↪00e+00 FLOPS = 1.53e+07 Tdav = 0.00 T = 0.01
 --> Site =    1-   2 .. Mmps =   10 Ndav =   1 E =    -75.7232592101 Error = 0.
↪00e+00 FLOPS = 1.32e+08 Tdav = 0.00 T = 0.06
 --> Site =    2-   3 .. Mmps =   35 Ndav =   1 E =    -75.7232592101 Error = 0.
↪00e+00 FLOPS = 1.09e+08 Tdav = 0.01 T = 0.10
 --> Site =    3-   4 .. Mmps =  126 Ndav =   2 E =    -75.7232592142 Error = 0.
↪00e+00 FLOPS = 2.49e+08 Tdav = 0.03 T = 0.23
 --> Site =    4-   5 .. Mmps =  250 Ndav =   6 E =    -75.7234052136 Error = 4.13e-
↪05 FLOPS = 7.38e+08 Tdav = 0.17 T = 0.53
 --> Site =    5-   6 .. Mmps =  250 Ndav =   7 E =    -75.7237452930 Error = 6.80e-
↪05 FLOPS = 1.77e+09 Tdav = 0.21 T = 0.72
 --> Site =    6-   7 .. Mmps =  250 Ndav =   6 E =    -75.7237136502 Error = 2.41e-
↪05 FLOPS = 2.80e+09 Tdav = 0.13 T = 0.50
 --> Site =    7-   8 .. Mmps =  250 Ndav =   6 E =    -75.7239111826 Error = 1.17e-
↪05 FLOPS = 3.07e+09 Tdav = 0.13 T = 0.54
 --> Site =    8-   9 .. Mmps =  250 Ndav =   5 E =    -75.7241648868 Error = 8.29e-
↪06 FLOPS = 2.44e+09 Tdav = 0.15 T = 0.59
 --> Site =    9-  10 .. Mmps =  250 Ndav =   5 E =    -75.7243750111 Error = 7.52e-
↪06 FLOPS = 3.08e+09 Tdav = 0.13 T = 0.59
 --> Site =   10-  11 .. Mmps =  250 Ndav =   5 E =    -75.7245369229 Error = 4.19e-
↪06 FLOPS = 2.91e+09 Tdav = 0.14 T = 0.65
 --> Site =   11-  12 .. Mmps =  250 Ndav =   5 E =    -75.7246985832 Error = 2.16e-
↪06 FLOPS = 2.88e+09 Tdav = 0.15 T = 1.39
 --> Site =   12-  13 .. Mmps =  250 Ndav =   5 E =    -75.7248097165 Error = 2.05e-
↪06 FLOPS = 2.57e+09 Tdav = 0.15 T = 1.60
 --> Site =   13-  14 .. Mmps =  250 Ndav =   4 E =    -75.7248481163 Error = 1.27e-
```

(continues on next page)

(continued from previous page)

```
↪06 FLOPS = 2.51e+09 Tdav = 0.10 T = 0.98
 --> Site =   14-  15 .. Mmps =  250 Ndav =   4 E =    -75.7248926895 Error = 9.45e-
↪07 FLOPS = 2.21e+09 Tdav = 0.09 T = 0.74
 --> Site =   15-  16 .. Mmps =  250 Ndav =   4 E =    -75.7249114972 Error = 8.19e-
↪07 FLOPS = 1.99e+09 Tdav = 0.07 T = 0.51
 --> Site =   16-  17 .. Mmps =  250 Ndav =   3 E =    -75.7249191345 Error = 2.75e-
↪07 FLOPS = 1.69e+09 Tdav = 0.04 T = 0.41
 --> Site =   17-  18 .. Mmps =  250 Ndav =   3 E =    -75.7249226753 Error = 2.33e-
↪07 FLOPS = 1.41e+09 Tdav = 0.03 T = 0.35
 --> Site =   18-  19 .. Mmps =  250 Ndav =   3 E =    -75.7249233359 Error = 8.79e-
↪08 FLOPS = 8.67e+08 Tdav = 0.02 T = 0.24
 --> Site =   19-  20 .. Mmps =  250 Ndav =   3 E =    -75.7249237309 Error = 4.94e-
↪08 FLOPS = 5.87e+08 Tdav = 0.02 T = 0.18
 --> Site =   20-  21 .. Mmps =  250 Ndav =   2 E =    -75.7249238259 Error = 2.23e-
↪08 FLOPS = 3.71e+08 Tdav = 0.01 T = 0.13
 --> Site =   21-  22 .. Mmps =  250 Ndav =   2 E =    -75.7249238396 Error = 4.19e-
↪09 FLOPS = 2.71e+08 Tdav = 0.01 T = 0.10
 --> Site =   22-  23 .. Mmps =  229 Ndav =   2 E =    -75.7249238890 Error = 1.19e-
↪21 FLOPS = 1.92e+08 Tdav = 0.00 T = 0.06
 --> Site =   23-  24 .. Mmps =   82 Ndav =   2 E =    -75.7249238917 Error = 7.54e-
↪21 FLOPS = 2.05e+08 Tdav = 0.00 T = 0.03
 --> Site =   24-  25 .. Mmps =   13 Ndav =   1 E =    -75.7249238917 Error = 5.36e-
↪21 FLOPS = 1.80e+07 Tdav = 0.00 T = 0.01
Time elapsed =     19.296 | E =     -75.7249238917 | DE = -1.52e-03 | DW = 6.80e-05
Time sweep =       11.410 | 3.76 GFLOP/SWP
 | Dmem = 6.96 MB (1%) | Imem = 2.38 MB (96%) | Hmem = 144 MB | Wmem = 69.4 KB |␣
↪Pmem = 5.39 MB
 | Tread = 0.227 | Twrite = 0.166 | Tfpread = 0.222 | Tfpwrite = 0.158 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 67.9 MB | cpsd = 54.6 MB
 | Trot = 0.321 | Tctr = 0.013 | Tint = 0.009 | Tmid = 0.000 | Tdctr = 0.123 | Tdiag␣
↪= 0.085 | Tinfo = 0.342
 | Teff = 7.574 | Tprt = 0.867 | Teig = 1.811 | Tblk = 10.397 | Tmve = 0.865 | Tdm =␣
↪0.062 | Tsplt = 0.032 | Tsvd = 0.000 | Torth = 0.000

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-08
 <-- Site =   24-  25 .. Mmps =    3 Ndav =   1 E =    -75.7249238917 Error = 0.
↪00e+00 FLOPS = 1.74e+07 Tdav = 0.00 T = 0.01
 <-- Site =   23-  24 .. Mmps =   10 Ndav =   1 E =    -75.7249238917 Error = 0.
↪00e+00 FLOPS = 1.66e+08 Tdav = 0.00 T = 0.02
 <-- Site =   22-  23 .. Mmps =   28 Ndav =   1 E =    -75.7249238917 Error = 0.
↪00e+00 FLOPS = 1.57e+08 Tdav = 0.00 T = 0.04
 <-- Site =   21-  22 .. Mmps =   57 Ndav =   2 E =    -75.7249238933 Error = 0.
↪00e+00 FLOPS = 2.69e+08 Tdav = 0.01 T = 0.09
```

(continues on next page)

(continued from previous page)

```
 <-- Site =   20-  21 .. Mmps =  114 Ndav =   2 E =    -75.7249239063 Error = 0.
↪00e+00 FLOPS = 2.03e+08 Tdav = 0.02 T = 0.18
 <-- Site =   19-  20 .. Mmps =  208 Ndav =   2 E =    -75.7249239233 Error = 0.
↪00e+00 FLOPS = 2.99e+08 Tdav = 0.02 T = 0.28
 <-- Site =   18-  19 .. Mmps =  250 Ndav =   2 E =    -75.7249239605 Error = 5.16e-
↪10 FLOPS = 4.58e+08 Tdav = 0.03 T = 0.41
 <-- Site =   17-  18 .. Mmps =  250 Ndav =   2 E =    -75.7249241027 Error = 1.81e-
↪08 FLOPS = 7.00e+08 Tdav = 0.04 T = 0.58
 <-- Site =   16-  17 .. Mmps =  250 Ndav =   2 E =    -75.7249242715 Error = 6.36e-
↪08 FLOPS = 8.59e+08 Tdav = 0.06 T = 0.68
 <-- Site =   15-  16 .. Mmps =  250 Ndav =   3 E =    -75.7249251140 Error = 9.84e-
↪08 FLOPS = 1.18e+09 Tdav = 0.09 T = 0.84
 <-- Site =   14-  15 .. Mmps =  250 Ndav =   3 E =    -75.7249261153 Error = 1.82e-
↪07 FLOPS = 2.19e+09 Tdav = 0.07 T = 0.97
 <-- Site =   13-  14 .. Mmps =  250 Ndav =   3 E =    -75.7249287653 Error = 3.00e-
↪07 FLOPS = 2.50e+09 Tdav = 0.08 T = 0.97
 <-- Site =   12-  13 .. Mmps =  250 Ndav =   3 E =    -75.7249339151 Error = 5.92e-
↪07 FLOPS = 2.57e+09 Tdav = 0.09 T = 1.57
 <-- Site =   11-  12 .. Mmps =  250 Ndav =   3 E =    -75.7249379223 Error = 6.35e-
↪07 FLOPS = 2.84e+09 Tdav = 0.09 T = 1.39
 <-- Site =   10-  11 .. Mmps =  250 Ndav =   4 E =    -75.7249575289 Error = 1.81e-
↪06 FLOPS = 2.90e+09 Tdav = 0.11 T = 0.65
 <-- Site =    9-  10 .. Mmps =  250 Ndav =   4 E =    -75.7249945619 Error = 4.71e-
↪06 FLOPS = 2.78e+09 Tdav = 0.11 T = 0.59
 <-- Site =    8-   9 .. Mmps =  250 Ndav =   4 E =    -75.7250110719 Error = 4.87e-
↪06 FLOPS = 2.79e+09 Tdav = 0.10 T = 0.56
 <-- Site =    7-   8 .. Mmps =  250 Ndav =   4 E =    -75.7250495206 Error = 7.48e-
↪06 FLOPS = 2.76e+09 Tdav = 0.09 T = 0.51
 <-- Site =    6-   7 .. Mmps =  250 Ndav =   5 E =    -75.7251768782 Error = 1.83e-
↪05 FLOPS = 2.76e+09 Tdav = 0.11 T = 0.48
 <-- Site =    5-   6 .. Mmps =  250 Ndav =   6 E =    -75.7254307257 Error = 6.87e-
↪05 FLOPS = 2.59e+09 Tdav = 0.13 T = 0.45
 <-- Site =    4-   5 .. Mmps =  250 Ndav =   7 E =    -75.7253660998 Error = 4.81e-
↪05 FLOPS = 1.36e+09 Tdav = 0.11 T = 0.35
 <-- Site =    3-   4 .. Mmps =  250 Ndav =   5 E =    -75.7251980323 Error = 2.26e-
↪06 FLOPS = 5.11e+08 Tdav = 0.04 T = 0.17
 <-- Site =    2-   3 .. Mmps =  250 Ndav =   1 E =    -75.7251980316 Error = 3.74e-
↪08 FLOPS = 1.98e+08 Tdav = 0.00 T = 0.09
 <-- Site =    1-   2 .. Mmps =  115 Ndav =   1 E =    -75.7251980316 Error = 1.97e-
↪20 FLOPS = 2.45e+08 Tdav = 0.00 T = 0.05
 <-- Site =    0-   1 .. Mmps =   16 Ndav =   1 E =    -75.7251980316 Error = 1.43e-
↪21 FLOPS = 3.13e+07 Tdav = 0.00 T = 0.02
Time elapsed =     31.390 | E =     -75.7254307257 | DE = -5.07e-04 | DW = 6.87e-05
Time sweep =       12.094 | 2.75 GFLOP/SWP
 | Dmem = 6.91 MB (1%) | Imem = 2.40 MB (96%) | Hmem = 143 MB | Wmem = 69.3 KB |␣
```

(continues on next page)

(continued from previous page)

```
↪Pmem = 5.38 MB
 | Tread = 0.139 | Twrite = 0.211 | Tfpread = 0.134 | Tfpwrite = 0.206 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 69.9 MB | cpsd = 57.1 MB
 | Trot = 0.383 | Tctr = 0.016 | Tint = 0.015 | Tmid = 0.000 | Tdctr = 0.014 | Tdiag␣
↪= 0.088 | Tinfo = 0.210
 | Teff = 8.373 | Tprt = 1.167 | Teig = 1.382 | Tblk = 11.077 | Tmve = 0.855 | Tdm =␣
↪0.060 | Tsplt = 0.036 | Tsvd = 0.000 | Torth = 0.000

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
 --> Site =    0-   1 .. Mmps =    3 Ndav =   1 E =    -75.7251980316 Error = 0.
↪00e+00 FLOPS = 1.57e+07 Tdav = 0.00 T = 0.01
 --> Site =    1-   2 .. Mmps =   10 Ndav =   1 E =    -75.7251980316 Error = 0.
↪00e+00 FLOPS = 1.64e+08 Tdav = 0.00 T = 0.04
 --> Site =    2-   3 .. Mmps =   35 Ndav =   1 E =    -75.7251980316 Error = 0.
↪00e+00 FLOPS = 1.91e+08 Tdav = 0.00 T = 0.06
 --> Site =    3-   4 .. Mmps =  126 Ndav =   1 E =    -75.7251980316 Error = 0.
↪00e+00 FLOPS = 3.54e+08 Tdav = 0.01 T = 0.12
 --> Site =    4-   5 .. Mmps =  418 Ndav =   7 E =    -75.7253661027 Error = 0.
↪00e+00 FLOPS = 1.13e+09 Tdav = 0.13 T = 0.33
 --> Site =    5-   6 .. Mmps =  500 Ndav =   9 E =    -75.7257735088 Error = 3.97e-
↪06 FLOPS = 3.86e+09 Tdav = 0.29 T = 0.76
 --> Site =    6-   7 .. Mmps =  500 Ndav =   6 E =    -75.7259667099 Error = 1.12e-
↪06 FLOPS = 2.82e+09 Tdav = 0.45 T = 1.36
 --> Site =    7-   8 .. Mmps =  500 Ndav =   5 E =    -75.7262124157 Error = 5.25e-
↪07 FLOPS = 2.77e+09 Tdav = 0.43 T = 1.82
 --> Site =    8-   9 .. Mmps =  500 Ndav =   5 E =    -75.7264691420 Error = 5.15e-
↪07 FLOPS = 4.90e+09 Tdav = 0.29 T = 1.48
 --> Site =    9-  10 .. Mmps =  500 Ndav =   5 E =    -75.7266957987 Error = 4.25e-
↪07 FLOPS = 5.13e+09 Tdav = 0.32 T = 1.27
 --> Site =   10-  11 .. Mmps =  500 Ndav =   5 E =    -75.7269168383 Error = 2.93e-
↪07 FLOPS = 5.27e+09 Tdav = 0.35 T = 1.36
 --> Site =   11-  12 .. Mmps =  500 Ndav =   5 E =    -75.7270750756 Error = 2.07e-
↪07 FLOPS = 5.19e+09 Tdav = 0.39 T = 2.26
 --> Site =   12-  13 .. Mmps =  500 Ndav =   4 E =    -75.7271854729 Error = 1.49e-
↪07 FLOPS = 4.74e+09 Tdav = 0.33 T = 3.38
 --> Site =   13-  14 .. Mmps =  500 Ndav =   4 E =    -75.7272582167 Error = 8.86e-
↪08 FLOPS = 2.58e+09 Tdav = 0.53 T = 2.51
 --> Site =   14-  15 .. Mmps =  500 Ndav =   4 E =    -75.7273066233 Error = 6.50e-
↪08 FLOPS = 2.61e+09 Tdav = 0.40 T = 2.62
 --> Site =   15-  16 .. Mmps =  500 Ndav =   4 E =    -75.7273314216 Error = 5.69e-
↪08 FLOPS = 3.87e+09 Tdav = 0.20 T = 1.05
 --> Site =   16-  17 .. Mmps =  500 Ndav =   4 E =    -75.7273471908 Error = 2.27e-
↪08 FLOPS = 3.28e+09 Tdav = 0.18 T = 0.90
```

(continues on next page)

(continued from previous page)

```
 --> Site =   17-  18 .. Mmps =  500 Ndav =   3 E =    -75.7273535904 Error = 1.15e-
↪08 FLOPS = 3.43e+09 Tdav = 0.09 T = 0.70
 --> Site =   18-  19 .. Mmps =  500 Ndav =   3 E =    -75.7273551591 Error = 4.46e-
↪09 FLOPS = 2.33e+09 Tdav = 0.06 T = 0.56
 --> Site =   19-  20 .. Mmps =  500 Ndav =   2 E =    -75.7273554387 Error = 2.43e-
↪09 FLOPS = 1.46e+09 Tdav = 0.03 T = 0.37
 --> Site =   20-  21 .. Mmps =  500 Ndav =   2 E =    -75.7273554986 Error = 6.12e-
↪10 FLOPS = 9.59e+08 Tdav = 0.02 T = 0.26
 --> Site =   21-  22 .. Mmps =  500 Ndav =   2 E =    -75.7273557488 Error = 1.40e-
↪10 FLOPS = 5.65e+08 Tdav = 0.01 T = 0.22
 --> Site =   22-  23 .. Mmps =  405 Ndav =   2 E =    -75.7273558056 Error = 1.56e-
↪20 FLOPS = 5.31e+08 Tdav = 0.01 T = 0.18
 --> Site =   23-  24 .. Mmps =   99 Ndav =   1 E =    -75.7273558056 Error = 1.27e-
↪20 FLOPS = 4.35e+08 Tdav = 0.00 T = 0.07
 --> Site =   24-  25 .. Mmps =   13 Ndav =   1 E =    -75.7273558056 Error = 1.32e-
↪20 FLOPS = 2.44e+07 Tdav = 0.00 T = 0.01
Time elapsed =     55.273 | E =     -75.7273558056 | DE = -1.93e-03 | DW = 3.97e-06
Time sweep =       23.883 | 15.4 GFLOP/SWP
 | Dmem = 23.6 MB (1%) | Imem = 3.02 MB (96%) | Hmem = 347 MB | Wmem = 146 KB | Pmem␣
↪= 11.2 MB
 | Tread = 0.549 | Twrite = 0.684 | Tfpread = 0.543 | Tfpwrite = 0.679 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 250 MB | cpsd = 201 MB
 | Trot = 0.947 | Tctr = 0.019 | Tint = 0.009 | Tmid = 0.000 | Tdctr = 0.437 | Tdiag␣
↪= 0.128 | Tinfo = 0.478
 | Teff = 14.022 | Tprt = 2.702 | Teig = 4.515 | Tblk = 21.566 | Tmve = 2.156 | Tdm␣
↪= 0.151 | Tsplt = 0.087 | Tsvd = 0.000 | Torth = 0.000

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
 <-- Site =   24-  25 .. Mmps =    3 Ndav =   1 E =    -75.7273558056 Error = 0.
↪00e+00 FLOPS = 2.30e+07 Tdav = 0.00 T = 0.01
 <-- Site =   23-  24 .. Mmps =   10 Ndav =   1 E =    -75.7273558056 Error = 0.
↪00e+00 FLOPS = 4.59e+08 Tdav = 0.00 T = 0.04
 <-- Site =   22-  23 .. Mmps =   31 Ndav =   1 E =    -75.7273558056 Error = 0.
↪00e+00 FLOPS = 4.62e+08 Tdav = 0.00 T = 0.08
 <-- Site =   21-  22 .. Mmps =   80 Ndav =   1 E =    -75.7273558056 Error = 0.
↪00e+00 FLOPS = 5.65e+08 Tdav = 0.01 T = 0.15
 <-- Site =   20-  21 .. Mmps =  173 Ndav =   2 E =    -75.7273558074 Error = 0.
↪00e+00 FLOPS = 9.56e+08 Tdav = 0.02 T = 0.23
 <-- Site =   19-  20 .. Mmps =  333 Ndav =   2 E =    -75.7273558176 Error = 0.
↪00e+00 FLOPS = 9.49e+08 Tdav = 0.04 T = 0.45
 <-- Site =   18-  19 .. Mmps =  500 Ndav =   2 E =    -75.7273558269 Error = 3.70e-
↪11 FLOPS = 1.88e+09 Tdav = 0.05 T = 0.66
 <-- Site =   17-  18 .. Mmps =  500 Ndav =   3 E =    -75.7273566664 Error = 5.75e-
```

(continues on next page)

(continued from previous page)

```
↪10 FLOPS = 3.45e+09 Tdav = 0.10 T = 1.11
 <-- Site =   16-  17 .. Mmps =  500 Ndav =   3 E =    -75.7273571781 Error = 3.00e-
↪09 FLOPS = 4.06e+09 Tdav = 0.15 T = 1.15
 <-- Site =   15-  16 .. Mmps =  500 Ndav =   3 E =    -75.7273587280 Error = 8.12e-
↪09 FLOPS = 2.75e+09 Tdav = 0.37 T = 2.13
 <-- Site =   14-  15 .. Mmps =  500 Ndav =   3 E =    -75.7273598091 Error = 1.82e-
↪08 FLOPS = 2.68e+09 Tdav = 0.53 T = 4.01
 <-- Site =   13-  14 .. Mmps =  500 Ndav =   3 E =    -75.7273614000 Error = 2.82e-
↪08 FLOPS = 5.68e+09 Tdav = 0.32 T = 3.38
 <-- Site =   12-  13 .. Mmps =  500 Ndav =   3 E =    -75.7273659236 Error = 4.68e-
↪08 FLOPS = 5.84e+09 Tdav = 0.34 T = 4.25
 <-- Site =   11-  12 .. Mmps =  500 Ndav =   4 E =    -75.7273764077 Error = 8.96e-
↪08 FLOPS = 6.26e+09 Tdav = 0.46 T = 4.03
 <-- Site =   10-  11 .. Mmps =  500 Ndav =   4 E =    -75.7273979421 Error = 3.39e-
↪07 FLOPS = 3.63e+09 Tdav = 0.74 T = 3.27
 <-- Site =    9-  10 .. Mmps =  500 Ndav =   4 E =    -75.7274389878 Error = 1.01e-
↪06 FLOPS = 6.92e+09 Tdav = 0.36 T = 1.80
 <-- Site =    8-   9 .. Mmps =  500 Ndav =   4 E =    -75.7274920505 Error = 1.60e-
↪06 FLOPS = 6.86e+09 Tdav = 0.33 T = 1.55
 <-- Site =    7-   8 .. Mmps =  500 Ndav =   5 E =    -75.7275654540 Error = 2.24e-
↪06 FLOPS = 6.66e+09 Tdav = 0.37 T = 1.45
 <-- Site =    6-   7 .. Mmps =  500 Ndav =   6 E =    -75.7276988453 Error = 4.25e-
↪06 FLOPS = 6.24e+09 Tdav = 0.45 T = 1.41
 <-- Site =    5-   6 .. Mmps =  500 Ndav =   9 E =    -75.7278860763 Error = 1.58e-
↪05 FLOPS = 5.60e+09 Tdav = 0.50 T = 1.23
 <-- Site =    4-   5 .. Mmps =  500 Ndav =   5 E =    -75.7278120707 Error = 1.86e-
↪07 FLOPS = 2.60e+09 Tdav = 0.12 T = 0.52
 <-- Site =    3-   4 .. Mmps =  500 Ndav =   1 E =    -75.7278120705 Error = 8.21e-
↪08 FLOPS = 6.21e+08 Tdav = 0.02 T = 0.28
 <-- Site =    2-   3 .. Mmps =  500 Ndav =   1 E =    -75.7278120705 Error = 5.04e-
↪10 FLOPS = 6.49e+08 Tdav = 0.00 T = 0.15
 <-- Site =    1-   2 .. Mmps =  114 Ndav =   1 E =    -75.7278120705 Error = 1.82e-
↪20 FLOPS = 6.14e+08 Tdav = 0.00 T = 0.10
 <-- Site =    0-   1 .. Mmps =   17 Ndav =   1 E =    -75.7278120705 Error = 1.40e-
↪22 FLOPS = 3.20e+07 Tdav = 0.00 T = 0.02
Time elapsed =     88.911 | E =     -75.7278860763 | DE = -5.30e-04 | DW = 1.58e-05
Time sweep =       33.638 | 24.4 GFLOP/SWP
 | Dmem = 26.7 MB (1%) | Imem = 3.23 MB (96%) | Hmem = 571 MB | Wmem = 243 KB | Pmem␣
↪= 20.5 MB
 | Tread = 0.476 | Twrite = 0.808 | Tfpread = 0.474 | Tfpwrite = 0.803 | Tmoread =␣
↪0.000 | Tasync = 0.000
 | data = 265 MB | cpsd = 216 MB
 | Trot = 1.120 | Tctr = 0.014 | Tint = 0.023 | Tmid = 0.000 | Tdctr = 0.015 | Tdiag␣
↪= 0.151 | Tinfo = 0.277
 | Teff = 20.032 | Tprt = 5.251 | Teig = 5.312 | Tblk = 31.156 | Tmve = 2.297 | Tdm␣
```

(continues on next page)

(continued from previous page)

```
↪= 0.254 | Tsplt = 0.115 | Tsvd = 0.000 | Torth = 0.000

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
 --> Site =    0-   1 .. Mmps =    3 Ndav =   1 E =    -75.7278120705 Error = 0.
↪00e+00 FLOPS = 3.19e+07 Tdav = 0.00 T = 0.01
 --> Site =    1-   2 .. Mmps =   10 Ndav =   1 E =    -75.7278120705 Error = 0.
↪00e+00 FLOPS = 6.34e+08 Tdav = 0.00 T = 0.07
 --> Site =    2-   3 .. Mmps =   35 Ndav =   1 E =    -75.7278120705 Error = 0.
↪00e+00 FLOPS = 4.70e+08 Tdav = 0.01 T = 0.10
 --> Site =    3-   4 .. Mmps =  126 Ndav =   1 E =    -75.7278120705 Error = 0.
↪00e+00 FLOPS = 9.50e+08 Tdav = 0.01 T = 0.22
 --> Site =    4-   5 .. Mmps =  421 Ndav =   1 E =    -75.7278120705 Error = 0.
↪00e+00 FLOPS = 2.34e+09 Tdav = 0.03 T = 0.36
 --> Site =    5-   6 .. Mmps =  500 Ndav =   6 E =    -75.7278862359 Error = 1.67e-
↪05 FLOPS = 5.65e+09 Tdav = 0.33 T = 1.12
 --> Site =    6-   7 .. Mmps =  500 Ndav =   5 E =    -75.7278386169 Error = 5.60e-
↪06 FLOPS = 3.81e+09 Tdav = 0.61 T = 2.23
 --> Site =    7-   8 .. Mmps =  500 Ndav =   4 E =    -75.7278422938 Error = 3.07e-
↪06 FLOPS = 6.79e+09 Tdav = 0.27 T = 1.85
 --> Site =    8-   9 .. Mmps =  500 Ndav =   4 E =    -75.7278572701 Error = 2.34e-
↪06 FLOPS = 6.56e+09 Tdav = 0.34 T = 1.54
 --> Site =    9-  10 .. Mmps =  500 Ndav =   4 E =    -75.7278659295 Error = 2.09e-
↪06 FLOPS = 6.69e+09 Tdav = 0.39 T = 1.75
 --> Site =   10-  11 .. Mmps =  500 Ndav =   4 E =    -75.7278694945 Error = 9.57e-
↪07 FLOPS = 6.95e+09 Tdav = 0.41 T = 1.97
 --> Site =   11-  12 .. Mmps =  500 Ndav =   4 E =    -75.7278751587 Error = 5.13e-
↪07 FLOPS = 6.69e+09 Tdav = 0.49 T = 3.95
 --> Site =   12-  13 .. Mmps =  500 Ndav =   3 E =    -75.7278789223 Error = 3.46e-
↪07 FLOPS = 6.40e+09 Tdav = 0.37 T = 6.04
 --> Site =   13-  14 .. Mmps =  500 Ndav =   3 E =    -75.7278801996 Error = 1.78e-
↪07 FLOPS = 6.17e+09 Tdav = 0.34 T = 2.84
 --> Site =   14-  15 .. Mmps =  500 Ndav =   3 E =    -75.7278808680 Error = 1.32e-
↪07 FLOPS = 5.88e+09 Tdav = 0.28 T = 2.13
 --> Site =   15-  16 .. Mmps =  500 Ndav =   3 E =    -75.7278813070 Error = 1.08e-
↪07 FLOPS = 5.85e+09 Tdav = 0.21 T = 1.53
 --> Site =   16-  17 .. Mmps =  500 Ndav =   2 E =    -75.7278812386 Error = 3.26e-
↪08 FLOPS = 4.82e+09 Tdav = 0.10 T = 1.18
 --> Site =   17-  18 .. Mmps =  500 Ndav =   2 E =    -75.7278812620 Error = 1.51e-
↪08 FLOPS = 1.98e+09 Tdav = 0.14 T = 1.22
 --> Site =   18-  19 .. Mmps =  500 Ndav =   2 E =    -75.7278812812 Error = 6.11e-
↪09 FLOPS = 1.35e+09 Tdav = 0.08 T = 1.09
 --> Site =   19-  20 .. Mmps =  500 Ndav =   2 E =    -75.7278812800 Error = 3.27e-
↪09 FLOPS = 8.75e+08 Tdav = 0.05 T = 0.69
 --> Site =   20-  21 .. Mmps =  500 Ndav =   1 E =    -75.7278812794 Error = 7.95e-
```

(continues on next page)

(continued from previous page)

```
↪10 FLOPS = 4.91e+08 Tdav = 0.02 T = 0.46
 --> Site =   21-  22 .. Mmps =  500 Ndav =   1 E =    -75.7278812791 Error = 1.62e-
↪10 FLOPS = 4.03e+08 Tdav = 0.01 T = 0.31
 --> Site =   22-  23 .. Mmps =  415 Ndav =   1 E =    -75.7278812791 Error = 1.67e-
↪20 FLOPS = 2.27e+08 Tdav = 0.01 T = 0.21
 --> Site =   23-  24 .. Mmps =   98 Ndav =   1 E =    -75.7278812791 Error = 1.23e-
↪20 FLOPS = 2.41e+08 Tdav = 0.00 T = 0.12
 --> Site =   24-  25 .. Mmps =   14 Ndav =   1 E =    -75.7278812791 Error = 1.78e-
↪21 FLOPS = 1.29e+07 Tdav = 0.00 T = 0.02
Time elapsed =    122.108 | E =     -75.7278862359 | DE = -1.60e-07 | DW = 1.67e-05
Time sweep =       33.197 | 23.6 GFLOP/SWP
 | Dmem = 26.7 MB (1%) | Imem = 3.30 MB (96%) | Hmem = 578 MB | Wmem = 241 KB | Pmem␣
↪= 20.4 MB
 | Tread = 0.960 | Twrite = 0.669 | Tfpread = 0.955 | Tfpwrite = 0.663 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 256 MB | cpsd = 205 MB
 | Trot = 0.953 | Tctr = 0.026 | Tint = 0.010 | Tmid = 0.000 | Tdctr = 0.516 | Tdiag␣
↪= 0.141 | Tinfo = 0.567
 | Teff = 21.205 | Tprt = 4.525 | Teig = 4.505 | Tblk = 30.746 | Tmve = 2.273 | Tdm␣
↪= 0.255 | Tsplt = 0.104 | Tsvd = 0.000 | Torth = 0.000

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
 <-- Site =   24-  25 .. Mmps =    3 Ndav =   1 E =    -75.7278812791 Error = 0.
↪00e+00 FLOPS = 1.24e+07 Tdav = 0.00 T = 0.01
 <-- Site =   23-  24 .. Mmps =   10 Ndav =   1 E =    -75.7278812791 Error = 0.
↪00e+00 FLOPS = 4.66e+08 Tdav = 0.00 T = 0.05
 <-- Site =   22-  23 .. Mmps =   31 Ndav =   1 E =    -75.7278812791 Error = 0.
↪00e+00 FLOPS = 4.57e+08 Tdav = 0.00 T = 0.08
 <-- Site =   21-  22 .. Mmps =   80 Ndav =   1 E =    -75.7278812791 Error = 0.
↪00e+00 FLOPS = 4.82e+08 Tdav = 0.01 T = 0.16
 <-- Site =   20-  21 .. Mmps =  174 Ndav =   1 E =    -75.7278812791 Error = 0.
↪00e+00 FLOPS = 8.99e+08 Tdav = 0.01 T = 0.20
 <-- Site =   19-  20 .. Mmps =  335 Ndav =   1 E =    -75.7278812791 Error = 0.
↪00e+00 FLOPS = 1.48e+09 Tdav = 0.01 T = 0.33
 <-- Site =   18-  19 .. Mmps =  500 Ndav =   2 E =    -75.7278812827 Error = 4.78e-
↪11 FLOPS = 2.55e+09 Tdav = 0.04 T = 0.55
 <-- Site =   17-  18 .. Mmps =  500 Ndav =   2 E =    -75.7278812948 Error = 6.98e-
↪10 FLOPS = 3.83e+09 Tdav = 0.07 T = 0.87
 <-- Site =   16-  17 .. Mmps =  500 Ndav =   2 E =    -75.7278813506 Error = 3.45e-
↪09 FLOPS = 4.74e+09 Tdav = 0.10 T = 1.13
 <-- Site =   15-  16 .. Mmps =  500 Ndav =   3 E =    -75.7278816037 Error = 1.15e-
↪08 FLOPS = 6.03e+09 Tdav = 0.21 T = 1.45
 <-- Site =   14-  15 .. Mmps =  500 Ndav =   3 E =    -75.7278818527 Error = 2.31e-
↪08 FLOPS = 6.04e+09 Tdav = 0.27 T = 2.09
```

(continues on next page)

(continued from previous page)

```
 <-- Site =   13-  14 .. Mmps =  500 Ndav =   3 E =    -75.7278822641 Error = 3.73e-
↪08 FLOPS = 6.12e+09 Tdav = 0.34 T = 2.83
 <-- Site =   12-  13 .. Mmps =  500 Ndav =   3 E =    -75.7278831562 Error = 8.81e-
↪08 FLOPS = 6.27e+09 Tdav = 0.37 T = 6.08
 <-- Site =   11-  12 .. Mmps =  500 Ndav =   3 E =    -75.7278842022 Error = 1.55e-
↪07 FLOPS = 6.42e+09 Tdav = 0.38 T = 3.81
 <-- Site =   10-  11 .. Mmps =  500 Ndav =   3 E =    -75.7278870423 Error = 5.16e-
↪07 FLOPS = 6.57e+09 Tdav = 0.33 T = 1.84
 <-- Site =    9-  10 .. Mmps =  500 Ndav =   4 E =    -75.7278941299 Error = 1.53e-
↪06 FLOPS = 6.90e+09 Tdav = 0.38 T = 1.71
 <-- Site =    8-   9 .. Mmps =  500 Ndav =   4 E =    -75.7278963607 Error = 1.78e-
↪06 FLOPS = 4.19e+09 Tdav = 0.54 T = 1.97
 <-- Site =    7-   8 .. Mmps =  500 Ndav =   4 E =    -75.7279044250 Error = 2.50e-
↪06 FLOPS = 3.82e+09 Tdav = 0.49 T = 2.37
 <-- Site =    6-   7 .. Mmps =  500 Ndav =   5 E =    -75.7279346950 Error = 5.06e-
↪06 FLOPS = 6.96e+09 Tdav = 0.33 T = 1.40
 <-- Site =    5-   6 .. Mmps =  500 Ndav =   6 E =    -75.7279905032 Error = 1.65e-
↪05 FLOPS = 5.71e+09 Tdav = 0.33 T = 1.09
 <-- Site =    4-   5 .. Mmps =  500 Ndav =   4 E =    -75.7279133365 Error = 1.85e-
↪07 FLOPS = 2.61e+09 Tdav = 0.10 T = 0.49
 <-- Site =    3-   4 .. Mmps =  500 Ndav =   1 E =    -75.7279133364 Error = 8.24e-
↪08 FLOPS = 9.12e+08 Tdav = 0.01 T = 0.28
 <-- Site =    2-   3 .. Mmps =  500 Ndav =   1 E =    -75.7279133364 Error = 5.13e-
↪10 FLOPS = 6.37e+08 Tdav = 0.00 T = 0.15
 <-- Site =    1-   2 .. Mmps =  114 Ndav =   1 E =    -75.7279133364 Error = 2.53e-
↪20 FLOPS = 6.61e+08 Tdav = 0.00 T = 0.11
 <-- Site =    0-   1 .. Mmps =   17 Ndav =   1 E =    -75.7279133364 Error = 1.37e-
↪22 FLOPS = 3.19e+07 Tdav = 0.00 T = 0.02
Time elapsed =    153.350 | E =     -75.7279905032 | DE = -1.04e-04 | DW = 1.65e-05
Time sweep =       31.242 | 22.4 GFLOP/SWP
 | Dmem = 26.8 MB (1%) | Imem = 3.30 MB (96%) | Hmem = 579 MB | Wmem = 242 KB | Pmem␣
↪= 20.4 MB
 | Tread = 0.443 | Twrite = 0.629 | Tfpread = 0.441 | Tfpwrite = 0.624 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 264 MB | cpsd = 215 MB
 | Trot = 0.938 | Tctr = 0.011 | Tint = 0.009 | Tmid = 0.000 | Tdctr = 0.014 | Tdiag␣
↪= 0.130 | Tinfo = 0.260
 | Teff = 19.693 | Tprt = 4.718 | Teig = 4.345 | Tblk = 29.192 | Tmve = 1.895 | Tdm␣
↪= 0.223 | Tsplt = 0.098 | Tsvd = 0.000 | Torth = 0.000

Sweep =    8 | Direction =  forward | Bond dimension =  750 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
 --> Site =    0-   1 .. Mmps =    3 Ndav =   1 E =    -75.7279133364 Error = 0.
↪00e+00 FLOPS = 1.73e+07 Tdav = 0.00 T = 0.01
 --> Site =    1-   2 .. Mmps =   10 Ndav =   1 E =    -75.7279133364 Error = 0.
```

(continues on next page)

(continued from previous page)

```
↪00e+00 FLOPS = 6.30e+08 Tdav = 0.00 T = 0.07
 --> Site =    2-   3 .. Mmps =   35 Ndav =   1 E =    -75.7279133364 Error = 0.
↪00e+00 FLOPS = 6.49e+08 Tdav = 0.00 T = 0.10
 --> Site =    3-   4 .. Mmps =  126 Ndav =   1 E =    -75.7279133364 Error = 0.
↪00e+00 FLOPS = 9.28e+08 Tdav = 0.01 T = 0.22
 --> Site =    4-   5 .. Mmps =  421 Ndav =   1 E =    -75.7279133364 Error = 0.
↪00e+00 FLOPS = 2.11e+09 Tdav = 0.03 T = 0.36
 --> Site =    5-   6 .. Mmps =  750 Ndav =   6 E =    -75.7279905058 Error = 1.40e-
↪06 FLOPS = 5.71e+09 Tdav = 0.33 T = 1.04
 --> Site =    6-   7 .. Mmps =  750 Ndav =   5 E =    -75.7280288171 Error = 1.35e-
↪06 FLOPS = 7.77e+09 Tdav = 0.59 T = 2.04
 --> Site =    7-   8 .. Mmps =  750 Ndav =   5 E =    -75.7280746779 Error = 7.40e-
↪07 FLOPS = 8.06e+09 Tdav = 0.64 T = 2.23
 --> Site =    8-   9 .. Mmps =  750 Ndav =   4 E =    -75.7281225653 Error = 5.97e-
↪07 FLOPS = 4.73e+09 Tdav = 1.03 T = 3.62
 --> Site =    9-  10 .. Mmps =  750 Ndav =   4 E =    -75.7281605371 Error = 5.28e-
↪07 FLOPS = 8.37e+09 Tdav = 0.66 T = 2.91
 --> Site =   10-  11 .. Mmps =  750 Ndav =   4 E =    -75.7281864144 Error = 3.50e-
↪07 FLOPS = 7.85e+09 Tdav = 0.80 T = 3.09
 --> Site =   11-  12 .. Mmps =  750 Ndav =   4 E =    -75.7282047508 Error = 1.92e-
↪07 FLOPS = 7.74e+09 Tdav = 0.92 T = 5.50
 --> Site =   12-  13 .. Mmps =  750 Ndav =   3 E =    -75.7282118759 Error = 1.38e-
↪07 FLOPS = 7.55e+09 Tdav = 0.66 T = 7.98
 --> Site =   13-  14 .. Mmps =  750 Ndav =   3 E =    -75.7282155349 Error = 6.74e-
↪08 FLOPS = 7.54e+09 Tdav = 0.60 T = 4.20
 --> Site =   14-  15 .. Mmps =  750 Ndav =   3 E =    -75.7282174239 Error = 5.27e-
↪08 FLOPS = 4.07e+09 Tdav = 0.84 T = 4.77
 --> Site =   15-  16 .. Mmps =  750 Ndav =   3 E =    -75.7282183509 Error = 4.31e-
↪08 FLOPS = 6.95e+09 Tdav = 0.37 T = 2.25
 --> Site =   16-  17 .. Mmps =  750 Ndav =   3 E =    -75.7282186451 Error = 1.32e-
↪08 FLOPS = 5.76e+09 Tdav = 0.26 T = 1.79
 --> Site =   17-  18 .. Mmps =  750 Ndav =   2 E =    -75.7282187353 Error = 7.57e-
↪09 FLOPS = 4.78e+09 Tdav = 0.12 T = 1.40
 --> Site =   18-  19 .. Mmps =  750 Ndav =   2 E =    -75.7282187395 Error = 2.89e-
↪09 FLOPS = 3.43e+09 Tdav = 0.07 T = 0.92
 --> Site =   19-  20 .. Mmps =  750 Ndav =   2 E =    -75.7282187406 Error = 1.45e-
↪09 FLOPS = 2.23e+09 Tdav = 0.04 T = 0.66
 --> Site =   20-  21 .. Mmps =  750 Ndav =   2 E =    -75.7282187410 Error = 3.01e-
↪10 FLOPS = 1.44e+09 Tdav = 0.02 T = 0.43
 --> Site =   21-  22 .. Mmps =  750 Ndav =   2 E =    -75.7282187494 Error = 4.14e-
↪11 FLOPS = 9.74e+08 Tdav = 0.02 T = 0.35
 --> Site =   22-  23 .. Mmps =  476 Ndav =   1 E =    -75.7282187494 Error = 3.64e-
↪21 FLOPS = 7.20e+08 Tdav = 0.00 T = 0.21
 --> Site =   23-  24 .. Mmps =  100 Ndav =   1 E =    -75.7282187494 Error = 2.12e-
↪20 FLOPS = 5.54e+08 Tdav = 0.00 T = 0.08
```

(continues on next page)

(continued from previous page)

```
 --> Site =   24-  25 .. Mmps =   13 Ndav =   1 E =    -75.7282187494 Error = 6.46e-
↪21 FLOPS = 2.47e+07 Tdav = 0.00 T = 0.02
Time elapsed =    199.782 | E =     -75.7282187494 | DE = -2.28e-04 | DW = 1.40e-06
Time sweep =       46.432 | 49.7 GFLOP/SWP
 | Dmem = 54.3 MB (1%) | Imem = 3.41 MB (96%) | Hmem = 908 MB | Wmem = 386 KB | Pmem␣
↪= 31.2 MB
 | Tread = 1.369 | Twrite = 1.375 | Tfpread = 1.363 | Tfpwrite = 1.370 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 546 MB | cpsd = 437 MB
 | Trot = 1.924 | Tctr = 0.022 | Tint = 0.010 | Tmid = 0.000 | Tdctr = 0.961 | Tdiag␣
↪= 0.165 | Tinfo = 0.572
 | Teff = 25.924 | Tprt = 7.634 | Teig = 8.021 | Tblk = 42.283 | Tmve = 3.984 | Tdm␣
↪= 0.402 | Tsplt = 0.159 | Tsvd = 0.000 | Torth = 0.000

Sweep =    9 | Direction = backward | Bond dimension =  750 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
 <-- Site =   24-  25 .. Mmps =    3 Ndav =   1 E =    -75.7282187494 Error = 0.
↪00e+00 FLOPS = 2.10e+07 Tdav = 0.00 T = 0.01
 <-- Site =   23-  24 .. Mmps =   10 Ndav =   1 E =    -75.7282187494 Error = 0.
↪00e+00 FLOPS = 5.54e+08 Tdav = 0.00 T = 0.05
 <-- Site =   22-  23 .. Mmps =   31 Ndav =   1 E =    -75.7282187494 Error = 0.
↪00e+00 FLOPS = 7.25e+08 Tdav = 0.00 T = 0.13
 <-- Site =   21-  22 .. Mmps =   81 Ndav =   1 E =    -75.7282187494 Error = 0.
↪00e+00 FLOPS = 8.47e+08 Tdav = 0.01 T = 0.24
 <-- Site =   20-  21 .. Mmps =  194 Ndav =   1 E =    -75.7282187494 Error = 0.
↪00e+00 FLOPS = 1.31e+09 Tdav = 0.01 T = 0.33
 <-- Site =   19-  20 .. Mmps =  445 Ndav =   2 E =    -75.7282187504 Error = 0.
↪00e+00 FLOPS = 2.14e+09 Tdav = 0.04 T = 0.53
 <-- Site =   18-  19 .. Mmps =  750 Ndav =   2 E =    -75.7282187520 Error = 9.83e-
↪13 FLOPS = 2.12e+09 Tdav = 0.12 T = 0.99
 <-- Site =   17-  18 .. Mmps =  750 Ndav =   2 E =    -75.7282187584 Error = 1.33e-
↪10 FLOPS = 3.16e+09 Tdav = 0.20 T = 2.49
 <-- Site =   16-  17 .. Mmps =  750 Ndav =   2 E =    -75.7282187840 Error = 8.12e-
↪10 FLOPS = 5.71e+09 Tdav = 0.21 T = 2.29
 <-- Site =   15-  16 .. Mmps =  750 Ndav =   2 E =    -75.7282189000 Error = 3.36e-
↪09 FLOPS = 6.82e+09 Tdav = 0.30 T = 2.64
 <-- Site =   14-  15 .. Mmps =  750 Ndav =   3 E =    -75.7282191096 Error = 1.08e-
↪08 FLOPS = 7.88e+09 Tdav = 0.63 T = 4.02
 <-- Site =   13-  14 .. Mmps =  750 Ndav =   3 E =    -75.7282194546 Error = 2.21e-
↪08 FLOPS = 6.67e+09 Tdav = 1.03 T = 7.12
 <-- Site =   12-  13 .. Mmps =  750 Ndav =   3 E =    -75.7282201965 Error = 3.87e-
↪08 FLOPS = 8.39e+09 Tdav = 0.89 T = 7.96
 <-- Site =   11-  12 .. Mmps =  750 Ndav =   3 E =    -75.7282216415 Error = 5.90e-
↪08 FLOPS = 8.62e+09 Tdav = 0.92 T = 8.87
 <-- Site =   10-  11 .. Mmps =  750 Ndav =   3 E =    -75.7282269952 Error = 1.89e-
```

(continues on next page)

(continued from previous page)

```
↪07 FLOPS = 8.78e+09 Tdav = 0.80 T = 3.86
 <-- Site =    9-  10 .. Mmps =  750 Ndav =   4 E =    -75.7282370875 Error = 4.83e-
↪07 FLOPS = 5.20e+09 Tdav = 1.61 T = 5.12
 <-- Site =    8-   9 .. Mmps =  750 Ndav =   4 E =    -75.7282479706 Error = 7.04e-
↪07 FLOPS = 8.90e+09 Tdav = 0.86 T = 3.35
 <-- Site =    7-   8 .. Mmps =  750 Ndav =   4 E =    -75.7282720747 Error = 1.31e-
↪06 FLOPS = 9.13e+09 Tdav = 0.71 T = 2.98
 <-- Site =    6-   7 .. Mmps =  750 Ndav =   5 E =    -75.7283208565 Error = 2.39e-
↪06 FLOPS = 9.17e+09 Tdav = 0.81 T = 2.81
 <-- Site =    5-   6 .. Mmps =  750 Ndav =   7 E =    -75.7283283738 Error = 2.06e-
↪06 FLOPS = 4.60e+09 Tdav = 0.84 T = 2.11
 <-- Site =    4-   5 .. Mmps =  750 Ndav =   3 E =    -75.7283189532 Error = 6.21e-
↪08 FLOPS = 1.87e+09 Tdav = 0.19 T = 1.19
 <-- Site =    3-   4 .. Mmps =  750 Ndav =   1 E =    -75.7283189531 Error = 5.46e-
↪08 FLOPS = 1.04e+09 Tdav = 0.03 T = 0.72
 <-- Site =    2-   3 .. Mmps =  750 Ndav =   1 E =    -75.7283189531 Error = 1.41e-
↪12 FLOPS = 4.81e+08 Tdav = 0.01 T = 0.38
 <-- Site =    1-   2 .. Mmps =  115 Ndav =   1 E =    -75.7283189531 Error = 2.46e-
↪20 FLOPS = 1.65e+08 Tdav = 0.02 T = 0.30
 <-- Site =    0-   1 .. Mmps =   16 Ndav =   1 E =    -75.7283189531 Error = 9.50e-
↪23 FLOPS = 3.44e+07 Tdav = 0.00 T = 0.02
Time elapsed =    260.462 | E =     -75.7283283738 | DE = -1.10e-04 | DW = 2.39e-06
Time sweep =       60.680 | 67.8 GFLOP/SWP
 | Dmem = 61.7 MB (1%) | Imem = 3.72 MB (96%) | Hmem = 1.26 GB | Wmem = 528 KB |␣
↪Pmem = 44.4 MB
 | Tread = 0.856 | Twrite = 1.461 | Tfpread = 0.854 | Tfpwrite = 1.455 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 583 MB | cpsd = 472 MB
 | Trot = 1.979 | Tctr = 0.017 | Tint = 0.011 | Tmid = 0.000 | Tdctr = 0.017 | Tdiag␣
↪= 0.199 | Tinfo = 0.373
 | Teff = 33.386 | Tprt = 12.049 | Teig = 10.269 | Tblk = 56.584 | Tmve = 3.937 |␣
↪Tdm = 0.538 | Tsplt = 0.210 | Tsvd = 0.000 | Torth = 0.000

Sweep =   10 | Direction =  forward | Bond dimension =  750 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
 --> Site =    0-   1 .. Mmps =    3 Ndav =   1 E =    -75.7283189531 Error = 0.
↪00e+00 FLOPS = 1.76e+07 Tdav = 0.00 T = 0.01
 --> Site =    1-   2 .. Mmps =   10 Ndav =   1 E =    -75.7283189531 Error = 0.
↪00e+00 FLOPS = 9.16e+08 Tdav = 0.00 T = 0.12
 --> Site =    2-   3 .. Mmps =   35 Ndav =   1 E =    -75.7283189531 Error = 0.
↪00e+00 FLOPS = 1.00e+09 Tdav = 0.01 T = 0.14
 --> Site =    3-   4 .. Mmps =  126 Ndav =   1 E =    -75.7283189531 Error = 0.
↪00e+00 FLOPS = 1.36e+09 Tdav = 0.02 T = 0.33
 --> Site =    4-   5 .. Mmps =  421 Ndav =   1 E =    -75.7283189531 Error = 0.
↪00e+00 FLOPS = 2.64e+09 Tdav = 0.04 T = 0.54
```

(continues on next page)

(continued from previous page)

```
 --> Site =    5-   6 .. Mmps =  750 Ndav =   4 E =    -75.7283283754 Error = 2.60e-
↪06 FLOPS = 6.53e+09 Tdav = 0.34 T = 1.30
 --> Site =    6-   7 .. Mmps =  750 Ndav =   4 E =    -75.7283309384 Error = 3.28e-
↪06 FLOPS = 8.74e+09 Tdav = 0.68 T = 2.63
 --> Site =    7-   8 .. Mmps =  750 Ndav =   4 E =    -75.7283268711 Error = 1.87e-
↪06 FLOPS = 8.90e+09 Tdav = 0.71 T = 2.94
 --> Site =    8-   9 .. Mmps =  750 Ndav =   4 E =    -75.7283266184 Error = 1.24e-
↪06 FLOPS = 6.66e+09 Tdav = 1.14 T = 4.24
 --> Site =    9-  10 .. Mmps =  750 Ndav =   4 E =    -75.7283277405 Error = 1.13e-
↪06 FLOPS = 9.05e+09 Tdav = 0.95 T = 4.06
 --> Site =   10-  11 .. Mmps =  750 Ndav =   3 E =    -75.7283275204 Error = 6.48e-
↪07 FLOPS = 8.75e+09 Tdav = 0.86 T = 3.93
 --> Site =   11-  12 .. Mmps =  750 Ndav =   3 E =    -75.7283280022 Error = 3.19e-
↪07 FLOPS = 7.68e+09 Tdav = 1.12 T = 9.04
 --> Site =   12-  13 .. Mmps =  750 Ndav =   3 E =    -75.7283281752 Error = 1.97e-
↪07 FLOPS = 8.56e+09 Tdav = 0.97 T = 8.13
 --> Site =   13-  14 .. Mmps =  750 Ndav =   3 E =    -75.7283280980 Error = 9.13e-
↪08 FLOPS = 8.46e+09 Tdav = 0.89 T = 7.24
 --> Site =   14-  15 .. Mmps =  750 Ndav =   3 E =    -75.7283280734 Error = 7.28e-
↪08 FLOPS = 7.93e+09 Tdav = 0.71 T = 4.26
 --> Site =   15-  16 .. Mmps =  750 Ndav =   2 E =    -75.7283280627 Error = 5.47e-
↪08 FLOPS = 7.18e+09 Tdav = 0.33 T = 3.20
 --> Site =   16-  17 .. Mmps =  750 Ndav =   2 E =    -75.7283279716 Error = 1.56e-
↪08 FLOPS = 4.51e+09 Tdav = 0.31 T = 3.85
 --> Site =   17-  18 .. Mmps =  750 Ndav =   2 E =    -75.7283279548 Error = 8.39e-
↪09 FLOPS = 3.92e+09 Tdav = 0.19 T = 2.01
 --> Site =   18-  19 .. Mmps =  750 Ndav =   1 E =    -75.7283279489 Error = 3.24e-
↪09 FLOPS = 2.44e+09 Tdav = 0.06 T = 1.22
 --> Site =   19-  20 .. Mmps =  750 Ndav =   1 E =    -75.7283279466 Error = 1.57e-
↪09 FLOPS = 2.11e+09 Tdav = 0.02 T = 0.78
 --> Site =   20-  21 .. Mmps =  750 Ndav =   1 E =    -75.7283279455 Error = 3.20e-
↪10 FLOPS = 1.00e+09 Tdav = 0.02 T = 0.65
 --> Site =   21-  22 .. Mmps =  750 Ndav =   1 E =    -75.7283279454 Error = 4.23e-
↪11 FLOPS = 7.96e+08 Tdav = 0.01 T = 0.50
 --> Site =   22-  23 .. Mmps =  482 Ndav =   1 E =    -75.7283279454 Error = 3.56e-
↪21 FLOPS = 5.53e+08 Tdav = 0.01 T = 0.36
 --> Site =   23-  24 .. Mmps =  102 Ndav =   1 E =    -75.7283279454 Error = 2.18e-
↪21 FLOPS = 4.61e+08 Tdav = 0.00 T = 0.14
 --> Site =   24-  25 .. Mmps =   13 Ndav =   1 E =    -75.7283279454 Error = 1.81e-
↪20 FLOPS = 3.60e+06 Tdav = 0.01 T = 0.03
Time elapsed =    322.357 | E =     -75.7283309384 | DE = -2.56e-06 | DW = 3.28e-06
Time sweep =       61.895 | 68.1 GFLOP/SWP
 | Dmem = 61.7 MB (1%) | Imem = 3.75 MB (96%) | Hmem = 1.27 GB | Wmem = 527 KB |␣
↪Pmem = 44.3 MB
 | Tread = 1.915 | Twrite = 1.830 | Tfpread = 1.910 | Tfpwrite = 1.821 | Tmporead =␣
```

(continues on next page)

(continued from previous page)

```
↪0.000 | Tasync = 0.000
 | data = 551 MB | cpsd = 441 MB
 | Trot = 2.066 | Tctr = 0.051 | Tint = 0.013 | Tmid = 0.000 | Tdctr = 1.035 | Tdiag␣
↪= 0.214 | Tinfo = 0.854
 | Teff = 34.752 | Tprt = 11.448 | Teig = 9.405 | Tblk = 56.715 | Tmve = 4.937 | Tdm␣
↪= 0.731 | Tsplt = 0.201 | Tsvd = 0.000 | Torth = 0.000

Sweep =   11 | Direction = backward | Bond dimension =  750 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
 <-- Site =   24-  25 .. Mmps =    3 Ndav =   1 E =    -75.7283279454 Error = 0.
↪00e+00 FLOPS = 2.01e+07 Tdav = 0.00 T = 0.01
 <-- Site =   23-  24 .. Mmps =   10 Ndav =   1 E =    -75.7283279454 Error = 0.
↪00e+00 FLOPS = 5.10e+08 Tdav = 0.00 T = 0.07
 <-- Site =   22-  23 .. Mmps =   31 Ndav =   1 E =    -75.7283279454 Error = 0.
↪00e+00 FLOPS = 7.02e+08 Tdav = 0.00 T = 0.16
 <-- Site =   21-  22 .. Mmps =   81 Ndav =   1 E =    -75.7283279454 Error = 0.
↪00e+00 FLOPS = 4.46e+08 Tdav = 0.02 T = 0.35
 <-- Site =   20-  21 .. Mmps =  194 Ndav =   1 E =    -75.7283279454 Error = 0.
↪00e+00 FLOPS = 9.86e+08 Tdav = 0.02 T = 0.39
 <-- Site =   19-  20 .. Mmps =  445 Ndav =   2 E =    -75.7283279466 Error = 0.
↪00e+00 FLOPS = 1.41e+09 Tdav = 0.07 T = 0.68
 <-- Site =   18-  19 .. Mmps =  750 Ndav =   2 E =    -75.7283279498 Error = 2.23e-
↪12 FLOPS = 3.32e+09 Tdav = 0.08 T = 1.02
 <-- Site =   17-  18 .. Mmps =  750 Ndav =   2 E =    -75.7283279556 Error = 1.58e-
↪10 FLOPS = 2.79e+09 Tdav = 0.27 T = 2.55
 <-- Site =   16-  17 .. Mmps =  750 Ndav =   2 E =    -75.7283279812 Error = 9.10e-
↪10 FLOPS = 5.51e+09 Tdav = 0.25 T = 3.18
 <-- Site =   15-  16 .. Mmps =  750 Ndav =   2 E =    -75.7283281146 Error = 3.98e-
↪09 FLOPS = 5.76e+09 Tdav = 0.41 T = 3.25
 <-- Site =   14-  15 .. Mmps =  750 Ndav =   2 E =    -75.7283282616 Error = 1.27e-
↪08 FLOPS = 6.77e+09 Tdav = 0.55 T = 4.89
 <-- Site =   13-  14 .. Mmps =  750 Ndav =   3 E =    -75.7283284869 Error = 2.57e-
↪08 FLOPS = 8.69e+09 Tdav = 0.86 T = 7.45
 <-- Site =   12-  13 .. Mmps =  750 Ndav =   3 E =    -75.7283290563 Error = 5.28e-
↪08 FLOPS = 8.53e+09 Tdav = 0.97 T = 8.75
 <-- Site =   11-  12 .. Mmps =  750 Ndav =   3 E =    -75.7283298183 Error = 8.40e-
↪08 FLOPS = 8.70e+09 Tdav = 0.98 T = 8.40
 <-- Site =   10-  11 .. Mmps =  750 Ndav =   3 E =    -75.7283318428 Error = 2.78e-
↪07 FLOPS = 8.57e+09 Tdav = 0.87 T = 3.94
 <-- Site =    9-  10 .. Mmps =  750 Ndav =   3 E =    -75.7283352731 Error = 6.34e-
↪07 FLOPS = 6.45e+09 Tdav = 0.99 T = 5.02
 <-- Site =    8-   9 .. Mmps =  750 Ndav =   4 E =    -75.7283367383 Error = 7.31e-
↪07 FLOPS = 8.78e+09 Tdav = 0.86 T = 3.33
 <-- Site =    7-   8 .. Mmps =  750 Ndav =   4 E =    -75.7283416559 Error = 1.34e-
↪06 FLOPS = 8.93e+09 Tdav = 0.71 T = 2.97
```

(continues on next page)

(continued from previous page)

```
 <-- Site =    6-   7 .. Mmps =  750 Ndav =   4 E =    -75.7283513281 Error = 2.59e-
↪06 FLOPS = 9.01e+09 Tdav = 0.66 T = 2.70
 <-- Site =    5-   6 .. Mmps =  750 Ndav =   5 E =    -75.7283489405 Error = 2.09e-
↪06 FLOPS = 3.68e+09 Tdav = 0.75 T = 2.46
 <-- Site =    4-   5 .. Mmps =  750 Ndav =   3 E =    -75.7283393442 Error = 6.23e-
↪08 FLOPS = 3.17e+09 Tdav = 0.11 T = 1.04
 <-- Site =    3-   4 .. Mmps =  750 Ndav =   1 E =    -75.7283393442 Error = 5.47e-
↪08 FLOPS = 1.47e+09 Tdav = 0.02 T = 0.43
 <-- Site =    2-   3 .. Mmps =  750 Ndav =   1 E =    -75.7283393442 Error = 1.54e-
↪12 FLOPS = 1.01e+09 Tdav = 0.01 T = 0.24
 <-- Site =    1-   2 .. Mmps =  114 Ndav =   1 E =    -75.7283393442 Error = 8.39e-
↪21 FLOPS = 8.75e+08 Tdav = 0.00 T = 0.17
 <-- Site =    0-   1 .. Mmps =   15 Ndav =   1 E =    -75.7283393442 Error = 5.09e-
↪21 FLOPS = 2.89e+07 Tdav = 0.00 T = 0.02
Time elapsed =    386.117 | E =     -75.7283513281 | DE = -2.04e-05 | DW = 2.59e-06
Time sweep =       63.760 | 64.9 GFLOP/SWP
 | Dmem = 61.1 MB (1%) | Imem = 3.78 MB (96%) | Hmem = 1.26 GB | Wmem = 527 KB |␣
↪Pmem = 44.3 MB
 | Tread = 1.025 | Twrite = 2.015 | Tfpread = 1.023 | Tfpwrite = 2.008 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 579 MB | cpsd = 469 MB
 | Trot = 2.248 | Tctr = 0.020 | Tint = 0.027 | Tmid = 0.000 | Tdctr = 0.032 | Tdiag␣
↪= 0.232 | Tinfo = 0.419
 | Teff = 35.424 | Tprt = 12.531 | Teig = 9.475 | Tblk = 58.609 | Tmve = 4.874 | Tdm␣
↪= 0.798 | Tsplt = 0.221 | Tsvd = 0.000 | Torth = 0.000

Sweep =   12 | Direction =  forward | Bond dimension = 1000 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
 --> Site =    0-   1 .. Mmps =    3 Ndav =   1 E =    -75.7283393442 Error = 0.
↪00e+00 FLOPS = 3.25e+07 Tdav = 0.00 T = 0.01
 --> Site =    1-   2 .. Mmps =   10 Ndav =   1 E =    -75.7283393442 Error = 0.
↪00e+00 FLOPS = 8.91e+08 Tdav = 0.00 T = 0.11
 --> Site =    2-   3 .. Mmps =   35 Ndav =   1 E =    -75.7283393442 Error = 0.
↪00e+00 FLOPS = 9.84e+08 Tdav = 0.01 T = 0.15
 --> Site =    3-   4 .. Mmps =  126 Ndav =   1 E =    -75.7283393442 Error = 0.
↪00e+00 FLOPS = 1.44e+09 Tdav = 0.02 T = 0.33
 --> Site =    4-   5 .. Mmps =  421 Ndav =   1 E =    -75.7283393442 Error = 0.
↪00e+00 FLOPS = 3.09e+09 Tdav = 0.04 T = 0.56
 --> Site =    5-   6 .. Mmps = 1000 Ndav =   5 E =    -75.7283489409 Error = 1.27e-
↪07 FLOPS = 6.81e+09 Tdav = 0.41 T = 1.35
 --> Site =    6-   7 .. Mmps = 1000 Ndav =   5 E =    -75.7283655946 Error = 1.18e-
↪06 FLOPS = 9.51e+09 Tdav = 1.12 T = 3.72
 --> Site =    7-   8 .. Mmps = 1000 Ndav =   4 E =    -75.7283799617 Error = 7.65e-
↪07 FLOPS = 9.54e+09 Tdav = 1.08 T = 4.73
 --> Site =    8-   9 .. Mmps = 1000 Ndav =   4 E =    -75.7283925464 Error = 5.41e-
```

(continues on next page)

(continued from previous page)

```
↪07 FLOPS = 9.75e+09 Tdav = 1.26 T = 5.23
 --> Site =    9-  10 .. Mmps = 1000 Ndav =   4 E =    -75.7284034409 Error = 5.04e-
↪07 FLOPS = 9.75e+09 Tdav = 1.44 T = 4.98
 --> Site =   10-  11 .. Mmps = 1000 Ndav =   4 E =    -75.7284117667 Error = 3.34e-
↪07 FLOPS = 6.84e+09 Tdav = 2.34 T = 7.27
 --> Site =   11-  12 .. Mmps = 1000 Ndav =   3 E =    -75.7284155318 Error = 1.55e-
↪07 FLOPS = 9.58e+09 Tdav = 1.43 T = 9.77
 --> Site =   12-  13 .. Mmps = 1000 Ndav =   3 E =    -75.7284173721 Error = 1.08e-
↪07 FLOPS = 9.40e+09 Tdav = 1.43 T = 11.99
 --> Site =   13-  14 .. Mmps = 1000 Ndav =   3 E =    -75.7284183239 Error = 5.68e-
↪08 FLOPS = 9.49e+09 Tdav = 1.26 T = 9.00
 --> Site =   14-  15 .. Mmps = 1000 Ndav =   3 E =    -75.7284188841 Error = 4.16e-
↪08 FLOPS = 8.93e+09 Tdav = 1.01 T = 6.45
 --> Site =   15-  16 .. Mmps = 1000 Ndav =   2 E =    -75.7284190260 Error = 3.30e-
↪08 FLOPS = 7.84e+09 Tdav = 0.48 T = 4.62
 --> Site =   16-  17 .. Mmps = 1000 Ndav =   2 E =    -75.7284190332 Error = 1.00e-
↪08 FLOPS = 6.80e+09 Tdav = 0.33 T = 3.10
 --> Site =   17-  18 .. Mmps = 1000 Ndav =   2 E =    -75.7284190450 Error = 5.38e-
↪09 FLOPS = 5.44e+09 Tdav = 0.23 T = 2.38
 --> Site =   18-  19 .. Mmps = 1000 Ndav =   2 E =    -75.7284190424 Error = 2.08e-
↪09 FLOPS = 4.26e+09 Tdav = 0.11 T = 1.41
 --> Site =   19-  20 .. Mmps = 1000 Ndav =   1 E =    -75.7284190400 Error = 9.14e-
↪10 FLOPS = 1.42e+09 Tdav = 0.06 T = 1.14
 --> Site =   20-  21 .. Mmps = 1000 Ndav =   1 E =    -75.7284190394 Error = 1.62e-
↪10 FLOPS = 9.08e+08 Tdav = 0.03 T = 1.13
 --> Site =   21-  22 .. Mmps = 1000 Ndav =   1 E =    -75.7284190394 Error = 1.13e-
↪11 FLOPS = 6.56e+08 Tdav = 0.02 T = 1.04
 --> Site =   22-  23 .. Mmps =  536 Ndav =   1 E =    -75.7284190394 Error = 4.04e-
↪21 FLOPS = 5.38e+08 Tdav = 0.01 T = 0.52
 --> Site =   23-  24 .. Mmps =   99 Ndav =   1 E =    -75.7284190394 Error = 1.53e-
↪20 FLOPS = 3.82e+08 Tdav = 0.00 T = 0.17
 --> Site =   24-  25 .. Mmps =   13 Ndav =   1 E =    -75.7284190394 Error = 3.29e-
↪21 FLOPS = 1.34e+07 Tdav = 0.00 T = 0.03
Time elapsed =    467.485 | E =     -75.7284190450 | DE = -6.77e-05 | DW = 1.18e-06
Time sweep =       81.368 | 114 GFLOP/SWP
 | Dmem = 94.8 MB (1%) | Imem = 4.12 MB (96%) | Hmem = 1.70 GB | Wmem = 677 KB |␣
↪Pmem = 57.5 MB
 | Tread = 2.618 | Twrite = 2.948 | Tfpread = 2.613 | Tfpwrite = 2.942 | Tmoread =␣
↪0.000 | Tasync = 0.000
 | data = 941 MB | cpsd = 751 MB
 | Trot = 4.111 | Tctr = 0.035 | Tint = 0.013 | Tmid = 0.000 | Tdctr = 1.709 | Tdiag␣
↪= 0.247 | Tinfo = 0.693
 | Teff = 41.250 | Tprt = 16.413 | Teig = 14.112 | Tblk = 73.261 | Tmve = 7.928 |␣
↪Tdm = 0.812 | Tsplt = 0.347 | Tsvd = 0.000 | Torth = 0.000
```

(continues on next page)

(continued from previous page)

```
Sweep =   13 | Direction = backward | Bond dimension = 1000 | Noise =  1.00e-05 | Dav threshold =  1.00e-08
 <-- Site =   24-  25 .. Mmps =    3 Ndav =   1 E =    -75.7284190394 Error = 0.00e+00 FLOPS = 1.22e+07 Tdav = 0.00 T = 0.01
 <-- Site =   23-  24 .. Mmps =   10 Ndav =   1 E =    -75.7284190394 Error = 0.00e+00 FLOPS = 4.41e+08 Tdav = 0.00 T = 0.09
 <-- Site =   22-  23 .. Mmps =   31 Ndav =   1 E =    -75.7284190394 Error = 0.00e+00 FLOPS = 8.98e+08 Tdav = 0.01 T = 0.23
 <-- Site =   21-  22 .. Mmps =   81 Ndav =   1 E =    -75.7284190394 Error = 0.00e+00 FLOPS = 1.10e+09 Tdav = 0.01 T = 0.33
 <-- Site =   20-  21 .. Mmps =  213 Ndav =   1 E =    -75.7284190394 Error = 0.00e+00 FLOPS = 1.57e+09 Tdav = 0.02 T = 0.44
 <-- Site =   19-  20 .. Mmps =  548 Ndav =   2 E =    -75.7284190409 Error = 0.00e+00 FLOPS = 2.80e+09 Tdav = 0.06 T = 0.74
 <-- Site =   18-  19 .. Mmps = 1000 Ndav =   2 E =    -75.7284190433 Error = 5.83e-13 FLOPS = 4.37e+09 Tdav = 0.11 T = 1.28
 <-- Site =   17-  18 .. Mmps = 1000 Ndav =   2 E =    -75.7284190494 Error = 4.79e-11 FLOPS = 5.88e+09 Tdav = 0.23 T = 2.51
 <-- Site =   16-  17 .. Mmps = 1000 Ndav =   2 E =    -75.7284190684 Error = 4.10e-10 FLOPS = 7.00e+09 Tdav = 0.38 T = 3.43
 <-- Site =   15-  16 .. Mmps = 1000 Ndav =   2 E =    -75.7284191759 Error = 1.35e-09 FLOPS = 4.83e+09 Tdav = 0.92 T = 6.01
 <-- Site =   14-  15 .. Mmps = 1000 Ndav =   2 E =    -75.7284193085 Error = 6.68e-09 FLOPS = 8.95e+09 Tdav = 0.81 T = 6.28
 <-- Site =   13-  14 .. Mmps = 1000 Ndav =   3 E =    -75.7284195147 Error = 1.56e-08 FLOPS = 9.43e+09 Tdav = 1.64 T = 10.82
 <-- Site =   12-  13 .. Mmps = 1000 Ndav =   3 E =    -75.7284200102 Error = 3.09e-08 FLOPS = 1.01e+10 Tdav = 1.80 T = 14.52
 <-- Site =   11-  12 .. Mmps = 1000 Ndav =   3 E =    -75.7284207093 Error = 4.49e-08 FLOPS = 9.81e+09 Tdav = 1.87 T = 13.34
 <-- Site =   10-  11 .. Mmps = 1000 Ndav =   3 E =    -75.7284233883 Error = 1.44e-07 FLOPS = 1.02e+10 Tdav = 1.59 T = 7.05
 <-- Site =    9-  10 .. Mmps = 1000 Ndav =   3 E =    -75.7284280297 Error = 2.90e-07 FLOPS = 9.67e+09 Tdav = 1.48 T = 7.33
 <-- Site =    8-   9 .. Mmps = 1000 Ndav =   4 E =    -75.7284320403 Error = 3.29e-07 FLOPS = 1.03e+10 Tdav = 1.61 T = 6.01
 <-- Site =    7-   8 .. Mmps = 1000 Ndav =   4 E =    -75.7284412951 Error = 5.93e-07 FLOPS = 1.03e+10 Tdav = 1.36 T = 6.28
 <-- Site =    6-   7 .. Mmps = 1000 Ndav =   4 E =    -75.7284581391 Error = 1.11e-06 FLOPS = 9.84e+09 Tdav = 1.23 T = 4.54
 <-- Site =    5-   6 .. Mmps = 1000 Ndav =   3 E =    -75.7284529738 Error = 1.37e-07 FLOPS = 7.49e+09 Tdav = 0.34 T = 1.82
 <-- Site =    4-   5 .. Mmps = 1000 Ndav =   1 E =    -75.7284529056 Error = 4.03e-08 FLOPS = 2.07e+09 Tdav = 0.09 T = 1.31
 <-- Site =    3-   4 .. Mmps = 1000 Ndav =   1 E =    -75.7284529055 Error = 3.68e-
```

(continues on next page)

(continued from previous page)

```
↪08 FLOPS = 1.06e+09 Tdav = 0.05 T = 1.14
 <-- Site =    2-   3 .. Mmps =  878 Ndav =   1 E =    -75.7284529055 Error = 1.00e-
↪19 FLOPS = 7.63e+08 Tdav = 0.01 T = 0.56
 <-- Site =    1-   2 .. Mmps =  117 Ndav =   1 E =    -75.7284529055 Error = 8.41e-
↪21 FLOPS = 7.23e+08 Tdav = 0.01 T = 0.34
 <-- Site =    0-   1 .. Mmps =   16 Ndav =   1 E =    -75.7284529055 Error = 1.71e-
↪22 FLOPS = 1.87e+07 Tdav = 0.00 T = 0.04
Time elapsed =    564.132 | E =     -75.7284581391 | DE = -3.91e-05 | DW = 1.11e-06
Time sweep =       96.647 | 134 GFLOP/SWP
 | Dmem = 107 MB (1%) | Imem = 4.19 MB (96%) | Hmem = 2.20 GB | Wmem = 836 KB | Pmem␣
↪= 73.4 MB
 | Tread = 1.918 | Twrite = 2.600 | Tfpread = 1.915 | Tfpwrite = 2.594 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 0.98 GB | cpsd = 812 MB
 | Trot = 3.880 | Tctr = 0.019 | Tint = 0.013 | Tmid = 0.000 | Tdctr = 0.025 | Tdiag␣
↪= 0.245 | Tinfo = 0.411
 | Teff = 51.855 | Tprt = 20.051 | Teig = 15.622 | Tblk = 89.360 | Tmve = 7.098 |␣
↪Tdm = 1.018 | Tsplt = 0.373 | Tsvd = 0.000 | Torth = 0.000

Sweep =   14 | Direction =  forward | Bond dimension = 1000 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
 --> Site =    0-   1 .. Mmps =    3 Ndav =   1 E =    -75.7284529055 Error = 0.
↪00e+00 FLOPS = 1.79e+07 Tdav = 0.00 T = 0.02
 --> Site =    1-   2 .. Mmps =   10 Ndav =   1 E =    -75.7284529055 Error = 0.
↪00e+00 FLOPS = 7.38e+08 Tdav = 0.01 T = 0.23
 --> Site =    2-   3 .. Mmps =   35 Ndav =   1 E =    -75.7284529055 Error = 0.
↪00e+00 FLOPS = 6.94e+08 Tdav = 0.01 T = 0.33
 --> Site =    3-   4 .. Mmps =  126 Ndav =   1 E =    -75.7284529055 Error = 0.
↪00e+00 FLOPS = 1.80e+09 Tdav = 0.03 T = 0.48
 --> Site =    4-   5 .. Mmps =  421 Ndav =   1 E =    -75.7284529055 Error = 0.
↪00e+00 FLOPS = 3.63e+09 Tdav = 0.05 T = 0.70
 --> Site =    5-   6 .. Mmps = 1000 Ndav =   2 E =    -75.7284529738 Error = 1.71e-
↪07 FLOPS = 6.80e+09 Tdav = 0.25 T = 1.48
 --> Site =    6-   7 .. Mmps = 1000 Ndav =   4 E =    -75.7284587601 Error = 1.80e-
↪06 FLOPS = 9.96e+09 Tdav = 1.22 T = 4.27
 --> Site =    7-   8 .. Mmps = 1000 Ndav =   4 E =    -75.7284553247 Error = 1.15e-
↪06 FLOPS = 9.56e+09 Tdav = 1.49 T = 6.21
 --> Site =    8-   9 .. Mmps = 1000 Ndav =   3 E =    -75.7284542008 Error = 7.90e-
↪07 FLOPS = 1.02e+10 Tdav = 1.25 T = 5.37
 --> Site =    9-  10 .. Mmps = 1000 Ndav =   3 E =    -75.7284546168 Error = 8.00e-
↪07 FLOPS = 1.03e+10 Tdav = 1.45 T = 6.36
 --> Site =   10-  11 .. Mmps = 1000 Ndav =   3 E =    -75.7284535922 Error = 4.82e-
↪07 FLOPS = 1.03e+10 Tdav = 1.68 T = 7.71
 --> Site =   11-  12 .. Mmps = 1000 Ndav =   3 E =    -75.7284530400 Error = 2.11e-
↪07 FLOPS = 1.00e+10 Tdav = 1.96 T = 13.45
```

(continues on next page)

(continued from previous page)

```
 --> Site =   12-  13 .. Mmps = 1000 Ndav =   3 E =    -75.7284528986 Error = 1.34e-
↪07 FLOPS = 1.02e+10 Tdav = 1.91 T = 14.71
 --> Site =   13-  14 .. Mmps = 1000 Ndav =   3 E =    -75.7284526735 Error = 6.38e-
↪08 FLOPS = 8.64e+09 Tdav = 1.93 T = 10.91
 --> Site =   14-  15 .. Mmps = 1000 Ndav =   2 E =    -75.7284525742 Error = 4.67e-
↪08 FLOPS = 8.18e+09 Tdav = 0.95 T = 9.61
 --> Site =   15-  16 .. Mmps = 1000 Ndav =   2 E =    -75.7284524975 Error = 3.64e-
↪08 FLOPS = 8.16e+09 Tdav = 0.59 T = 4.83
 --> Site =   16-  17 .. Mmps = 1000 Ndav =   2 E =    -75.7284524118 Error = 1.08e-
↪08 FLOPS = 7.20e+09 Tdav = 0.40 T = 3.67
 --> Site =   17-  18 .. Mmps = 1000 Ndav =   2 E =    -75.7284523921 Error = 5.46e-
↪09 FLOPS = 5.92e+09 Tdav = 0.24 T = 2.79
 --> Site =   18-  19 .. Mmps = 1000 Ndav =   1 E =    -75.7284523849 Error = 2.12e-
↪09 FLOPS = 2.32e+09 Tdav = 0.11 T = 2.54
 --> Site =   19-  20 .. Mmps = 1000 Ndav =   1 E =    -75.7284523824 Error = 9.56e-
↪10 FLOPS = 2.68e+09 Tdav = 0.03 T = 1.29
 --> Site =   20-  21 .. Mmps = 1000 Ndav =   1 E =    -75.7284523817 Error = 1.65e-
↪10 FLOPS = 1.60e+09 Tdav = 0.02 T = 0.67
 --> Site =   21-  22 .. Mmps = 1000 Ndav =   1 E =    -75.7284523816 Error = 1.18e-
↪11 FLOPS = 8.43e+08 Tdav = 0.02 T = 0.52
 --> Site =   22-  23 .. Mmps =  545 Ndav =   1 E =    -75.7284523816 Error = 5.57e-
↪21 FLOPS = 9.75e+08 Tdav = 0.01 T = 0.30
 --> Site =   23-  24 .. Mmps =  101 Ndav =   1 E =    -75.7284523816 Error = 1.30e-
↪20 FLOPS = 6.55e+08 Tdav = 0.00 T = 0.10
 --> Site =   24-  25 .. Mmps =   13 Ndav =   1 E =    -75.7284523816 Error = 7.57e-
↪21 FLOPS = 2.49e+07 Tdav = 0.00 T = 0.02
Time elapsed =    662.879 | E =     -75.7284587601 | DE = -6.21e-07 | DW = 1.80e-06
Time sweep =       98.747 | 136 GFLOP/SWP
 | Dmem = 107 MB (1%) | Imem = 4.22 MB (96%) | Hmem = 2.22 GB | Wmem = 842 KB | Pmem␣
↪= 73.9 MB
 | Tread = 3.315 | Twrite = 2.578 | Tfpread = 3.309 | Tfpwrite = 2.572 | Tmoread =␣
↪0.000 | Tasync = 0.000
 | data = 948 MB | cpsd = 757 MB
 | Trot = 3.841 | Tctr = 0.031 | Tint = 0.012 | Tmid = 0.000 | Tdctr = 1.624 | Tdiag␣
↪= 0.259 | Tinfo = 0.710
 | Teff = 52.876 | Tprt = 20.752 | Teig = 15.605 | Tblk = 91.187 | Tmve = 7.387 |␣
↪Tdm = 1.056 | Tsplt = 0.321 | Tsvd = 0.000 | Torth = 0.000

Sweep =   15 | Direction = backward | Bond dimension = 1000 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-08
 <-- Site =   24-  25 .. Mmps =    3 Ndav =   1 E =    -75.7284523816 Error = 0.
↪00e+00 FLOPS = 2.36e+07 Tdav = 0.00 T = 0.01
 <-- Site =   23-  24 .. Mmps =   10 Ndav =   1 E =    -75.7284523816 Error = 0.
↪00e+00 FLOPS = 6.86e+08 Tdav = 0.00 T = 0.05
 <-- Site =   22-  23 .. Mmps =   31 Ndav =   1 E =    -75.7284523816 Error = 0.
```

(continues on next page)

(continued from previous page)

```
↪00e+00 FLOPS = 7.96e+08 Tdav = 0.01 T = 0.19
 <-- Site =   21-  22 .. Mmps =   87 Ndav =   1 E =    -75.7284523816 Error = 0.
↪00e+00 FLOPS = 1.13e+09 Tdav = 0.01 T = 0.35
 <-- Site =   20-  21 .. Mmps =  240 Ndav =   1 E =    -75.7284523816 Error = 0.
↪00e+00 FLOPS = 1.60e+09 Tdav = 0.02 T = 0.43
 <-- Site =   19-  20 .. Mmps =  581 Ndav =   2 E =    -75.7284523826 Error = 0.
↪00e+00 FLOPS = 2.85e+09 Tdav = 0.06 T = 0.77
 <-- Site =   18-  19 .. Mmps = 1000 Ndav =   2 E =    -75.7284523853 Error = 8.65e-
↪13 FLOPS = 4.51e+09 Tdav = 0.11 T = 1.31
 <-- Site =   17-  18 .. Mmps = 1000 Ndav =   2 E =    -75.7284523924 Error = 5.31e-
↪11 FLOPS = 6.02e+09 Tdav = 0.24 T = 2.55
 <-- Site =   16-  17 .. Mmps = 1000 Ndav =   2 E =    -75.7284524143 Error = 4.63e-
↪10 FLOPS = 4.92e+09 Tdav = 0.59 T = 4.33
 <-- Site =   15-  16 .. Mmps = 1000 Ndav =   2 E =    -75.7284525190 Error = 1.56e-
↪09 FLOPS = 8.18e+09 Tdav = 0.59 T = 5.01
 <-- Site =   14-  15 .. Mmps = 1000 Ndav =   2 E =    -75.7284526414 Error = 7.46e-
↪09 FLOPS = 9.04e+09 Tdav = 0.86 T = 6.42
 <-- Site =   13-  14 .. Mmps = 1000 Ndav =   3 E =    -75.7284528326 Error = 1.76e-
↪08 FLOPS = 1.02e+10 Tdav = 1.64 T = 10.82
 <-- Site =   12-  13 .. Mmps = 1000 Ndav =   3 E =    -75.7284532372 Error = 3.64e-
↪08 FLOPS = 1.00e+10 Tdav = 1.93 T = 14.86
 <-- Site =   11-  12 .. Mmps = 1000 Ndav =   3 E =    -75.7284537826 Error = 5.62e-
↪08 FLOPS = 1.02e+10 Tdav = 1.93 T = 13.58
 <-- Site =   10-  11 .. Mmps = 1000 Ndav =   3 E =    -75.7284555385 Error = 1.90e-
↪07 FLOPS = 6.41e+09 Tdav = 2.69 T = 8.42
 <-- Site =    9-  10 .. Mmps = 1000 Ndav =   3 E =    -75.7284581756 Error = 3.83e-
↪07 FLOPS = 1.01e+10 Tdav = 1.47 T = 6.21
 <-- Site =    8-   9 .. Mmps = 1000 Ndav =   3 E =    -75.7284591921 Error = 3.50e-
↪07 FLOPS = 5.93e+09 Tdav = 2.14 T = 6.93
 <-- Site =    7-   8 .. Mmps = 1000 Ndav =   4 E =    -75.7284625688 Error = 6.14e-
↪07 FLOPS = 1.02e+10 Tdav = 1.40 T = 5.12
 <-- Site =    6-   7 .. Mmps = 1000 Ndav =   4 E =    -75.7284677459 Error = 1.16e-
↪06 FLOPS = 1.00e+10 Tdav = 1.22 T = 5.32
 <-- Site =    5-   6 .. Mmps = 1000 Ndav =   3 E =    -75.7284617497 Error = 1.37e-
↪07 FLOPS = 7.41e+09 Tdav = 0.34 T = 2.43
 <-- Site =    4-   5 .. Mmps = 1000 Ndav =   1 E =    -75.7284616787 Error = 4.03e-
↪08 FLOPS = 3.83e+09 Tdav = 0.05 T = 0.96
 <-- Site =    3-   4 .. Mmps = 1000 Ndav =   1 E =    -75.7284616787 Error = 3.68e-
↪08 FLOPS = 1.79e+09 Tdav = 0.03 T = 0.62
 <-- Site =    2-   3 .. Mmps =  876 Ndav =   1 E =    -75.7284616787 Error = 9.42e-
↪20 FLOPS = 1.26e+09 Tdav = 0.01 T = 0.32
 <-- Site =    1-   2 .. Mmps =  114 Ndav =   1 E =    -75.7284616787 Error = 1.96e-
↪20 FLOPS = 8.27e+08 Tdav = 0.01 T = 0.22
 <-- Site =    0-   1 .. Mmps =   16 Ndav =   1 E =    -75.7284616787 Error = 3.39e-
↪22 FLOPS = 3.26e+07 Tdav = 0.00 T = 0.02
```

(continues on next page)

(continued from previous page)

```
Time elapsed =    760.317 | E =     -75.7284677459 | DE = -8.99e-06 | DW = 1.16e-06
Time sweep =       97.438 | 137 GFLOP/SWP
 | Dmem = 107 MB (1%) | Imem = 4.25 MB (96%) | Hmem = 2.22 GB | Wmem = 843 KB | Pmem␣
↪= 73.8 MB
 | Tread = 1.613 | Twrite = 2.533 | Tfpread = 1.611 | Tfpwrite = 2.525 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 0.98 GB | cpsd = 811 MB
 | Trot = 3.692 | Tctr = 0.013 | Tint = 0.009 | Tmid = 0.000 | Tdctr = 0.013 | Tdiag␣
↪= 0.267 | Tinfo = 0.381
 | Teff = 50.126 | Tprt = 21.197 | Teig = 17.344 | Tblk = 90.447 | Tmve = 6.818 |␣
↪Tdm = 0.997 | Tsplt = 0.327 | Tsvd = 0.000 | Torth = 0.000

Sweep =   16 | Direction =  forward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-08
 --> Site =    0-   1 .. Mmps =    3 Ndav =   1 E =    -75.7284616787 Error = 0.
↪00e+00 FLOPS = 3.06e+07 Tdav = 0.00 T = 0.01
 --> Site =    1-   2 .. Mmps =   10 Ndav =   1 E =    -75.7284616787 Error = 0.
↪00e+00 FLOPS = 1.01e+09 Tdav = 0.01 T = 0.14
 --> Site =    2-   3 .. Mmps =   35 Ndav =   1 E =    -75.7284616787 Error = 0.
↪00e+00 FLOPS = 1.27e+09 Tdav = 0.01 T = 0.19
 --> Site =    3-   4 .. Mmps =  126 Ndav =   1 E =    -75.7284616787 Error = 0.
↪00e+00 FLOPS = 1.81e+09 Tdav = 0.03 T = 0.43
 --> Site =    4-   5 .. Mmps =  400 Ndav =   1 E =    -75.7284616787 Error = 1.88e-
↪20 FLOPS = 3.79e+09 Tdav = 0.05 T = 0.59
 --> Site =    5-   6 .. Mmps = 1000 Ndav =   2 E =    -75.7284617488 Error = 6.52e-
↪10 FLOPS = 7.02e+09 Tdav = 0.24 T = 1.02
 --> Site =    6-   7 .. Mmps = 1000 Ndav =   4 E =    -75.7284682378 Error = 1.14e-
↪06 FLOPS = 1.01e+10 Tdav = 1.25 T = 2.75
 --> Site =    7-   8 .. Mmps = 1000 Ndav =   4 E =    -75.7284659931 Error = 6.78e-
↪07 FLOPS = 6.10e+09 Tdav = 2.45 T = 4.62
 --> Site =    8-   9 .. Mmps = 1000 Ndav =   3 E =    -75.7284649892 Error = 3.86e-
↪07 FLOPS = 1.04e+10 Tdav = 1.29 T = 3.96
 --> Site =    9-  10 .. Mmps = 1000 Ndav =   3 E =    -75.7284659664 Error = 4.61e-
↪07 FLOPS = 1.05e+10 Tdav = 1.53 T = 4.31
 --> Site =   10-  11 .. Mmps = 1000 Ndav =   3 E =    -75.7284651123 Error = 2.15e-
↪07 FLOPS = 6.58e+09 Tdav = 2.84 T = 6.63
 --> Site =   11-  12 .. Mmps = 1000 Ndav =   3 E =    -75.7284646124 Error = 5.06e-
↪08 FLOPS = 9.25e+09 Tdav = 2.36 T = 10.39
 --> Site =   12-  13 .. Mmps = 1000 Ndav =   3 E =    -75.7284647790 Error = 2.67e-
↪08 FLOPS = 1.05e+10 Tdav = 2.16 T = 13.10
 --> Site =   13-  14 .. Mmps = 1000 Ndav =   3 E =    -75.7284648817 Error = 5.96e-
↪09 FLOPS = 1.05e+10 Tdav = 2.02 T = 9.64
 --> Site =   14-  15 .. Mmps = 1000 Ndav =   2 E =    -75.7284649812 Error = 6.27e-
↪10 FLOPS = 5.94e+09 Tdav = 1.86 T = 6.46
 --> Site =   15-  16 .. Mmps = 1000 Ndav =   2 E =    -75.7284650866 Error = 5.13e-
```

(continues on next page)

(continued from previous page)

```
↪10 FLOPS = 9.58e+09 Tdav = 0.84 T = 4.27
 --> Site =   16-  17 .. Mmps =  965 Ndav =   2 E =    -75.7284651314 Error = 6.82e-
↪20 FLOPS = 8.44e+09 Tdav = 0.67 T = 3.14
 --> Site =   17-  18 .. Mmps =  951 Ndav =   2 E =    -75.7284651428 Error = 4.34e-
↪20 FLOPS = 8.17e+09 Tdav = 0.39 T = 2.27
 --> Site =   18-  19 .. Mmps =  685 Ndav =   1 E =    -75.7284651428 Error = 3.40e-
↪20 FLOPS = 6.80e+09 Tdav = 0.11 T = 1.36
 --> Site =   19-  20 .. Mmps =  405 Ndav =   1 E =    -75.7284651428 Error = 1.65e-
↪20 FLOPS = 2.50e+09 Tdav = 0.07 T = 0.85
 --> Site =   20-  21 .. Mmps =  223 Ndav =   1 E =    -75.7284651428 Error = 1.59e-
↪20 FLOPS = 8.39e+08 Tdav = 0.03 T = 0.49
 --> Site =   21-  22 .. Mmps =   90 Ndav =   1 E =    -75.7284651428 Error = 8.10e-
↪21 FLOPS = 1.97e+08 Tdav = 0.01 T = 0.22
 --> Site =   22-  23 .. Mmps =   33 Ndav =   1 E =    -75.7284651428 Error = 8.89e-
↪21 FLOPS = 2.25e+07 Tdav = 0.00 T = 0.08
 --> Site =   23-  24 .. Mmps =   14 Ndav =   1 E =    -75.7284651428 Error = 4.67e-
↪21 FLOPS = 3.41e+06 Tdav = 0.00 T = 0.03
 --> Site =   24-  25 .. Mmps =    5 Ndav =   1 E =    -75.7284651428 Error = 1.10e-
↪20 FLOPS = 3.41e+05 Tdav = 0.00 T = 0.01
Time elapsed =    837.468 | E =     -75.7284682378 | DE = -4.92e-07 | DW = 1.14e-06
Time sweep =       77.151 | 160 GFLOP/SWP
 | Dmem = 107 MB (1%) | Imem = 4.35 MB (96%) | Hmem = 2.29 GB | Wmem = 868 KB | Pmem␣
↪= 0 B
 | Tread = 3.232 | Twrite = 2.558 | Tfpread = 3.224 | Tfpwrite = 2.551 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 896 MB | cpsd = 718 MB
 | Trot = 3.896 | Tctr = 0.030 | Tint = 0.010 | Tmid = 0.000 | Tdctr = 1.684 | Tdiag␣
↪= 0.324 | Tinfo = 0.721
 | Teff = 48.552 | Tprt = 0.000 | Teig = 20.206 | Tblk = 69.477 | Tmve = 7.492 | Tdm␣
↪= 0.012 | Tsplt = 0.300 | Tsvd = 0.000 | Torth = 0.000

Sweep =   17 | Direction = backward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-08
 <-- Site =   24-  25 .. Mmps =    3 Ndav =   1 E =    -75.7284651428 Error = 0.
↪00e+00 FLOPS = 2.15e+05 Tdav = 0.00 T = 0.01
 <-- Site =   23-  24 .. Mmps =   10 Ndav =   1 E =    -75.7284651428 Error = 0.
↪00e+00 FLOPS = 5.75e+05 Tdav = 0.02 T = 0.04
 <-- Site =   22-  23 .. Mmps =   31 Ndav =   1 E =    -75.7284651428 Error = 0.
↪00e+00 FLOPS = 2.18e+07 Tdav = 0.00 T = 0.05
 <-- Site =   21-  22 .. Mmps =   87 Ndav =   1 E =    -75.7284651428 Error = 0.
↪00e+00 FLOPS = 2.05e+08 Tdav = 0.01 T = 0.17
 <-- Site =   20-  21 .. Mmps =  222 Ndav =   1 E =    -75.7284651428 Error = 7.00e-
↪28 FLOPS = 8.02e+08 Tdav = 0.03 T = 0.33
 <-- Site =   19-  20 .. Mmps =  401 Ndav =   1 E =    -75.7284651428 Error = 2.02e-
↪26 FLOPS = 2.49e+09 Tdav = 0.06 T = 0.85
```

(continues on next page)

(continued from previous page)

```
 <-- Site =   18-  19 .. Mmps =  679 Ndav =   1 E =    -75.7284651428 Error = 1.50e-
↪25 FLOPS = 6.77e+09 Tdav = 0.10 T = 1.38
 <-- Site =   17-  18 .. Mmps =  943 Ndav =   1 E =    -75.7284651428 Error = 7.26e-
↪24 FLOPS = 7.42e+09 Tdav = 0.20 T = 1.53
 <-- Site =   16-  17 .. Mmps =  959 Ndav =   1 E =    -75.7284651428 Error = 1.14e-
↪22 FLOPS = 8.40e+09 Tdav = 0.32 T = 2.37
 <-- Site =   15-  16 .. Mmps = 1000 Ndav =   2 E =    -75.7284651480 Error = 5.14e-
↪10 FLOPS = 9.15e+09 Tdav = 0.84 T = 3.35
 <-- Site =   14-  15 .. Mmps = 1000 Ndav =   2 E =    -75.7284651491 Error = 6.06e-
↪10 FLOPS = 7.31e+09 Tdav = 1.49 T = 6.78
 <-- Site =   13-  14 .. Mmps = 1000 Ndav =   2 E =    -75.7284651954 Error = 6.11e-
↪09 FLOPS = 1.02e+10 Tdav = 1.29 T = 7.81
 <-- Site =   12-  13 .. Mmps = 1000 Ndav =   3 E =    -75.7284653506 Error = 2.68e-
↪08 FLOPS = 1.05e+10 Tdav = 2.07 T = 13.10
 <-- Site =   11-  12 .. Mmps = 1000 Ndav =   3 E =    -75.7284655124 Error = 4.89e-
↪08 FLOPS = 1.05e+10 Tdav = 2.04 T = 11.76
 <-- Site =   10-  11 .. Mmps = 1000 Ndav =   3 E =    -75.7284666297 Error = 2.13e-
↪07 FLOPS = 8.86e+09 Tdav = 2.08 T = 5.47
 <-- Site =    9-  10 .. Mmps = 1000 Ndav =   3 E =    -75.7284682020 Error = 4.56e-
↪07 FLOPS = 1.04e+10 Tdav = 1.53 T = 5.40
 <-- Site =    8-   9 .. Mmps = 1000 Ndav =   3 E =    -75.7284677680 Error = 3.84e-
↪07 FLOPS = 1.04e+10 Tdav = 1.29 T = 3.69
 <-- Site =    7-   8 .. Mmps = 1000 Ndav =   4 E =    -75.7284694920 Error = 6.79e-
↪07 FLOPS = 1.05e+10 Tdav = 1.42 T = 3.50
 <-- Site =    6-   7 .. Mmps = 1000 Ndav =   4 E =    -75.7284724833 Error = 1.16e-
↪06 FLOPS = 6.96e+09 Tdav = 1.82 T = 4.40
 <-- Site =    5-   6 .. Mmps = 1000 Ndav =   2 E =    -75.7284659031 Error = 6.95e-
↪10 FLOPS = 7.29e+09 Tdav = 0.23 T = 1.16
 <-- Site =    4-   5 .. Mmps =  440 Ndav =   1 E =    -75.7284658998 Error = 4.79e-
↪19 FLOPS = 3.68e+09 Tdav = 0.05 T = 0.75
 <-- Site =    3-   4 .. Mmps =  195 Ndav =   1 E =    -75.7284658998 Error = 4.46e-
↪19 FLOPS = 8.13e+08 Tdav = 0.01 T = 0.26
 <-- Site =    2-   3 .. Mmps =   72 Ndav =   1 E =    -75.7284658998 Error = 1.22e-
↪19 FLOPS = 1.56e+08 Tdav = 0.00 T = 0.09
 <-- Site =    1-   2 .. Mmps =   28 Ndav =   1 E =    -75.7284658998 Error = 1.72e-
↪20 FLOPS = 2.95e+07 Tdav = 0.00 T = 0.03
 <-- Site =    0-   1 .. Mmps =    7 Ndav =   1 E =    -75.7284658998 Error = 9.44e-
↪33 FLOPS = 2.17e+06 Tdav = 0.00 T = 0.01
Time elapsed =    911.909 | E =     -75.7284724833 | DE = -4.25e-06 | DW = 1.16e-06
Time sweep =       74.441 | 146 GFLOP/SWP
 | Dmem = 105 MB (1%) | Imem = 4.38 MB (96%) | Hmem = 2.26 GB | Wmem = 868 KB | Pmem␣
↪= 0 B
 | Tread = 1.613 | Twrite = 2.349 | Tfpread = 1.610 | Tfpwrite = 2.342 | Tmoread =␣
↪0.000 | Tasync = 0.000
 | data = 897 MB | cpsd = 722 MB
```

(continues on next page)

(continued from previous page)

```
 | Trot = 3.363 | Tctr = 0.011 | Tint = 0.010 | Tmid = 0.000 | Tdctr = 0.019 | Tdiag␣
↪= 0.301 | Tinfo = 0.374
 | Teff = 50.472 | Tprt = 0.000 | Teig = 16.908 | Tblk = 68.016 | Tmve = 6.263 | Tdm␣
↪= 0.012 | Tsplt = 0.272 | Tsvd = 0.000 | Torth = 0.000

Sweep =   18 | Direction =  forward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-08
 --> Site =    0-   1 .. Mmps =    3 Ndav =   1 E =    -75.7284658998 Error = 0.
↪00e+00 FLOPS = 1.13e+06 Tdav = 0.00 T = 0.01
 --> Site =    1-   2 .. Mmps =   10 Ndav =   1 E =    -75.7284658998 Error = 0.
↪00e+00 FLOPS = 2.79e+07 Tdav = 0.00 T = 0.02
 --> Site =    2-   3 .. Mmps =   35 Ndav =   1 E =    -75.7284658998 Error = 0.
↪00e+00 FLOPS = 1.49e+08 Tdav = 0.00 T = 0.06
 --> Site =    3-   4 .. Mmps =  126 Ndav =   1 E =    -75.7284658998 Error = 0.
↪00e+00 FLOPS = 8.26e+08 Tdav = 0.01 T = 0.18
 --> Site =    4-   5 .. Mmps =  405 Ndav =   1 E =    -75.7284658998 Error = 1.17e-
↪20 FLOPS = 3.76e+09 Tdav = 0.05 T = 0.63
 --> Site =    5-   6 .. Mmps = 1000 Ndav =   2 E =    -75.7284659055 Error = 6.49e-
↪10 FLOPS = 7.36e+09 Tdav = 0.23 T = 1.01
 --> Site =    6-   7 .. Mmps = 1000 Ndav =   4 E =    -75.7284724840 Error = 1.16e-
↪06 FLOPS = 9.92e+09 Tdav = 1.27 T = 2.77
 --> Site =    7-   8 .. Mmps = 1000 Ndav =   4 E =    -75.7284698798 Error = 6.89e-
↪07 FLOPS = 7.80e+09 Tdav = 1.90 T = 3.91
 --> Site =    8-   9 .. Mmps = 1000 Ndav =   3 E =    -75.7284683562 Error = 3.90e-
↪07 FLOPS = 1.02e+10 Tdav = 1.31 T = 4.71
 --> Site =    9-  10 .. Mmps = 1000 Ndav =   3 E =    -75.7284689591 Error = 4.67e-
↪07 FLOPS = 1.04e+10 Tdav = 1.52 T = 4.21
 --> Site =   10-  11 .. Mmps = 1000 Ndav =   3 E =    -75.7284675418 Error = 2.18e-
↪07 FLOPS = 6.64e+09 Tdav = 2.77 T = 6.12
 --> Site =   11-  12 .. Mmps = 1000 Ndav =   3 E =    -75.7284665194 Error = 5.02e-
↪08 FLOPS = 1.04e+10 Tdav = 2.06 T = 10.60
 --> Site =   12-  13 .. Mmps = 1000 Ndav =   3 E =    -75.7284663879 Error = 2.71e-
↪08 FLOPS = 1.05e+10 Tdav = 2.05 T = 13.22
 --> Site =   13-  14 .. Mmps = 1000 Ndav =   2 E =    -75.7284662465 Error = 6.23e-
↪09 FLOPS = 1.02e+10 Tdav = 1.29 T = 8.93
 --> Site =   14-  15 .. Mmps = 1000 Ndav =   2 E =    -75.7284662060 Error = 6.23e-
↪10 FLOPS = 9.93e+09 Tdav = 1.09 T = 5.47
 --> Site =   15-  16 .. Mmps = 1000 Ndav =   2 E =    -75.7284662074 Error = 5.48e-
↪10 FLOPS = 9.34e+09 Tdav = 0.83 T = 5.36
 --> Site =   16-  17 .. Mmps =  967 Ndav =   1 E =    -75.7284662024 Error = 3.90e-
↪20 FLOPS = 8.47e+09 Tdav = 0.32 T = 2.68
 --> Site =   17-  18 .. Mmps =  950 Ndav =   1 E =    -75.7284662024 Error = 4.26e-
↪20 FLOPS = 8.44e+09 Tdav = 0.18 T = 1.86
 --> Site =   18-  19 .. Mmps =  697 Ndav =   2 E =    -75.7284662036 Error = 3.05e-
↪20 FLOPS = 6.95e+09 Tdav = 0.20 T = 1.34
```

(continues on next page)

(continued from previous page)

```
 --> Site =   19-  20 .. Mmps =  405 Ndav =   1 E =    -75.7284662036 Error = 1.72e-
↪20 FLOPS = 4.05e+09 Tdav = 0.04 T = 0.67
 --> Site =   20-  21 .. Mmps =  224 Ndav =   1 E =    -75.7284662036 Error = 1.54e-
↪20 FLOPS = 1.61e+09 Tdav = 0.01 T = 0.26
 --> Site =   21-  22 .. Mmps =   88 Ndav =   1 E =    -75.7284662036 Error = 1.27e-
↪20 FLOPS = 3.63e+08 Tdav = 0.01 T = 0.13
 --> Site =   22-  23 .. Mmps =   32 Ndav =   1 E =    -75.7284662036 Error = 5.37e-
↪21 FLOPS = 4.01e+07 Tdav = 0.00 T = 0.05
 --> Site =   23-  24 .. Mmps =   12 Ndav =   1 E =    -75.7284662036 Error = 9.11e-
↪21 FLOPS = 6.01e+06 Tdav = 0.00 T = 0.02
 --> Site =   24-  25 .. Mmps =    6 Ndav =   1 E =    -75.7284662036 Error = 1.38e-
↪23 FLOPS = 4.06e+05 Tdav = 0.00 T = 0.01
Time elapsed =    986.291 | E =     -75.7284724840 | DE = -6.84e-10 | DW = 1.16e-06
Time sweep =       74.383 | 147 GFLOP/SWP
 | Dmem = 105 MB (1%) | Imem = 4.38 MB (96%) | Hmem = 2.26 GB | Wmem = 868 KB | Pmem␣
↪= 0 B
 | Tread = 2.960 | Twrite = 2.410 | Tfpread = 2.950 | Tfpwrite = 2.404 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 897 MB | cpsd = 718 MB
 | Trot = 3.358 | Tctr = 0.030 | Tint = 0.009 | Tmid = 0.000 | Tdctr = 1.426 | Tdiag␣
↪= 0.266 | Tinfo = 0.664
 | Teff = 49.805 | Tprt = 0.000 | Teig = 17.154 | Tblk = 67.566 | Tmve = 6.669 | Tdm␣
↪= 0.012 | Tsplt = 0.276 | Tsvd = 0.000 | Torth = 0.000

Sweep =   19 | Direction = backward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-08
 <-- Site =   24-  25 .. Mmps =    3 Ndav =   1 E =    -75.7284662036 Error = 0.
↪00e+00 FLOPS = 2.74e+05 Tdav = 0.00 T = 0.01
 <-- Site =   23-  24 .. Mmps =   10 Ndav =   1 E =    -75.7284662036 Error = 0.
↪00e+00 FLOPS = 6.05e+06 Tdav = 0.00 T = 0.01
 <-- Site =   22-  23 .. Mmps =   31 Ndav =   1 E =    -75.7284662036 Error = 0.
↪00e+00 FLOPS = 3.90e+07 Tdav = 0.00 T = 0.03
 <-- Site =   21-  22 .. Mmps =   87 Ndav =   1 E =    -75.7284662036 Error = 0.
↪00e+00 FLOPS = 3.27e+08 Tdav = 0.01 T = 0.11
 <-- Site =   20-  21 .. Mmps =  221 Ndav =   1 E =    -75.7284662036 Error = 1.32e-
↪27 FLOPS = 1.62e+09 Tdav = 0.01 T = 0.19
 <-- Site =   19-  20 .. Mmps =  400 Ndav =   1 E =    -75.7284662036 Error = 2.59e-
↪26 FLOPS = 3.95e+09 Tdav = 0.04 T = 0.51
 <-- Site =   18-  19 .. Mmps =  690 Ndav =   1 E =    -75.7284662036 Error = 2.07e-
↪25 FLOPS = 5.99e+09 Tdav = 0.12 T = 1.04
 <-- Site =   17-  18 .. Mmps =  943 Ndav =   1 E =    -75.7284662036 Error = 1.01e-
↪23 FLOPS = 5.25e+09 Tdav = 0.28 T = 2.60
 <-- Site =   16-  17 .. Mmps =  959 Ndav =   1 E =    -75.7284662036 Error = 7.85e-
↪23 FLOPS = 8.36e+09 Tdav = 0.32 T = 2.86
 <-- Site =   15-  16 .. Mmps = 1000 Ndav =   2 E =    -75.7284662086 Error = 5.45e-
```

(continues on next page)

(continued from previous page)

```
↪10 FLOPS = 9.27e+09 Tdav = 0.83 T = 3.34
 <-- Site =   14-  15 .. Mmps = 1000 Ndav =   2 E =    -75.7284662090 Error = 6.23e-
↪10 FLOPS = 9.92e+09 Tdav = 1.09 T = 5.16
 <-- Site =   13-  14 .. Mmps = 1000 Ndav =   2 E =    -75.7284662528 Error = 6.20e-
↪09 FLOPS = 1.03e+10 Tdav = 1.28 T = 8.74
 <-- Site =   12-  13 .. Mmps = 1000 Ndav =   3 E =    -75.7284664059 Error = 2.75e-
↪08 FLOPS = 1.05e+10 Tdav = 2.06 T = 13.07
 <-- Site =   11-  12 .. Mmps = 1000 Ndav =   3 E =    -75.7284665605 Error = 5.05e-
↪08 FLOPS = 9.21e+09 Tdav = 2.32 T = 11.59
 <-- Site =   10-  11 .. Mmps = 1000 Ndav =   3 E =    -75.7284677844 Error = 2.22e-
↪07 FLOPS = 1.04e+10 Tdav = 1.78 T = 5.09
 <-- Site =    9-  10 .. Mmps = 1000 Ndav =   3 E =    -75.7284694053 Error = 4.75e-
↪07 FLOPS = 7.20e+09 Tdav = 2.20 T = 5.01
 <-- Site =    8-   9 .. Mmps = 1000 Ndav =   3 E =    -75.7284688173 Error = 3.92e-
↪07 FLOPS = 1.03e+10 Tdav = 1.30 T = 4.44
 <-- Site =    7-   8 .. Mmps = 1000 Ndav =   4 E =    -75.7284705035 Error = 6.97e-
↪07 FLOPS = 1.05e+10 Tdav = 1.40 T = 3.46
 <-- Site =    6-   7 .. Mmps = 1000 Ndav =   4 E =    -75.7284733043 Error = 1.17e-
↪06 FLOPS = 1.03e+10 Tdav = 1.23 T = 2.79
 <-- Site =    5-   6 .. Mmps = 1000 Ndav =   3 E =    -75.7284666776 Error = 7.38e-
↪10 FLOPS = 7.19e+09 Tdav = 0.35 T = 1.28
 <-- Site =    4-   5 .. Mmps =  442 Ndav =   1 E =    -75.7284666741 Error = 4.38e-
↪19 FLOPS = 2.10e+09 Tdav = 0.09 T = 1.34
 <-- Site =    3-   4 .. Mmps =  184 Ndav =   1 E =    -75.7284666741 Error = 4.13e-
↪19 FLOPS = 4.48e+08 Tdav = 0.02 T = 0.45
 <-- Site =    2-   3 .. Mmps =   73 Ndav =   1 E =    -75.7284666741 Error = 1.17e-
↪19 FLOPS = 7.30e+07 Tdav = 0.01 T = 0.15
 <-- Site =    1-   2 .. Mmps =   30 Ndav =   1 E =    -75.7284666741 Error = 1.15e-
↪20 FLOPS = 1.59e+07 Tdav = 0.00 T = 0.05
 <-- Site =    0-   1 .. Mmps =    7 Ndav =   1 E =    -75.7284666741 Error = 2.94e-
↪21 FLOPS = 1.39e+06 Tdav = 0.00 T = 0.01
Time elapsed =   1059.782 | E =     -75.7284733043 | DE = -8.20e-07 | DW = 1.17e-06
Time sweep =       73.491 | 147 GFLOP/SWP
 | Dmem = 105 MB (1%) | Imem = 4.38 MB (96%) | Hmem = 2.26 GB | Wmem = 868 KB | Pmem␣
↪= 0 B
 | Tread = 1.571 | Twrite = 2.384 | Tfpread = 1.569 | Tfpwrite = 2.378 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 897 MB | cpsd = 722 MB
 | Trot = 3.762 | Tctr = 0.019 | Tint = 0.010 | Tmid = 0.000 | Tdctr = 0.015 | Tdiag␣
↪= 0.293 | Tinfo = 0.418
 | Teff = 49.124 | Tprt = 0.000 | Teig = 16.764 | Tblk = 66.518 | Tmve = 6.799 | Tdm␣
↪= 0.012 | Tsplt = 0.291 | Tsvd = 0.000 | Torth = 0.000


Sweep =   20 | Direction = backward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
```

(continues on next page)

(continued from previous page)

```
↪Dav threshold =  1.00e-09
 <-- Site =    0 .. Mmps =    2 Ndav =   1 E =    -75.7284666741 Error = 5.29e-50␣
↪FLOPS = 7.61e+04 Tdav = 0.00 T = 0.01
Time elapsed =   1059.812 | E =     -75.7284666741 | DE = 6.63e-06 | DW = 5.29e-50
Time sweep =        0.030 | 181 FLOP/SWP
 | Dmem = 1.19 KB (4%) | Imem = 752 B (73%) | Hmem = 1.33 KB | Wmem = 56 B | Pmem =␣
↪0 B
 | Tread = 0.001 | Twrite = 0.001 | Tfpread = 0.001 | Tfpwrite = 0.001 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 1.13 KB | cpsd = 904 B
 | Trot = 0.000 | Tctr = 0.000 | Tint = 0.000 | Tmid = 0.000 | Tdctr = 0.000 | Tdiag␣
↪= 0.001 | Tinfo = 0.001
 | Teff = 0.002 | Tprt = 0.000 | Teig = 0.002 | Tblk = 0.008 | Tmve = 0.000 | Tdm =␣
↪0.000 | Tsplt = 0.000 | Tsvd = 0.000 | Torth = 0.000

Sweep =   21 | Direction =  forward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
 --> Site =    0 .. Mmps =    3 Ndav =   1 E =    -75.7284666741 Error = 0.00e+00␣
↪FLOPS = 7.69e+04 Tdav = 0.00 T = 0.01
 --> Site =    1 .. Mmps =   10 Ndav =   1 E =    -75.7284666741 Error = 0.00e+00␣
↪FLOPS = 1.83e+06 Tdav = 0.00 T = 0.01
 --> Site =    2 .. Mmps =   35 Ndav =   1 E =    -75.7284666741 Error = 0.00e+00␣
↪FLOPS = 8.83e+06 Tdav = 0.00 T = 0.02
 --> Site =    3 .. Mmps =  126 Ndav =   1 E =    -75.7284666741 Error = 0.00e+00␣
↪FLOPS = 2.06e+07 Tdav = 0.01 T = 0.05
 --> Site =    4 .. Mmps =  402 Ndav =   1 E =    -75.7284666741 Error = 1.12e-20␣
↪FLOPS = 2.28e+08 Tdav = 0.03 T = 0.17
 --> Site =    5 .. Mmps = 1000 Ndav =   2 E =    -75.7284666743 Error = 8.67e-18␣
↪FLOPS = 2.10e+09 Tdav = 0.14 T = 0.61
 --> Site =    6 .. Mmps = 1000 Ndav =   1 E =    -75.7284666743 Error = 5.59e-18␣
↪FLOPS = 7.78e+09 Tdav = 0.12 T = 1.32
 --> Site =    7 .. Mmps = 1000 Ndav =   3 E =    -75.7284667238 Error = 2.54e-18␣
↪FLOPS = 8.87e+09 Tdav = 0.44 T = 1.57
 --> Site =    8 .. Mmps = 1000 Ndav =   3 E =    -75.7284667587 Error = 4.49e-18␣
↪FLOPS = 8.54e+09 Tdav = 0.56 T = 2.01
 --> Site =    9 .. Mmps = 1000 Ndav =   3 E =    -75.7284669171 Error = 2.15e-18␣
↪FLOPS = 9.15e+09 Tdav = 0.66 T = 2.26
 --> Site =   10 .. Mmps = 1000 Ndav =   3 E =    -75.7284669447 Error = 3.23e-18␣
↪FLOPS = 8.70e+09 Tdav = 0.85 T = 2.70
 --> Site =   11 .. Mmps = 1000 Ndav =   3 E =    -75.7284669720 Error = 5.17e-18␣
↪FLOPS = 5.37e+09 Tdav = 1.65 T = 4.84
 --> Site =   12 .. Mmps = 1000 Ndav =   3 E =    -75.7284669921 Error = 1.92e-18␣
↪FLOPS = 9.11e+09 Tdav = 1.03 T = 7.74
 --> Site =   13 .. Mmps = 1000 Ndav =   2 E =    -75.7284670000 Error = 1.47e-17␣
↪FLOPS = 5.60e+09 Tdav = 1.08 T = 5.90
```

(continues on next page)

(continued from previous page)

```
 --> Site =   14 .. Mmps = 1000 Ndav =   2 E =    -75.7284670024 Error = 8.33e-18␣
↪FLOPS = 9.27e+09 Tdav = 0.54 T = 3.72
 --> Site =   15 .. Mmps = 1000 Ndav =   2 E =    -75.7284670037 Error = 5.26e-19␣
↪FLOPS = 8.20e+09 Tdav = 0.52 T = 2.63
 --> Site =   16 .. Mmps =  963 Ndav =   2 E =    -75.7284670042 Error = 5.67e-20␣
↪FLOPS = 7.83e+09 Tdav = 0.38 T = 1.99
 --> Site =   17 .. Mmps =  949 Ndav =   2 E =    -75.7284670047 Error = 5.48e-20␣
↪FLOPS = 7.84e+09 Tdav = 0.28 T = 1.71
 --> Site =   18 .. Mmps =  694 Ndav =   1 E =    -75.7284670047 Error = 3.95e-20␣
↪FLOPS = 3.59e+09 Tdav = 0.17 T = 1.74
 --> Site =   19 .. Mmps =  405 Ndav =   1 E =    -75.7284670047 Error = 2.14e-20␣
↪FLOPS = 2.21e+09 Tdav = 0.07 T = 1.05
 --> Site =   20 .. Mmps =  224 Ndav =   1 E =    -75.7284670047 Error = 2.30e-20␣
↪FLOPS = 8.61e+08 Tdav = 0.03 T = 0.44
 --> Site =   21 .. Mmps =   88 Ndav =   1 E =    -75.7284670047 Error = 2.21e-20␣
↪FLOPS = 2.33e+08 Tdav = 0.01 T = 0.23
 --> Site =   22 .. Mmps =   32 Ndav =   1 E =    -75.7284670047 Error = 1.80e-20␣
↪FLOPS = 2.18e+07 Tdav = 0.00 T = 0.08
 --> Site =   23 .. Mmps =   12 Ndav =   1 E =    -75.7284670047 Error = 4.12e-21␣
↪FLOPS = 3.25e+06 Tdav = 0.00 T = 0.03
 --> Site =   24 .. Mmps =    4 Ndav =   1 E =    -75.7284670047 Error = 1.03e-20␣
↪FLOPS = 2.52e+05 Tdav = 0.00 T = 0.02
 --> Site =   25 .. Mmps =    2 Ndav =   1 E =    -75.7284670047 Error = 0.00e+00␣
↪FLOPS = 2.78e+04 Tdav = 0.00 T = 0.01
Time elapsed =   1102.836 | E =     -75.7284670047 | DE = -3.31e-07 | DW = 1.47e-17
Time sweep =       43.023 | 58.5 GFLOP/SWP
 | Dmem = 104 MB (0%) | Imem = 2.59 MB (94%) | Hmem = 1.22 GB | Wmem = 434 KB | Pmem␣
↪= 0 B
 | Tread = 3.210 | Twrite = 2.515 | Tfpread = 3.193 | Tfpwrite = 2.499 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 896 MB | cpsd = 718 MB
 | Trot = 3.370 | Tctr = 0.028 | Tint = 0.013 | Tmid = 0.000 | Tdctr = 1.624 | Tdiag␣
↪= 0.221 | Tinfo = 0.771
 | Teff = 26.925 | Tprt = 0.000 | Teig = 8.580 | Tblk = 35.965 | Tmve = 6.878 | Tdm␣
↪= 0.010 | Tsplt = 0.291 | Tsvd = 0.000 | Torth = 0.000

Sweep =   22 | Direction = backward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
 <-- Site =   25 .. Mmps =    3 Ndav =   1 E =    -75.7284670047 Error = 0.00e+00␣
↪FLOPS = 1.23e+04 Tdav = 0.01 T = 0.01
 <-- Site =   24 .. Mmps =   10 Ndav =   1 E =    -75.7284670047 Error = 0.00e+00␣
↪FLOPS = 2.42e+05 Tdav = 0.00 T = 0.01
 <-- Site =   23 .. Mmps =   31 Ndav =   1 E =    -75.7284670047 Error = 0.00e+00␣
↪FLOPS = 3.27e+06 Tdav = 0.00 T = 0.02
 <-- Site =   22 .. Mmps =   87 Ndav =   1 E =    -75.7284670047 Error = 0.00e+00␣
```

(continues on next page)

(continued from previous page)

```
↪FLOPS = 2.08e+07 Tdav = 0.00 T = 0.05
 <-- Site =   21 .. Mmps =  221 Ndav =   1 E =    -75.7284670047 Error = 5.20e-28␣
↪FLOPS = 1.44e+08 Tdav = 0.02 T = 0.17
 <-- Site =   20 .. Mmps =  400 Ndav =   1 E =    -75.7284670047 Error = 2.30e-26␣
↪FLOPS = 1.41e+09 Tdav = 0.02 T = 0.23
 <-- Site =   19 .. Mmps =  690 Ndav =   1 E =    -75.7284670047 Error = 1.43e-25␣
↪FLOPS = 3.79e+09 Tdav = 0.04 T = 0.54
 <-- Site =   18 .. Mmps =  943 Ndav =   1 E =    -75.7284670047 Error = 1.02e-23␣
↪FLOPS = 6.52e+09 Tdav = 0.10 T = 1.05
 <-- Site =   17 .. Mmps =  959 Ndav =   1 E =    -75.7284670047 Error = 1.15e-22␣
↪FLOPS = 7.73e+09 Tdav = 0.14 T = 1.64
 <-- Site =   16 .. Mmps = 1000 Ndav =   1 E =    -75.7284670047 Error = 2.86e-22␣
↪FLOPS = 7.40e+09 Tdav = 0.20 T = 2.46
 <-- Site =   15 .. Mmps = 1000 Ndav =   1 E =    -75.7284670047 Error = 5.20e-21␣
↪FLOPS = 8.13e+09 Tdav = 0.26 T = 2.90
 <-- Site =   14 .. Mmps = 1000 Ndav =   1 E =    -75.7284670047 Error = 5.34e-21␣
↪FLOPS = 8.36e+09 Tdav = 0.30 T = 5.29
 <-- Site =   13 .. Mmps = 1000 Ndav =   2 E =    -75.7284670048 Error = 7.05e-21␣
↪FLOPS = 9.10e+09 Tdav = 0.67 T = 6.38
 <-- Site =   12 .. Mmps = 1000 Ndav =   2 E =    -75.7284670054 Error = 1.90e-20␣
↪FLOPS = 8.89e+09 Tdav = 0.70 T = 11.64
 <-- Site =   11 .. Mmps = 1000 Ndav =   2 E =    -75.7284670066 Error = 3.96e-20␣
↪FLOPS = 8.67e+09 Tdav = 0.68 T = 10.25
 <-- Site =   10 .. Mmps = 1000 Ndav =   2 E =    -75.7284670125 Error = 6.95e-20␣
↪FLOPS = 8.81e+09 Tdav = 0.56 T = 3.88
 <-- Site =    9 .. Mmps = 1000 Ndav =   2 E =    -75.7284670245 Error = 1.27e-19␣
↪FLOPS = 8.54e+09 Tdav = 0.47 T = 3.14
 <-- Site =    8 .. Mmps = 1000 Ndav =   3 E =    -75.7284670554 Error = 2.05e-19␣
↪FLOPS = 6.23e+09 Tdav = 0.77 T = 4.56
 <-- Site =    7 .. Mmps = 1000 Ndav =   3 E =    -75.7284671268 Error = 6.81e-19␣
↪FLOPS = 8.59e+09 Tdav = 0.45 T = 2.49
 <-- Site =    6 .. Mmps = 1000 Ndav =   3 E =    -75.7284673355 Error = 4.94e-19␣
↪FLOPS = 8.07e+09 Tdav = 0.34 T = 1.98
 <-- Site =    5 .. Mmps =  437 Ndav =   2 E =    -75.7284673359 Error = 4.54e-19␣
↪FLOPS = 3.56e+09 Tdav = 0.08 T = 1.02
 <-- Site =    4 .. Mmps =  194 Ndav =   1 E =    -75.7284673359 Error = 4.09e-19␣
↪FLOPS = 4.37e+08 Tdav = 0.02 T = 0.62
 <-- Site =    3 .. Mmps =   72 Ndav =   1 E =    -75.7284673359 Error = 1.09e-19␣
↪FLOPS = 4.33e+07 Tdav = 0.01 T = 0.19
 <-- Site =    2 .. Mmps =   26 Ndav =   1 E =    -75.7284673359 Error = 1.96e-20␣
↪FLOPS = 1.51e+07 Tdav = 0.00 T = 0.05
 <-- Site =    1 .. Mmps =   10 Ndav =   1 E =    -75.7284673359 Error = 3.13e-35␣
↪FLOPS = 2.44e+06 Tdav = 0.00 T = 0.02
 <-- Site =    0 .. Mmps =    2 Ndav =   1 E =    -75.7284673359 Error = 1.17e-49␣
↪FLOPS = 2.41e+05 Tdav = 0.00 T = 0.01
```

(continues on next page)

(continued from previous page)

```
Time elapsed =   1163.616 | E =     -75.7284673359 | DE = -3.31e-07 | DW = 6.81e-19
Time sweep =       60.780 | 43.7 GFLOP/SWP
 | Dmem = 104 MB (0%) | Imem = 2.59 MB (94%) | Hmem = 1.22 GB | Wmem = 434 KB | Pmem␣
↪= 0 B
 | Tread = 2.898 | Twrite = 2.629 | Tfpread = 2.891 | Tfpwrite = 2.622 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 897 MB | cpsd = 722 MB
 | Trot = 3.761 | Tctr = 22.210 | Tint = 0.010 | Tmid = 0.000 | Tdctr = 0.014 |␣
↪Tdiag = 0.188 | Tinfo = 0.585
 | Teff = 23.670 | Tprt = 0.000 | Teig = 5.833 | Tblk = 29.905 | Tmve = 30.707 | Tdm␣
↪= 0.009 | Tsplt = 0.257 | Tsvd = 0.000 | Torth = 0.000

Sweep =   23 | Direction =  forward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
 --> Site =    0 .. Mmps =    3 Ndav =   1 E =    -75.7284673359 Error = 0.00e+00␣
↪FLOPS = 1.85e+05 Tdav = 0.00 T = 0.01
 --> Site =    1 .. Mmps =   10 Ndav =   1 E =    -75.7284673359 Error = 0.00e+00␣
↪FLOPS = 2.36e+06 Tdav = 0.00 T = 0.01
 --> Site =    2 .. Mmps =   35 Ndav =   1 E =    -75.7284673359 Error = 0.00e+00␣
↪FLOPS = 1.52e+07 Tdav = 0.00 T = 0.01
 --> Site =    3 .. Mmps =  126 Ndav =   1 E =    -75.7284673359 Error = 0.00e+00␣
↪FLOPS = 4.40e+07 Tdav = 0.01 T = 0.03
 --> Site =    4 .. Mmps =  403 Ndav =   1 E =    -75.7284673359 Error = 9.76e-21␣
↪FLOPS = 4.36e+08 Tdav = 0.02 T = 0.10
 --> Site =    5 .. Mmps = 1000 Ndav =   1 E =    -75.7284673359 Error = 5.70e-18␣
↪FLOPS = 3.67e+09 Tdav = 0.04 T = 0.33
 --> Site =    6 .. Mmps = 1000 Ndav =   1 E =    -75.7284673359 Error = 2.87e-17␣
↪FLOPS = 7.94e+09 Tdav = 0.12 T = 0.94
 --> Site =    7 .. Mmps = 1000 Ndav =   2 E =    -75.7284673395 Error = 2.43e-18␣
↪FLOPS = 8.38e+09 Tdav = 0.31 T = 1.45
 --> Site =    8 .. Mmps = 1000 Ndav =   3 E =    -75.7284673544 Error = 3.60e-18␣
↪FLOPS = 5.00e+09 Tdav = 0.96 T = 3.27
 --> Site =    9 .. Mmps = 1000 Ndav =   3 E =    -75.7284676613 Error = 4.94e-18␣
↪FLOPS = 9.03e+09 Tdav = 0.67 T = 2.55
 --> Site =   10 .. Mmps = 1000 Ndav =   2 E =    -75.7284676751 Error = 1.54e-18␣
↪FLOPS = 8.49e+09 Tdav = 0.58 T = 2.52
 --> Site =   11 .. Mmps = 1000 Ndav =   2 E =    -75.7284676868 Error = 5.66e-18␣
↪FLOPS = 8.59e+09 Tdav = 0.69 T = 3.02
 --> Site =   12 .. Mmps = 1000 Ndav =   2 E =    -75.7284676969 Error = 1.26e-17␣
↪FLOPS = 8.89e+09 Tdav = 0.70 T = 9.20
 --> Site =   13 .. Mmps = 1000 Ndav =   2 E =    -75.7284677017 Error = 9.44e-19␣
↪FLOPS = 9.08e+09 Tdav = 0.67 T = 4.40
 --> Site =   14 .. Mmps = 1000 Ndav =   2 E =    -75.7284677030 Error = 1.41e-18␣
↪FLOPS = 4.89e+09 Tdav = 1.03 T = 4.34
 --> Site =   15 .. Mmps = 1000 Ndav =   2 E =    -75.7284677040 Error = 1.04e-18␣
```

(continues on next page)

(continued from previous page)

```
↪FLOPS = 9.01e+09 Tdav = 0.47 T = 3.48
 --> Site =   16 .. Mmps =  965 Ndav =   2 E =    -75.7284677043 Error = 4.86e-20␣
↪FLOPS = 7.63e+09 Tdav = 0.39 T = 2.12
 --> Site =   17 .. Mmps =  949 Ndav =   1 E =    -75.7284677043 Error = 4.68e-20␣
↪FLOPS = 6.97e+09 Tdav = 0.16 T = 1.44
 --> Site =   18 .. Mmps =  695 Ndav =   1 E =    -75.7284677043 Error = 4.93e-20␣
↪FLOPS = 6.49e+09 Tdav = 0.10 T = 1.07
 --> Site =   19 .. Mmps =  406 Ndav =   1 E =    -75.7284677043 Error = 2.37e-20␣
↪FLOPS = 3.90e+09 Tdav = 0.04 T = 0.66
 --> Site =   20 .. Mmps =  222 Ndav =   1 E =    -75.7284677043 Error = 2.46e-20␣
↪FLOPS = 8.77e+08 Tdav = 0.03 T = 0.26
 --> Site =   21 .. Mmps =   89 Ndav =   1 E =    -75.7284677043 Error = 1.72e-20␣
↪FLOPS = 3.94e+08 Tdav = 0.01 T = 0.12
 --> Site =   22 .. Mmps =   32 Ndav =   1 E =    -75.7284677043 Error = 1.32e-20␣
↪FLOPS = 3.96e+07 Tdav = 0.00 T = 0.05
 --> Site =   23 .. Mmps =   12 Ndav =   1 E =    -75.7284677043 Error = 2.05e-21␣
↪FLOPS = 6.04e+06 Tdav = 0.00 T = 0.02
 --> Site =   24 .. Mmps =    5 Ndav =   1 E =    -75.7284677043 Error = 9.61e-22␣
↪FLOPS = 4.08e+05 Tdav = 0.00 T = 0.01
 --> Site =   25 .. Mmps =    2 Ndav =   1 E =    -75.7284677043 Error = 0.00e+00␣
↪FLOPS = 7.02e+04 Tdav = 0.00 T = 0.01
Time elapsed =   1205.214 | E =     -75.7284677043 | DE = -3.68e-07 | DW = 2.87e-17
Time sweep =       41.599 | 48.2 GFLOP/SWP
 | Dmem = 104 MB (0%) | Imem = 2.59 MB (94%) | Hmem = 1.22 GB | Wmem = 434 KB | Pmem␣
↪= 0 B
 | Tread = 2.961 | Twrite = 2.365 | Tfpread = 2.955 | Tfpwrite = 2.358 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 896 MB | cpsd = 718 MB
 | Trot = 3.494 | Tctr = 0.028 | Tint = 0.010 | Tmid = 0.000 | Tdctr = 1.497 | Tdiag␣
↪= 0.200 | Tinfo = 0.663
 | Teff = 27.282 | Tprt = 0.000 | Teig = 6.971 | Tblk = 34.680 | Tmve = 6.748 | Tdm␣
↪= 0.010 | Tsplt = 0.269 | Tsvd = 0.000 | Torth = 0.000

Sweep =   24 | Direction = backward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
 <-- Site =   25 .. Mmps =    3 Ndav =   1 E =    -75.7284677043 Error = 0.00e+00␣
↪FLOPS = 2.74e+04 Tdav = 0.00 T = 0.01
 <-- Site =   24 .. Mmps =   10 Ndav =   1 E =    -75.7284677043 Error = 0.00e+00␣
↪FLOPS = 2.31e+05 Tdav = 0.00 T = 0.01
 <-- Site =   23 .. Mmps =   31 Ndav =   1 E =    -75.7284677043 Error = 0.00e+00␣
↪FLOPS = 3.28e+06 Tdav = 0.00 T = 0.02
 <-- Site =   22 .. Mmps =   87 Ndav =   1 E =    -75.7284677043 Error = 0.00e+00␣
↪FLOPS = 3.60e+07 Tdav = 0.00 T = 0.03
 <-- Site =   21 .. Mmps =  221 Ndav =   1 E =    -75.7284677043 Error = 7.05e-28␣
↪FLOPS = 3.92e+08 Tdav = 0.01 T = 0.10
```

(continues on next page)

(continued from previous page)

```
 <-- Site =   20 .. Mmps =  400 Ndav =   1 E =    -75.7284677043 Error = 2.34e-26␣
↪FLOPS = 1.47e+09 Tdav = 0.02 T = 0.21
 <-- Site =   19 .. Mmps =  690 Ndav =   1 E =    -75.7284677043 Error = 1.69e-25␣
↪FLOPS = 2.29e+09 Tdav = 0.07 T = 0.79
 <-- Site =   18 .. Mmps =  943 Ndav =   1 E =    -75.7284677043 Error = 9.58e-24␣
↪FLOPS = 3.28e+09 Tdav = 0.19 T = 1.99
 <-- Site =   17 .. Mmps =  959 Ndav =   1 E =    -75.7284677043 Error = 1.26e-22␣
↪FLOPS = 4.42e+09 Tdav = 0.25 T = 3.24
 <-- Site =   16 .. Mmps = 1000 Ndav =   1 E =    -75.7284677043 Error = 4.28e-22␣
↪FLOPS = 7.49e+09 Tdav = 0.20 T = 3.11
 <-- Site =   15 .. Mmps = 1000 Ndav =   1 E =    -75.7284677043 Error = 6.39e-21␣
↪FLOPS = 7.44e+09 Tdav = 0.29 T = 3.06
 <-- Site =   14 .. Mmps = 1000 Ndav =   1 E =    -75.7284677043 Error = 4.05e-21␣
↪FLOPS = 8.21e+09 Tdav = 0.31 T = 4.49
 <-- Site =   13 .. Mmps = 1000 Ndav =   1 E =    -75.7284677043 Error = 7.03e-21␣
↪FLOPS = 8.51e+09 Tdav = 0.36 T = 7.80
 <-- Site =   12 .. Mmps = 1000 Ndav =   2 E =    -75.7284677048 Error = 2.47e-20␣
↪FLOPS = 6.82e+09 Tdav = 0.91 T = 11.64
 <-- Site =   11 .. Mmps = 1000 Ndav =   2 E =    -75.7284677057 Error = 3.42e-20␣
↪FLOPS = 8.68e+09 Tdav = 0.68 T = 8.63
 <-- Site =   10 .. Mmps = 1000 Ndav =   2 E =    -75.7284677096 Error = 7.68e-20␣
↪FLOPS = 7.81e+09 Tdav = 0.63 T = 5.43
 <-- Site =    9 .. Mmps = 1000 Ndav =   2 E =    -75.7284677172 Error = 1.16e-19␣
↪FLOPS = 8.81e+09 Tdav = 0.46 T = 3.14
 <-- Site =    8 .. Mmps = 1000 Ndav =   3 E =    -75.7284677781 Error = 1.96e-19␣
↪FLOPS = 8.51e+09 Tdav = 0.56 T = 2.91
 <-- Site =    7 .. Mmps = 1000 Ndav =   3 E =    -75.7284678610 Error = 8.15e-19␣
↪FLOPS = 8.69e+09 Tdav = 0.44 T = 2.52
 <-- Site =    6 .. Mmps = 1000 Ndav =   3 E =    -75.7284679691 Error = 4.55e-19␣
↪FLOPS = 4.74e+09 Tdav = 0.58 T = 2.72
 <-- Site =    5 .. Mmps =  442 Ndav =   2 E =    -75.7284679695 Error = 4.48e-19␣
↪FLOPS = 2.32e+09 Tdav = 0.13 T = 1.71
 <-- Site =    4 .. Mmps =  191 Ndav =   1 E =    -75.7284679695 Error = 3.78e-19␣
↪FLOPS = 4.41e+08 Tdav = 0.02 T = 0.64
 <-- Site =    3 .. Mmps =   71 Ndav =   1 E =    -75.7284679695 Error = 1.08e-19␣
↪FLOPS = 4.52e+07 Tdav = 0.01 T = 0.23
 <-- Site =    2 .. Mmps =   26 Ndav =   1 E =    -75.7284679695 Error = 1.55e-20␣
↪FLOPS = 1.54e+07 Tdav = 0.00 T = 0.05
 <-- Site =    1 .. Mmps =    7 Ndav =   1 E =    -75.7284679695 Error = 1.10e-37␣
↪FLOPS = 2.49e+06 Tdav = 0.00 T = 0.02
 <-- Site =    0 .. Mmps =    4 Ndav =   1 E =    -75.7284679695 Error = 4.84e-50␣
↪FLOPS = 7.05e+04 Tdav = 0.00 T = 0.01
Time elapsed =   1269.879 | E =     -75.7284679695 | DE = -2.65e-07 | DW = 8.15e-19
Time sweep =       64.664 | 40.8 GFLOP/SWP
 | Dmem = 104 MB (0%) | Imem = 2.59 MB (94%) | Hmem = 1.22 GB | Wmem = 434 KB | Pmem␣
```

(continues on next page)

(continued from previous page)

```
↪= 0 B
 | Tread = 3.170 | Twrite = 2.818 | Tfpread = 3.165 | Tfpwrite = 2.808 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 897 MB | cpsd = 722 MB
 | Trot = 3.754 | Tctr = 21.607 | Tint = 0.011 | Tmid = 0.000 | Tdctr = 0.017 |␣
↪Tdiag = 0.227 | Tinfo = 0.667
 | Teff = 27.358 | Tprt = 0.000 | Teig = 6.107 | Tblk = 33.921 | Tmve = 30.562 | Tdm␣
↪= 0.010 | Tsplt = 0.292 | Tsvd = 0.000 | Torth = 0.000

Sweep =   25 | Direction =  forward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
 --> Site =    0 .. Mmps =    3 Ndav =   1 E =    -75.7284679695 Error = 0.00e+00␣
↪FLOPS = 6.90e+04 Tdav = 0.00 T = 0.01
 --> Site =    1 .. Mmps =   10 Ndav =   1 E =    -75.7284679695 Error = 0.00e+00␣
↪FLOPS = 1.25e+06 Tdav = 0.00 T = 0.01
 --> Site =    2 .. Mmps =   35 Ndav =   1 E =    -75.7284679695 Error = 0.00e+00␣
↪FLOPS = 1.51e+07 Tdav = 0.00 T = 0.01
 --> Site =    3 .. Mmps =  126 Ndav =   1 E =    -75.7284679695 Error = 0.00e+00␣
↪FLOPS = 4.39e+07 Tdav = 0.01 T = 0.03
 --> Site =    4 .. Mmps =  403 Ndav =   1 E =    -75.7284679695 Error = 1.66e-20␣
↪FLOPS = 4.04e+08 Tdav = 0.02 T = 0.11
 --> Site =    5 .. Mmps = 1000 Ndav =   1 E =    -75.7284679695 Error = 2.07e-17␣
↪FLOPS = 3.42e+09 Tdav = 0.04 T = 0.32
 --> Site =    6 .. Mmps = 1000 Ndav =   1 E =    -75.7284679695 Error = 4.73e-18␣
↪FLOPS = 7.34e+09 Tdav = 0.13 T = 0.94
 --> Site =    7 .. Mmps = 1000 Ndav =   2 E =    -75.7284679720 Error = 2.24e-18␣
↪FLOPS = 8.50e+09 Tdav = 0.30 T = 1.43
 --> Site =    8 .. Mmps = 1000 Ndav =   2 E =    -75.7284679827 Error = 2.75e-18␣
↪FLOPS = 8.31e+09 Tdav = 0.38 T = 1.86
 --> Site =    9 .. Mmps = 1000 Ndav =   3 E =    -75.7284680835 Error = 3.17e-18␣
↪FLOPS = 8.76e+09 Tdav = 0.69 T = 2.35
 --> Site =   10 .. Mmps = 1000 Ndav =   2 E =    -75.7284680933 Error = 4.54e-18␣
↪FLOPS = 5.06e+09 Tdav = 0.97 T = 3.06
 --> Site =   11 .. Mmps = 1000 Ndav =   2 E =    -75.7284681010 Error = 1.68e-18␣
↪FLOPS = 8.31e+09 Tdav = 0.71 T = 4.07
 --> Site =   12 .. Mmps = 1000 Ndav =   2 E =    -75.7284681081 Error = 2.61e-18␣
↪FLOPS = 9.01e+09 Tdav = 0.69 T = 7.58
 --> Site =   13 .. Mmps = 1000 Ndav =   2 E =    -75.7284681121 Error = 1.12e-17␣
↪FLOPS = 9.15e+09 Tdav = 0.66 T = 6.03
 --> Site =   14 .. Mmps = 1000 Ndav =   2 E =    -75.7284681131 Error = 2.84e-18␣
↪FLOPS = 8.93e+09 Tdav = 0.56 T = 3.83
 --> Site =   15 .. Mmps = 1000 Ndav =   2 E =    -75.7284681137 Error = 5.02e-19␣
↪FLOPS = 8.32e+09 Tdav = 0.51 T = 2.52
 --> Site =   16 .. Mmps =  964 Ndav =   2 E =    -75.7284681138 Error = 4.93e-20␣
↪FLOPS = 7.79e+09 Tdav = 0.38 T = 2.03
```

(continues on next page)

(continued from previous page)

```
 --> Site =   17 .. Mmps =  948 Ndav =   2 E =    -75.7284681140 Error = 4.08e-20␣
↪FLOPS = 4.55e+09 Tdav = 0.48 T = 2.57
 --> Site =   18 .. Mmps =  696 Ndav =   1 E =    -75.7284681140 Error = 4.24e-20␣
↪FLOPS = 4.35e+09 Tdav = 0.14 T = 1.75
 --> Site =   19 .. Mmps =  405 Ndav =   1 E =    -75.7284681140 Error = 1.12e-20␣
↪FLOPS = 4.04e+09 Tdav = 0.04 T = 0.61
 --> Site =   20 .. Mmps =  224 Ndav =   1 E =    -75.7284681140 Error = 2.42e-20␣
↪FLOPS = 1.64e+09 Tdav = 0.01 T = 0.23
 --> Site =   21 .. Mmps =   89 Ndav =   1 E =    -75.7284681140 Error = 2.13e-20␣
↪FLOPS = 4.01e+08 Tdav = 0.01 T = 0.12
 --> Site =   22 .. Mmps =   33 Ndav =   1 E =    -75.7284681140 Error = 2.11e-20␣
↪FLOPS = 3.61e+07 Tdav = 0.00 T = 0.05
 --> Site =   23 .. Mmps =   12 Ndav =   1 E =    -75.7284681140 Error = 8.85e-21␣
↪FLOPS = 6.49e+06 Tdav = 0.00 T = 0.02
 --> Site =   24 .. Mmps =    4 Ndav =   1 E =    -75.7284681140 Error = 3.33e-21␣
↪FLOPS = 4.39e+05 Tdav = 0.00 T = 0.01
 --> Site =   25 .. Mmps =    3 Ndav =   1 E =    -75.7284681140 Error = 0.00e+00␣
↪FLOPS = 5.05e+04 Tdav = 0.00 T = 0.01
Time elapsed =   1311.600 | E =     -75.7284681140 | DE = -1.45e-07 | DW = 2.07e-17
Time sweep =       41.722 | 47.7 GFLOP/SWP
 | Dmem = 104 MB (0%) | Imem = 2.59 MB (94%) | Hmem = 1.22 GB | Wmem = 434 KB | Pmem␣
↪= 0 B
 | Tread = 3.051 | Twrite = 2.479 | Tfpread = 3.046 | Tfpwrite = 2.472 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 896 MB | cpsd = 718 MB
 | Trot = 3.428 | Tctr = 0.027 | Tint = 0.011 | Tmid = 0.000 | Tdctr = 1.547 | Tdiag␣
↪= 0.202 | Tinfo = 0.663
 | Teff = 27.571 | Tprt = 0.000 | Teig = 6.752 | Tblk = 34.750 | Tmve = 6.805 | Tdm␣
↪= 0.010 | Tsplt = 0.267 | Tsvd = 0.000 | Torth = 0.000

Sweep =   26 | Direction = backward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
 <-- Site =   25 .. Mmps =    3 Ndav =   1 E =    -75.7284681140 Error = 0.00e+00␣
↪FLOPS = 3.89e+04 Tdav = 0.00 T = 0.01
 <-- Site =   24 .. Mmps =   10 Ndav =   1 E =    -75.7284681140 Error = 0.00e+00␣
↪FLOPS = 2.91e+05 Tdav = 0.00 T = 0.01
 <-- Site =   23 .. Mmps =   31 Ndav =   1 E =    -75.7284681140 Error = 0.00e+00␣
↪FLOPS = 3.57e+06 Tdav = 0.00 T = 0.01
 <-- Site =   22 .. Mmps =   87 Ndav =   1 E =    -75.7284681140 Error = 0.00e+00␣
↪FLOPS = 3.94e+07 Tdav = 0.00 T = 0.03
 <-- Site =   21 .. Mmps =  221 Ndav =   1 E =    -75.7284681140 Error = 1.66e-27␣
↪FLOPS = 3.99e+08 Tdav = 0.01 T = 0.09
 <-- Site =   20 .. Mmps =  400 Ndav =   1 E =    -75.7284681140 Error = 2.81e-26␣
↪FLOPS = 1.63e+09 Tdav = 0.01 T = 0.21
 <-- Site =   19 .. Mmps =  690 Ndav =   1 E =    -75.7284681140 Error = 1.38e-25␣
```

(continues on next page)

(continued from previous page)

```
↪FLOPS = 3.91e+09 Tdav = 0.04 T = 0.52
 <-- Site =   18 .. Mmps =  943 Ndav =   1 E =    -75.7284681140 Error = 7.82e-24␣
↪FLOPS = 6.55e+09 Tdav = 0.09 T = 1.12
 <-- Site =   17 .. Mmps =  959 Ndav =   1 E =    -75.7284681140 Error = 1.09e-22␣
↪FLOPS = 7.01e+09 Tdav = 0.16 T = 1.68
 <-- Site =   16 .. Mmps = 1000 Ndav =   1 E =    -75.7284681140 Error = 2.90e-22␣
↪FLOPS = 7.32e+09 Tdav = 0.20 T = 2.45
 <-- Site =   15 .. Mmps = 1000 Ndav =   1 E =    -75.7284681140 Error = 5.48e-21␣
↪FLOPS = 5.71e+09 Tdav = 0.37 T = 3.06
 <-- Site =   14 .. Mmps = 1000 Ndav =   1 E =    -75.7284681140 Error = 4.05e-21␣
↪FLOPS = 7.96e+09 Tdav = 0.32 T = 5.96
 <-- Site =   13 .. Mmps = 1000 Ndav =   1 E =    -75.7284681140 Error = 8.58e-21␣
↪FLOPS = 8.36e+09 Tdav = 0.36 T = 6.17
 <-- Site =   12 .. Mmps = 1000 Ndav =   2 E =    -75.7284681145 Error = 1.78e-20␣
↪FLOPS = 8.73e+09 Tdav = 0.71 T = 11.72
 <-- Site =   11 .. Mmps = 1000 Ndav =   2 E =    -75.7284681152 Error = 3.76e-20␣
↪FLOPS = 8.70e+09 Tdav = 0.68 T = 10.21
 <-- Site =   10 .. Mmps = 1000 Ndav =   2 E =    -75.7284681180 Error = 7.76e-20␣
↪FLOPS = 8.72e+09 Tdav = 0.56 T = 3.91
 <-- Site =    9 .. Mmps = 1000 Ndav =   2 E =    -75.7284681232 Error = 1.01e-19␣
↪FLOPS = 8.93e+09 Tdav = 0.45 T = 3.13
 <-- Site =    8 .. Mmps = 1000 Ndav =   2 E =    -75.7284681381 Error = 2.58e-19␣
↪FLOPS = 7.41e+09 Tdav = 0.43 T = 4.29
 <-- Site =    7 .. Mmps = 1000 Ndav =   3 E =    -75.7284681651 Error = 3.86e-19␣
↪FLOPS = 8.61e+09 Tdav = 0.45 T = 2.56
 <-- Site =    6 .. Mmps = 1000 Ndav =   3 E =    -75.7284681904 Error = 3.92e-19␣
↪FLOPS = 8.14e+09 Tdav = 0.34 T = 1.95
 <-- Site =    5 .. Mmps =  446 Ndav =   1 E =    -75.7284681904 Error = 4.21e-19␣
↪FLOPS = 3.70e+09 Tdav = 0.04 T = 0.96
 <-- Site =    4 .. Mmps =  188 Ndav =   1 E =    -75.7284681904 Error = 4.23e-19␣
↪FLOPS = 4.52e+08 Tdav = 0.02 T = 0.59
 <-- Site =    3 .. Mmps =   74 Ndav =   1 E =    -75.7284681904 Error = 1.11e-19␣
↪FLOPS = 4.30e+07 Tdav = 0.01 T = 0.20
 <-- Site =    2 .. Mmps =   27 Ndav =   1 E =    -75.7284681904 Error = 1.87e-20␣
↪FLOPS = 1.51e+07 Tdav = 0.00 T = 0.05
 <-- Site =    1 .. Mmps =    8 Ndav =   1 E =    -75.7284681904 Error = 3.33e-33␣
↪FLOPS = 2.78e+06 Tdav = 0.00 T = 0.02
 <-- Site =    0 .. Mmps =    2 Ndav =   1 E =    -75.7284681904 Error = 5.20e-34␣
↪FLOPS = 1.49e+05 Tdav = 0.00 T = 0.01
Time elapsed =   1372.679 | E =     -75.7284681904 | DE = -7.64e-08 | DW = 4.23e-19
Time sweep =       61.079 | 39.2 GFLOP/SWP
 | Dmem = 104 MB (0%) | Imem = 2.59 MB (94%) | Hmem = 1.22 GB | Wmem = 434 KB | Pmem␣
↪= 0 B
 | Tread = 2.884 | Twrite = 2.570 | Tfpread = 2.880 | Tfpwrite = 2.564 | Tmoread =␣
↪0.000 | Tasync = 0.000
```

(continues on next page)

(continued from previous page)

```
 | data = 897 MB | cpsd = 722 MB
 | Trot = 3.570 | Tctr = 23.663 | Tint = 0.009 | Tmid = 0.000 | Tdctr = 0.013 |␣
↪Tdiag = 0.189 | Tinfo = 0.599
 | Teff = 23.247 | Tprt = 0.000 | Teig = 5.264 | Tblk = 28.929 | Tmve = 32.002 | Tdm␣
↪= 0.009 | Tsplt = 0.253 | Tsvd = 0.000 | Torth = 0.000

Sweep =   27 | Direction =  forward | Bond dimension = 1000 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
 --> Site =    0 .. Mmps =    3 Ndav =   1 E =    -75.7284681904 Error = 0.00e+00␣
↪FLOPS = 1.74e+05 Tdav = 0.00 T = 0.01
 --> Site =    1 .. Mmps =   10 Ndav =   1 E =    -75.7284681904 Error = 0.00e+00␣
↪FLOPS = 2.72e+06 Tdav = 0.00 T = 0.01
 --> Site =    2 .. Mmps =   35 Ndav =   1 E =    -75.7284681904 Error = 0.00e+00␣
↪FLOPS = 1.57e+07 Tdav = 0.00 T = 0.02
 --> Site =    3 .. Mmps =  126 Ndav =   1 E =    -75.7284681904 Error = 0.00e+00␣
↪FLOPS = 4.44e+07 Tdav = 0.01 T = 0.03
 --> Site =    4 .. Mmps =  403 Ndav =   1 E =    -75.7284681904 Error = 1.51e-20␣
↪FLOPS = 4.46e+08 Tdav = 0.02 T = 0.10
 --> Site =    5 .. Mmps = 1000 Ndav =   1 E =    -75.7284681904 Error = 2.56e-17␣
↪FLOPS = 3.61e+09 Tdav = 0.04 T = 0.32
 --> Site =    6 .. Mmps = 1000 Ndav =   1 E =    -75.7284681904 Error = 3.53e-17␣
↪FLOPS = 7.05e+09 Tdav = 0.13 T = 0.95
 --> Site =    7 .. Mmps = 1000 Ndav =   2 E =    -75.7284681924 Error = 2.03e-18␣
↪FLOPS = 8.17e+09 Tdav = 0.32 T = 1.45
 --> Site =    8 .. Mmps = 1000 Ndav =   2 E =    -75.7284681995 Error = 1.70e-18␣
↪FLOPS = 4.62e+09 Tdav = 0.69 T = 2.95
 --> Site =    9 .. Mmps = 1000 Ndav =   2 E =    -75.7284682176 Error = 3.48e-18␣
↪FLOPS = 8.32e+09 Tdav = 0.48 T = 2.53
 --> Site =   10 .. Mmps = 1000 Ndav =   2 E =    -75.7284682243 Error = 3.39e-18␣
↪FLOPS = 8.80e+09 Tdav = 0.56 T = 2.41
 --> Site =   11 .. Mmps = 1000 Ndav =   2 E =    -75.7284682294 Error = 7.84e-19␣
↪FLOPS = 8.78e+09 Tdav = 0.67 T = 3.07
 --> Site =   12 .. Mmps = 1000 Ndav =   2 E =    -75.7284682339 Error = 5.07e-18␣
↪FLOPS = 8.75e+09 Tdav = 0.71 T = 9.13
 --> Site =   13 .. Mmps = 1000 Ndav =   2 E =    -75.7284682366 Error = 1.08e-18␣
↪FLOPS = 9.30e+09 Tdav = 0.65 T = 4.54
 --> Site =   14 .. Mmps = 1000 Ndav =   2 E =    -75.7284682373 Error = 1.83e-18␣
↪FLOPS = 6.05e+09 Tdav = 0.83 T = 4.03
 --> Site =   15 .. Mmps = 1000 Ndav =   2 E =    -75.7284682375 Error = 6.20e-19␣
↪FLOPS = 8.47e+09 Tdav = 0.50 T = 3.86
 --> Site =   16 .. Mmps =  967 Ndav =   1 E =    -75.7284682375 Error = 5.23e-20␣
↪FLOPS = 6.79e+09 Tdav = 0.22 T = 1.94
 --> Site =   17 .. Mmps =  951 Ndav =   1 E =    -75.7284682375 Error = 3.88e-20␣
↪FLOPS = 7.70e+09 Tdav = 0.14 T = 1.47
 --> Site =   18 .. Mmps =  695 Ndav =   2 E =    -75.7284682376 Error = 4.72e-20␣
```

(continues on next page)

(continued from previous page)

```
↪FLOPS = 6.92e+09 Tdav = 0.18 T = 1.17
 --> Site =   19 .. Mmps =  404 Ndav =   1 E =    -75.7284682376 Error = 2.70e-20␣
↪FLOPS = 4.06e+09 Tdav = 0.04 T = 0.68
 --> Site =   20 .. Mmps =  225 Ndav =   1 E =    -75.7284682376 Error = 2.20e-20␣
↪FLOPS = 1.62e+09 Tdav = 0.01 T = 0.24
 --> Site =   21 .. Mmps =   90 Ndav =   1 E =    -75.7284682376 Error = 2.68e-20␣
↪FLOPS = 3.78e+08 Tdav = 0.01 T = 0.12
 --> Site =   22 .. Mmps =   34 Ndav =   1 E =    -75.7284682376 Error = 9.59e-21␣
↪FLOPS = 3.96e+07 Tdav = 0.00 T = 0.05
 --> Site =   23 .. Mmps =   12 Ndav =   1 E =    -75.7284682376 Error = 1.40e-20␣
↪FLOPS = 6.79e+06 Tdav = 0.00 T = 0.02
 --> Site =   24 .. Mmps =    5 Ndav =   1 E =    -75.7284682376 Error = 3.49e-21␣
↪FLOPS = 4.37e+05 Tdav = 0.00 T = 0.01
 --> Site =   25 .. Mmps =    2 Ndav =   1 E =    -75.7284682376 Error = 0.00e+00␣
↪FLOPS = 4.22e+04 Tdav = 0.00 T = 0.01
Time elapsed =   1413.959 | E =     -75.7284682376 | DE = -4.72e-08 | DW = 3.53e-17
Time sweep =       41.280 | 44.0 GFLOP/SWP
 | Dmem = 104 MB (0%) | Imem = 2.59 MB (94%) | Hmem = 1.22 GB | Wmem = 434 KB | Pmem␣
↪= 0 B
 | Tread = 2.953 | Twrite = 2.578 | Tfpread = 2.948 | Tfpwrite = 2.572 | Tmporead =␣
↪0.000 | Tasync = 0.000
 | data = 896 MB | cpsd = 718 MB
 | Trot = 3.546 | Tctr = 0.027 | Tint = 0.010 | Tmid = 0.000 | Tdctr = 1.466 | Tdiag␣
↪= 0.191 | Tinfo = 0.662
 | Teff = 27.438 | Tprt = 0.000 | Teig = 6.220 | Tblk = 34.094 | Tmve = 7.013 | Tdm␣
↪= 0.010 | Tsplt = 0.268 | Tsvd = 0.000 | Torth = 0.000

ATTENTION: DMRG is not converged to desired tolerance of 1.000e-08
DMRG energy (variational) =  -75.728468237610826
```

4.3.3 Restarting from backup MPS

We can set the `restart_dir` in `DMRGDriver` to a different directory name than `scratch`, so that after each sweep, `block2` will store a copy the MPS (int the left canonicalized or right canonicalized form) in the given `restart_dir` directory.

We can then restart using the data from this directory when the `scratch` folder is removed. To do this, we can copy the data from `restart_dir` to the `scratch` for the new DMRG calculation, and use `driver.load_mps` to load the MPS.

Note that for this case, since after each sweep, the MPS is either left canonicalized or right canonicalized, the next sweep direction can be automatically determined without the need to set the `forward` parameter in `driver.dmrg`.

4.4 DMRG with Spin-Orbit-Coupling

[1]:
```
!pip install block2==0.5.3rc5 -qq --progress-bar off --extra-index-url=https://block-
↪hczhai.github.io/block2-preview/pypi/
!pip install pyscf==2.3.0 -qq --progress-bar off
!pip install git+https://github.com/pyscf/dmrgscf -qq --progress-bar off
!wget -q https://raw.githubusercontent.com/pyscf/dmrgscf/master/pyscf/dmrgscf/
↪settings.py.example
!mv settings.py.example $(pip show pyscf-dmrgscf | grep 'Location' | tr ' ' '\n' |␣
↪tail -n 1)/pyscf/dmrgscf/settings.py
```

```
  Preparing metadata (setup.py) . . .  done
  Building wheel for pyscf-dmrgscf (setup.py) . . .  done
```

4.4.1 Introduction

In this tutorial we explain how to perform quantum chemistry DMRG calculation with the spin-orbit-coupling (SOC) term in the Hamiltonian. As a testing system, we consider the low energy spectrum of $[DyCl_6]^{3-}$, which has been studied in

- H. Zhai, and G. K.-L. Chan. A comparison between the one- and two-step spin-orbit coupling approaches based on the ab initio Density Matrix Renormalization Group. *J. Chem. Phys.* **157**, 164108 (2022). doi: 10.1063/5.0107805

The experimental data can be found in the supporting information Table S10 of:

- D. Aravena, M. Atanasov, and F. Neese. Periodic trends in lanthanide compounds through the eyes of multireference ab initio theory. *Inorg. Chem.* **55**, 4457-4469 (2016). doi: 10.1021/acs.inorgchem.6b00244

[2]:
```
from pyscf import scf, mcscf, gto, dmrgscf, lib
from pyblock2._pyscf import scf as b2scf
from pyblock2._pyscf import mcscf as b2mcscf
from pyblock2.driver.core import DMRGDriver, SymmetryTypes
from pyblock2._pyscf.ao2mo import soc_integrals as itgsoc
import numpy as np
import os

dmrgscf.settings.BLOCKEXE = os.popen("which block2main").read().strip()
dmrgscf.settings.MPIPREFIX = ''
```

```
/usr/local/lib/python3.10/dist-packages/pyscf/dft/libxc.py:772: UserWarning: Since␣
↪PySCF-2.3, B3LYP (and B3P86) are changed to the VWN-RPA variant, the same to the␣
↪B3LYP functional in Gaussian and ORCA (issue 1480). To restore the VWN5 definition,
↪ you can put the setting "B3LYP_WITH_VWN5 = True" in pyscf_conf.py
  warnings.warn('Since PySCF-2.3, B3LYP (and B3P86) are changed to the VWN-RPA␣
↪variant, '
```

4.4.2 Mean-Field Calculations

We first do mean-field calculations without SOC to get a set of starting orbitals.

[3]:
```
b = 2.72
coords = [["Dy", [0, 0, 0]]] + [["Cl", [0] * p + [i] + [0] * (2 - p)] for p in [0, 1,
↪ 2] for i in [-b, b]]
mol = gto.M(atom=coords, basis={"Dy": "ano@9s8p5d4f", "Cl": "ano@4s3p"}, verbose=3,␣
↪spin=5, charge=-3)
print('basis = dz nelec = %d nao = %d' % (mol.nelectron, mol.nao))

mf = scf.UKS(mol).x2c()
mf = b2scf.smearing_(mf, sigma=0.2, method="fermi", fit_spin=True)
mf.xc = "bp86"
mf.verbose = 4

dm0 = b2scf.get_metal_init_guess(mol, orb="4f", atom_idxs=[0], coupling="+", atomic_
↪spin=5)
mf.kernel(dm0=dm0)
dmao = np.einsum('yij->ij', mf.make_rdm1(), optimize=True)

lo_coeff, lo_occ, lo_energy = b2mcscf.get_uno(mf)
selected = b2mcscf.select_active_space(mol, lo_coeff, lo_occ, ao_labels=["Dy-4f"],␣
↪atom_order=[0])
lo_coeff, lo_occ, lo_energy, nactorb, nactelec = b2mcscf.sort_orbitals(mol, lo_coeff,
    lo_occ, lo_energy, cas_list=selected, do_loc=True, split_low=0.1, split_high=1.9)

b2scf.mulliken_pop_dmao(mol, mf.make_rdm1())

mf = scf.RHF(mol).x2c()
mf.mo_coeff = lo_coeff
mf.mo_occ = np.array([int(np.round(x) + 0.1) for x in lo_occ])
assert sum(mf.mo_occ) == mol.nelectron
mf.mo_energy = lo_energy
```

```
basis = dz nelec = 171 nao = 164


******** <class 'pyscf.x2c.sfx2c1e.sfx2c1e.<locals>.SFX2C1E_SCF'> ********
method = SFX2C1E_SCF-UKS
initial guess = minao
damping factor = 0
level_shift factor = 0
DIIS = <class 'pyscf.scf.diis.CDIIS'>
diis_start_cycle = 1
diis_space = 8
SCF conv_tol = 1e-09
SCF conv_tol_grad = None
```

(continues on next page)

(continued from previous page)

```
SCF max_cycles = 50
direct_scf = True
direct_scf_tol = 1e-13
chkfile to save SCF result = /content/tmpuiy5aay9
max_memory 4000 MB (current use 163 MB)
number electrons alpha = 88  beta = 83
XC library pyscf.dft.libxc version 6.1.0
    S. Lehtola, C. Steigemann, M. J.T. Oliveira, and M. A.L. Marques.,  SoftwareX 7,␣
↪1-5 (2018)
XC functionals = bp86
    A. D. Becke.,  Phys. Rev. A 38, 3098 (1988)
    J. P. Perdew.,  Phys. Rev. B 33, 8822 (1986)
small_rho_cutoff = 1e-07
Set gradient conv threshold to 3.16228e-05
Overwritten attributes  get_grad get_occ energy_tot  of <class 'pyscf.x2c.sfx2c1e.
↪sfx2c1e.<locals>.SFX2C1E_SCF'>
init E= -14787.0872479682

WARN: alpha nocc = 88  HOMO 0.0611863318980825 >= LUMO 0.0613044136748146


WARN: beta  nocc = 83  HOMO 0.0739533459338481 >= LUMO 0.0739533459486078

    sigma = 0.2  Optimized mu = [-0.01549092 -0.10180943]  entropy = 58.5500123445
    Total E(T) = -14913.1615642987  Free energy = -14924.8715667676  E0 = -14919.
↪0165655332
cycle= 1 E= -14913.1615642987  delta_E= -126  |g|= 12.9  |ddm|=  6.1

WARN: alpha nocc = 88  HOMO -0.121609089741254 >= LUMO -0.121609089734184


WARN: beta  nocc = 83  HOMO -0.23372079326522 >= LUMO -0.233720793220507


WARN: system HOMO -0.109118498567311 >= system LUMO -0.233720793220507

    sigma = 0.2  Optimized mu = [-0.21224349 -0.32945095]  entropy = 47.6143433007
    Total E(T) = -14720.9861471891  Free energy = -14730.5090158493  E0 = -14725.
↪7475815192
cycle= 2 E= -14720.9861471891  delta_E=  192  |g|= 52.6  |ddm|= 7.48

WARN: alpha nocc = 88  HOMO -0.0245777519477351 >= LUMO -0.0245777519129139


WARN: beta  nocc = 83  HOMO -0.0299227438160497 >= LUMO -0.0299227438085627
```

(continues on next page)

(continued from previous page)

```
WARN: system HOMO -0.0143431707523865 >= system LUMO -0.0299227438085627

    sigma = 0.2  Optimized mu = [-0.02278235 -0.1425376 ]  entropy = 48.1342878081
    Total E(T) = -14819.6546270882  Free energy = -14829.2814846498  E0 = -14824.
↪468055869
cycle= 3 E= -14819.6546270882  delta_E= -98.7  |g|= 37.7  |ddm|=  2.5

WARN: alpha nocc = 88  HOMO -0.0232792163297702 >= LUMO -0.0232792163029307


WARN: beta  nocc = 83  HOMO -0.027664984550311 >= LUMO -0.0276649844907769


WARN: system HOMO -0.0122535293260926 >= system LUMO -0.0276649844907769

    sigma = 0.2  Optimized mu = [-0.02233909 -0.14063446]  entropy = 48.4897684516
    Total E(T) = -14826.1731924567  Free energy = -14835.871146147  E0 = -14831.
↪0221693018
cycle= 4 E= -14826.1731924567  delta_E= -6.52  |g|=   36  |ddm|= 0.276
  alpha nocc = 88  HOMO = 0.00846605306704732  LUMO = 0.0416407638907885

WARN: beta  nocc = 83  HOMO 0.0131852927675519 >= LUMO 0.0131852928294732

    sigma = 0.2  Optimized mu = [0.12771296 0.01817416]  entropy = 50.6679557528
    Total E(T) = -14878.0616222321  Free energy = -14888.1952133827  E0 = -14883.
↪1284178074
cycle= 5 E= -14878.0616222321  delta_E= -51.9  |g|=   26  |ddm|= 2.33
  alpha nocc = 88  HOMO = 0.00761147574577788  LUMO = 0.0121697431976167

WARN: beta  nocc = 83  HOMO 0.0108548019978724 >= LUMO 0.0108548020335531

    sigma = 0.2  Optimized mu = [0.11060168 0.00166636]  entropy = 50.9645078306
    Total E(T) = -14878.2337623554  Free energy = -14888.4266639215  E0 = -14883.
↪3302131385
cycle= 6 E= -14878.2337623554  delta_E= -0.172  |g|= 25.5  |ddm|= 0.335
  alpha nocc = 88  HOMO = 0.0276942235776991  LUMO = 0.298610217463403

WARN: beta  nocc = 83  HOMO 0.0340199452252861 >= LUMO 0.034019945234256

    sigma = 0.2  Optimized mu = [0.21728538 0.11188526]  entropy = 53.9560278764
    Total E(T) = -14911.2741368555  Free energy = -14922.0653424308  E0 = -14916.
↪6697396431
cycle= 7 E= -14911.2741368555  delta_E=  -33  |g|= 13.6  |ddm|= 1.98
```

(continues on next page)

(continued from previous page)

```
  alpha nocc = 88  HOMO = 0.13381609792731  LUMO = 0.288456084791265

WARN: beta  nocc = 83  HOMO 0.16480594038153 >= LUMO 0.164805940383059

    sigma = 0.2  Optimized mu = [0.25668441 0.16930681]  entropy = 60.3824311798
    Total E(T) = -14923.9028739442  Free energy = -14935.9793601802  E0 = -14929.
↪9411170622
cycle= 8 E= -14923.9028739442  delta_E= -12.6  |g|= 1.39  |ddm|= 1.76
  alpha nocc = 88  HOMO = 0.0921890164678569  LUMO = 0.292843791767242

WARN: beta  nocc = 83  HOMO 0.121292379771426 >= LUMO 0.12129237977444

    sigma = 0.2  Optimized mu = [0.26092675 0.17153212]  entropy = 59.6350760692
    Total E(T) = -14924.1241708051  Free energy = -14936.0511860189  E0 = -14930.
↪087678412
cycle= 9 E= -14924.1241708051  delta_E= -0.221  |g|= 0.97  |ddm|= 0.303
  alpha nocc = 88  HOMO = 0.0911511800536998  LUMO = 0.288212939422363

WARN: beta  nocc = 83  HOMO 0.12085074130689 >= LUMO 0.120850741315004

    sigma = 0.2  Optimized mu = [0.25274845 0.16418413]  entropy = 59.9452854704
    Total E(T) = -14924.0969557785  Free energy = -14936.0860128726  E0 = -14930.
↪0914843255
cycle= 10 E= -14924.0969557785  delta_E= 0.0272  |g|= 0.542  |ddm|= 0.126
  alpha nocc = 88  HOMO = 0.114234928592505  LUMO = 0.289794835426367

WARN: beta  nocc = 83  HOMO 0.14423729560329 >= LUMO 0.144237295608166

    sigma = 0.2  Optimized mu = [0.25964362 0.17126877]  entropy = 60.0214174303
    Total E(T) = -14924.1028556452  Free energy = -14936.1071391313  E0 = -14930.
↪1049973883
cycle= 11 E= -14924.1028556452  delta_E= -0.0059  |g|= 0.239  |ddm|= 0.125
  alpha nocc = 88  HOMO = 0.106063479664111  LUMO = 0.289200857284636

WARN: beta  nocc = 83  HOMO 0.135947391800556 >= LUMO 0.135947391801812

    sigma = 0.2  Optimized mu = [0.25685265 0.16838371]  entropy = 59.9816253629
    Total E(T) = -14924.115749333  Free energy = -14936.1120744055  E0 = -14930.
↪1139118692
cycle= 12 E= -14924.115749333  delta_E= -0.0129  |g|= 0.0216  |ddm|= 0.0476
  alpha nocc = 88  HOMO = 0.106903607487204  LUMO = 0.289331834697122

WARN: beta  nocc = 83  HOMO 0.136788886346519 >= LUMO 0.136788886349247

    sigma = 0.2  Optimized mu = [0.25729131 0.16884598]  entropy = 59.991989718
```

(continues on next page)

(continued from previous page)

```
    Total E(T) = -14924.1137185514  Free energy = -14936.112116495  E0 = -14930.
↪1129175232
cycle= 13 E= -14924.1137185514  delta_E= 0.00203  |g|= 0.00327  |ddm|= 0.00658
  alpha nocc = 88  HOMO = 0.106970896101911  LUMO = 0.289323970464076

WARN: beta  nocc = 83  HOMO 0.136853198214426 >= LUMO 0.136853198215269

    sigma = 0.2  Optimized mu = [0.2572915  0.16884265]  entropy = 59.9906378915
    Total E(T) = -14924.1139901545  Free energy = -14936.1121177328  E0 = -14930.
↪1130539437
cycle= 14 E= -14924.1139901545  delta_E= -0.000272  |g|= 0.000105  |ddm|= 0.000591
  alpha nocc = 88  HOMO = 0.106977598962225  LUMO = 0.289328998365118

WARN: beta  nocc = 83  HOMO 0.136859044165478 >= LUMO 0.136859044169837

    sigma = 0.2  Optimized mu = [0.25729958 0.16885066]  entropy = 59.9906457967
    Total E(T) = -14924.1139885756  Free energy = -14936.1121177349  E0 = -14930.
↪1130531553
cycle= 15 E= -14924.1139885756  delta_E= 1.58e-06  |g|= 2.08e-05  |ddm|= 8.82e-05
  alpha nocc = 88  HOMO = 0.106976831045835  LUMO = 0.289328537204721

WARN: beta  nocc = 83  HOMO 0.136857660075519 >= LUMO 0.136857660081439

    sigma = 0.2  Optimized mu = [0.25729862 0.16884968]  entropy = 59.9906548735
    Total E(T) = -14924.1139867603  Free energy = -14936.112117735  E0 = -14930.
↪1130522476
cycle= 16 E= -14924.1139867603  delta_E= 1.82e-06  |g|= 1.66e-05  |ddm|= 1.22e-05
  alpha nocc = 88  HOMO = 0.106976875910355  LUMO = 0.289328533787231

WARN: beta  nocc = 83  HOMO 0.136857649407593 >= LUMO 0.13685764941068

    sigma = 0.2  Optimized mu = [0.25729862 0.16884967]  entropy = 59.9906550584
    Total E(T) = -14924.1139867234  Free energy = -14936.1121177351  E0 = -14930.
↪1130522292
cycle= 17 E= -14924.1139867234  delta_E= 3.69e-08  |g|= 8.09e-06  |ddm|= 4.21e-06
  alpha nocc = 88  HOMO = 0.106977070517948  LUMO = 0.289328536598835

WARN: beta  nocc = 83  HOMO 0.136857689526215 >= LUMO 0.136857689532169

    sigma = 0.2  Optimized mu = [0.25729866 0.1688497 ]  entropy = 59.9906555645
    Total E(T) = -14924.1139866221  Free energy = -14936.112117735  E0 = -14930.
↪1130521786
cycle= 18 E= -14924.1139866221  delta_E= 1.01e-07  |g|= 7.21e-07  |ddm|= 1.52e-06
  alpha nocc = 88  HOMO = 0.106977063879805  LUMO = 0.289328536502427
```

(continues on next page)

(continued from previous page)

```
WARN: beta  nocc = 83  HOMO 0.136857674935149 >= LUMO 0.13685767493636

    sigma = 0.2  Optimized mu = [0.25729866 0.1688497 ]  entropy = 59.9906555094
    Total E(T) = -14924.1139866332  Free energy = -14936.112117735  E0 = -14930.
↪1130521841
cycle= 19 E= -14924.1139866332  delta_E= -1.1e-08  |g|= 3.04e-07  |ddm|= 6.21e-08
  alpha nocc = 88  HOMO = 0.106977063133108  LUMO = 0.289328536324841

WARN: beta  nocc = 83  HOMO 0.136857662726047 >= LUMO 0.136857662726672

    sigma = 0.2  Optimized mu = [0.25729866 0.1688497 ]  entropy = 59.990655473
    Total E(T) = -14924.1139866405  Free energy = -14936.1121177351  E0 = -14930.
↪1130521878
cycle= 20 E= -14924.1139866405  delta_E= -7.31e-09  |g|= 1.91e-08  |ddm|= 4.94e-08
  alpha nocc = 88  HOMO = 0.106977055367124  LUMO = 0.289328535445213

WARN: beta  nocc = 83  HOMO 0.136857660473417 >= LUMO 0.136857660475939

    sigma = 0.2  Optimized mu = [0.25729866 0.16884969]  entropy = 59.9906553948
    Total E(T) = -14924.1139866561  Free energy = -14936.1121177351  E0 = -14930.
↪1130521956
cycle= 21 E= -14924.1139866561  delta_E= -1.56e-08  |g|= 8.81e-08  |ddm|= 4.62e-08
  alpha nocc = 88  HOMO = 0.106977057273827  LUMO = 0.289328535914515

WARN: beta  nocc = 83  HOMO 0.136857661866812 >= LUMO 0.136857661869012

    sigma = 0.2  Optimized mu = [0.25729866 0.1688497 ]  entropy = 59.9906554608
    Total E(T) = -14924.1139866429  Free energy = -14936.112117735  E0 = -14930.
↪113052189
cycle= 22 E= -14924.1139866429  delta_E= 1.32e-08  |g|= 8.46e-08  |ddm|= 2.74e-08
  alpha nocc = 88  HOMO = 0.106977058322305  LUMO = 0.289328536045372

WARN: beta  nocc = 83  HOMO 0.136857661896088 >= LUMO 0.136857661898237

    sigma = 0.2  Optimized mu = [0.25729866 0.1688497 ]  entropy = 59.9906554615
    Total E(T) = -14924.1139866427  Free energy = -14936.112117735  E0 = -14930.
↪1130521888
cycle= 23 E= -14924.1139866427  delta_E= 1.71e-10  |g|= 7.34e-08  |ddm|= 4.91e-09
  alpha nocc = 88  HOMO = 0.106977091624998  LUMO = 0.289328537155713

WARN: beta  nocc = 83  HOMO 0.136857691496918 >= LUMO 0.136857691498817

    sigma = 0.2  Optimized mu = [0.25729867 0.1688497 ]  entropy = 59.9906556412
    Total E(T) = -14924.1139866068  Free energy = -14936.112117735  E0 = -14930.
↪1130521709
```

(continues on next page)

(continued from previous page)

```
Extra cycle  E= -14924.1139866068  delta_E= 3.59e-08  |g|= 1.06e-06  |ddm|= 1.66e-07
converged SCF energy = -14924.1139866068  <S^2> = -1.1368684e-13  2S+1 = 1
Nalpha = 88, Nbeta 83, Sz = 5, Norb = 164
Idemponency of DM: 0.5590169948595674
active orbital indices [83 84 85 86 87]
Num active orbitals 5
core orbs: (164, 83)
act  orbs: (164, 5)
vir  orbs: (164, 76)

[pm_loc_kernel]
 mocoeff.shape= (164, 83)
 tol= 1e-06
 maxcycle= 1000
 partition= 7 \n [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,␣
↪19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
↪ 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,␣
↪60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
↪ 81, 82, 83, 84, 85], [86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98], [99,␣
↪100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111], [112, 113, 114, 115,␣
↪116, 117, 118, 119, 120, 121, 122, 123, 124], [125, 126, 127, 128, 129, 130, 131,␣
↪132, 133, 134, 135, 136, 137], [138, 139, 140, 141, 142, 143, 144, 145, 146, 147,␣
↪148, 149, 150], [151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163]]
 initial funval =  33.16272108187299
icycle= 0 delta= 29.19971105434567 fun= 62.36243213621866
icycle= 1 delta= 13.438486671684773 fun= 75.80091880790343
icycle= 2 delta= 5.05954903052021 fun= 80.86046783842365
icycle= 3 delta= 1.3886437451091505 fun= 82.2491115835328
icycle= 4 delta= 0.020998758507891263 fun= 82.27011034204068
icycle= 5 delta= 0.0046055512769261125 fun= 82.2747158933176
icycle= 6 delta= 0.0005141643961265244 fun= 82.27523005771373
icycle= 7 delta= 0.00011100705764772061 fun= 82.27534106477138
icycle= 8 delta= 8.348600036268924e-06 fun= 82.27534941337142
icycle= 9 delta= 8.927262712634137e-07 fun= 82.27535030609769
CONG: PMloc converged!

[pm_loc_kernel]
 mocoeff.shape= (164, 5)
 tol= 1e-06
 maxcycle= 1000
 partition= 7 \n [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,␣
↪19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
↪ 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,␣
↪60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
↪ 81, 82, 83, 84, 85], [86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98], [99,␣
```

(continues on next page)

(continued from previous page)

```
↪100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111], [112, 113, 114, 115,␣
↪116, 117, 118, 119, 120, 121, 122, 123, 124], [125, 126, 127, 128, 129, 130, 131,␣
↪132, 133, 134, 135, 136, 137], [138, 139, 140, 141, 142, 143, 144, 145, 146, 147,␣
↪148, 149, 150], [151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163]]
 initial funval =  4.905871498873693
icycle= 0 delta= 0.00013091761467721026 fun= 4.90600241648837
icycle= 1 delta= 1.6384733158646628e-05 fun= 4.906018801221529
icycle= 2 delta= 4.299752835633212e-06 fun= 4.906023100974364
icycle= 3 delta= 1.5937957090927954e-08 fun= 4.906023116912321
CONG: PMloc converged!
\nLowdin population for LMOs:
[C   0] occ =  2.00000 fii =  -1966.529 (  0-Dy-     1s = 1.000) (  0-Dy-   4dxy = 0.
↪000) (  0-Dy-   4dyz = 0.000)
[C   1] occ =  2.00000 fii =    -60.381 (  4-Cl-     1s = 0.570) (  4-Cl-    2py = 0.
↪304) (  4-Cl-    2pz = 0.089)
[C   2] occ =  2.00000 fii =    -10.617 (  6-Cl-    2px = 0.699) (  6-Cl-    2py = 0.
↪256) (  6-Cl-     1s = 0.041)
[C   3] occ =  2.00000 fii =    -56.133 (  5-Cl-     1s = 0.525) (  5-Cl-    2pz = 0.
↪304) (  5-Cl-    2py = 0.092)
[C   4] occ =  2.00000 fii =     -6.941 (  4-Cl-    2px = 0.819) (  4-Cl-    2pz = 0.
↪177) (  4-Cl-     1s = 0.002)
[C   5] occ =  2.00000 fii =     -7.021 (  1-Cl-    2py = 0.629) (  1-Cl-    2pz = 0.
↪366) (  1-Cl-     1s = 0.003)
[C   6] occ =  2.00000 fii =     -7.795 (  2-Cl-    2py = 0.979) (  2-Cl-     1s = 0.
↪011) (  2-Cl-    2pz = 0.008)
[C   7] occ =  2.00000 fii =    -22.455 (  5-Cl-    2px = 0.741) (  5-Cl-     1s = 0.
↪167) (  5-Cl-    2py = 0.085)
[C   8] occ =  2.00000 fii =     -8.823 (  3-Cl-    2px = 0.656) (  3-Cl-    2pz = 0.
↪320) (  3-Cl-     1s = 0.022)
[C   9] occ =  2.00000 fii =     -6.929 (  2-Cl-    2pz = 0.989) (  2-Cl-    2py = 0.
↪007) (  2-Cl-     1s = 0.002)
[C  10] occ =  2.00000 fii =     -9.180 (  6-Cl-    2py = 0.741) (  6-Cl-    2px = 0.
↪230) (  6-Cl-     1s = 0.026)
[C  11] occ =  2.00000 fii =    -10.384 (  5-Cl-    2py = 0.798) (  5-Cl-    2px = 0.
↪160) (  5-Cl-     1s = 0.039)
[C  12] occ =  2.00000 fii =    -20.005 (  4-Cl-    2pz = 0.713) (  4-Cl-     1s = 0.
↪141) (  4-Cl-    2px = 0.138)
[C  13] occ =  2.00000 fii =    -34.179 (  3-Cl-    2pz = 0.531) (  3-Cl-     1s = 0.
↪292) (  3-Cl-    2px = 0.164)
[C  14] occ =  2.00000 fii =    -35.433 (  1-Cl-    2pz = 0.404) (  1-Cl-     1s = 0.
↪305) (  1-Cl-    2py = 0.277)
[C  15] occ =  2.00000 fii =    -67.593 (  3-Cl-     1s = 0.647) (  3-Cl-    2px = 0.
↪179) (  3-Cl-    2pz = 0.148)
[C  16] occ =  2.00000 fii =    -75.573 (  0-Dy-    3pz = 0.418) (  0-Dy-  3dz^2 = 0.
↪135) (  0-Dy-    3py = 0.114)
```

(continues on next page)

(continued from previous page)

```
[C  17] occ =  2.00000 fii =    -90.803 (  6-Cl-     1s = 0.894) (  6-Cl-    2px = 0.
↪069) (  6-Cl-    2pz = 0.028)
[C  18] occ =  2.00000 fii =    -95.872 (  2-Cl-     1s = 0.948) (  2-Cl-    2px = 0.
↪030) (  2-Cl-    2py = 0.013)
[C  19] occ =  2.00000 fii =    -69.289 (  1-Cl-     1s = 0.665) (  1-Cl-    2pz = 0.
↪227) (  1-Cl-    2py = 0.092)
[C  20] occ =  2.00000 fii =   -137.829 (  0-Dy-    2pz = 0.321) (  0-Dy-    4px = 0.
↪124) (  0-Dy-   3dyz = 0.109)
[C  21] occ =  2.00000 fii =   -183.170 (  0-Dy-    2py = 0.210) (  0-Dy-     2s = 0.
↪201) (  0-Dy-   3dxy = 0.179)
[C  22] occ =  2.00000 fii =    -44.171 (  0-Dy-    3px = 0.276) (  0-Dy-    3py = 0.
↪246) (  0-Dy-    4py = 0.197)
[C  23] occ =  2.00000 fii =    -58.139 (  0-Dy-3dx2-y2 = 0.243) (  0-Dy-    4px = 0.
↪189) (  0-Dy-   3dyz = 0.082)
[C  24] occ =  2.00000 fii =   -136.058 (  0-Dy-    2pz = 0.244) (  0-Dy-   3dyz = 0.
↪212) (  0-Dy-    2py = 0.136)
[C  25] occ =  2.00000 fii =    -80.454 (  0-Dy-3dx2-y2 = 0.387) (  0-Dy-   3dxy = 0.
↪134) (  0-Dy-    2py = 0.086)
[C  26] occ =  2.00000 fii =   -185.374 (  0-Dy-     2s = 0.491) (  0-Dy-   3dxy = 0.
↪116) (  0-Dy-  3dz^2 = 0.073)
[C  27] occ =  2.00000 fii =    -83.298 (  0-Dy-     3s = 0.370) (  0-Dy-    2px = 0.
↪114) (  0-Dy-     4s = 0.113)
[C  28] occ =  2.00000 fii =   -110.461 (  0-Dy-    2pz = 0.138) (  0-Dy-    3px = 0.
↪124) (  0-Dy-   3dyz = 0.094)
[C  29] occ =  2.00000 fii =   -128.348 (  0-Dy-    2px = 0.241) (  0-Dy-    3px = 0.
↪115) (  0-Dy-    3py = 0.107)
[C  30] occ =  2.00000 fii =    -43.508 (  0-Dy-    4py = 0.214) (  0-Dy-    4pz = 0.
↪192) (  0-Dy-    3pz = 0.183)
[C  31] occ =  2.00000 fii =     -7.685 (  1-Cl-    2px = 0.931) (  1-Cl-     2s = 0.
↪050) (  1-Cl-     1s = 0.009)
[C  32] occ =  2.00000 fii =     -8.833 (  3-Cl-    2py = 0.917) (  3-Cl-     2s = 0.
↪052) (  3-Cl-     1s = 0.022)
[C  33] occ =  2.00000 fii =     -8.829 (  6-Cl-    2pz = 0.917) (  6-Cl-     2s = 0.
↪052) (  6-Cl-     1s = 0.022)
[C  34] occ =  2.00000 fii =     -8.832 (  2-Cl-    2px = 0.917) (  2-Cl-     2s = 0.
↪052) (  2-Cl-     1s = 0.022)
[C  35] occ =  2.00000 fii =    -58.486 (  0-Dy-  3dz^2 = 0.334) (  0-Dy-   3dxy = 0.
↪097) (  0-Dy-   3dyz = 0.096)
[C  36] occ =  2.00000 fii =    -86.441 (  0-Dy-    4pz = 0.273) (  0-Dy-    3py = 0.
↪152) (  0-Dy-    2px = 0.149)
[C  37] occ =  2.00000 fii =    -43.187 (  0-Dy-   3dxz = 0.508) (  0-Dy-   3dyz = 0.
↪131) (  0-Dy-   4dyz = 0.097)
[C  38] occ =  2.00000 fii =    -93.548 (  0-Dy-    4px = 0.268) (  0-Dy-    2px = 0.
↪139) (  0-Dy-     3s = 0.139)
[C  39] occ =  2.00000 fii =    -32.099 (  4-Cl-    2py = 0.638) (  4-Cl-     1s = 0.
```

(continues on next page)

(continued from previous page)

```
↪269) (  4-Cl-     2s = 0.056)
[C  40] occ =  2.00000 fii =    -74.895 (  0-Dy-     4s = 0.242) (  0-Dy-    4py = 0.
↪195) (  0-Dy-     3s = 0.085)
[C  41] occ =  2.00000 fii =    -62.091 (  0-Dy-   4dxy = 0.584) (  0-Dy-    2py = 0.
↪136) (  0-Dy-     3s = 0.095)
[C  42] occ =  2.00000 fii =    -37.444 (  0-Dy-   4dyz = 0.376) (  0-Dy-   4dxz = 0.
↪166) (  0-Dy-   3dxy = 0.108)
[C  43] occ =  2.00000 fii =    -25.185 (  0-Dy-   4dxz = 0.468) (  0-Dy-   4dyz = 0.
↪324) (  0-Dy-   4dxy = 0.040)
[C  44] occ =  2.00000 fii =     -9.732 (  0-Dy-4dx2-y2 = 0.654) (  0-Dy-  4dz^2 = 0.
↪278) (  0-Dy-3dx2-y2 = 0.027)
[C  45] occ =  2.00000 fii =     -7.154 (  0-Dy-  4dz^2 = 0.666) (  0-Dy-4dx2-y2 = 0.
↪297) (  0-Dy-  3dz^2 = 0.024)
[C  46] occ =  2.00000 fii =    -19.510 (  5-Cl-    2pz = 0.376) (  5-Cl-     2s = 0.
↪342) (  5-Cl-     1s = 0.135)
[C  47] occ =  2.00000 fii =    -18.867 (  5-Cl-     2s = 0.357) (  5-Cl-    2pz = 0.
↪311) (  5-Cl-     3s = 0.159)
[C  48] occ =  2.00000 fii =     -7.915 (  2-Cl-     2s = 0.641) (  2-Cl-     3s = 0.
↪255) (  2-Cl-    2px = 0.049)
[C  49] occ =  2.00000 fii =     -7.917 (  4-Cl-     2s = 0.641) (  4-Cl-     3s = 0.
↪255) (  4-Cl-    2py = 0.049)
[C  50] occ =  2.00000 fii =     -7.915 (  6-Cl-     2s = 0.641) (  6-Cl-     3s = 0.
↪255) (  6-Cl-    2pz = 0.049)
[C  51] occ =  2.00000 fii =     -7.915 (  3-Cl-     2s = 0.641) (  3-Cl-     3s = 0.
↪255) (  3-Cl-    2py = 0.049)
[C  52] occ =  2.00000 fii =     -7.915 (  1-Cl-     2s = 0.641) (  1-Cl-     3s = 0.
↪255) (  1-Cl-    2px = 0.049)
[C  53] occ =  2.00000 fii =     -0.592 (  0-Dy-    5py = 0.951) (  0-Dy-    6py = 0.
↪010) (  3-Cl-    4py = 0.009)
[C  54] occ =  2.00000 fii =     -0.601 (  0-Dy-    5pz = 0.947) (  0-Dy-    6pz = 0.
↪010) (  5-Cl-    4pz = 0.010)
[C  55] occ =  2.00000 fii =     -1.700 (  0-Dy-    5px = 0.477) (  0-Dy-     5s = 0.
↪413) (  0-Dy-     4s = 0.023)
[C  56] occ =  2.00000 fii =     -1.701 (  0-Dy-    5px = 0.476) (  0-Dy-     5s = 0.
↪413) (  0-Dy-     4s = 0.023)
[C  57] occ =  2.00000 fii =     -2.660 (  1-Cl-     3s = 0.383) (  1-Cl-    3px = 0.
↪265) (  1-Cl-     2s = 0.262)
[C  58] occ =  2.00000 fii =     -2.660 (  3-Cl-     3s = 0.383) (  3-Cl-    3py = 0.
↪265) (  3-Cl-     2s = 0.262)
[C  59] occ =  2.00000 fii =     -2.660 (  6-Cl-     3s = 0.383) (  6-Cl-    3pz = 0.
↪265) (  6-Cl-     2s = 0.262)
[C  60] occ =  2.00000 fii =     -2.660 (  2-Cl-     3s = 0.383) (  2-Cl-    3px = 0.
↪265) (  2-Cl-     2s = 0.262)
[C  61] occ =  2.00000 fii =     -2.660 (  4-Cl-     3s = 0.383) (  4-Cl-    3py = 0.
↪265) (  4-Cl-     2s = 0.262)
```

(continues on next page)

(continued from previous page)

```
[C  62] occ =  2.00000 fii =     -2.654 (  5-Cl-     3s = 0.383) (  5-Cl-     3pz = 0.
↪265) (  5-Cl-     2s = 0.261)
[C  63] occ =  2.00000 fii =     -0.381 (  1-Cl-    3px = 0.590) (  1-Cl-     3s = 0.
↪156) (  0-Dy-    6px = 0.049)
[C  64] occ =  2.00000 fii =     -0.382 (  6-Cl-    3pz = 0.590) (  6-Cl-     3s = 0.
↪156) (  0-Dy-    6pz = 0.049)
[C  65] occ =  2.00000 fii =     -0.382 (  2-Cl-    3px = 0.590) (  2-Cl-     3s = 0.
↪156) (  0-Dy-    6px = 0.049)
[C  66] occ =  2.00000 fii =     -0.381 (  4-Cl-    3py = 0.590) (  4-Cl-     3s = 0.
↪156) (  0-Dy-    6py = 0.049)
[C  67] occ =  2.00000 fii =     -0.382 (  5-Cl-    3pz = 0.590) (  5-Cl-     3s = 0.
↪156) (  0-Dy-    6pz = 0.049)
[C  68] occ =  2.00000 fii =     -0.381 (  3-Cl-    3py = 0.590) (  3-Cl-     3s = 0.
↪156) (  0-Dy-    6py = 0.049)
[C  69] occ =  1.99999 fii =     -0.005 (  4-Cl-    3pz = 0.918) (  0-Dy-   7f+2 = 0.
↪010) (  0-Dy-   6dyz = 0.010)
[C  70] occ =  1.99999 fii =     -0.005 (  5-Cl-    3py = 0.690) (  5-Cl-    3px = 0.
↪237) (  0-Dy-   7f-1 = 0.012)
[C  71] occ =  1.99999 fii =     -0.005 (  3-Cl-    3pz = 0.918) (  0-Dy-   7f+2 = 0.
↪010) (  0-Dy-   6dyz = 0.010)
[C  72] occ =  1.99999 fii =     -0.005 (  6-Cl-    3py = 0.690) (  6-Cl-    3px = 0.
↪237) (  0-Dy-   7f-1 = 0.012)
[C  73] occ =  1.99999 fii =     -0.005 (  2-Cl-    3pz = 0.924) (  0-Dy-   7f+2 = 0.
↪010) (  0-Dy-   6dxz = 0.010)
[C  74] occ =  1.99999 fii =     -0.005 (  1-Cl-    3pz = 0.924) (  0-Dy-   7f+2 = 0.
↪010) (  0-Dy-   6dxz = 0.010)
[C  75] occ =  1.99999 fii =     -0.004 (  2-Cl-    3py = 0.923) (  0-Dy-   7f-3 = 0.
↪015) (  0-Dy-   6dxy = 0.010)
[C  76] occ =  1.99999 fii =     -0.004 (  1-Cl-    3py = 0.923) (  0-Dy-   7f-3 = 0.
↪015) (  0-Dy-   6dxy = 0.010)
[C  77] occ =  1.99999 fii =     -0.003 (  3-Cl-    3px = 0.917) (  0-Dy-   7f+3 = 0.
↪015) (  0-Dy-   6dxy = 0.010)
[C  78] occ =  1.99999 fii =     -0.003 (  4-Cl-    3px = 0.917) (  0-Dy-   7f+3 = 0.
↪015) (  0-Dy-   6dxy = 0.010)
[C  79] occ =  1.99999 fii =     -0.003 (  6-Cl-    3px = 0.690) (  6-Cl-    3py = 0.
↪236) (  0-Dy-   7f+1 = 0.012)
[C  80] occ =  1.99999 fii =     -0.003 (  5-Cl-    3px = 0.690) (  5-Cl-    3py = 0.
↪236) (  0-Dy-   7f+1 = 0.012)
[C  81] occ =  1.99999 fii =      0.120 (  0-Dy-   4f-1 = 0.410) (  0-Dy-   4f-3 = 0.
↪246) (  0-Dy-   4f+1 = 0.214)
[C  82] occ =  1.99999 fii =      0.121 (  0-Dy-   4f+2 = 0.974) (  0-Dy-   4f+1 = 0.
↪012) (  0-Dy-   4f+3 = 0.007)
[A  83] occ =  1.00000 fii =      0.122 (  0-Dy-   4f-3 = 0.617) (  0-Dy-   4f-1 = 0.
↪366) (  4-Cl-    3py = 0.005)
[A  84] occ =  1.00000 fii =      0.122 (  0-Dy-   4f+0 = 0.983) (  5-Cl-    3pz = 0.
```

(continues on next page)

(continued from previous page)

```
↪005) (  6-Cl-    3pz = 0.005)
[A  85] occ =  1.00000 fii =      0.121 (  0-Dy-   4f+1 = 0.399) (  0-Dy-   4f+3 = 0.
↪236) (  0-Dy-   4f-1 = 0.212)
[A  86] occ =  1.00000 fii =      0.122 (  0-Dy-   4f-2 = 1.000) (  0-Dy-   6f-2 = 0.
↪000) (  0-Dy-   5f-2 = 0.000)
[A  87] occ =  1.00000 fii =      0.122 (  0-Dy-   4f+3 = 0.617) (  0-Dy-   4f+1 = 0.
↪366) (  1-Cl-    3px = 0.005)
[V  88] occ =  0.00001 fii =      2.226 (  0-Dy-   5f+2 = 0.996) (  2-Cl-    3pz = 0.
↪001) (  1-Cl-    3pz = 0.001)
[V  89] occ =  0.00001 fii =      1.269 (  0-Dy-    6s = 0.800) (  0-Dy-    5s = 0.
↪028) (  5-Cl-     3s = 0.018)
[V  90] occ =  0.00001 fii =      0.728 (  2-Cl-    4pz = 0.990) (  2-Cl-    3pz = 0.
↪004) (  6-Cl-    3px = 0.001)
[V  91] occ =  0.00001 fii =      0.728 (  3-Cl-    4pz = 0.990) (  3-Cl-    3pz = 0.
↪004) (  6-Cl-    3py = 0.001)
[V  92] occ =  0.00001 fii =      0.728 (  4-Cl-    4pz = 0.990) (  4-Cl-    3pz = 0.
↪004) (  5-Cl-    3py = 0.001)
[V  93] occ =  0.00001 fii =      0.728 (  1-Cl-    4pz = 0.990) (  1-Cl-    3pz = 0.
↪004) (  5-Cl-    3px = 0.001)
[V  94] occ =  0.00001 fii =      0.728 (  5-Cl-    4py = 0.990) (  5-Cl-    3py = 0.
↪004) (  4-Cl-    3pz = 0.001)
[V  95] occ =  0.00001 fii =      0.728 (  6-Cl-    4py = 0.990) (  6-Cl-    3py = 0.
↪004) (  3-Cl-    3pz = 0.001)
[V  96] occ =  0.00001 fii =      0.728 (  2-Cl-    4py = 0.990) (  2-Cl-    3py = 0.
↪004) (  4-Cl-    3px = 0.001)
[V  97] occ =  0.00001 fii =      0.728 (  1-Cl-    4py = 0.990) (  1-Cl-    3py = 0.
↪004) (  3-Cl-    3px = 0.001)
[V  98] occ =  0.00001 fii =      0.728 (  5-Cl-    4px = 0.990) (  5-Cl-    3px = 0.
↪004) (  1-Cl-    3pz = 0.001)
[V  99] occ =  0.00001 fii =      0.728 (  6-Cl-    4px = 0.990) (  6-Cl-    3px = 0.
↪004) (  2-Cl-    3pz = 0.001)
[V 100] occ =  0.00001 fii =      0.728 (  3-Cl-    4px = 0.990) (  3-Cl-    3px = 0.
↪004) (  1-Cl-    3py = 0.001)
[V 101] occ =  0.00001 fii =      0.728 (  4-Cl-    4px = 0.990) (  4-Cl-    3px = 0.
↪004) (  2-Cl-    3py = 0.001)
[V 102] occ =  0.00000 fii =      0.708 (  0-Dy-  5dz^2 = 0.876) (  5-Cl-    3pz = 0.
↪025) (  6-Cl-    3pz = 0.025)
[V 103] occ =  0.00000 fii =      0.708 (  0-Dy-5dx2-y2 = 0.876) (  4-Cl-    3py = 0.
↪018) (  3-Cl-    3py = 0.018)
[V 104] occ =  0.00000 fii =      2.246 (  0-Dy-   5f-1 = 0.992) (  5-Cl-    3py = 0.
↪002) (  6-Cl-    3py = 0.002)
[V 105] occ =  0.00000 fii =      0.682 (  6-Cl-    4pz = 0.966) (  0-Dy-    5s = 0.
↪010) (  0-Dy-    5pz = 0.008)
[V 106] occ =  0.00000 fii =      0.682 (  5-Cl-    4pz = 0.966) (  0-Dy-    5s = 0.
↪010) (  0-Dy-    5pz = 0.008)
```

(continues on next page)

(continued from previous page)

```
[V 107] occ =  0.00000 fii =     0.682 (  4-Cl-    4py = 0.966) (  0-Dy-     5s = 0.
↪010) (  0-Dy-    5py = 0.008)
[V 108] occ =  0.00000 fii =     0.682 (  1-Cl-    4px = 0.966) (  0-Dy-     5s = 0.
↪010) (  0-Dy-    5px = 0.008)
[V 109] occ =  0.00000 fii =     0.682 (  2-Cl-    4px = 0.966) (  0-Dy-     5s = 0.
↪010) (  0-Dy-    5px = 0.008)
[V 110] occ =  0.00000 fii =     0.682 (  3-Cl-    4py = 0.966) (  0-Dy-     5s = 0.
↪010) (  0-Dy-    5py = 0.008)
[V 111] occ =  0.00000 fii =     0.694 (  0-Dy-     7s = 0.941) (  5-Cl-    3pz = 0.
↪006) (  6-Cl-    3pz = 0.006)
[V 112] occ =  0.00000 fii =     0.463 (  0-Dy-   5dxy = 0.965) (  4-Cl-    3px = 0.
↪009) (  2-Cl-    3py = 0.009)
[V 113] occ =  0.00000 fii =     0.463 (  0-Dy-   5dyz = 0.965) (  4-Cl-    3pz = 0.
↪009) (  3-Cl-    3pz = 0.009)
[V 114] occ =  0.00000 fii =     0.463 (  0-Dy-   5dxz = 0.965) (  5-Cl-    3px = 0.
↪009) (  2-Cl-    3pz = 0.009)
[V 115] occ =  0.00000 fii =     1.098 (  5-Cl-     4s = 0.976) (  5-Cl-     3s = 0.
↪010) (  0-Dy-    5pz = 0.005)
[V 116] occ =  0.00000 fii =     1.098 (  6-Cl-     4s = 0.976) (  6-Cl-     3s = 0.
↪010) (  0-Dy-    5pz = 0.005)
[V 117] occ =  0.00000 fii =     1.098 (  4-Cl-     4s = 0.976) (  4-Cl-     3s = 0.
↪010) (  0-Dy-    5py = 0.005)
[V 118] occ =  0.00000 fii =     1.098 (  2-Cl-     4s = 0.976) (  2-Cl-     3s = 0.
↪010) (  0-Dy-    5px = 0.005)
[V 119] occ =  0.00000 fii =     1.098 (  1-Cl-     4s = 0.976) (  1-Cl-     3s = 0.
↪010) (  0-Dy-    5px = 0.005)
[V 120] occ =  0.00000 fii =     1.098 (  3-Cl-     4s = 0.976) (  3-Cl-     3s = 0.
↪010) (  0-Dy-    5py = 0.005)
[V 121] occ =  0.00000 fii =     1.034 (  0-Dy-    6py = 0.848) (  4-Cl-     3s = 0.
↪036) (  3-Cl-     3s = 0.036)
[V 122] occ =  0.00000 fii =     1.034 (  0-Dy-    6px = 0.848) (  1-Cl-     3s = 0.
↪036) (  2-Cl-     3s = 0.036)
[V 123] occ =  0.00000 fii =     1.034 (  0-Dy-    6pz = 0.848) (  5-Cl-     3s = 0.
↪036) (  6-Cl-     3s = 0.036)
[V 124] occ =  0.00000 fii =     2.260 (  0-Dy-   5f-3 = 0.990) (  1-Cl-    3py = 0.
↪002) (  2-Cl-    3py = 0.002)
[V 125] occ =  0.00000 fii =     2.246 (  0-Dy-   5f+1 = 0.992) (  5-Cl-    3px = 0.
↪002) (  6-Cl-    3px = 0.002)
[V 126] occ =  0.00000 fii =     1.986 (  0-Dy-     8s = 0.973) (  5-Cl-    3pz = 0.
↪004) (  6-Cl-    3pz = 0.004)
[V 127] occ =  0.00000 fii =     6.040 (  0-Dy-     9s = 0.991) (  1-Cl-     3s = 0.
↪001) (  6-Cl-     3s = 0.001)
[V 128] occ =  0.00000 fii =     2.260 (  0-Dy-   5f+3 = 0.990) (  4-Cl-    3px = 0.
↪002) (  3-Cl-    3px = 0.002)
[V 129] occ =  0.00000 fii =     1.921 (  0-Dy-    7px = 0.944) (  2-Cl-     3s = 0.
```

(continues on next page)

(continued from previous page)

```
↪019) (  1-Cl-     3s = 0.019)
[V 130] occ =  0.00000 fii =      1.921 (  0-Dy-    7py = 0.944) (  3-Cl-     3s = 0.
↪019) (  4-Cl-     3s = 0.019)
[V 131] occ =  0.00000 fii =      1.921 (  0-Dy-    7pz = 0.944) (  5-Cl-     3s = 0.
↪019) (  6-Cl-     3s = 0.019)
[V 132] occ =  0.00000 fii =      4.197 (  0-Dy-    8px = 0.969) (  2-Cl-    3px = 0.
↪007) (  1-Cl-    3px = 0.007)
[V 133] occ =  0.00000 fii =      4.197 (  0-Dy-    8py = 0.969) (  3-Cl-    3py = 0.
↪007) (  4-Cl-    3py = 0.007)
[V 134] occ =  0.00000 fii =      4.197 (  0-Dy-    8pz = 0.969) (  5-Cl-    3pz = 0.
↪007) (  6-Cl-    3pz = 0.007)
[V 135] occ =  0.00000 fii =      3.403 (  0-Dy-   6f+2 = 0.978) (  2-Cl-    3pz = 0.
↪005) (  4-Cl-    3pz = 0.005)
[V 136] occ =  0.00000 fii =      3.595 (  0-Dy-   7dxz = 0.971) (  5-Cl-    3px = 0.
↪007) (  1-Cl-    3pz = 0.007)
[V 137] occ =  0.00000 fii =      3.595 (  0-Dy-   7dxy = 0.971) (  1-Cl-    3py = 0.
↪007) (  3-Cl-    3px = 0.007)
[V 138] occ =  0.00000 fii =      3.595 (  0-Dy-   7dyz = 0.971) (  5-Cl-    3py = 0.
↪007) (  4-Cl-    3pz = 0.007)
[V 139] occ =  0.00000 fii =      8.178 (  0-Dy-    9px = 0.974) (  2-Cl-    3px = 0.
↪011) (  1-Cl-    3px = 0.011)
[V 140] occ =  0.00000 fii =      8.178 (  0-Dy-    9py = 0.974) (  3-Cl-    3py = 0.
↪011) (  4-Cl-    3py = 0.011)
[V 141] occ =  0.00000 fii =      8.178 (  0-Dy-    9pz = 0.974) (  5-Cl-    3pz = 0.
↪011) (  6-Cl-    3pz = 0.011)
[V 142] occ =  0.00000 fii =      6.337 (  0-Dy-   7f+2 = 0.958) (  4-Cl-    3pz = 0.
↪010) (  2-Cl-    3pz = 0.010)
[V 143] occ =  0.00000 fii =      3.363 (  0-Dy-   6f-1 = 0.965) (  5-Cl-    3py = 0.
↪009) (  6-Cl-    3py = 0.009)
[V 144] occ =  0.00000 fii =      4.101 (  0-Dy-7dx2-y2 = 0.936) (  3-Cl-     3s = 0.
↪014) (  4-Cl-     3s = 0.014)
[V 145] occ =  0.00000 fii =      4.101 (  0-Dy-  7dz^2 = 0.936) (  5-Cl-     3s = 0.
↪018) (  6-Cl-     3s = 0.018)
[V 146] occ =  0.00000 fii =      3.336 (  0-Dy-   6f-3 = 0.957) (  1-Cl-    3py = 0.
↪008) (  2-Cl-    3py = 0.008)
[V 147] occ =  0.00000 fii =      6.748 (  0-Dy-   7f-1 = 0.943) (  5-Cl-    3py = 0.
↪017) (  6-Cl-    3py = 0.017)
[V 148] occ =  0.00000 fii =      6.748 (  0-Dy-   7f+1 = 0.943) (  5-Cl-    3px = 0.
↪017) (  6-Cl-    3px = 0.017)
[V 149] occ =  0.00000 fii =      3.363 (  0-Dy-   6f+1 = 0.965) (  5-Cl-    3px = 0.
↪009) (  6-Cl-    3px = 0.009)
[V 150] occ =  0.00000 fii =      7.022 (  0-Dy-   7f-3 = 0.933) (  1-Cl-    3py = 0.
↪016) (  2-Cl-    3py = 0.016)
[V 151] occ =  0.00000 fii =      3.336 (  0-Dy-   6f+3 = 0.957) (  4-Cl-    3px = 0.
↪008) (  3-Cl-    3px = 0.008)
```

(continues on next page)

(continued from previous page)

```
[V 152] occ =  0.00000 fii =      7.022 (  0-Dy-   7f+3 = 0.933) (  3-Cl-    3px = 0.
↪016) (  4-Cl-    3px = 0.016)
[V 153] occ =  0.00000 fii =      1.466 (  0-Dy-  6dz^2 = 0.882) (  5-Cl-     3s = 0.
↪028) (  6-Cl-     3s = 0.028)
[V 154] occ =  0.00000 fii =      1.466 (  0-Dy-6dx2-y2 = 0.882) (  2-Cl-     3s = 0.
↪021) (  1-Cl-     3s = 0.021)
[V 155] occ =  0.00000 fii =      2.280 (  0-Dy-   5f+0 = 0.986) (  6-Cl-    3pz = 0.
↪003) (  5-Cl-    3pz = 0.003)
[V 156] occ =  0.00000 fii =      3.296 (  0-Dy-   6f+0 = 0.945) (  6-Cl-     3s = 0.
↪012) (  5-Cl-     3s = 0.012)
[V 157] occ =  0.00000 fii =      1.134 (  0-Dy-   6dxy = 0.958) (  4-Cl-    3px = 0.
↪010) (  2-Cl-    3py = 0.010)
[V 158] occ =  0.00000 fii =      1.134 (  0-Dy-   6dxz = 0.958) (  5-Cl-    3px = 0.
↪010) (  2-Cl-    3pz = 0.010)
[V 159] occ =  0.00000 fii =      1.134 (  0-Dy-   6dyz = 0.958) (  4-Cl-    3pz = 0.
↪010) (  3-Cl-    3pz = 0.010)
[V 160] occ =  0.00000 fii =      7.433 (  0-Dy-   7f+0 = 0.919) (  6-Cl-     3s = 0.
↪022) (  5-Cl-     3s = 0.022)
[V 161] occ =  0.00000 fii =      3.642 (  0-Dy-   6f-2 = 1.000) (  0-Dy-   4f-2 = 0.
↪000) (  0-Dy-   5f-2 = 0.000)
[V 162] occ =  0.00000 fii =      5.314 (  0-Dy-   7f-2 = 1.000) (  0-Dy-   4f-2 = 0.
↪000) (  0-Dy-   6f-2 = 0.000)
[V 163] occ = -0.00000 fii =      2.207 (  0-Dy-   5f-2 = 1.000) (  0-Dy-   4f-2 = 0.
↪000) (  0-Dy-   6f-2 = 0.000)
   [  0] occ = 2.00000 | (0-Dy-1s       = 1.000) (0-Dy-4dxy     = 0.000) (0-Dy-4dyz ␣
↪   = 0.000)
   [  1] occ = 2.00000 | (4-Cl-1s       = 0.570) (4-Cl-2py      = 0.304) (4-Cl-2pz  ␣
↪   = 0.089)
   [  2] occ = 2.00000 | (6-Cl-2px      = 0.699) (6-Cl-2py      = 0.256) (6-Cl-1s   ␣
↪   = 0.041)
   [  3] occ = 2.00000 | (5-Cl-1s       = 0.525) (5-Cl-2pz      = 0.304) (5-Cl-2py  ␣
↪   = 0.092)
   [  4] occ = 2.00000 | (4-Cl-2px      = 0.819) (4-Cl-2pz      = 0.177) (4-Cl-1s   ␣
↪   = 0.002)
   [  5] occ = 2.00000 | (1-Cl-2py      = 0.629) (1-Cl-2pz      = 0.366) (1-Cl-1s   ␣
↪   = 0.003)
   [  6] occ = 2.00000 | (2-Cl-2py      = 0.979) (2-Cl-1s       = 0.011) (2-Cl-2pz  ␣
↪   = 0.008)
   [  7] occ = 2.00000 | (5-Cl-2px      = 0.741) (5-Cl-1s       = 0.167) (5-Cl-2py  ␣
↪   = 0.085)
   [  8] occ = 2.00000 | (3-Cl-2px      = 0.656) (3-Cl-2pz      = 0.320) (3-Cl-1s   ␣
↪   = 0.022)
   [  9] occ = 2.00000 | (2-Cl-2pz      = 0.989) (2-Cl-2py      = 0.007) (2-Cl-1s   ␣
↪   = 0.002)
   [ 10] occ = 2.00000 | (6-Cl-2py      = 0.741) (6-Cl-2px      = 0.230) (6-Cl-1s   ␣
```

(continues on next page)

(continued from previous page)

```
↪   = 0.026)
  [ 11] occ = 2.00000 | (5-Cl-2py      = 0.798) (5-Cl-2px      = 0.160) (5-Cl-1s    ␣
↪   = 0.039)
  [ 12] occ = 2.00000 | (4-Cl-2pz      = 0.713) (4-Cl-1s       = 0.141) (4-Cl-2px   ␣
↪   = 0.138)
  [ 13] occ = 2.00000 | (3-Cl-2pz      = 0.531) (3-Cl-1s       = 0.292) (3-Cl-2px   ␣
↪   = 0.164)
  [ 14] occ = 2.00000 | (1-Cl-2pz      = 0.404) (1-Cl-1s       = 0.305) (1-Cl-2py   ␣
↪   = 0.277)
  [ 15] occ = 2.00000 | (3-Cl-1s       = 0.647) (3-Cl-2px      = 0.179) (3-Cl-2pz   ␣
↪   = 0.148)
  [ 16] occ = 2.00000 | (0-Dy-3pz      = 0.418) (0-Dy-3dz^2    = 0.135) (0-Dy-3py   ␣
↪   = 0.114)
  [ 17] occ = 2.00000 | (6-Cl-1s       = 0.894) (6-Cl-2px      = 0.069) (6-Cl-2pz   ␣
↪   = 0.028)
  [ 18] occ = 2.00000 | (2-Cl-1s       = 0.948) (2-Cl-2px      = 0.030) (2-Cl-2py   ␣
↪   = 0.013)
  [ 19] occ = 2.00000 | (1-Cl-1s       = 0.665) (1-Cl-2pz      = 0.227) (1-Cl-2py   ␣
↪   = 0.092)
  [ 20] occ = 2.00000 | (0-Dy-2pz      = 0.321) (0-Dy-4px      = 0.124) (0-Dy-3dyz  ␣
↪   = 0.109)
  [ 21] occ = 2.00000 | (0-Dy-2py      = 0.210) (0-Dy-2s       = 0.201) (0-Dy-3dxy  ␣
↪   = 0.179)
  [ 22] occ = 2.00000 | (0-Dy-3px      = 0.276) (0-Dy-3py      = 0.246) (0-Dy-4py   ␣
↪   = 0.197)
  [ 23] occ = 2.00000 | (0-Dy-3dx2-y2  = 0.243) (0-Dy-4px      = 0.189) (0-Dy-3dyz  ␣
↪   = 0.082)
  [ 24] occ = 2.00000 | (0-Dy-2pz      = 0.244) (0-Dy-3dyz     = 0.212) (0-Dy-2py   ␣
↪   = 0.136)
  [ 25] occ = 2.00000 | (0-Dy-3dx2-y2  = 0.387) (0-Dy-3dxy     = 0.134) (0-Dy-2py   ␣
↪   = 0.086)
  [ 26] occ = 2.00000 | (0-Dy-2s       = 0.491) (0-Dy-3dxy     = 0.116) (0-Dy-3dz^2␣
↪   = 0.073)
  [ 27] occ = 2.00000 | (0-Dy-3s       = 0.370) (0-Dy-2px      = 0.114) (0-Dy-4s    ␣
↪   = 0.113)
  [ 28] occ = 2.00000 | (0-Dy-2pz      = 0.138) (0-Dy-3px      = 0.124) (0-Dy-3dyz  ␣
↪   = 0.094)
  [ 29] occ = 2.00000 | (0-Dy-2px      = 0.241) (0-Dy-3px      = 0.115) (0-Dy-3py   ␣
↪   = 0.107)
  [ 30] occ = 2.00000 | (0-Dy-4py      = 0.214) (0-Dy-4pz      = 0.192) (0-Dy-3pz   ␣
↪   = 0.183)
  [ 31] occ = 2.00000 | (1-Cl-2px      = 0.931) (1-Cl-2s       = 0.051) (1-Cl-1s    ␣
↪   = 0.009)
  [ 32] occ = 2.00000 | (3-Cl-2py      = 0.918) (3-Cl-2s       = 0.053) (3-Cl-1s    ␣
↪   = 0.022)
```

(continues on next page)

(continued from previous page)

```
  [ 33] occ = 2.00000 | (6-Cl-2pz      = 0.918) (6-Cl-2s       = 0.053) (6-Cl-1s    ␣
↪   = 0.022)
  [ 34] occ = 2.00000 | (2-Cl-2px      = 0.918) (2-Cl-2s       = 0.053) (2-Cl-1s    ␣
↪   = 0.022)
  [ 35] occ = 2.00000 | (0-Dy-3dz^2    = 0.334) (0-Dy-3dxy     = 0.097) (0-Dy-3dyz  ␣
↪   = 0.096)
  [ 36] occ = 2.00000 | (0-Dy-4pz      = 0.273) (0-Dy-3py      = 0.152) (0-Dy-2px   ␣
↪   = 0.149)
  [ 37] occ = 2.00000 | (0-Dy-3dxz     = 0.508) (0-Dy-3dyz     = 0.131) (0-Dy-4dyz  ␣
↪   = 0.097)
  [ 38] occ = 2.00000 | (0-Dy-4px      = 0.268) (0-Dy-2px      = 0.139) (0-Dy-3s    ␣
↪   = 0.139)
  [ 39] occ = 2.00000 | (4-Cl-2py      = 0.638) (4-Cl-1s       = 0.269) (4-Cl-2s    ␣
↪   = 0.057)
  [ 40] occ = 2.00000 | (0-Dy-4s       = 0.242) (0-Dy-4py      = 0.195) (0-Dy-3s    ␣
↪   = 0.085)
  [ 41] occ = 2.00000 | (0-Dy-4dxy     = 0.584) (0-Dy-2py      = 0.136) (0-Dy-3s    ␣
↪   = 0.095)
  [ 42] occ = 2.00000 | (0-Dy-4dyz     = 0.376) (0-Dy-4dxz     = 0.166) (0-Dy-3dxy  ␣
↪   = 0.108)
  [ 43] occ = 2.00000 | (0-Dy-4dxz     = 0.468) (0-Dy-4dyz     = 0.324) (0-Dy-4dxy  ␣
↪   = 0.040)
  [ 44] occ = 2.00000 | (0-Dy-4dx2-y2  = 0.654) (0-Dy-4dz^2    = 0.278) (0-Dy-3dx2-
↪y2  = 0.027)
  [ 45] occ = 2.00000 | (0-Dy-4dz^2    = 0.666) (0-Dy-4dx2-y2  = 0.297) (0-Dy-3dz^2␣
↪   = 0.024)
  [ 46] occ = 2.00000 | (5-Cl-2pz      = 0.376) (5-Cl-2s       = 0.338) (5-Cl-1s    ␣
↪   = 0.135)
  [ 47] occ = 2.00000 | (5-Cl-2s       = 0.348) (5-Cl-2pz      = 0.312) (5-Cl-3s    ␣
↪   = 0.182)
  [ 48] occ = 2.00000 | (2-Cl-2s       = 0.628) (2-Cl-3s       = 0.288) (2-Cl-2px   ␣
↪   = 0.049)
  [ 49] occ = 2.00000 | (4-Cl-2s       = 0.628) (4-Cl-3s       = 0.288) (4-Cl-2py   ␣
↪   = 0.049)
  [ 50] occ = 2.00000 | (6-Cl-2s       = 0.628) (6-Cl-3s       = 0.288) (6-Cl-2pz   ␣
↪   = 0.049)
  [ 51] occ = 2.00000 | (3-Cl-2s       = 0.628) (3-Cl-3s       = 0.288) (3-Cl-2py   ␣
↪   = 0.049)
  [ 52] occ = 2.00000 | (1-Cl-2s       = 0.628) (1-Cl-3s       = 0.288) (1-Cl-2px   ␣
↪   = 0.049)
  [ 53] occ = 2.00000 | (0-Dy-5py      = 1.017) (0-Dy-4py      = 0.004) (3-Cl-3py   ␣
↪   = 0.003)
  [ 54] occ = 2.00000 | (0-Dy-5pz      = 1.013) (0-Dy-5s       = 0.006) (0-Dy-4pz   ␣
↪   = 0.004)
  [ 55] occ = 2.00000 | (0-Dy-5px      = 0.510) (0-Dy-5s       = 0.473) (0-Dy-4s    ␣
```

(continues on next page)

(continued from previous page)

```
↪   = 0.026)
  [ 56] occ = 2.00000 | (0-Dy-5px      = 0.509) (0-Dy-5s       = 0.473) (0-Dy-4s    ␣
↪   = 0.026)
  [ 57] occ = 2.00000 | (1-Cl-3s       = 0.453) (1-Cl-2s       = 0.279) (1-Cl-3px   ␣
↪   = 0.254)
  [ 58] occ = 2.00000 | (3-Cl-3s       = 0.453) (3-Cl-2s       = 0.279) (3-Cl-3py   ␣
↪   = 0.254)
  [ 59] occ = 2.00000 | (6-Cl-3s       = 0.453) (6-Cl-2s       = 0.279) (6-Cl-3pz   ␣
↪   = 0.254)
  [ 60] occ = 2.00000 | (2-Cl-3s       = 0.453) (2-Cl-2s       = 0.279) (2-Cl-3px   ␣
↪   = 0.254)
  [ 61] occ = 2.00000 | (4-Cl-3s       = 0.453) (4-Cl-2s       = 0.279) (4-Cl-3py   ␣
↪   = 0.254)
  [ 62] occ = 2.00000 | (5-Cl-3s       = 0.454) (5-Cl-2s       = 0.277) (5-Cl-3pz   ␣
↪   = 0.254)
  [ 63] occ = 2.00000 | (1-Cl-3px      = 0.678) (1-Cl-3s       = 0.224) (1-Cl-2s    ␣
↪   = 0.030)
  [ 64] occ = 2.00000 | (6-Cl-3pz      = 0.678) (6-Cl-3s       = 0.224) (6-Cl-2s    ␣
↪   = 0.030)
  [ 65] occ = 2.00000 | (2-Cl-3px      = 0.678) (2-Cl-3s       = 0.224) (2-Cl-2s    ␣
↪   = 0.030)
  [ 66] occ = 2.00000 | (4-Cl-3py      = 0.678) (4-Cl-3s       = 0.224) (4-Cl-2s    ␣
↪   = 0.030)
  [ 67] occ = 2.00000 | (5-Cl-3pz      = 0.678) (5-Cl-3s       = 0.224) (5-Cl-2s    ␣
↪   = 0.030)
  [ 68] occ = 2.00000 | (3-Cl-3py      = 0.678) (3-Cl-3s       = 0.224) (3-Cl-2s    ␣
↪   = 0.030)
  [ 69] occ = 1.99999 | (4-Cl-3pz      = 0.969) (4-Cl-3px      = 0.009) (4-Cl-2pz   ␣
↪   = 0.001)
  [ 70] occ = 1.99999 | (5-Cl-3py      = 0.729) (5-Cl-3px      = 0.250) (5-Cl-2py   ␣
↪   = 0.001)
  [ 71] occ = 1.99999 | (3-Cl-3pz      = 0.969) (3-Cl-3px      = 0.009) (3-Cl-2pz   ␣
↪   = 0.001)
  [ 72] occ = 1.99999 | (6-Cl-3py      = 0.729) (6-Cl-3px      = 0.250) (0-Dy-5dyz  ␣
↪   = 0.001)
  [ 73] occ = 1.99999 | (2-Cl-3pz      = 0.975) (2-Cl-3py      = 0.003) (2-Cl-2pz   ␣
↪   = 0.001)
  [ 74] occ = 1.99999 | (1-Cl-3pz      = 0.975) (1-Cl-3py      = 0.003) (1-Cl-2pz   ␣
↪   = 0.001)
  [ 75] occ = 1.99999 | (2-Cl-3py      = 0.975) (2-Cl-3pz      = 0.003) (0-Dy-5dxy  ␣
↪   = 0.001)
  [ 76] occ = 1.99999 | (1-Cl-3py      = 0.975) (1-Cl-3pz      = 0.003) (0-Dy-5dxy  ␣
↪   = 0.001)
  [ 77] occ = 1.99999 | (3-Cl-3px      = 0.968) (3-Cl-3pz      = 0.009) (0-Dy-5dxy  ␣
↪   = 0.001)
```

(continues on next page)

(continued from previous page)

```
  [ 78] occ = 1.99999 | (4-Cl-3px      = 0.968) (4-Cl-3pz      = 0.009) (0-Dy-5dxy
↪   = 0.001)
  [ 79] occ = 1.99999 | (6-Cl-3px      = 0.728) (6-Cl-3py      = 0.250) (0-Dy-5dxz
↪   = 0.001)
  [ 80] occ = 1.99999 | (5-Cl-3px      = 0.728) (5-Cl-3py      = 0.249) (0-Dy-5dxz
↪   = 0.001)
* [ 81] occ = 1.99999 | (0-Dy-4f-1     = 0.410) (0-Dy-4f-3     = 0.246) (0-Dy-4f+1
↪   = 0.214)
* [ 82] occ = 1.99999 | (0-Dy-4f+2     = 0.975) (0-Dy-4f+1     = 0.012) (0-Dy-4f+3
↪   = 0.007)
* [ 83] occ = 1.00000 | (0-Dy-4f-3     = 0.620) (0-Dy-4f-1     = 0.368) (4-Cl-3py
↪   = 0.007)
* [ 84] occ = 1.00000 | (0-Dy-4f+0     = 0.989) (5-Cl-3pz      = 0.007) (6-Cl-3pz
↪   = 0.007)
* [ 85] occ = 1.00000 | (0-Dy-4f+1     = 0.400) (0-Dy-4f+3     = 0.236) (0-Dy-4f-1
↪   = 0.212)
* [ 86] occ = 1.00000 | (0-Dy-4f-2     = 1.000) (0-Dy-6f-2     = 0.000) (0-Dy-5f-2
↪   = 0.000)
* [ 87] occ = 1.00000 | (0-Dy-4f+3     = 0.620) (0-Dy-4f+1     = 0.368) (1-Cl-3px
↪   = 0.007)
  [ 88] occ = 0.00001 | (0-Dy-5f+2     = 1.026) (1-Cl-3pz      = 0.005) (2-Cl-3pz
↪   = 0.005)
  [ 89] occ = 0.00001 | (0-Dy-6s       = 19.659) (0-Dy-5s       = 2.301) (5-Cl-3s
↪    = 0.620)
  [ 90] occ = 0.00001 | (2-Cl-4pz      = 1.531) (0-Dy-6pz      = 0.066) (2-Cl-3pz
↪   = 0.054)
  [ 91] occ = 0.00001 | (3-Cl-4pz      = 1.531) (0-Dy-6pz      = 0.066) (3-Cl-3pz
↪   = 0.054)
  [ 92] occ = 0.00001 | (4-Cl-4pz      = 1.531) (0-Dy-6pz      = 0.066) (4-Cl-3pz
↪   = 0.054)
  [ 93] occ = 0.00001 | (1-Cl-4pz      = 1.531) (0-Dy-6pz      = 0.066) (1-Cl-3pz
↪   = 0.054)
  [ 94] occ = 0.00001 | (5-Cl-4py      = 1.531) (0-Dy-6py      = 0.066) (5-Cl-3py
↪   = 0.054)
  [ 95] occ = 0.00001 | (6-Cl-4py      = 1.531) (0-Dy-6py      = 0.066) (6-Cl-3py
↪   = 0.054)
  [ 96] occ = 0.00001 | (2-Cl-4py      = 1.531) (0-Dy-6py      = 0.066) (2-Cl-3py
↪   = 0.054)
  [ 97] occ = 0.00001 | (1-Cl-4py      = 1.531) (0-Dy-6py      = 0.066) (1-Cl-3py
↪   = 0.054)
  [ 98] occ = 0.00001 | (5-Cl-4px      = 1.531) (0-Dy-6px      = 0.066) (5-Cl-3px
↪   = 0.054)
  [ 99] occ = 0.00001 | (6-Cl-4px      = 1.531) (0-Dy-6px      = 0.066) (6-Cl-3px
↪   = 0.054)
  [100] occ = 0.00001 | (3-Cl-4px      = 1.531) (0-Dy-6px      = 0.066) (3-Cl-3px
```

(continues on next page)

(continued from previous page)

```
↪   = 0.054)
  [101] occ = 0.00001 | (4-Cl-4px      = 1.531) (0-Dy-6px      = 0.066) (4-Cl-3px  ␣
↪   = 0.054)
  [102] occ = 0.00000 | (0-Dy-5dz^2    = 2.427) (0-Dy-6dz^2    = 0.539) (5-Cl-3s   ␣
↪   = 0.417)
  [103] occ = 0.00000 | (0-Dy-5dx2-y2  = 2.427) (0-Dy-6dx2-y2  = 0.539) (4-Cl-3s   ␣
↪   = 0.313)
  [104] occ = 0.00000 | (0-Dy-5f-1     = 1.058) (3-Cl-3s       = 0.022) (4-Cl-3s   ␣
↪   = 0.022)
  [105] occ = 0.00000 | (6-Cl-4pz      = 1.910) (0-Dy-6s       = 0.208) (0-Dy-6pz  ␣
↪   = 0.196)
  [106] occ = 0.00000 | (5-Cl-4pz      = 1.910) (0-Dy-6s       = 0.208) (0-Dy-6pz  ␣
↪   = 0.196)
  [107] occ = 0.00000 | (4-Cl-4py      = 1.910) (0-Dy-6s       = 0.208) (0-Dy-6py  ␣
↪   = 0.196)
  [108] occ = 0.00000 | (1-Cl-4px      = 1.910) (0-Dy-6s       = 0.208) (0-Dy-6px  ␣
↪   = 0.196)
  [109] occ = 0.00000 | (2-Cl-4px      = 1.910) (0-Dy-6s       = 0.208) (0-Dy-6px  ␣
↪   = 0.196)
  [110] occ = 0.00000 | (3-Cl-4py      = 1.910) (0-Dy-6s       = 0.208) (0-Dy-6py  ␣
↪   = 0.196)
  [111] occ = 0.00000 | (0-Dy-7s       = 1.663) (0-Dy-5s       = 0.090) (0-Dy-6s   ␣
↪   = 0.054)
  [112] occ = 0.00000 | (0-Dy-5dxy     = 1.236) (4-Cl-3px      = 0.038) (2-Cl-3py  ␣
↪   = 0.038)
  [113] occ = 0.00000 | (0-Dy-5dyz     = 1.236) (4-Cl-3pz      = 0.038) (3-Cl-3pz  ␣
↪   = 0.038)
  [114] occ = 0.00000 | (0-Dy-5dxz     = 1.236) (5-Cl-3px      = 0.038) (2-Cl-3pz  ␣
↪   = 0.038)
  [115] occ = 0.00000 | (5-Cl-4s       = 3.066) (5-Cl-3s       = 0.934) (0-Dy-6pz  ␣
↪   = 0.871)
  [116] occ = 0.00000 | (6-Cl-4s       = 3.066) (6-Cl-3s       = 0.934) (0-Dy-6pz  ␣
↪   = 0.871)
  [117] occ = 0.00000 | (4-Cl-4s       = 3.066) (4-Cl-3s       = 0.934) (0-Dy-6py  ␣
↪   = 0.871)
  [118] occ = 0.00000 | (2-Cl-4s       = 3.066) (2-Cl-3s       = 0.934) (0-Dy-6px  ␣
↪   = 0.871)
  [119] occ = 0.00000 | (1-Cl-4s       = 3.066) (1-Cl-3s       = 0.934) (0-Dy-6px  ␣
↪   = 0.871)
  [120] occ = 0.00000 | (3-Cl-4s       = 3.066) (3-Cl-3s       = 0.934) (0-Dy-6py  ␣
↪   = 0.871)
  [121] occ = 0.00000 | (0-Dy-6py      = 9.853) (4-Cl-3s       = 1.899) (3-Cl-3s   ␣
↪   = 1.899)
  [122] occ = 0.00000 | (0-Dy-6px      = 9.853) (1-Cl-3s       = 1.899) (2-Cl-3s   ␣
↪   = 1.899)
```

(continues on next page)

(continued from previous page)

```
  [123] occ = 0.00000 | (0-Dy-6pz      = 9.853) (5-Cl-3s       = 1.899) (6-Cl-3s   ␣
↪   = 1.899)
  [124] occ = 0.00000 | (0-Dy-5f-3     = 1.080) (3-Cl-3s       = 0.037) (4-Cl-3s   ␣
↪   = 0.037)
  [125] occ = 0.00000 | (0-Dy-5f+1     = 1.058) (1-Cl-3s       = 0.022) (2-Cl-3s   ␣
↪   = 0.022)
  [126] occ = 0.00000 | (0-Dy-8s       = 1.416) (0-Dy-6s       = 0.290) (0-Dy-5s   ␣
↪   = 0.079)
  [127] occ = 0.00000 | (0-Dy-9s       = 1.182) (0-Dy-6s       = 0.241) (0-Dy-5s   ␣
↪   = 0.052)
  [128] occ = 0.00000 | (0-Dy-5f+3     = 1.080) (2-Cl-3s       = 0.037) (1-Cl-3s   ␣
↪   = 0.037)
  [129] occ = 0.00000 | (0-Dy-7px      = 3.006) (0-Dy-6px      = 1.406) (2-Cl-3s   ␣
↪   = 0.696)
  [130] occ = 0.00000 | (0-Dy-7py      = 3.006) (0-Dy-6py      = 1.406) (3-Cl-3s   ␣
↪   = 0.696)
  [131] occ = 0.00000 | (0-Dy-7pz      = 3.006) (0-Dy-6pz      = 1.406) (6-Cl-3s   ␣
↪   = 0.696)
  [132] occ = 0.00000 | (0-Dy-8px      = 1.480) (0-Dy-6px      = 0.177) (2-Cl-3s   ␣
↪   = 0.108)
  [133] occ = 0.00000 | (0-Dy-8py      = 1.480) (0-Dy-6py      = 0.177) (3-Cl-3s   ␣
↪   = 0.108)
  [134] occ = 0.00000 | (0-Dy-8pz      = 1.480) (0-Dy-6pz      = 0.177) (6-Cl-3s   ␣
↪   = 0.108)
  [135] occ = 0.00000 | (0-Dy-6f+2     = 1.164) (4-Cl-3pz      = 0.031) (2-Cl-3pz  ␣
↪   = 0.031)
  [136] occ = 0.00000 | (0-Dy-7dxz     = 1.406) (5-Cl-3px      = 0.050) (1-Cl-3pz  ␣
↪   = 0.050)
  [137] occ = 0.00000 | (0-Dy-7dxy     = 1.406) (1-Cl-3py      = 0.050) (3-Cl-3px  ␣
↪   = 0.050)
  [138] occ = 0.00000 | (0-Dy-7dyz     = 1.406) (5-Cl-3py      = 0.050) (4-Cl-3pz  ␣
↪   = 0.050)
  [139] occ = 0.00000 | (0-Dy-9px      = 1.273) (2-Cl-3px      = 0.069) (1-Cl-3px  ␣
↪   = 0.069)
  [140] occ = 0.00000 | (0-Dy-9py      = 1.273) (3-Cl-3py      = 0.069) (4-Cl-3py  ␣
↪   = 0.069)
  [141] occ = 0.00000 | (0-Dy-9pz      = 1.273) (5-Cl-3pz      = 0.069) (6-Cl-3pz  ␣
↪   = 0.069)
  [142] occ = 0.00000 | (0-Dy-7f+2     = 1.349) (4-Cl-3pz      = 0.062) (2-Cl-3pz  ␣
↪   = 0.062)
  [143] occ = 0.00000 | (0-Dy-6f-1     = 1.299) (3-Cl-3s       = 0.102) (4-Cl-3s   ␣
↪   = 0.102)
  [144] occ = 0.00000 | (0-Dy-7dx2-y2  = 1.896) (0-Dy-6dx2-y2  = 0.314) (3-Cl-3s   ␣
↪   = 0.256)
  [145] occ = 0.00000 | (0-Dy-7dz^2    = 1.896) (6-Cl-3s       = 0.341) (5-Cl-3s   ␣
```

(continues on next page)

(continued from previous page)

```
↪   = 0.341)
   [146] occ = 0.00000 | (0-Dy-6f-3     = 1.393) (3-Cl-3s       = 0.170) (4-Cl-3s   ␣
↪   = 0.170)
   [147] occ = 0.00000 | (0-Dy-7f-1     = 1.518) (3-Cl-3s       = 0.148) (4-Cl-3s   ␣
↪   = 0.148)
   [148] occ = 0.00000 | (0-Dy-7f+1     = 1.518) (1-Cl-3s       = 0.148) (2-Cl-3s   ␣
↪   = 0.148)
   [149] occ = 0.00000 | (0-Dy-6f+1     = 1.299) (1-Cl-3s       = 0.102) (2-Cl-3s   ␣
↪   = 0.102)
   [150] occ = 0.00000 | (0-Dy-7f-3     = 1.636) (3-Cl-3s       = 0.247) (4-Cl-3s   ␣
↪   = 0.247)
   [151] occ = 0.00000 | (0-Dy-6f+3     = 1.393) (2-Cl-3s       = 0.170) (1-Cl-3s   ␣
↪   = 0.170)
   [152] occ = 0.00000 | (0-Dy-7f+3     = 1.636) (1-Cl-3s       = 0.247) (2-Cl-3s   ␣
↪   = 0.247)
   [153] occ = 0.00000 | (0-Dy-6dz^2    = 3.625) (5-Cl-3s       = 0.711) (6-Cl-3s   ␣
↪   = 0.711)
   [154] occ = 0.00000 | (0-Dy-6dx2-y2  = 3.625) (1-Cl-3s       = 0.533) (3-Cl-3s   ␣
↪   = 0.533)
   [155] occ = 0.00000 | (0-Dy-5f+0     = 1.113) (6-Cl-3s       = 0.059) (5-Cl-3s   ␣
↪   = 0.059)
   [156] occ = 0.00000 | (0-Dy-6f+0     = 1.540) (6-Cl-3s       = 0.272) (5-Cl-3s   ␣
↪   = 0.272)
   [157] occ = 0.00000 | (0-Dy-6dxy     = 1.595) (4-Cl-3px      = 0.069) (2-Cl-3py  ␣
↪   = 0.069)
   [158] occ = 0.00000 | (0-Dy-6dxz     = 1.595) (5-Cl-3px      = 0.069) (2-Cl-3pz  ␣
↪   = 0.069)
   [159] occ = 0.00000 | (0-Dy-6dyz     = 1.595) (4-Cl-3pz      = 0.069) (3-Cl-3pz  ␣
↪   = 0.069)
   [160] occ = 0.00000 | (0-Dy-7f+0     = 1.822) (6-Cl-3s       = 0.395) (5-Cl-3s   ␣
↪   = 0.395)
   [161] occ = 0.00000 | (0-Dy-6f-2     = 1.000) (0-Dy-4f-2     = 0.000) (0-Dy-5f-2 ␣
↪   = 0.000)
   [162] occ = 0.00000 | (0-Dy-7f-2     = 1.000) (0-Dy-4f-2     = 0.000) (0-Dy-6f-2 ␣
↪   = 0.000)
   [163] occ =-0.00000 | (0-Dy-5f-2     = 1.000) (0-Dy-4f-2     = 0.000) (0-Dy-6f-2 ␣
↪   = 0.000)
cas list =  [81, 82, 83, 84, 85, 86, 87]
split localization at 0.1 ~ 1.9
active occ =  8.999985635920426 [1.99999282 1.99999282 1.         1.         1.     ␣
↪   1.
 1.        ]
low orbs =  []

[pm_loc_kernel]
```

(continues on next page)

(continued from previous page)

```
 mocoeff.shape= (164, 0)
 tol= 1e-06
 maxcycle= 1000
 partition= 7 \n [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,␣
↪19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
↪ 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,␣
↪60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
↪ 81, 82, 83, 84, 85], [86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98], [99,␣
↪100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111], [112, 113, 114, 115,␣
↪116, 117, 118, 119, 120, 121, 122, 123, 124], [125, 126, 127, 128, 129, 130, 131,␣
↪132, 133, 134, 135, 136, 137], [138, 139, 140, 141, 142, 143, 144, 145, 146, 147,␣
↪148, 149, 150], [151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163]]
 initial funval =  0.0
icycle= 0 delta= 0.0 fun= 0.0
CONG: PMloc converged!
mid orbs =  [2 3 4 5 6]

[pm_loc_kernel]
 mocoeff.shape= (164, 5)
 tol= 1e-06
 maxcycle= 1000
 partition= 7 \n [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,␣
↪19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
↪ 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,␣
↪60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
↪ 81, 82, 83, 84, 85], [86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98], [99,␣
↪100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111], [112, 113, 114, 115,␣
↪116, 117, 118, 119, 120, 121, 122, 123, 124], [125, 126, 127, 128, 129, 130, 131,␣
↪132, 133, 134, 135, 136, 137], [138, 139, 140, 141, 142, 143, 144, 145, 146, 147,␣
↪148, 149, 150], [151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163]]
 initial funval =  4.9060231169123165
icycle= 0 delta= 2.921607285039791e-11 fun= 4.906023116941532
CONG: PMloc converged!
high orbs =  [0 1]

[pm_loc_kernel]
 mocoeff.shape= (164, 2)
 tol= 1e-06
 maxcycle= 1000
 partition= 7 \n [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18,␣
↪19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
↪ 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,␣
↪60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
↪ 81, 82, 83, 84, 85], [86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98], [99,␣
↪100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111], [112, 113, 114, 115,␣
```

(continues on next page)

(continued from previous page)

```
↪116, 117, 118, 119, 120, 121, 122, 123, 124], [125, 126, 127, 128, 129, 130, 131,␣
↪132, 133, 134, 135, 136, 137], [138, 139, 140, 141, 142, 143, 144, 145, 146, 147,␣
↪148, 149, 150], [151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163]]
 initial funval =  2.0002973634558763
icycle= 0 delta= 4.1152794603266643e-19 fun= 2.0002973634558763
CONG: PMloc converged!
NACTORB = 7 NACTELEC = 9 NCORE = 81
 ** Mulliken atomic charges    ( Nelec_alpha | Nelec_beta )    magnetism **
charge of    0Dy =     -5.18560  (    36.66128     34.52432 )      2.13696
charge of    1Cl =      0.36427  (     8.55645      8.07928 )      0.47717
charge of    2Cl =      0.36427  (     8.55645      8.07928 )      0.47717
charge of    3Cl =      0.36427  (     8.55645      8.07928 )      0.47717
charge of    4Cl =      0.36427  (     8.55645      8.07928 )      0.47717
charge of    5Cl =      0.36427  (     8.55645      8.07928 )      0.47717
charge of    6Cl =      0.36427  (     8.55645      8.07928 )      0.47717
```

4.4.3 CASSCF Calculations

Next, we do state-averaged CASSCF for all sextet (pure total spin) states (without SOC). Unlike the procedure used in the first reference listed above, here we only do state-average for sextet, so that the result can be compariable with the "CASSCF-HS" results in the reference literature.

[4]:
```
lib.param.TMPDIR = os.path.abspath("./tmp")

mc = mcscf.CASSCF(mf, nactorb, nactelec)
mc.fcisolver = dmrgscf.DMRGCI(mol, maxM=1500, tol=1E-10)
mc.fcisolver.spin = 5
mc.fcisolver.nroots = 21
mc.fcisolver.runtimeDir = os.path.abspath(lib.param.TMPDIR)
mc.fcisolver.scratchDirectory = os.path.abspath(lib.param.TMPDIR)
mc.fcisolver.threads = 4
mc.fcisolver.memory = int(mol.max_memory / 1000) # mem in GB
mc.fcisolver.block_extra_keyword = ["real_density_matrix",
    "davidson_soft_max_iter 1600", "noreorder", "cutoff 1E-24"]
mc = mcscf.state_average_(mc, [1.0 / 21] * 21)
mc.verbose = 4
mc.kernel()
mf.mo_coeff = mc.mo_coeff

ncore = mc.ncore
ncas = mc.ncas
print('ncore = ', ncore, ' ncas = ', ncas)
```

```
******** <class 'pyscf.mcscf.addons._state_average_mcscf_solver.<locals>.
```

(continues on next page)

(continued from previous page)

```
↪StateAverageMCSCF'> ********
CAS (7e+2e, 7o), ncore = 81, nvir = 76
max_cycle_macro = 50
max_cycle_micro = 4
conv_tol = 1e-07
conv_tol_grad = None
orbital rotation max_stepsize = 0.02
orbital rotation threshold for CI restart = 0.01
augmented hessian ah_max_cycle = 30
augmented hessian ah_conv_tol = 1e-12
augmented hessian ah_linear dependence = 1e-14
augmented hessian ah_level shift = 1e-08
augmented hessian ah_start_tol = 2.5
augmented hessian ah_start_cycle = 3
augmented hessian ah_grad_trust_region = 3
kf_trust_region = 3
kf_interval = 4
ci_response_space = 4
ci_grad_trust_region = 3
with_dep4 0
natorb = False
canonicalization = True
sorting_mo_energy = False
ao2mo_level = 2
chkfile = /content/tmpryjxvk8l
max_memory 4000 MB (current use 911 MB)
internal_rotation = False

******** Block flags ********
executable             = /usr/local/bin/block2main
BLOCKEXE_COMPRESS_NEVPT= /path/to/serially/compiled/Block/block.spin_adapted
Block version          = 2.0
mpiprefix              =
scratchDirectory       = /content/tmp
integralFile           = /content/tmp/FCIDUMP
configFile             = /content/tmp/dmrg.conf
outputFile             = /content/tmp/dmrg.out
maxIter                = 38
scheduleSweeps         = [0, 4, 8, 12, 14, 16, 18, 20, 22, 24, 26]
scheduleMaxMs          = [200, 400, 800, 1500, 1500, 1500, 1500, 1500, 1500, 1500,␣
↪1500]
scheduleTols           = [0.0001, 0.0001, 0.0001, 0.0001, 1e-05, 1.0000000000000002e-
↪06, 1.0000000000000002e-07, 1.0000000000000002e-08, 1.0000000000000003e-09, 1.
↪0000000000000003e-10, 1.0000000000000001e-11]
scheduleNoises         = [0.0001, 0.0001, 0.0001, 0.0001, 1e-05, 1.0000000000000002e-
```

(continues on next page)

(continued from previous page)

```
↪06, 1.0000000000000002e-07, 1.0000000000000002e-08, 1.0000000000000003e-09, 1.
↪0000000000000003e-10, 0.0]
twodot_to_onedot       = 30
tol                    = 1e-10
maxM                   = 1500
dmrg switch tol        = 0.001
wfnsym                 = 1
fullrestart            = False
num_thrds              = 4
memory                 = 4

State-average over 21 states with weights [0.047619047619047616, 0.
↪047619047619047616, 0.047619047619047616, 0.047619047619047616, 0.
↪047619047619047616, 0.047619047619047616, 0.047619047619047616, 0.
↪047619047619047616, 0.047619047619047616, 0.047619047619047616, 0.
↪047619047619047616, 0.047619047619047616, 0.047619047619047616, 0.
↪047619047619047616, 0.047619047619047616, 0.047619047619047616, 0.
↪047619047619047616, 0.047619047619047616, 0.047619047619047616, 0.
↪047619047619047616, 0.047619047619047616]
CASCI E = -14918.4288385784  S^2 = 8.7500000
Set conv_tol_grad to 0.000316228
macro iter   1 ( 21 JK    4 micro), CASSCF E = -14918.8905225755  dE = -4.61683997e-
↪01  S^2 = 8.7500000
               |grad[o]|=  1.7  |ddm|=4.63e-15  |maxRot[o]|=0.0733
macro iter   2 ( 21 JK    4 micro), CASSCF E = -14918.8907530630  dE = -2.30487482e-
↪04  S^2 = 8.7500000
               |grad[o]|=0.0365  |ddm|=5.48e-15  |maxRot[o]|=0.00358
macro iter   3 (  6 JK    1 micro), CASSCF E = -14918.8907530657  dE = -2.73212208e-
↪09  S^2 = 8.7500000
               |grad[o]|=0.000314  |ddm|=3.32e-14  |maxRot[o]|=5.87e-06
1-step CASSCF converged in   3 macro ( 48 JK   9 micro) steps
CASSCF canonicalization
Density matrix diagonal elements [1.28571429 1.28571429 1.28571429 1.28571429 1.
↪28571429 1.28571429
 1.28571429]
CASSCF energy = -14918.8907530657
CASCI E = -14918.8907530657  E(CI) = -45.4381192621731  S^2 = 8.7500000
CASCI state-averaged energy = -14918.8907530658
CASCI energy for each state
  State 0 weight 0.047619  E = -14918.9254735699 S^2 = 8.7500000
  State 1 weight 0.047619  E = -14918.9254735631 S^2 = 8.7500000
  State 2 weight 0.047619  E = -14918.9254735574 S^2 = 8.7500000
  State 3 weight 0.047619  E = -14918.9247583778 S^2 = 8.7500000
  State 4 weight 0.047619  E = -14918.9247583706 S^2 = 8.7500000
  State 5 weight 0.047619  E = -14918.9247583638 S^2 = 8.7500000
```

(continues on next page)

(continued from previous page)

```
  State 6 weight 0.047619  E = -14918.9243548149 S^2 = 8.7500000
  State 7 weight 0.047619  E = -14918.9243547955 S^2 = 8.7500000
  State 8 weight 0.047619  E = -14918.9241469348 S^2 = 8.7500000
  State 9 weight 0.047619  E = -14918.9241469319 S^2 = 8.7500000
  State 10 weight 0.047619  E = -14918.9241469158 S^2 = 8.7500000
  State 11 weight 0.047619  E = -14918.891088414 S^2 = 8.7500000
  State 12 weight 0.047619  E = -14918.8910883948 S^2 = 8.7500000
  State 13 weight 0.047619  E = -14918.8910883749 S^2 = 8.7500000
  State 14 weight 0.047619  E = -14918.890571941 S^2 = 8.7500000
  State 15 weight 0.047619  E = -14918.8905719226 S^2 = 8.7500000
  State 16 weight 0.047619  E = -14918.8905719121 S^2 = 8.7500000
  State 17 weight 0.047619  E = -14918.8902125684 S^2 = 8.7500000
  State 18 weight 0.047619  E = -14918.7662582457 S^2 = 8.7500000
  State 19 weight 0.047619  E = -14918.7662582269 S^2 = 8.7500000
  State 20 weight 0.047619  E = -14918.7662581846 S^2 = 8.7500000
ncore =  81  ncas =  7
```

4.4.4 1-Step SOC-DMRG

Now we do 1-step CASCI-DMRG within the (9e, 7o) active space with the SOC effect considered using SOMF approximation.

[5]:
```
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itgsoc.get_rhf_somf_integrals(
    mf, ncore, ncas, pg_symm=False, dmao=dmao, amfi=True, x2c1e=False, x2c2e=False
)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SGFCPX, stack_mem=4 <<␣
↪30, n_threads=4)
idx = driver.orbital_reordering(np.abs(h1e), np.abs(g2e))
print('reordering = ', idx)
h1e = h1e[idx][:, idx]
g2e = g2e[idx][:, idx][:, :, idx][:, :, :, idx]

driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=0, orb_sym=orb_sym)
h1e[np.abs(h1e) < 1e-10] = 0
g2e[np.abs(g2e) < 1e-10] = 0

mpo = driver.get_qc_mpo(h1e, g2e, ecore=ecore)
ket = driver.get_random_mps(tag="KET", bond_dim=250, nroots=16)
bond_dims = [500] * 5 + [1000] * 5 + [1500] * 5
noises = [1e-4] * 5 + [1e-5] * 5 + [1e-6] * 5 + [0]
thrds = [1e-5] * 5 + [1e-5] * 5 + [1e-7] * 5 + [1E-8]
energies = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1, dav_max_iter=400, cutoff=1E-24)
```

(continues on next page)

(continued from previous page)

```
au2cm = 219474.63111558527
au2ev = 27.21139
e0 = energies[0]

ener_cm = []

for ix, ex in enumerate(energies):
    ener_cm.append((ex - e0) * au2cm)
    print("%5d %20.10f Ha %15.6f eV %10.4f cm-1" % (ix, ex, (ex - e0) * au2ev, (ex -␣
↪e0) * au2cm))
```

```
reordering =  [ 5 13 11  1  9  3  7  6  2  8  0 10 12  4]
integral symmetrize error =  0.0
integral cutoff error =  0.0
mpo terms =       4139

Build MPO | Nsites =    14 | Nterms =       4139 | Algorithm = FastBIP | Cutoff = 1.
↪00e-20
 Site =     0 /    14 .. Mmpo =     9 DW = 0.00e+00 NNZ =        9 SPT = 0.0000 Tmvc␣
↪= 0.001 T = 0.007
 Site =     1 /    14 .. Mmpo =    26 DW = 0.00e+00 NNZ =       51 SPT = 0.7821 Tmvc␣
↪= 0.001 T = 0.006
 Site =     2 /    14 .. Mmpo =    37 DW = 0.00e+00 NNZ =      103 SPT = 0.8929 Tmvc␣
↪= 0.001 T = 0.008
 Site =     3 /    14 .. Mmpo =    44 DW = 0.00e+00 NNZ =      243 SPT = 0.8507 Tmvc␣
↪= 0.001 T = 0.007
 Site =     4 /    14 .. Mmpo =    47 DW = 0.00e+00 NNZ =      173 SPT = 0.9163 Tmvc␣
↪= 0.001 T = 0.007
 Site =     5 /    14 .. Mmpo =    54 DW = 0.00e+00 NNZ =      151 SPT = 0.9405 Tmvc␣
↪= 0.001 T = 0.007
 Site =     6 /    14 .. Mmpo =    65 DW = 0.00e+00 NNZ =      103 SPT = 0.9707 Tmvc␣
↪= 0.001 T = 0.016
 Site =     7 /    14 .. Mmpo =    54 DW = 0.00e+00 NNZ =     2522 SPT = 0.2815 Tmvc␣
↪= 0.001 T = 0.023
 Site =     8 /    14 .. Mmpo =    47 DW = 0.00e+00 NNZ =      154 SPT = 0.9393 Tmvc␣
↪= 0.000 T = 0.005
 Site =     9 /    14 .. Mmpo =    44 DW = 0.00e+00 NNZ =      178 SPT = 0.9139 Tmvc␣
↪= 0.000 T = 0.004
 Site =    10 /    14 .. Mmpo =    37 DW = 0.00e+00 NNZ =      250 SPT = 0.8464 Tmvc␣
↪= 0.000 T = 0.003
 Site =    11 /    14 .. Mmpo =    26 DW = 0.00e+00 NNZ =      112 SPT = 0.8836 Tmvc␣
↪= 0.000 T = 0.006
 Site =    12 /    14 .. Mmpo =     9 DW = 0.00e+00 NNZ =       64 SPT = 0.7265 Tmvc␣
↪= 0.000 T = 0.003
 Site =    13 /    14 .. Mmpo =     1 DW = 0.00e+00 NNZ =        9 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.002
```

(continues on next page)

(continued from previous page)

```
Ttotal =      0.104 Tmvc-total = 0.010 MPO bond dimension =    65 MaxDW = 0.00e+00
NNZ =         4122 SIZE =        21898 SPT = 0.8118

Rank =     0 Ttotal =      0.161 MPO method = FastBipartite bond dimension =      65␣
↪NNZ =         4122 SIZE =        21898 SPT = 0.8118

Sweep =    0 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-05
Time elapsed =      9.125 | E[ 16] =  -14918.9542835604 -14918.9542422643 -14918.
↪9541715016 -14918.9541203618 -14918.9539513808 -14918.9538646833 -14918.9536668910␣
↪-14918.9535939114 -14918.9535317178 -14918.9534750957 -14918.9534488166 -14918.
↪9533890762 -14918.9533552697 -14918.9379111868 -14918.9378379441 -14918.9376703868␣
↪| DW = 1.19e-21

Sweep =    1 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-05
Time elapsed =     13.200 | E[ 16] =  -14918.9542835604 -14918.9542422643 -14918.
↪9541715016 -14918.9541203618 -14918.9539513808 -14918.9538646833 -14918.9536668910␣
↪-14918.9535939114 -14918.9535317178 -14918.9534750957 -14918.9534488166 -14918.
↪9533890762 -14918.9533552697 -14918.9379111868 -14918.9378379441 -14918.9376703868␣
↪| DE = 1.99e-13 | DW = 1.85e-21

Sweep =    2 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-05
Time elapsed =     17.348 | E[ 16] =  -14918.9542835604 -14918.9542422643 -14918.
↪9541715016 -14918.9541203618 -14918.9539513808 -14918.9538646833 -14918.9536668910␣
↪-14918.9535939114 -14918.9535317178 -14918.9534750957 -14918.9534488166 -14918.
↪9533890762 -14918.9533552697 -14918.9379111868 -14918.9378379441 -14918.9376703868␣
↪| DE = 7.82e-14 | DW = 1.23e-21

Sweep =    3 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-05
Time elapsed =     22.978 | E[ 16] =  -14918.9542835604 -14918.9542422643 -14918.
↪9541715016 -14918.9541203618 -14918.9539513808 -14918.9538646833 -14918.9536668910␣
↪-14918.9535939114 -14918.9535317178 -14918.9534750957 -14918.9534488166 -14918.
↪9533890762 -14918.9533552697 -14918.9379111868 -14918.9378379441 -14918.9376703868␣
↪| DE = 7.11e-15 | DW = 1.63e-21

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-05
Time elapsed =     27.088 | E[ 16] =  -14918.9542835604 -14918.9542422643 -14918.
↪9541715016 -14918.9541203618 -14918.9539513808 -14918.9538646833 -14918.9536668910␣
↪-14918.9535939114 -14918.9535317178 -14918.9534750957 -14918.9534488166 -14918.
↪9533890762 -14918.9533552697 -14918.9379111868 -14918.9378379441 -14918.9376703868␣
↪| DE = -3.55e-14 | DW = 2.26e-21
```

(continues on next page)

(continued from previous page)

```
Sweep =    5 | Direction = backward | Bond dimension = 1000 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-05
Time elapsed =     31.333 | E[ 16] =  -14918.9542835604 -14918.9542422643 -14918.
↪9541715016 -14918.9541203618 -14918.9539513808 -14918.9538646833 -14918.9536668910␣
↪-14918.9535939114 -14918.9535317178 -14918.9534750957 -14918.9534488166 -14918.
↪9533890762 -14918.9533552697 -14918.9379111868 -14918.9378379441 -14918.9376703868␣
↪| DE = 0.00e+00 | DW = 8.46e-24

Sweep =    6 | Direction =  forward | Bond dimension = 1000 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-05
Time elapsed =     37.183 | E[ 16] =  -14918.9542835604 -14918.9542422643 -14918.
↪9541715016 -14918.9541203618 -14918.9539513808 -14918.9538646833 -14918.9536668910␣
↪-14918.9535939114 -14918.9535317178 -14918.9534750957 -14918.9534488166 -14918.
↪9533890762 -14918.9533552697 -14918.9379111868 -14918.9378379441 -14918.9376703868␣
↪| DE = 2.13e-14 | DW = 1.21e-23

Sweep =    7 | Direction = backward | Bond dimension = 1000 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-05
Time elapsed =     41.357 | E[ 16] =  -14918.9542835604 -14918.9542422643 -14918.
↪9541715016 -14918.9541203618 -14918.9539513808 -14918.9538646833 -14918.9536668910␣
↪-14918.9535939114 -14918.9535317178 -14918.9534750957 -14918.9534488166 -14918.
↪9533890762 -14918.9533552697 -14918.9379111868 -14918.9378379441 -14918.9376703868␣
↪| DE = -2.84e-14 | DW = 7.85e-24

Sweep =    8 | Direction =  forward | Bond dimension = 1000 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-05
Time elapsed =     45.562 | E[ 16] =  -14918.9542835604 -14918.9542422643 -14918.
↪9541715016 -14918.9541203618 -14918.9539513808 -14918.9538646833 -14918.9536668910␣
↪-14918.9535939114 -14918.9535317178 -14918.9534750957 -14918.9534488166 -14918.
↪9533890762 -14918.9533552697 -14918.9379111868 -14918.9378379441 -14918.9376703868␣
↪| DE = 7.11e-15 | DW = 1.59e-23

Sweep =    9 | Direction = backward | Bond dimension = 1000 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-05
Time elapsed =     51.314 | E[ 16] =  -14918.9542835604 -14918.9542422643 -14918.
↪9541715016 -14918.9541203618 -14918.9539513808 -14918.9538646833 -14918.9536668910␣
↪-14918.9535939114 -14918.9535317178 -14918.9534750957 -14918.9534488166 -14918.
↪9533890762 -14918.9533552697 -14918.9379111868 -14918.9378379441 -14918.9376703868␣
↪| DE = 7.11e-15 | DW = 1.14e-23

Sweep =   10 | Direction =  forward | Bond dimension = 1500 | Noise =  1.00e-06 |␣
↪Dav threshold =  1.00e-07
Time elapsed =     60.111 | E[ 16] =  -14918.9543415921 -14918.9543413136 -14918.
↪9542607917 -14918.9542606937 -14918.9542606291 -14918.9542604100 -14918.9539727528␣
```

(continues on next page)

(continued from previous page)

```
↪-14918.9539724846 -14918.9535923209 -14918.9535922145 -14918.9535921710 -14918.
↪9535919583 -14918.9534292711 -14918.9534291702 -14918.9534289740 -14918.9534288629␣
↪| DE = -1.58e-02 | DW = 1.28e-23

Sweep =   11 | Direction = backward | Bond dimension = 1500 | Noise =  1.00e-06 |␣
↪Dav threshold =  1.00e-07
Time elapsed =     65.816 | E[ 16] =  -14918.9543415921 -14918.9543413136 -14918.
↪9542607917 -14918.9542606937 -14918.9542606291 -14918.9542604100 -14918.9539727528␣
↪-14918.9539724846 -14918.9535923209 -14918.9535922145 -14918.9535921710 -14918.
↪9535919583 -14918.9534292711 -14918.9534291702 -14918.9534289740 -14918.9534288629␣
↪| DE = -7.11e-15 | DW = 7.59e-24

Sweep =   12 | Direction =  forward | Bond dimension = 1500 | Noise =  1.00e-06 |␣
↪Dav threshold =  1.00e-07
Time elapsed =     69.943 | E[ 16] =  -14918.9543415921 -14918.9543413136 -14918.
↪9542607917 -14918.9542606937 -14918.9542606291 -14918.9542604100 -14918.9539727528␣
↪-14918.9539724846 -14918.9535923209 -14918.9535922145 -14918.9535921710 -14918.
↪9535919583 -14918.9534292711 -14918.9534291702 -14918.9534289740 -14918.9534288629␣
↪| DE = 7.11e-15 | DW = 1.22e-23

Sweep =   13 | Direction = backward | Bond dimension = 1500 | Noise =  1.00e-06 |␣
↪Dav threshold =  1.00e-07
Time elapsed =     74.265 | E[ 16] =  -14918.9543415921 -14918.9543413136 -14918.
↪9542607917 -14918.9542606937 -14918.9542606291 -14918.9542604100 -14918.9539727528␣
↪-14918.9539724846 -14918.9535923209 -14918.9535922145 -14918.9535921710 -14918.
↪9535919583 -14918.9534292711 -14918.9534291702 -14918.9534289740 -14918.9534288629␣
↪| DE = -7.11e-15 | DW = 7.32e-24

Sweep =   14 | Direction =  forward | Bond dimension = 1500 | Noise =  1.00e-06 |␣
↪Dav threshold =  1.00e-07
Time elapsed =     79.784 | E[ 16] =  -14918.9543415921 -14918.9543413136 -14918.
↪9542607917 -14918.9542606937 -14918.9542606291 -14918.9542604100 -14918.9539727528␣
↪-14918.9539724846 -14918.9535923209 -14918.9535922145 -14918.9535921710 -14918.
↪9535919583 -14918.9534292711 -14918.9534291702 -14918.9534289740 -14918.9534288629␣
↪| DE = 0.00e+00 | DW = 1.33e-23

Sweep =   15 | Direction = backward | Bond dimension = 1500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-08
Time elapsed =     83.718 | E[ 16] =  -14918.9543418220 -14918.9543417819 -14918.
↪9542610635 -14918.9542610615 -14918.9542610409 -14918.9542610257 -14918.9539731921␣
↪-14918.9539731834 -14918.9535927283 -14918.9535926953 -14918.9535926795 -14918.
↪9535926594 -14918.9534297026 -14918.9534296962 -14918.9534296756 -14918.9534296597␣
↪| DE = -7.97e-07 | DW = 7.14e-25

Sweep =   16 | Direction =  forward | Bond dimension = 1500 | Noise =  0.00e+00 |␣
```

(continues on next page)

(continued from previous page)

```
↪Dav threshold =  1.00e-09
Time elapsed =     87.875 | E[ 16] =  -14918.9543418595 -14918.9543418558 -14918.
↪9542611149 -14918.9542611131 -14918.9542611062 -14918.9542611038 -14918.9539732438␣
↪-14918.9539732424 -14918.9535927726 -14918.9535927707 -14918.9535927596 -14918.
↪9535927572 -14918.9534297875 -14918.9534297860 -14918.9534297683 -14918.9534297667␣
↪| DE = -1.07e-07 | DW = 1.19e-24

Sweep =   17 | Direction = backward | Bond dimension = 1500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =     92.162 | E[ 16] =  -14918.9543418595 -14918.9543418558 -14918.
↪9542611149 -14918.9542611131 -14918.9542611062 -14918.9542611038 -14918.9539732438␣
↪-14918.9539732424 -14918.9535927726 -14918.9535927707 -14918.9535927596 -14918.
↪9535927572 -14918.9534297875 -14918.9534297860 -14918.9534297683 -14918.9534297667␣
↪| DE = 4.97e-14 | DW = 5.28e-26

    0    -14918.9543418595 Ha        0.000000 eV     0.0000 cm-1
    1    -14918.9543418558 Ha        0.000000 eV     0.0008 cm-1
    2    -14918.9542611149 Ha        0.002197 eV    17.7214 cm-1
    3    -14918.9542611131 Ha        0.002197 eV    17.7218 cm-1
    4    -14918.9542611062 Ha        0.002197 eV    17.7233 cm-1
    5    -14918.9542611038 Ha        0.002197 eV    17.7238 cm-1
    6    -14918.9539732438 Ha        0.010031 eV    80.9018 cm-1
    7    -14918.9539732424 Ha        0.010031 eV    80.9021 cm-1
    8    -14918.9535927726 Ha        0.020384 eV   164.4056 cm-1
    9    -14918.9535927707 Ha        0.020384 eV   164.4060 cm-1
   10    -14918.9535927596 Ha        0.020384 eV   164.4084 cm-1
   11    -14918.9535927572 Ha        0.020384 eV   164.4089 cm-1
   12    -14918.9534297875 Ha        0.024819 eV   200.1767 cm-1
   13    -14918.9534297860 Ha        0.024819 eV   200.1770 cm-1
   14    -14918.9534297683 Ha        0.024819 eV   200.1809 cm-1
   15    -14918.9534297667 Ha        0.024819 eV   200.1812 cm-1
```

4.4.5 Plot the results

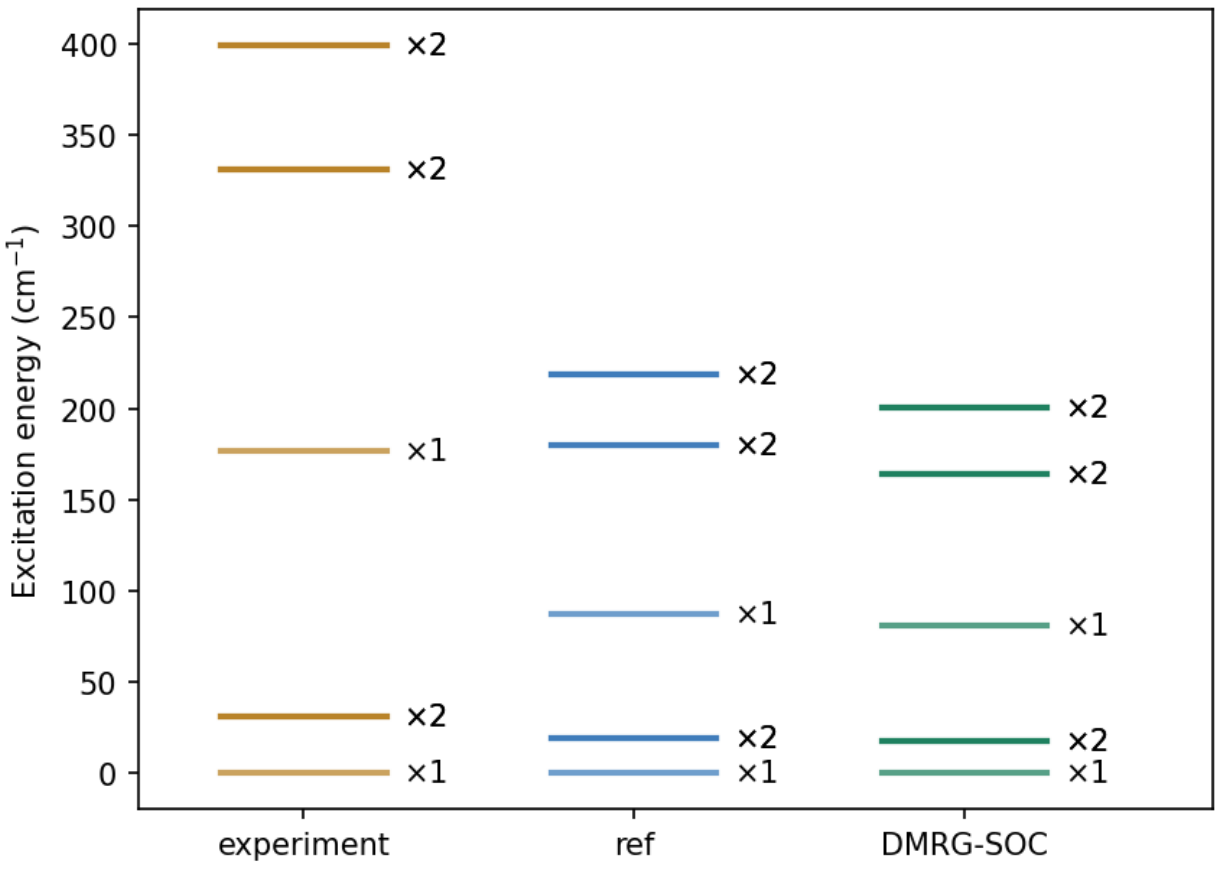
In the following we plot the results. The experiment and ref data are from Table S10 of the second reference listed above. The difference between the experiment data and the CASSCF data is mainly from the missing dynamic correlation. The difference between ref and DMRG-SOC is mainly from the basis set and different method used.

[6]:
```
import matplotlib.pyplot as plt
import numpy as np

ener_exp = [0, 31, 31, 177, 331, 331, 399, 399]
ener_ref = [0, 19.2, 19.2, 87.2, 179.9, 179.9, 218.9, 218.9]
ener = [round((i + j ) / 2, 1) for i, j in zip(ener_cm[::2], ener_cm[1::2])]
```

(continues on next page)

(continued from previous page)

```
for i in range(len(ener_exp)):
    plt.text(2.1, ener_exp[i], "$\\times%d$" % ener_exp.count(ener_exp[i]),␣
↪verticalalignment='center', fontsize=10)
    plt.plot([1, 2], [ener_exp[i]] * 2, '-', linewidth=2, color="#B1740F", alpha=0.
↪67)
    plt.text(4.1, ener_ref[i], "$\\times%d$" % ener_ref.count(ener_ref[i]),␣
↪verticalalignment='center', fontsize=10)
    plt.plot([3, 4], [ener_ref[i]] * 2, '-', linewidth=2, color="#296EB4", alpha=0.
↪67)
    plt.text(6.1, ener[i], "$\\times%d$" % ener.count(ener[i]), verticalalignment=
↪'center', fontsize=10)
    plt.plot([5, 6], [ener[i]] * 2, '-', linewidth=2, color="#04724D", alpha=0.67)
plt.xlim((0.5, 7))
plt.xticks([1.5, 3.5, 5.5], ["experiment", "ref", "DMRG-SOC"])
plt.ylabel("Excitation energy ($\\mathrm{cm^{-1}}$)")
plt.gcf().set_dpi(150)
plt.show()
```



4.5 Green's Function and TD-DMRG

[1]:
```
!pip install block2==0.5.2rc13 -qq --progress-bar off --extra-index-url=https://
↪block-hczhai.github.io/block2-preview/pypi/
!pip install pyscf==2.3.0 -qq --progress-bar off
```

In the following example, we calculate the electron removal (IP) part of one-particle Green's function for Hydrogen chain (H_6) in the minimal basis:

$$G_{ij}^{-}(\omega) = \langle \Psi_0 | a_j^\dagger \frac{1}{\omega + \hat{H}_0 - E_0 + i\eta} a_i | \Psi_0 \rangle$$

where $|\Psi_0\rangle$ is the ground state, $i = j = 2$ (counting from zero), $\eta = 0.005$.

The Green's function can be computed using DMRG in either the frequency domain (dynamical DMRG) or the time domain (time-dependent DMRG).

4.5.1 Dynamical DMRG

We first solve the response equation to find the Green's function in the frequency domain.

Note that the return value of `driver.greens_function` method is the state:

$$|X_i\rangle := \frac{1}{\omega + \hat{H}_0 - E_0 + i\eta} a_i |\Psi_0\rangle$$

Therefore, to get the value for Green's function with $i \neq j$, one can simply use `driver.expectation` to compute the dot product of $|X_i\rangle$ with some other $a_j|\Psi_0\rangle$.

In the `SU2` mode:

[2]:
```
import numpy as np
from pyblock2._pyscf.ao2mo import integrals as itg
from pyblock2.driver.core import DMRGDriver, SymmetryTypes
from pyscf import gto, scf, lo

BOHR = 0.52917721092
R = 1.8 * BOHR
N = 6

mol = gto.M(atom=[['H', (i * R, 0, 0)] for i in range(N)], basis="sto6g", symmetry=
↪"c1", verbose=0)
mf = scf.RHF(mol).run(conv_tol=1E-14)

mf.mo_coeff = lo.orth.lowdin(mol.intor('cint1e_ovlp_sph'))
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf, ncore=0,␣
↪ncas=None, g2e_symm=8)
```

(continues on next page)

(continued from previous page)

```
driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SU2, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

bond_dims = [150] * 4 + [200] * 4
noises = [1e-4] * 4 + [1e-5] * 4 + [0]
thrds = [1e-10] * 8

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, integral_cutoff=1E-8,␣
↪iprint=1)
ket = driver.get_random_mps(tag="KET", bond_dim=150, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('Ground state energy = %20.15f' % energy)

isite = 2
mpo.const_e -= energy
eta = 0.005

dmpo = driver.get_site_mpo(op='D', site_index=isite, iprint=0)
dket = driver.get_random_mps(tag="DKET", bond_dim=200, center=ket.center, left_
↪vacuum=dmpo.left_vacuum)
driver.multiply(dket, dmpo, ket, n_sweeps=10, bond_dims=[200], thrds=[1E-10] * 10,␣
↪iprint=1)

freqs = np.arange(-0.8, -0.2, 0.01)
gfmat = np.zeros((len(freqs), ), dtype=complex)
for iw, freq in enumerate(freqs):
    bra = driver.copy_mps(dket, tag="BRA") # initial guess
    gfmat[iw] = driver.greens_function(bra, mpo, dmpo, ket, freq, eta, n_sweeps=6,
        bra_bond_dims=[200], ket_bond_dims=[200], thrds=[1E-6] * 10, iprint=0)
    print("FREQ = %8.2f GF[%d,%d] = %12.6f + %12.6f i" % (freq, isite, isite,␣
↪gfmat[iw].real, gfmat[iw].imag))

ldos = -1 / np.pi * gfmat.imag

import matplotlib.pyplot as plt
plt.grid(which='major', axis='both', alpha=0.5)
plt.plot(freqs, ldos, linestyle='-', marker='o', markersize=4, mfc='white', mec="
↪#7FB685", color="#7FB685")
plt.xlabel("Frequency $\\omega$")
plt.ylabel("LDOS")
plt.show()
```

```
integral symmetrize error =  0.0
integral cutoff error =  0.0
```

(continues on next page)

(continued from previous page)

```
mpo terms =        863

Build MPO | Nsites =     6 | Nterms =        863 | Algorithm = FastBIP | Cutoff = 1.
↪00e-20
 Site =     0 /     6 .. Mmpo =    13 DW = 0.00e+00 NNZ =       13 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.008
 Site =     1 /     6 .. Mmpo =    34 DW = 0.00e+00 NNZ =      100 SPT = 0.7738 Tmvc␣
↪= 0.000 T = 0.004
 Site =     2 /     6 .. Mmpo =    56 DW = 0.00e+00 NNZ =      185 SPT = 0.9028 Tmvc␣
↪= 0.000 T = 0.004
 Site =     3 /     6 .. Mmpo =    34 DW = 0.00e+00 NNZ =      419 SPT = 0.7799 Tmvc␣
↪= 0.001 T = 0.004
 Site =     4 /     6 .. Mmpo =    14 DW = 0.00e+00 NNZ =      105 SPT = 0.7794 Tmvc␣
↪= 0.000 T = 0.003
 Site =     5 /     6 .. Mmpo =     1 DW = 0.00e+00 NNZ =       14 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.002
Ttotal =      0.027 Tmvc-total = 0.002 MPO bond dimension =    56 MaxDW = 0.00e+00
NNZ =          836 SIZE =         4753 SPT = 0.8241

Rank =     0 Ttotal =      0.057 MPO method = FastBipartite bond dimension =      56␣
↪NNZ =          836 SIZE =         4753 SPT = 0.8241

Sweep =    0 | Direction =  forward | Bond dimension =  150 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.061 | E =      -3.2667431000 | DW = 1.41e-20

Sweep =    1 | Direction = backward | Bond dimension =  150 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.105 | E =      -3.2667431000 | DE = 7.11e-15 | DW = 9.98e-21

Sweep =    2 | Direction =  forward | Bond dimension =  150 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.148 | E =      -3.2667431000 | DE = 0.00e+00 | DW = 1.25e-20

Sweep =    3 | Direction = backward | Bond dimension =  150 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.197 | E =      -3.2667431000 | DE = -1.78e-15 | DW = 1.88e-20

Sweep =    4 | Direction =  forward | Bond dimension =  200 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.238 | E =      -3.2667431000 | DE = 1.78e-15 | DW = 1.13e-20

Sweep =    5 | Direction = backward | Bond dimension =  200 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.280 | E =      -3.2667431000 | DE = 1.78e-15 | DW = 1.19e-20
```

(continues on next page)

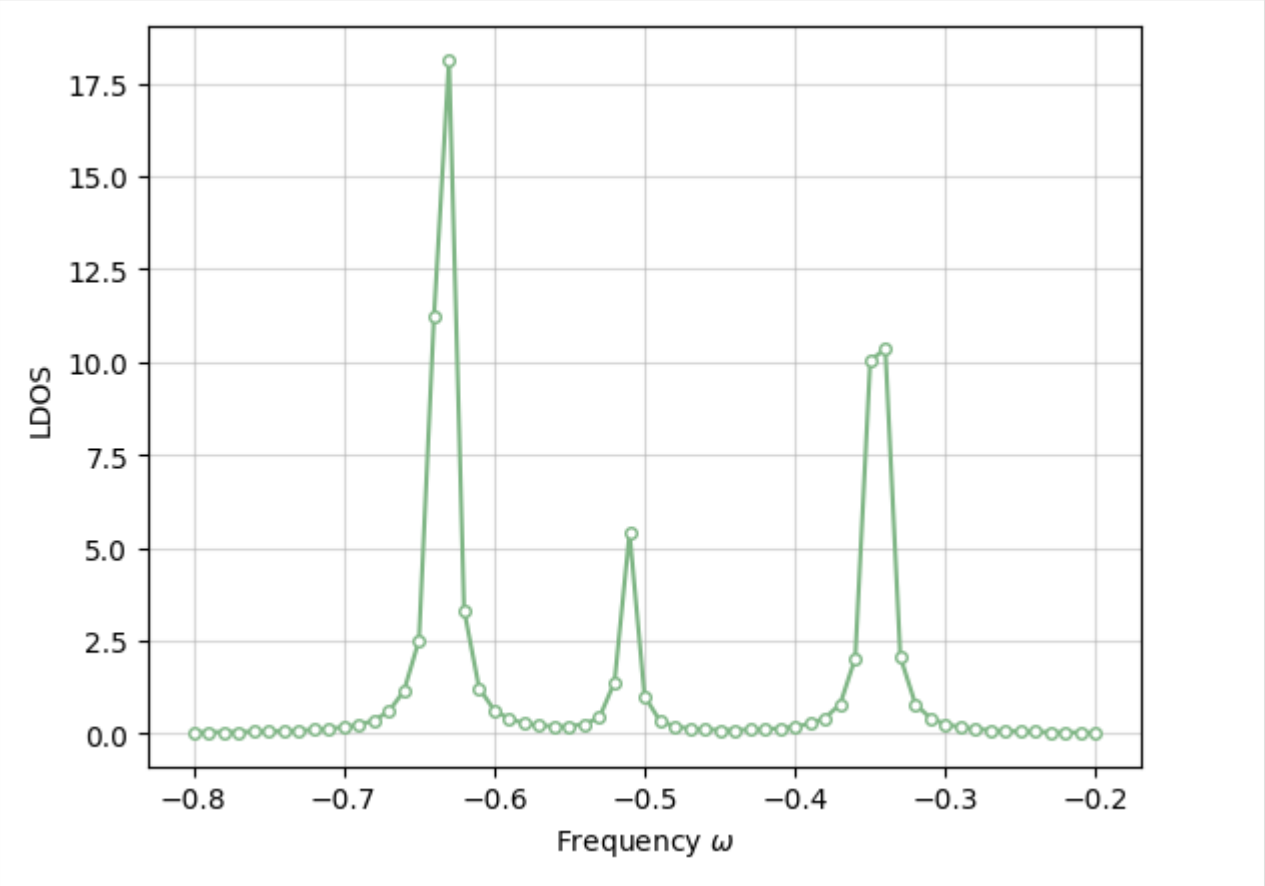
(continued from previous page)

```
Sweep =    6 | Direction =  forward | Bond dimension =  200 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.319 | E =      -3.2667431000 | DE = -1.78e-15 | DW = 9.42e-21

Sweep =    7 | Direction = backward | Bond dimension =  200 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.365 | E =      -3.2667431000 | DE = -1.78e-15 | DW = 2.01e-20

Sweep =    8 | Direction =  forward | Bond dimension =  200 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      0.400 | E =      -3.2667431000 | DE = 3.55e-15 | DW = 1.53e-20

Ground state energy =   -3.266743099950662

Sweep =    0 | Direction = backward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.018 | F =    0.9962046481 | DW = 5.23e-25

Sweep =    1 | Direction =  forward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.038 | F =    0.9970609307 | DF = 8.56e-04 | DW = 8.99e-25

Sweep =    2 | Direction = backward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.059 | F =    0.9970609307 | DF = 4.44e-16 | DW = 6.59e-25

Sweep =    3 | Direction =  forward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.079 | F =    0.9970609307 | DF = -3.33e-16 | DW = 3.32e-25

Sweep =    4 | Direction = backward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.101 | F =    0.9970609307 | DF = -3.33e-16 | DW = 3.88e-25

FREQ =    -0.80 GF[2,2] =    -2.858060 +    -0.132949 i
FREQ =    -0.79 GF[2,2] =    -3.127639 +    -0.137862 i
FREQ =    -0.78 GF[2,2] =    -3.410730 +    -0.146788 i
FREQ =    -0.77 GF[2,2] =    -3.716042 +    -0.159910 i
FREQ =    -0.76 GF[2,2] =    -4.052436 +    -0.177910 i
FREQ =    -0.75 GF[2,2] =    -4.429247 +    -0.202036 i
FREQ =    -0.74 GF[2,2] =    -4.862689 +    -0.234242 i
FREQ =    -0.73 GF[2,2] =    -5.370446 +    -0.277828 i
FREQ =    -0.72 GF[2,2] =    -5.979802 +    -0.338034 i
FREQ =    -0.71 GF[2,2] =    -6.730787 +    -0.424057 i
FREQ =    -0.70 GF[2,2] =    -7.689106 +    -0.552665 i
FREQ =    -0.69 GF[2,2] =    -8.964426 +    -0.757635 i
FREQ =    -0.68 GF[2,2] =   -10.763740 +    -1.118686 i
FREQ =    -0.67 GF[2,2] =   -13.510429 +    -1.904433 i
FREQ =    -0.66 GF[2,2] =   -17.418881 +    -3.612921 i
```

(continues on next page)

(continued from previous page)

```
FREQ =    -0.65 GF[2,2] =   -26.296063 +    -7.892317 i
FREQ =    -0.64 GF[2,2] =   -44.979411 +   -35.247680 i
FREQ =    -0.63 GF[2,2] =    41.659891 +   -56.954077 i
FREQ =    -0.62 GF[2,2] =    27.132451 +   -10.471813 i
FREQ =    -0.61 GF[2,2] =    16.312481 +    -3.861684 i
FREQ =    -0.60 GF[2,2] =    11.092739 +    -2.003940 i
FREQ =    -0.59 GF[2,2] =     8.036382 +    -1.254680 i
FREQ =    -0.58 GF[2,2] =     5.972372 +    -0.893065 i
FREQ =    -0.57 GF[2,2] =     4.418047 +    -0.708471 i
FREQ =    -0.56 GF[2,2] =     3.108176 +    -0.629033 i
FREQ =    -0.55 GF[2,2] =     1.874834 +    -0.643365 i
FREQ =    -0.54 GF[2,2] =     0.501622 +    -0.805649 i
FREQ =    -0.53 GF[2,2] =    -1.428740 +    -1.392787 i
FREQ =    -0.52 GF[2,2] =    -5.038213 +    -4.367329 i
FREQ =    -0.51 GF[2,2] =     5.385607 +   -16.985511 i
FREQ =    -0.50 GF[2,2] =     8.132057 +    -3.176944 i
FREQ =    -0.49 GF[2,2] =     5.112053 +    -1.122502 i
FREQ =    -0.48 GF[2,2] =     3.547000 +    -0.626599 i
FREQ =    -0.47 GF[2,2] =     2.530196 +    -0.443406 i
FREQ =    -0.46 GF[2,2] =     1.745955 +    -0.363108 i
FREQ =    -0.45 GF[2,2] =     1.066544 +    -0.329559 i
FREQ =    -0.44 GF[2,2] =     0.421145 +    -0.324585 i
FREQ =    -0.43 GF[2,2] =    -0.234970 +    -0.342450 i
FREQ =    -0.42 GF[2,2] =    -0.955736 +    -0.386035 i
FREQ =    -0.41 GF[2,2] =    -1.792136 +    -0.464805 i
FREQ =    -0.40 GF[2,2] =    -2.833594 +    -0.601140 i
FREQ =    -0.39 GF[2,2] =    -4.230434 +    -0.846750 i
FREQ =    -0.38 GF[2,2] =    -6.289553 +    -1.335939 i
FREQ =    -0.37 GF[2,2] =    -9.737590 +    -2.501588 i
FREQ =    -0.36 GF[2,2] =   -16.688375 +    -6.387690 i
FREQ =    -0.35 GF[2,2] =   -29.656382 +   -31.528685 i
FREQ =    -0.34 GF[2,2] =    34.225016 +   -32.546412 i
FREQ =    -0.33 GF[2,2] =    21.430321 +    -6.498027 i
FREQ =    -0.32 GF[2,2] =    14.419157 +    -2.511719 i
FREQ =    -0.31 GF[2,2] =    10.986939 +    -1.318801 i
FREQ =    -0.30 GF[2,2] =     8.980094 +    -0.814240 i
FREQ =    -0.29 GF[2,2] =     7.662527 +    -0.555185 i
FREQ =    -0.28 GF[2,2] =     6.728338 +    -0.404730 i
FREQ =    -0.27 GF[2,2] =     6.027647 +    -0.309463 i
FREQ =    -0.26 GF[2,2] =     5.481284 +    -0.245307 i
FREQ =    -0.25 GF[2,2] =     5.041000 +    -0.199912 i
FREQ =    -0.24 GF[2,2] =     4.677884 +    -0.166589 i
FREQ =    -0.23 GF[2,2] =     4.372155 +    -0.141336 i
FREQ =    -0.22 GF[2,2] =     4.110743 +    -0.121720 i
FREQ =    -0.21 GF[2,2] =     3.883890 +    -0.106156 i
FREQ =    -0.20 GF[2,2] =     3.685138 +    -0.093570 i
```



In the SZ mode:

[3]:
```
import numpy as np
from pyblock2._pyscf.ao2mo import integrals as itg
from pyblock2.driver.core import DMRGDriver, SymmetryTypes
from pyscf import gto, scf, lo

BOHR = 0.52917721092
R = 1.8 * BOHR
N = 6

mol = gto.M(atom=[['H', (i * R, 0, 0)] for i in range(N)], basis="sto6g", symmetry=
↪"c1", verbose=0)
mf = scf.RHF(mol).run(conv_tol=1E-14)

mf.mo_coeff = lo.orth.lowdin(mol.intor('cint1e_ovlp_sph'))
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf, ncore=0,␣
↪ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SZ, n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)
```

(continues on next page)

(continued from previous page)

```
bond_dims = [150] * 4 + [200] * 4
noises = [1e-4] * 4 + [1e-5] * 4 + [0]
thrds = [1e-10] * 8

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, integral_cutoff=1E-8,␣
↪iprint=1)
ket = driver.get_random_mps(tag="KET", bond_dim=150, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('Ground state energy = %20.15f' % energy)

isite = 2
mpo.const_e -= energy
eta = 0.005

dmpo = driver.get_site_mpo(op='d', site_index=isite, iprint=0) # only alpha spin
dket = driver.get_random_mps(tag="DKET", bond_dim=200, center=ket.center,␣
↪target=dmpo.op.q_label + ket.info.target)
driver.multiply(dket, dmpo, ket, n_sweeps=10, bond_dims=[200], thrds=[1E-10] * 10,␣
↪iprint=1)

freqs = np.arange(-0.8, -0.2, 0.01)
gfmat = np.zeros((len(freqs), ), dtype=complex)
for iw, freq in enumerate(freqs):
    bra = driver.copy_mps(dket, tag="BRA") # initial guess
    gfmat[iw] = driver.greens_function(bra, mpo, dmpo, ket, freq, eta, n_sweeps=6,
        bra_bond_dims=[200], ket_bond_dims=[200], thrds=[1E-6] * 10, iprint=0)
    print("FREQ = %8.2f GF[%d,%d] = %12.6f + %12.6f i" % (freq, isite, isite,␣
↪gfmat[iw].real, gfmat[iw].imag))

ldos = -2 / np.pi * gfmat.imag # account for both spin

import matplotlib.pyplot as plt
plt.grid(which='major', axis='both', alpha=0.5)
plt.plot(freqs, ldos, linestyle='-', marker='o', markersize=4, mfc='white', mec="
↪#7FB685", color="#7FB685")
plt.xlabel("Frequency $\\omega$")
plt.ylabel("LDOS")
plt.show()
```

```
integral symmetrize error =  0.0
integral cutoff error =  0.0
mpo terms =       2286

Build MPO | Nsites =     6 | Nterms =       2286 | Algorithm = FastBIP | Cutoff = 1.
```

(continues on next page)

(continued from previous page)

```
↪00e-20
 Site =     0 /     6 .. Mmpo =    26 DW = 0.00e+00 NNZ =       26 SPT = 0.0000 Tmvc␣
↪= 0.001 T = 0.008
 Site =     1 /     6 .. Mmpo =    66 DW = 0.00e+00 NNZ =      243 SPT = 0.8584 Tmvc␣
↪= 0.001 T = 0.011
 Site =     2 /     6 .. Mmpo =   110 DW = 0.00e+00 NNZ =      459 SPT = 0.9368 Tmvc␣
↪= 0.002 T = 0.009
 Site =     3 /     6 .. Mmpo =    66 DW = 0.00e+00 NNZ =     1147 SPT = 0.8420 Tmvc␣
↪= 0.001 T = 0.016
 Site =     4 /     6 .. Mmpo =    26 DW = 0.00e+00 NNZ =      243 SPT = 0.8584 Tmvc␣
↪= 0.000 T = 0.006
 Site =     5 /     6 .. Mmpo =     1 DW = 0.00e+00 NNZ =       26 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.008
Ttotal =      0.057 Tmvc-total = 0.005 MPO bond dimension =   110 MaxDW = 0.00e+00
NNZ =         2144 SIZE =        18004 SPT = 0.8809

Rank =     0 Ttotal =      0.097 MPO method = FastBipartite bond dimension =     110␣
↪NNZ =         2144 SIZE =        18004 SPT = 0.8809

Sweep =    0 | Direction =  forward | Bond dimension =  150 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.168 | E =      -3.2667431000 | DW = 1.33e-20

Sweep =    1 | Direction = backward | Bond dimension =  150 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.305 | E =      -3.2667431000 | DE = -7.11e-15 | DW = 1.29e-20

Sweep =    2 | Direction =  forward | Bond dimension =  150 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.432 | E =      -3.2667431000 | DE = 0.00e+00 | DW = 2.12e-20

Sweep =    3 | Direction = backward | Bond dimension =  150 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.557 | E =      -3.2667431000 | DE = 3.55e-15 | DW = 1.89e-20

Sweep =    4 | Direction =  forward | Bond dimension =  200 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.678 | E =      -3.2667431000 | DE = -1.78e-15 | DW = 1.98e-20

Sweep =    5 | Direction = backward | Bond dimension =  200 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.811 | E =      -3.2667431000 | DE = -5.33e-15 | DW = 3.57e-20

Sweep =    6 | Direction =  forward | Bond dimension =  200 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
```

(continues on next page)

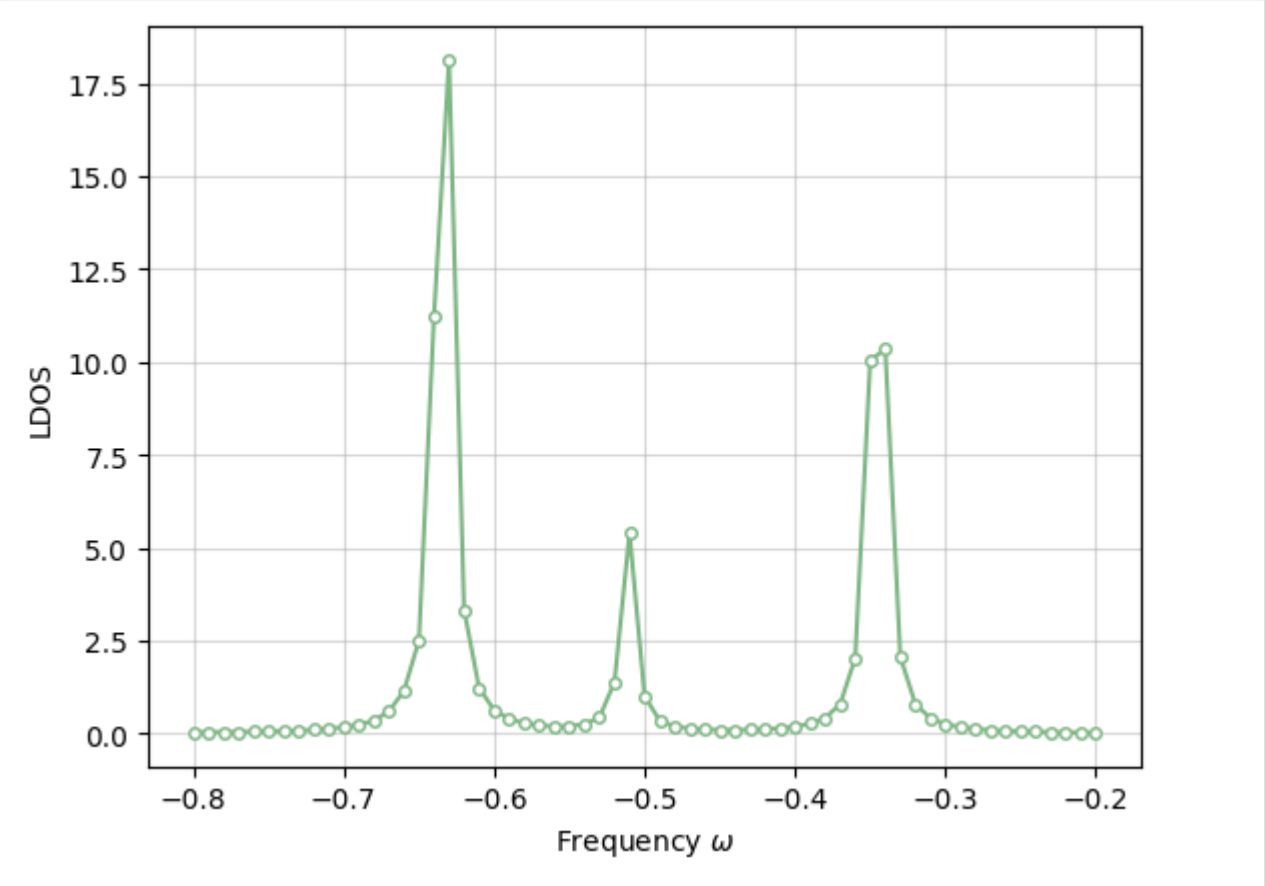
(continued from previous page)

```
Time elapsed =      0.935 | E =      -3.2667431000 | DE = 0.00e+00 | DW = 1.87e-20

Sweep =    7 | Direction = backward | Bond dimension =  200 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.070 | E =      -3.2667431000 | DE = 7.11e-15 | DW = 1.72e-20

Sweep =    8 | Direction =  forward | Bond dimension =  200 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      1.172 | E =      -3.2667431000 | DE = -5.33e-15 | DW = 4.35e-20

Ground state energy =   -3.266743099973319

Sweep =    0 | Direction = backward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.019 | F =    0.6660221816 | DW = 9.82e-26

Sweep =    1 | Direction =  forward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.042 | F =    0.7050281723 | DF = 3.90e-02 | DW = 1.70e-24

Sweep =    2 | Direction = backward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.063 | F =    0.7050281723 | DF = 6.66e-16 | DW = 2.63e-25

Sweep =    3 | Direction =  forward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.084 | F =    0.7050281723 | DF = 3.33e-16 | DW = 1.94e-24

Sweep =    4 | Direction = backward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.105 | F =    0.7050281723 | DF = 1.11e-16 | DW = 8.09e-25

FREQ =    -0.80 GF[2,2] =    -1.429314 +    -0.066472 i
FREQ =    -0.79 GF[2,2] =    -1.563695 +    -0.068960 i
FREQ =    -0.78 GF[2,2] =    -1.705301 +    -0.073389 i
FREQ =    -0.77 GF[2,2] =    -1.857857 +    -0.079957 i
FREQ =    -0.76 GF[2,2] =    -2.026014 +    -0.088950 i
FREQ =    -0.75 GF[2,2] =    -2.215113 +    -0.101005 i
FREQ =    -0.74 GF[2,2] =    -2.431836 +    -0.117119 i
FREQ =    -0.73 GF[2,2] =    -2.685052 +    -0.138881 i
FREQ =    -0.72 GF[2,2] =    -2.989529 +    -0.168989 i
FREQ =    -0.71 GF[2,2] =    -3.364958 +    -0.212001 i
FREQ =    -0.70 GF[2,2] =    -3.843595 +    -0.276232 i
FREQ =    -0.69 GF[2,2] =    -4.481702 +    -0.378669 i
FREQ =    -0.68 GF[2,2] =    -5.379774 +    -0.559234 i
FREQ =    -0.67 GF[2,2] =    -6.754026 +    -0.952219 i
FREQ =    -0.66 GF[2,2] =    -8.711280 +    -1.806963 i
FREQ =    -0.65 GF[2,2] =   -13.147719 +    -3.946547 i
FREQ =    -0.64 GF[2,2] =   -22.488537 +   -17.623137 i
FREQ =    -0.63 GF[2,2] =    20.830470 +   -28.478123 i
```

(continues on next page)

(continued from previous page)

```
FREQ =    -0.62 GF[2,2] =    13.568650 +    -5.237022 i
FREQ =    -0.61 GF[2,2] =     8.158524 +    -1.931326 i
FREQ =    -0.60 GF[2,2] =     5.546668 +    -1.001994 i
FREQ =    -0.59 GF[2,2] =     4.015779 +    -0.627121 i
FREQ =    -0.58 GF[2,2] =     2.986254 +    -0.446332 i
FREQ =    -0.57 GF[2,2] =     2.206705 +    -0.354016 i
FREQ =    -0.56 GF[2,2] =     1.553827 +    -0.314387 i
FREQ =    -0.55 GF[2,2] =     0.934667 +    -0.321487 i
FREQ =    -0.54 GF[2,2] =     0.247498 +    -0.402597 i
FREQ =    -0.53 GF[2,2] =    -0.716559 +    -0.696260 i
FREQ =    -0.52 GF[2,2] =    -2.517667 +    -2.183247 i
FREQ =    -0.51 GF[2,2] =     2.694685 +    -8.495055 i
FREQ =    -0.50 GF[2,2] =     4.068420 +    -1.589597 i
FREQ =    -0.49 GF[2,2] =     2.556421 +    -0.561628 i
FREQ =    -0.48 GF[2,2] =     1.776438 +    -0.313770 i
FREQ =    -0.47 GF[2,2] =     1.266039 +    -0.222005 i
FREQ =    -0.46 GF[2,2] =     0.873240 +    -0.181737 i
FREQ =    -0.45 GF[2,2] =     0.533170 +    -0.164892 i
FREQ =    -0.44 GF[2,2] =     0.210275 +    -0.162307 i
FREQ =    -0.43 GF[2,2] =    -0.119248 +    -0.171350 i
FREQ =    -0.42 GF[2,2] =    -0.478803 +    -0.193119 i
FREQ =    -0.41 GF[2,2] =    -0.898305 +    -0.232404 i
FREQ =    -0.40 GF[2,2] =    -1.418566 +    -0.300549 i
FREQ =    -0.39 GF[2,2] =    -2.115432 +    -0.423477 i
FREQ =    -0.38 GF[2,2] =    -3.145615 +    -0.668038 i
FREQ =    -0.37 GF[2,2] =    -4.871226 +    -1.251034 i
FREQ =    -0.36 GF[2,2] =    -8.345991 +    -3.194131 i
FREQ =    -0.35 GF[2,2] =   -14.828832 +   -15.765406 i
FREQ =    -0.34 GF[2,2] =    17.111094 +   -16.271567 i
FREQ =    -0.33 GF[2,2] =    10.713170 +    -3.248626 i
FREQ =    -0.32 GF[2,2] =     7.207938 +    -1.255671 i
FREQ =    -0.31 GF[2,2] =     5.490817 +    -0.659108 i
FREQ =    -0.30 GF[2,2] =     4.488220 +    -0.406981 i
FREQ =    -0.29 GF[2,2] =     3.829440 +    -0.277473 i
FREQ =    -0.28 GF[2,2] =     3.362384 +    -0.202273 i
FREQ =    -0.27 GF[2,2] =     3.011931 +    -0.154637 i
FREQ =    -0.26 GF[2,2] =     2.738941 +    -0.122582 i
FREQ =    -0.25 GF[2,2] =     2.518611 +    -0.099880 i
FREQ =    -0.24 GF[2,2] =     2.336457 +    -0.083188 i
FREQ =    -0.23 GF[2,2] =     2.183385 +    -0.070557 i
FREQ =    -0.22 GF[2,2] =     2.053131 +    -0.060779 i
FREQ =    -0.21 GF[2,2] =     1.939693 +    -0.052990 i
FREQ =    -0.20 GF[2,2] =     1.840458 +    -0.046715 i
```



4.5.2 Time-Dependent DMRG

Here we use real time TD-DMRG and Fast Fourier Transform (FFT) to calculate the Green's function. This is obtained from a Fourier transform from time domain to frequency domain:

$$G_{ij}(t) = -i\langle\Psi_0|a_j^\dagger \mathrm{e}^{-i(\hat{H}_0-E_0)t}a_i|\Psi_0\rangle$$
$$G_{ij}(\omega) = \int_{-\infty}^{\infty} \mathrm{d}t \mathrm{e}^{-i\omega t}G_{ij}(t)\mathrm{e}^{-\eta t}$$

where $\mathrm{e}^{-\eta t}$ is a broading factor.

In the `SU2` mode:

[4]:
```
import numpy as np
from pyblock2._pyscf.ao2mo import integrals as itg
from pyblock2.driver.core import DMRGDriver, SymmetryTypes
from pyscf import gto, scf, lo

BOHR = 0.52917721092
R = 1.8 * BOHR
```

(continues on next page)

(continued from previous page)

```
N = 6

mol = gto.M(atom=[['H', (i * R, 0, 0)] for i in range(N)], basis="sto6g", symmetry=
↪"c1", verbose=0)
mf = scf.RHF(mol).run(conv_tol=1E-14)

mf.mo_coeff = lo.orth.lowdin(mol.intor('cint1e_ovlp_sph'))
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf, ncore=0,␣
↪ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SU2 | SymmetryTypes.CPX,
↪ n_threads=4)
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

bond_dims = [150] * 4 + [200] * 4
noises = [1e-4] * 4 + [1e-5] * 4 + [0]
thrds = [1e-10] * 8

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, integral_cutoff=1E-8,␣
↪iprint=1)
ket = driver.get_random_mps(tag="KET", bond_dim=150, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('Ground state energy = %20.15f' % energy)

isite = 2
mpo.const_e -= energy
eta = 0.005

dmpo = driver.get_site_mpo(op='D', site_index=isite, iprint=0)
dket = driver.get_random_mps(tag="DKET", bond_dim=200, center=ket.center, left_
↪vacuum=dmpo.left_vacuum)
driver.multiply(dket, dmpo, ket, n_sweeps=10, bond_dims=[200], thrds=[1E-10] * 10,␣
↪iprint=1)

impo = driver.get_identity_mpo()
dbra = driver.copy_mps(dket, tag='DBRA')

dt = 0.2
t = 500.0
nstep = int(t / dt)
rtgf = np.zeros((nstep, ), dtype=complex)
rtgf[0] = driver.expectation(dket, impo, dket)
for it in range(nstep - 1):
    if it % (nstep // 100) == 0:
```

(continues on next page)

(continued from previous page)

```
        print("it = %5d (%4.1f %%)" % (it, it * 100 / nstep))
    dbra = driver.td_dmrg(mpo, dbra, -dt * 1j, -dt * 1j, final_mps_tag='DBRA',␣
↪hermitian=True, bond_dims=[200], iprint=0)
    rtgf[it + 1] = driver.expectation(dbra, impo, dket)

def gf_fft(eta, dt, rtgf, npts):
    frq = np.fft.fftfreq(npts, dt)
    frq = np.fft.fftshift(frq) * 2.0 * np.pi
    fftinp = -1j * rtgf * np.exp(-eta * dt * np.arange(0, npts))
    return frq, np.fft.fftshift(np.fft.fft(fftinp)) * dt

frq, frq_gf = gf_fft(eta, dt, rtgf, len(rtgf))
frq_gf = frq_gf[(frq >= -0.8) & (frq < -0.2)]
frq = frq[(frq >= -0.8) & (frq < -0.2)]

ldos = -1 / np.pi * frq_gf.imag

import matplotlib.pyplot as plt
plt.grid(which='major', axis='both', alpha=0.5)
plt.plot(frq, ldos, linestyle='-', marker='o', markersize=4, mfc='white', mec="
↪#7FB685", color="#7FB685")
plt.xlabel("Frequency $\\omega$")
plt.ylabel("LDOS")
plt.show()
```

```
integral symmetrize error =  0.0
integral cutoff error =  0.0
mpo terms =        863

Build MPO | Nsites =     6 | Nterms =        863 | Algorithm = FastBIP | Cutoff = 1.
↪00e-20
 Site =     0 /     6 .. Mmpo =    13 DW = 0.00e+00 NNZ =       13 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.014
 Site =     1 /     6 .. Mmpo =    34 DW = 0.00e+00 NNZ =      100 SPT = 0.7738 Tmvc␣
↪= 0.000 T = 0.008
 Site =     2 /     6 .. Mmpo =    56 DW = 0.00e+00 NNZ =      185 SPT = 0.9028 Tmvc␣
↪= 0.000 T = 0.008
 Site =     3 /     6 .. Mmpo =    34 DW = 0.00e+00 NNZ =      419 SPT = 0.7799 Tmvc␣
↪= 0.000 T = 0.016
 Site =     4 /     6 .. Mmpo =    14 DW = 0.00e+00 NNZ =      105 SPT = 0.7794 Tmvc␣
↪= 0.000 T = 0.013
 Site =     5 /     6 .. Mmpo =     1 DW = 0.00e+00 NNZ =       14 SPT = 0.0000 Tmvc␣
↪= 0.001 T = 0.027
Ttotal =      0.087 Tmvc-total = 0.002 MPO bond dimension =    56 MaxDW = 0.00e+00
NNZ =          836 SIZE =         4753 SPT = 0.8241
```

(continues on next page)

(continued from previous page)

```
Rank =     0 Ttotal =      0.190 MPO method = FastBipartite bond dimension =      56␣
↪NNZ =          836 SIZE =         4753 SPT = 0.8241

Sweep =    0 | Direction =  forward | Bond dimension =  150 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.138 | E =      -3.2667431000 | DW = 5.73e-21

Sweep =    1 | Direction = backward | Bond dimension =  150 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.227 | E =      -3.2667431000 | DE = 4.09e-14 | DW = 1.64e-20

Sweep =    2 | Direction =  forward | Bond dimension =  150 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.327 | E =      -3.2667431000 | DE = 0.00e+00 | DW = 9.52e-21

Sweep =    3 | Direction = backward | Bond dimension =  150 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.428 | E =      -3.2667431000 | DE = -1.78e-15 | DW = 1.55e-20

Sweep =    4 | Direction =  forward | Bond dimension =  200 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.521 | E =      -3.2667431000 | DE = -1.78e-15 | DW = 6.12e-21

Sweep =    5 | Direction = backward | Bond dimension =  200 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.634 | E =      -3.2667431000 | DE = -3.55e-15 | DW = 2.47e-20

Sweep =    6 | Direction =  forward | Bond dimension =  200 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.781 | E =      -3.2667431000 | DE = 7.11e-15 | DW = 7.01e-21

Sweep =    7 | Direction = backward | Bond dimension =  200 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.876 | E =      -3.2667431000 | DE = 1.78e-15 | DW = 1.12e-20

Sweep =    8 | Direction =  forward | Bond dimension =  200 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      0.997 | E =      -3.2667431000 | DE = 1.78e-15 | DW = 1.93e-20

Ground state energy =   -3.266743099951065

Sweep =    0 | Direction = backward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.057 | F = (0.9960879032,0.0000000000) | DW = 3.05e-25

Sweep =    1 | Direction =  forward | BRA bond dimension =  200 | Noise =  0.00e+00
```

(continues on next page)

(continued from previous page)

```
Time elapsed =      0.120 | F = (0.9970594762,0.0000000000) | DF = (9.72e-04,0.
↪00e+00) | DW = 9.73e-25

Sweep =    2 | Direction = backward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.168 | F = (0.9970594762,0.0000000000) | DF = (4.44e-16,0.
↪00e+00) | DW = 1.57e-25

Sweep =    3 | Direction =  forward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.213 | F = (0.9970594762,0.0000000000) | DF = (-5.55e-16,0.
↪00e+00) | DW = 3.36e-25

Sweep =    4 | Direction = backward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.262 | F = (0.9970594762,0.0000000000) | DF = (2.22e-16,0.
↪00e+00) | DW = 1.04e-24

it =     0 ( 0.0 %)
it =    25 ( 1.0 %)
it =    50 ( 2.0 %)
it =    75 ( 3.0 %)
it =   100 ( 4.0 %)
it =   125 ( 5.0 %)
it =   150 ( 6.0 %)
it =   175 ( 7.0 %)
it =   200 ( 8.0 %)
it =   225 ( 9.0 %)
it =   250 (10.0 %)
it =   275 (11.0 %)
it =   300 (12.0 %)
it =   325 (13.0 %)
it =   350 (14.0 %)
it =   375 (15.0 %)
it =   400 (16.0 %)
it =   425 (17.0 %)
it =   450 (18.0 %)
it =   475 (19.0 %)
it =   500 (20.0 %)
it =   525 (21.0 %)
it =   550 (22.0 %)
it =   575 (23.0 %)
it =   600 (24.0 %)
it =   625 (25.0 %)
it =   650 (26.0 %)
it =   675 (27.0 %)
it =   700 (28.0 %)
it =   725 (29.0 %)
```

(continues on next page)

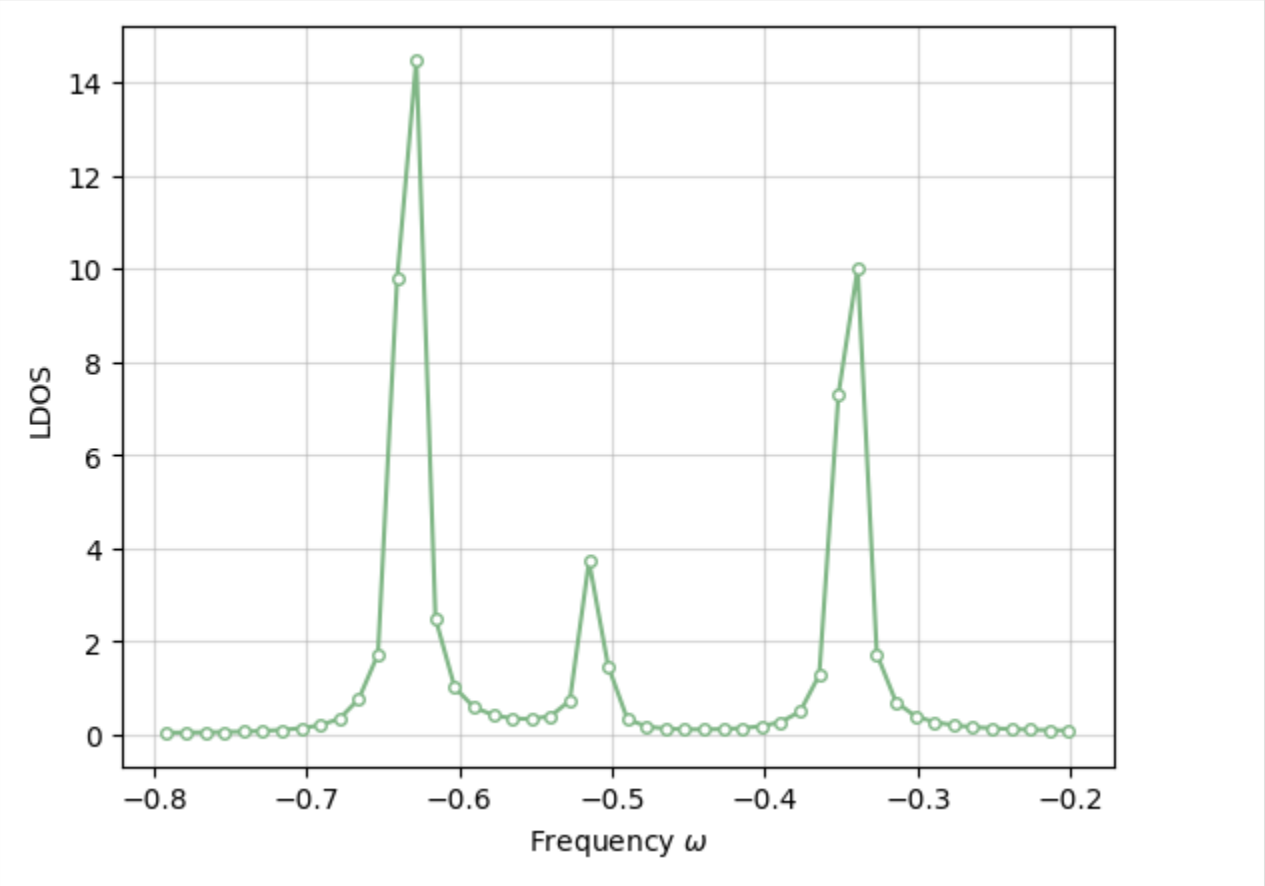
(continued from previous page)

```
it =   750 (30.0 %)
it =   775 (31.0 %)
it =   800 (32.0 %)
it =   825 (33.0 %)
it =   850 (34.0 %)
it =   875 (35.0 %)
it =   900 (36.0 %)
it =   925 (37.0 %)
it =   950 (38.0 %)
it =   975 (39.0 %)
it =  1000 (40.0 %)
it =  1025 (41.0 %)
it =  1050 (42.0 %)
it =  1075 (43.0 %)
it =  1100 (44.0 %)
it =  1125 (45.0 %)
it =  1150 (46.0 %)
it =  1175 (47.0 %)
it =  1200 (48.0 %)
it =  1225 (49.0 %)
it =  1250 (50.0 %)
it =  1275 (51.0 %)
it =  1300 (52.0 %)
it =  1325 (53.0 %)
it =  1350 (54.0 %)
it =  1375 (55.0 %)
it =  1400 (56.0 %)
it =  1425 (57.0 %)
it =  1450 (58.0 %)
it =  1475 (59.0 %)
it =  1500 (60.0 %)
it =  1525 (61.0 %)
it =  1550 (62.0 %)
it =  1575 (63.0 %)
it =  1600 (64.0 %)
it =  1625 (65.0 %)
it =  1650 (66.0 %)
it =  1675 (67.0 %)
it =  1700 (68.0 %)
it =  1725 (69.0 %)
it =  1750 (70.0 %)
it =  1775 (71.0 %)
it =  1800 (72.0 %)
it =  1825 (73.0 %)
it =  1850 (74.0 %)
```

(continues on next page)

(continued from previous page)

```
it =  1875 (75.0 %)
it =  1900 (76.0 %)
it =  1925 (77.0 %)
it =  1950 (78.0 %)
it =  1975 (79.0 %)
it =  2000 (80.0 %)
it =  2025 (81.0 %)
it =  2050 (82.0 %)
it =  2075 (83.0 %)
it =  2100 (84.0 %)
it =  2125 (85.0 %)
it =  2150 (86.0 %)
it =  2175 (87.0 %)
it =  2200 (88.0 %)
it =  2225 (89.0 %)
it =  2250 (90.0 %)
it =  2275 (91.0 %)
it =  2300 (92.0 %)
it =  2325 (93.0 %)
it =  2350 (94.0 %)
it =  2375 (95.0 %)
it =  2400 (96.0 %)
it =  2425 (97.0 %)
it =  2450 (98.0 %)
it =  2475 (99.0 %)
```



In the SZ mode:

[5]:
```
import numpy as np
from pyblock2._pyscf.ao2mo import integrals as itg
from pyblock2.driver.core import DMRGDriver, SymmetryTypes
from pyscf import gto, scf, lo

BOHR = 0.52917721092
R = 1.8 * BOHR
N = 6

mol = gto.M(atom=[['H', (i * R, 0, 0)] for i in range(N)], basis="sto6g", symmetry=
↪"c1", verbose=0)
mf = scf.RHF(mol).run(conv_tol=1E-14)

mf.mo_coeff = lo.orth.lowdin(mol.intor('cint1e_ovlp_sph'))
ncas, n_elec, spin, ecore, h1e, g2e, orb_sym = itg.get_rhf_integrals(mf, ncore=0,␣
↪ncas=None, g2e_symm=8)

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SZ | SymmetryTypes.CPX,␣
↪n_threads=4)
```

(continues on next page)

(continued from previous page)

```
driver.initialize_system(n_sites=ncas, n_elec=n_elec, spin=spin, orb_sym=orb_sym)

bond_dims = [150] * 4 + [200] * 4
noises = [1e-4] * 4 + [1e-5] * 4 + [0]
thrds = [1e-10] * 8

mpo = driver.get_qc_mpo(h1e=h1e, g2e=g2e, ecore=ecore, integral_cutoff=1E-8,␣
↪iprint=1)
ket = driver.get_random_mps(tag="KET", bond_dim=150, nroots=1)
energy = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('Ground state energy = %20.15f' % energy)

isite = 2
mpo.const_e -= energy
eta = 0.005

dmpo = driver.get_site_mpo(op='d', site_index=isite, iprint=0) # only alpha spin
dket = driver.get_random_mps(tag="DKET", bond_dim=200, center=ket.center,␣
↪target=dmpo.op.q_label + ket.info.target)
driver.multiply(dket, dmpo, ket, n_sweeps=10, bond_dims=[200], thrds=[1E-10] * 10,␣
↪iprint=1)

impo = driver.get_identity_mpo()
dbra = driver.copy_mps(dket, tag='DBRA')

dt = 0.2
t = 500.0
nstep = int(t / dt)
rtgf = np.zeros((nstep, ), dtype=complex)
rtgf[0] = driver.expectation(dket, impo, dket)
for it in range(nstep - 1):
    if it % (nstep // 100) == 0:
        print("it = %5d (%4.1f %%)" % (it, it * 100 / nstep))
    dbra = driver.td_dmrg(mpo, dbra, -dt * 1j, -dt * 1j, final_mps_tag='DBRA',␣
↪hermitian=True, bond_dims=[200], iprint=0)
    rtgf[it + 1] = driver.expectation(dbra, impo, dket)

def gf_fft(eta, dt, rtgf, npts):
    frq = np.fft.fftfreq(npts, dt)
    frq = np.fft.fftshift(frq) * 2.0 * np.pi
    fftinp = -1j * rtgf * np.exp(-eta * dt * np.arange(0, npts))
    return frq, np.fft.fftshift(np.fft.fft(fftinp)) * dt

frq, frq_gf = gf_fft(eta, dt, rtgf, len(rtgf))
```

(continues on next page)

(continued from previous page)

```
frq_gf = frq_gf[(frq >= -0.8) & (frq < -0.2)]
frq = frq[(frq >= -0.8) & (frq < -0.2)]

ldos = -2 / np.pi * frq_gf.imag

import matplotlib.pyplot as plt
plt.grid(which='major', axis='both', alpha=0.5)
plt.plot(frq, ldos, linestyle='-', marker='o', markersize=4, mfc='white', mec="
↪#7FB685", color="#7FB685")
plt.xlabel("Frequency $\\omega$")
plt.ylabel("LDOS")
plt.show()
```

```
integral symmetrize error =  0.0
integral cutoff error =  0.0
mpo terms =       2286

Build MPO | Nsites =     6 | Nterms =       2286 | Algorithm = FastBIP | Cutoff = 1.
↪00e-20
 Site =     0 /     6 .. Mmpo =    26 DW = 0.00e+00 NNZ =       26 SPT = 0.0000 Tmvc␣
↪= 0.001 T = 0.016
 Site =     1 /     6 .. Mmpo =    66 DW = 0.00e+00 NNZ =      243 SPT = 0.8584 Tmvc␣
↪= 0.001 T = 0.016
 Site =     2 /     6 .. Mmpo =   110 DW = 0.00e+00 NNZ =      459 SPT = 0.9368 Tmvc␣
↪= 0.001 T = 0.014
 Site =     3 /     6 .. Mmpo =    66 DW = 0.00e+00 NNZ =     1147 SPT = 0.8420 Tmvc␣
↪= 0.001 T = 0.013
 Site =     4 /     6 .. Mmpo =    26 DW = 0.00e+00 NNZ =      243 SPT = 0.8584 Tmvc␣
↪= 0.000 T = 0.004
 Site =     5 /     6 .. Mmpo =     1 DW = 0.00e+00 NNZ =       26 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.003
Ttotal =      0.066 Tmvc-total = 0.004 MPO bond dimension =   110 MaxDW = 0.00e+00
NNZ =         2144 SIZE =        18004 SPT = 0.8809

Rank =     0 Ttotal =      0.124 MPO method = FastBipartite bond dimension =     110␣
↪NNZ =         2144 SIZE =        18004 SPT = 0.8809

Sweep =    0 | Direction =  forward | Bond dimension =  150 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.230 | E =      -3.2667431000 | DW = 7.12e-17

Sweep =    1 | Direction = backward | Bond dimension =  150 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.413 | E =      -3.2667431000 | DE = -1.78e-15 | DW = 6.71e-17

Sweep =    2 | Direction =  forward | Bond dimension =  150 | Noise =  1.00e-04 |␣
```

(continues on next page)

(continued from previous page)

```
↪Dav threshold =  1.00e-10
Time elapsed =      0.563 | E =      -3.2667431000 | DE = 3.55e-15 | DW = 6.86e-17

Sweep =    3 | Direction = backward | Bond dimension =  150 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.770 | E =      -3.2667431000 | DE = 0.00e+00 | DW = 7.22e-17

Sweep =    4 | Direction =  forward | Bond dimension =  200 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.951 | E =      -3.2667431000 | DE = -3.55e-15 | DW = 1.68e-20

Sweep =    5 | Direction = backward | Bond dimension =  200 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.109 | E =      -3.2667431000 | DE = 0.00e+00 | DW = 2.60e-20

Sweep =    6 | Direction =  forward | Bond dimension =  200 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.261 | E =      -3.2667431000 | DE = 0.00e+00 | DW = 1.87e-20

Sweep =    7 | Direction = backward | Bond dimension =  200 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.415 | E =      -3.2667431000 | DE = 0.00e+00 | DW = 3.66e-20

Sweep =    8 | Direction =  forward | Bond dimension =  200 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      1.526 | E =      -3.2667431000 | DE = -1.78e-15 | DW = 5.13e-20

Ground state energy =   -3.266743099950349

Sweep =    0 | Direction = backward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.022 | F = (0.6266622731,0.0000000000) | DW = 7.22e-25

Sweep =    1 | Direction =  forward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.051 | F = (0.7050283557,0.0000000000) | DF = (7.84e-02,0.
↪00e+00) | DW = 9.91e-25

Sweep =    2 | Direction = backward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.076 | F = (0.7050283557,0.0000000000) | DF = (4.44e-16,0.
↪00e+00) | DW = 7.16e-25

Sweep =    3 | Direction =  forward | BRA bond dimension =  200 | Noise =  0.00e+00
Time elapsed =      0.112 | F = (0.7050283557,0.0000000000) | DF = (-1.11e-16,0.
↪00e+00) | DW = 1.77e-24

Sweep =    4 | Direction = backward | BRA bond dimension =  200 | Noise =  0.00e+00
```

(continues on next page)

(continued from previous page)

```
Time elapsed =      0.148 | F = (0.7050283557,0.0000000000) | DF = (-1.11e-16,0.
↪00e+00) | DW = 7.19e-25

it =     0 ( 0.0 %)
it =    25 ( 1.0 %)
it =    50 ( 2.0 %)
it =    75 ( 3.0 %)
it =   100 ( 4.0 %)
it =   125 ( 5.0 %)
it =   150 ( 6.0 %)
it =   175 ( 7.0 %)
it =   200 ( 8.0 %)
it =   225 ( 9.0 %)
it =   250 (10.0 %)
it =   275 (11.0 %)
it =   300 (12.0 %)
it =   325 (13.0 %)
it =   350 (14.0 %)
it =   375 (15.0 %)
it =   400 (16.0 %)
it =   425 (17.0 %)
it =   450 (18.0 %)
it =   475 (19.0 %)
it =   500 (20.0 %)
it =   525 (21.0 %)
it =   550 (22.0 %)
it =   575 (23.0 %)
it =   600 (24.0 %)
it =   625 (25.0 %)
it =   650 (26.0 %)
it =   675 (27.0 %)
it =   700 (28.0 %)
it =   725 (29.0 %)
it =   750 (30.0 %)
it =   775 (31.0 %)
it =   800 (32.0 %)
it =   825 (33.0 %)
it =   850 (34.0 %)
it =   875 (35.0 %)
it =   900 (36.0 %)
it =   925 (37.0 %)
it =   950 (38.0 %)
it =   975 (39.0 %)
it =  1000 (40.0 %)
it =  1025 (41.0 %)
```

(continues on next page)

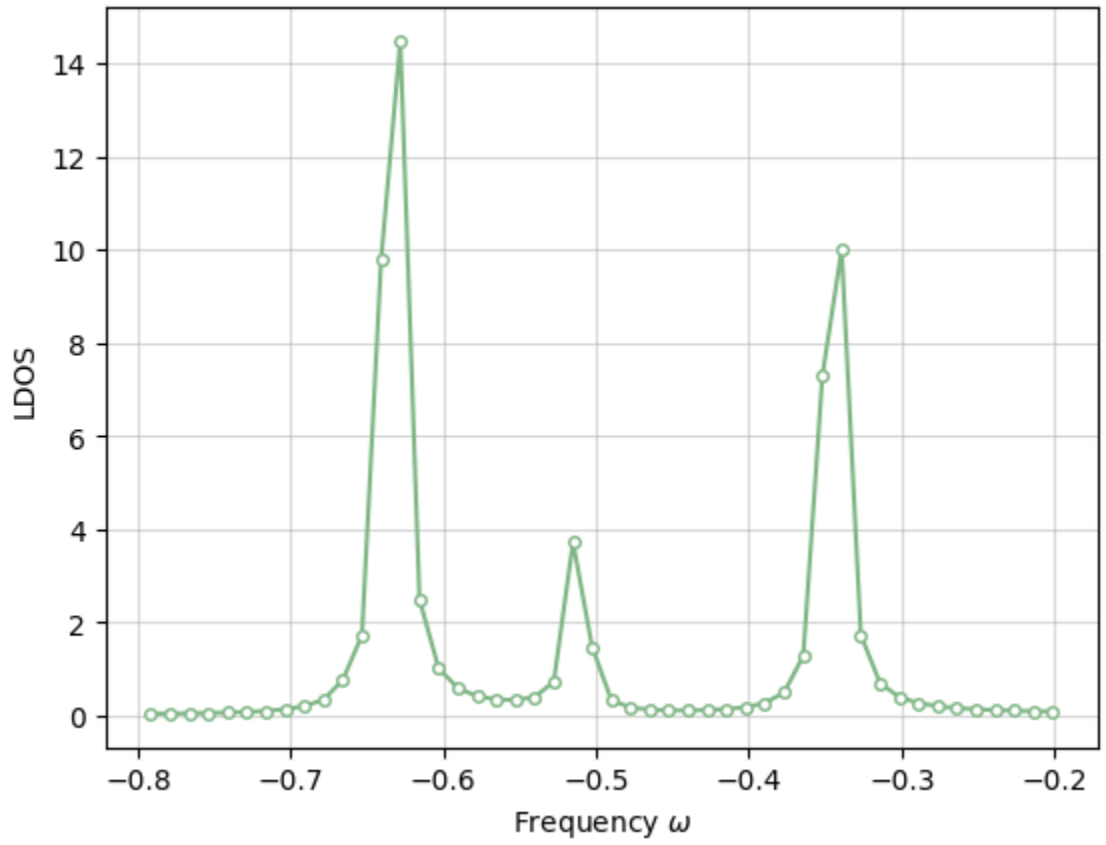
(continued from previous page)

```
it =  1050 (42.0 %)
it =  1075 (43.0 %)
it =  1100 (44.0 %)
it =  1125 (45.0 %)
it =  1150 (46.0 %)
it =  1175 (47.0 %)
it =  1200 (48.0 %)
it =  1225 (49.0 %)
it =  1250 (50.0 %)
it =  1275 (51.0 %)
it =  1300 (52.0 %)
it =  1325 (53.0 %)
it =  1350 (54.0 %)
it =  1375 (55.0 %)
it =  1400 (56.0 %)
it =  1425 (57.0 %)
it =  1450 (58.0 %)
it =  1475 (59.0 %)
it =  1500 (60.0 %)
it =  1525 (61.0 %)
it =  1550 (62.0 %)
it =  1575 (63.0 %)
it =  1600 (64.0 %)
it =  1625 (65.0 %)
it =  1650 (66.0 %)
it =  1675 (67.0 %)
it =  1700 (68.0 %)
it =  1725 (69.0 %)
it =  1750 (70.0 %)
it =  1775 (71.0 %)
it =  1800 (72.0 %)
it =  1825 (73.0 %)
it =  1850 (74.0 %)
it =  1875 (75.0 %)
it =  1900 (76.0 %)
it =  1925 (77.0 %)
it =  1950 (78.0 %)
it =  1975 (79.0 %)
it =  2000 (80.0 %)
it =  2025 (81.0 %)
it =  2050 (82.0 %)
it =  2075 (83.0 %)
it =  2100 (84.0 %)
it =  2125 (85.0 %)
it =  2150 (86.0 %)
```

(continues on next page)

(continued from previous page)

```
it =  2175 (87.0 %)
it =  2200 (88.0 %)
it =  2225 (89.0 %)
it =  2250 (90.0 %)
it =  2275 (91.0 %)
it =  2300 (92.0 %)
it =  2325 (93.0 %)
it =  2350 (94.0 %)
it =  2375 (95.0 %)
it =  2400 (96.0 %)
it =  2425 (97.0 %)
it =  2450 (98.0 %)
it =  2475 (99.0 %)
```



4.6 Custom Hamiltonians

[1]:
```
!pip install block2==0.5.3rc6 -qq --progress-bar off --extra-index-url=https://block-
↪hczhai.github.io/block2-preview/pypi/
```

In this tutorial, we provide an example python scripts for performing DMRG using custom Hamiltonians, where the operators and states at local Hilbert space at every site can be redefined. It is also possible to use different local Hilbert space for different sites. New letters can be introduced for representing new operators (the operator name can only be a single lower or upper case character).

Note the following examples are only supposed to work in the Abelian symmetry modes (`SZ`, `SZ|CPX`, `SGF`, `SGFCPX`, `SAny`, or `SAny|CPX`).

4.6.1 The Hubbard Model

In the following example, we implement a custom Hamiltonian for the Hubbard model. In the standard implementation, the on-site term was represented as `cdCD`. Here we instead introduce a single letter `N` for the `cdCD` term. For each letter in `cdCDN` (representing elementary operators), we define its matrix representation in the local basis in `site_ops`. The quantum number and number of states in each quantum number at each site (which defines the local Hilbert space) is set in `site_basis`.

[2]:
```
from pyblock2.driver.core import DMRGDriver, SymmetryTypes, MPOAlgorithmTypes
import numpy as np

L = 8
U = 2
N_ELEC = 8

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SZ, n_threads=4)
driver.initialize_system(n_sites=L, n_elec=N_ELEC, spin=0)

# [Part A] Set states and matrix representation of operators in local Hilbert space
site_basis, site_ops = [], []
Q = driver.bw.SX # quantum number wrapper (n_elec, 2 * spin, point group irrep)

for k in range(L):
    basis = [(Q(0, 0, 0), 1), (Q(1, 1, 0), 1), (Q(1, -1, 0), 1), (Q(2, 0, 0), 1)] #␣
↪[0ab2]
    ops = {
        "": np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]),   #␣
↪identity
        "c": np.array([[0, 0, 0, 0], [1, 0, 0, 0], [0, 0, 0, 0], [0, 0, 1, 0]]),  #␣
↪alpha+
        "d": np.array([[0, 1, 0, 0], [0, 0, 0, 0], [0, 0, 0, 1], [0, 0, 0, 0]]),  #␣
```

(continues on next page)

(continued from previous page)

```
↪alpha
        "C": np.array([[0, 0, 0, 0], [0, 0, 0, 0], [1, 0, 0, 0], [0, -1, 0, 0]]), #␣
↪beta+
        "D": np.array([[0, 0, 1, 0], [0, 0, 0, -1], [0, 0, 0, 0], [0, 0, 0, 0]]), #␣
↪beta
        "N": np.array([[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 1]]),  #␣
↪cdCD
    }
    site_basis.append(basis)
    site_ops.append(ops)

# [Part B] Set Hamiltonian terms
driver.ghamil = driver.get_custom_hamiltonian(site_basis, site_ops)
b = driver.expr_builder()

b.add_term("cd", np.array([[i, i + 1, i + 1, i] for i in range(L - 1)]).ravel(), -1)
b.add_term("CD", np.array([[i, i + 1, i + 1, i] for i in range(L - 1)]).ravel(), -1)
b.add_term("N", np.array([i for i in range(L)]), U)

# [Part C] Perform DMRG
mpo = driver.get_mpo(b.finalize(adjust_order=True, fermionic_ops="cdCD"), algo_
↪type=MPOAlgorithmTypes.FastBipartite)
mps = driver.get_random_mps(tag="KET", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, mps, n_sweeps=10, bond_dims=[250] * 4 + [500] * 4,
    noises=[1e-4] * 4 + [1e-5] * 4 + [0], thrds=[1e-10] * 8, dav_max_iter=30,␣
↪iprint=1)
print("DMRG energy = %20.15f" % energy)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.379 | E =      -6.2256341447 | DW = 2.65e-16

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.487 | E =      -6.2256341447 | DE = -1.15e-14 | DW = 4.93e-16

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.607 | E =      -6.2256341447 | DE = -2.66e-15 | DW = 9.63e-17

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.729 | E =      -6.2256341447 | DE = 1.78e-15 | DW = 1.20e-16

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
```

(continues on next page)

(continued from previous page)

```
↪Dav threshold =  1.00e-10
Time elapsed =      0.877 | E =      -6.2256341447 | DE = 3.55e-15 | DW = 4.71e-20

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.989 | E =      -6.2256341447 | DE = -3.55e-15 | DW = 3.71e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.119 | E =      -6.2256341447 | DE = -7.11e-15 | DW = 3.62e-20

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.328 | E =      -6.2256341447 | DE = 0.00e+00 | DW = 5.12e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      1.423 | E =      -6.2256341447 | DE = 3.55e-15 | DW = 4.24e-20

DMRG energy =   -6.225634144657922
```

4.6.2 The Hubbard-Holstein Model

The above script can be easily extended to treat phonons.

[3]:
```
from pyblock2.driver.core import DMRGDriver, SymmetryTypes, MPOAlgorithmTypes
import numpy as np

N_SITES_ELEC, N_SITES_PH, N_ELEC = 4, 4, 4
N_PH, U, OMEGA, G = 11, 2, 0.25, 0.5
L = N_SITES_ELEC + N_SITES_PH

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SZ, n_threads=4)
driver.initialize_system(n_sites=L, n_elec=N_ELEC, spin=0)

# [Part A] Set states and matrix representation of operators in local Hilbert space
site_basis, site_ops = [], []
Q = driver.bw.SX # quantum number wrapper (n_elec, 2 * spin, point group irrep)

for k in range(L):
    if k < N_SITES_ELEC:
        # electron part
        basis = [(Q(0, 0, 0), 1), (Q(1, 1, 0), 1), (Q(1, -1, 0), 1), (Q(2, 0, 0),␣
↪1)] # [0ab2]
        ops = {
```

(continues on next page)

(continued from previous page)

```
            "": np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]), ␣
↪ # identity
            "c": np.array([[0, 0, 0, 0], [1, 0, 0, 0], [0, 0, 0, 0], [0, 0, 1, 0]]),␣
↪ # alpha+
            "d": np.array([[0, 1, 0, 0], [0, 0, 0, 0], [0, 0, 0, 1], [0, 0, 0, 0]]),␣
↪ # alpha
            "C": np.array([[0, 0, 0, 0], [0, 0, 0, 0], [1, 0, 0, 0], [0, -1, 0, 0]]),
↪ # beta+
            "D": np.array([[0, 0, 1, 0], [0, 0, 0, -1], [0, 0, 0, 0], [0, 0, 0, 0]]),
↪ # beta
        }
    else:
        # phonon part
        basis = [(Q(0, 0, 0), N_PH)]
        ops = {
            "": np.identity(N_PH), # identity
            "E": np.diag(np.sqrt(np.arange(1, N_PH)), k=-1), # ph+
            "F": np.diag(np.sqrt(np.arange(1, N_PH)), k=1),  # ph
        }
    site_basis.append(basis)
    site_ops.append(ops)

# [Part B] Set Hamiltonian terms in Hubbard-Holstein model
driver.ghamil = driver.get_custom_hamiltonian(site_basis, site_ops)
b = driver.expr_builder()

# electron part
b.add_term("cd", np.array([[i, i + 1, i + 1, i] for i in range(N_SITES_ELEC - 1)]).
↪ravel(), -1)
b.add_term("CD", np.array([[i, i + 1, i + 1, i] for i in range(N_SITES_ELEC - 1)]).
↪ravel(), -1)
b.add_term("cdCD", np.array([[i, i, i, i] for i in range(N_SITES_ELEC)]).ravel(), U)

# phonon part
b.add_term("EF", np.array([[i + N_SITES_ELEC, ] * 2 for i in range(N_SITES_PH)]).
↪ravel(), OMEGA)

# interaction part
b.add_term("cdE", np.array([[i, i, i + N_SITES_ELEC] for i in range(N_SITES_ELEC)]).
↪ravel(), G)
b.add_term("cdF", np.array([[i, i, i + N_SITES_ELEC] for i in range(N_SITES_ELEC)]).
↪ravel(), G)
b.add_term("CDE", np.array([[i, i, i + N_SITES_ELEC] for i in range(N_SITES_ELEC)]).
↪ravel(), G)
b.add_term("CDF", np.array([[i, i, i + N_SITES_ELEC] for i in range(N_SITES_ELEC)]).
```

(continues on next page)

(continued from previous page)

```
↪ravel(), G)

# [Part C] Perform DMRG
mpo = driver.get_mpo(b.finalize(adjust_order=True, fermionic_ops="cdCD"), algo_
↪type=MPOAlgorithmTypes.FastBipartite)
mps = driver.get_random_mps(tag="KET", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, mps, n_sweeps=10, bond_dims=[250] * 4 + [500] * 4,
    noises=[1e-4] * 4 + [1e-5] * 4 + [0], thrds=[1e-10] * 8, dav_max_iter=30,␣
↪iprint=1)
print("DMRG energy = %20.15f" % energy)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     90.266 | E =      -6.9568929201 | DW = 3.62e-09

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    119.298 | E =      -6.9568932112 | DE = -2.91e-07 | DW = 3.07e-19

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    130.142 | E =      -6.9568932112 | DE = -4.44e-15 | DW = 1.24e-19

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    139.619 | E =      -6.9568932112 | DE = -6.22e-15 | DW = 6.77e-20

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    148.704 | E =      -6.9568932112 | DE = 5.33e-15 | DW = 8.92e-20

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    158.074 | E =      -6.9568932112 | DE = 8.88e-16 | DW = 7.14e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    167.573 | E =      -6.9568932112 | DE = 8.88e-16 | DW = 7.69e-20

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =    176.769 | E =      -6.9568932112 | DE = -1.78e-15 | DW = 7.20e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
```

(continues on next page)

(continued from previous page)

```
Time elapsed =    185.427 | E =      -6.9568932112 | DE = 0.00e+00 | DW = 8.02e-20

DMRG energy =   -6.956893211182315
```

4.6.3 Custom Symmetry Groups

In the following we show how to set the custom symmetry groups. The symmetry mode `SymmetryTypes.SAny` (or `SymmetryTypes.SAny | SymmetryTypes.CPX` if complex number is required) should be used for this purpose. Currently we support the definition of symmetry group as an arbitrary direct product of up to six Abelian symmetry sub-groups. Possible sub-group names are "U1", "Z1", "Z2", "Z3", ..., "Z2055", "U1Fermi", "Z1Fermi", "Z2Fermi", "Z3Fermi", ..., "Z2055Fermi", "LZ", and "AbelianPG". The names with the suffix "Fermi" should be used for Fermion symmetries. The names without the suffix "Fermi" should be used for spin or Boson symmetries. The `DMRGDriver.set_symmetry_groups(sub_group_name_1: str, sub_group_name_2: str, ...)` method can be used to set the symmetry sub-groups. The number of arguments in the quantum number wrapper `Q` should then match the number of sub-group names given in `DMRGDriver.set_symmetry_groups`.

As a first example, we use the custom symmetry group syntax to recompute the Hubbard model. We first use $U(1) \times U(1)$ symmetry, which should be equivalent to the previous `SZ` mode example.

[4]:
```
from pyblock2.driver.core import DMRGDriver, SymmetryTypes, MPOAlgorithmTypes
import numpy as np

L = 8
U = 2
N_ELEC = 8
TWO_SZ = 0

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SAny, n_threads=4)

# quantum number wrapper (U1 / n_elec, U1 / 2*Sz)
driver.set_symmetry_groups("U1Fermi", "U1Fermi")
Q = driver.bw.SX

# [Part A] Set states and matrix representation of operators in local Hilbert space
site_basis, site_ops = [], []

for k in range(L):
    basis = [(Q(0, 0), 1), (Q(1, 1), 1), (Q(1, -1), 1), (Q(2, 0), 1)] # [0ab2]
    ops = {
        "": np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]),   #␣
↪identity
        "c": np.array([[0, 0, 0, 0], [1, 0, 0, 0], [0, 0, 0, 0], [0, 0, 1, 0]]),  #␣
↪alpha+
```

(continues on next page)

(continued from previous page)

```
        "d": np.array([[0, 1, 0, 0], [0, 0, 0, 0], [0, 0, 0, 1], [0, 0, 0, 0]]),  #␣
↪alpha
        "C": np.array([[0, 0, 0, 0], [0, 0, 0, 0], [1, 0, 0, 0], [0, -1, 0, 0]]), #␣
↪beta+
        "D": np.array([[0, 0, 1, 0], [0, 0, 0, -1], [0, 0, 0, 0], [0, 0, 0, 0]]), #␣
↪beta
    }
    site_basis.append(basis)
    site_ops.append(ops)

# [Part B] Set Hamiltonian terms
driver.initialize_system(n_sites=L, vacuum=Q(0, 0), target=Q(N_ELEC, TWO_SZ), hamil_
↪init=False)
driver.ghamil = driver.get_custom_hamiltonian(site_basis, site_ops)
b = driver.expr_builder()

b.add_term("cd", np.array([[i, i + 1, i + 1, i] for i in range(L - 1)]).ravel(), -1)
b.add_term("CD", np.array([[i, i + 1, i + 1, i] for i in range(L - 1)]).ravel(), -1)
b.add_term("cdCD", np.array([i for i in range(L) for _ in range(4)]), U)

# [Part C] Perform DMRG
mpo = driver.get_mpo(b.finalize(adjust_order=True, fermionic_ops="cdCD"), algo_
↪type=MPOAlgorithmTypes.FastBipartite)
mps = driver.get_random_mps(tag="KET", bond_dim=250, nroots=1)
energy = driver.dmrg(mpo, mps, n_sweeps=10, bond_dims=[250] * 4 + [500] * 4,
    noises=[1e-4] * 4 + [1e-5] * 4 + [0], thrds=[1e-10] * 8, dav_max_iter=30,␣
↪iprint=1)
print("DMRG energy = %20.15f" % energy)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.176 | E =      -6.2256341447 | DW = 5.28e-16

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.243 | E =      -6.2256341447 | DE = -9.77e-15 | DW = 6.68e-16

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.313 | E =      -6.2256341447 | DE = -8.88e-16 | DW = 1.63e-16

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.377 | E =      -6.2256341447 | DE = -4.44e-15 | DW = 2.17e-16
```

(continues on next page)

(continued from previous page)

```
Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.446 | E =      -6.2256341447 | DE = 8.88e-16 | DW = 2.98e-20

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.556 | E =      -6.2256341447 | DE = 1.78e-15 | DW = 4.40e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.672 | E =      -6.2256341447 | DE = 8.88e-16 | DW = 2.10e-20

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.777 | E =      -6.2256341447 | DE = 1.78e-15 | DW = 3.94e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      0.852 | E =      -6.2256341447 | DE = -4.44e-15 | DW = 3.77e-20

DMRG energy =   -6.225634144658390
```

As a second example, we recompute the Hubbard model using $Z_2 \times Z_2$ symmetry. This time we cannot easily target the $N_{\text{elec}} = 8$ symmetry sector. Instead, we compute a few excited states and compute the $\langle N \rangle$ to identify the correct state.

[5]:
```
from pyblock2.driver.core import DMRGDriver, SymmetryTypes, MPOAlgorithmTypes
import numpy as np

L = 8
U = 2
N_ELEC = 8
TWO_SZ = 0

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SAny, n_threads=4)

# quantum number wrapper (Z2 / n_elec, Z2 / 2*Sz)
driver.set_symmetry_groups("Z2Fermi", "Z2Fermi")
Q = driver.bw.SX

# [Part A] Set states and matrix representation of operators in local Hilbert space
site_basis, site_ops = [], []

for k in range(L):
    basis = [(Q(0, 0), 2), (Q(1, 1), 2)] # [02ab]
    ops = {
```

(continues on next page)

(continued from previous page)

```
        # note the order of row and column is different from the U1xU1 case
        "": np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]]),   #␣
↪identity
        "c": np.array([[0, 0, 0, 0], [0, 0, 0, 1], [1, 0, 0, 0], [0, 0, 0, 0]]),  #␣
↪alpha+
        "d": np.array([[0, 0, 1, 0], [0, 0, 0, 0], [0, 0, 0, 0], [0, 1, 0, 0]]),  #␣
↪alpha
        "C": np.array([[0, 0, 0, 0], [0, 0, -1, 0], [0, 0, 0, 0], [1, 0, 0, 0]]), #␣
↪beta+
        "D": np.array([[0, 0, 0, 1], [0, 0, 0, 0], [0, -1, 0, 0], [0, 0, 0, 0]]), #␣
↪beta
    }
    site_basis.append(basis)
    site_ops.append(ops)

# [Part B] Set Hamiltonian terms
driver.initialize_system(n_sites=L, vacuum=Q(0, 0), target=Q(N_ELEC % 2, TWO_SZ % 2),
↪ hamil_init=False)
driver.ghamil = driver.get_custom_hamiltonian(site_basis, site_ops)
b = driver.expr_builder()

b.add_term("cd", np.array([[i, i + 1, i + 1, i] for i in range(L - 1)]).ravel(), -1)
b.add_term("CD", np.array([[i, i + 1, i + 1, i] for i in range(L - 1)]).ravel(), -1)
b.add_term("cdCD", np.array([i for i in range(L) for _ in range(4)]), U)

# [Part C] Perform state-averaged DMRG
mpo = driver.get_mpo(b.finalize(adjust_order=True, fermionic_ops="cdCD"), algo_
↪type=MPOAlgorithmTypes.FastBipartite)
mps = driver.get_random_mps(tag="KET", bond_dim=250, nroots=10)
energies = driver.dmrg(mpo, mps, n_sweeps=10, bond_dims=[250] * 4 + [500] * 4,
    noises=[1e-4] * 4 + [1e-5] * 4 + [0], thrds=[1e-10] * 8, dav_max_iter=200,␣
↪iprint=1)

# [Part D] Check particle number expectations
b = driver.expr_builder()
b.add_term("cd", np.array([[i, i] for i in range(L)]).ravel(), 1)
b.add_term("CD", np.array([[i, i] for i in range(L)]).ravel(), 1)
partile_n_mpo = driver.get_mpo(b.finalize(adjust_order=True, fermionic_ops="cdCD"),␣
↪algo_type=MPOAlgorithmTypes.FastBipartite)

kets = [driver.split_mps(mps, ir, tag="KET-%d" % ir) for ir in range(mps.nroots)]
for ir in range(mps.nroots):
    n_expt = driver.expectation(kets[ir], partile_n_mpo, kets[ir])
    print("Root = %d <E> = %20.15f <N> = %10.3f" % (ir, energies[ir], n_expt))
```

(continues on next page)

(continued from previous page)

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 | Dav threshold =  1.00e-10
Time elapsed =     35.522 | E[ 10] =      -6.9958183038     -6.5705972882     -6.5705972844     -6.5705972810     -6.2773135424     -6.2256341358     -6.1122535707     -6.1122534909     -6.1122534352     -6.0320900882 | DW = 4.44e-09

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 | Dav threshold =  1.00e-10
Time elapsed =     39.172 | E[ 10] =      -6.9958183059     -6.5705972895     -6.5705972895     -6.5705972895     -6.2773135514     -6.2256341447     -6.1122535739     -6.1122535739     -6.1122535739     -6.0320902499 | DE = -1.62e-07 | DW = 1.21e-09

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 | Dav threshold =  1.00e-10
Time elapsed =     41.919 | E[ 10] =      -6.9958183059     -6.5705972895     -6.5705972895     -6.5705972895     -6.2773135514     -6.2256341447     -6.1122535739     -6.1122535739     -6.1122535739     -6.0320902499 | DE = 4.26e-12 | DW = 1.15e-09

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 | Dav threshold =  1.00e-10
Time elapsed =     44.670 | E[ 10] =      -6.9958183059     -6.5705972895     -6.5705972895     -6.5705972895     -6.2773135514     -6.2256341447     -6.1122535739     -6.1122535739     -6.1122535739     -6.0320902499 | DE = -2.69e-12 | DW = 1.11e-09

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 | Dav threshold =  1.00e-10
Time elapsed =     49.388 | E[ 10] =      -6.9958183059     -6.5705972895     -6.5705972895     -6.5705972895     -6.2773135514     -6.2256341447     -6.1122535739     -6.1122535739     -6.1122535739     -6.0320902499 | DE = -1.55e-13 | DW = 8.14e-17

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 | Dav threshold =  1.00e-10
Time elapsed =     52.005 | E[ 10] =      -6.9958183059     -6.5705972895     -6.5705972895     -6.5705972895     -6.2773135514     -6.2256341447     -6.1122535739     -6.1122535739     -6.1122535739     -6.0320902499 | DE = 4.44e-15 | DW = 1.67e-17

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 | Dav threshold =  1.00e-10
Time elapsed =     54.521 | E[ 10] =      -6.9958183059     -6.5705972895     -6.5705972895     -6.5705972895     -6.2773135514     -6.2256341447     -6.1122535739
```

(continues on next page)

(continued from previous page)

```
↪    -6.1122535739     -6.1122535739     -6.0320902499 | DE = -1.78e-15 | DW = 2.
↪12e-17

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     57.156 | E[ 10] =      -6.9958183059     -6.5705972895     -6.
↪5705972895     -6.5705972895     -6.2773135514     -6.2256341447     -6.1122535739␣
↪    -6.1122535739     -6.1122535739     -6.0320902499 | DE = 0.00e+00 | DW = 5.80e-
↪18

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =     59.782 | E[ 10] =      -6.9958183059     -6.5705972895     -6.
↪5705972895     -6.5705972895     -6.2773135514     -6.2256341447     -6.1122535739␣
↪    -6.1122535739     -6.1122535739     -6.0320902499 | DE = -8.88e-16 | DW = 1.
↪09e-18

Root = 0 <E> =   -6.995818305900327 <N> =      6.000
Root = 1 <E> =   -6.570597289542966 <N> =      6.000
Root = 2 <E> =   -6.570597289542286 <N> =      6.000
Root = 3 <E> =   -6.570597289541241 <N> =      6.000
Root = 4 <E> =   -6.277313551398623 <N> =      6.000
Root = 5 <E> =   -6.225634144677048 <N> =      8.000
Root = 6 <E> =   -6.112253573865427 <N> =      6.000
Root = 7 <E> =   -6.112253573864907 <N> =      6.000
Root = 8 <E> =   -6.112253573863808 <N> =      6.000
Root = 9 <E> =   -6.032090249941684 <N> =      4.000
```

4.6.4 SU(3) Heisenberg Model

In this example we will find the ground state of the 1D SU(3) Heisenberg model using the custom symemtry group syntax. We will only use Abelian symmetry groups (for quantum numbers S_z and Q_z) for this problem. The model used here can be found in Eq. (2) in *Phys. Rev. B* **79**, 012408 (2009).

[6]:
```
from pyblock2.driver.core import DMRGDriver, SymmetryTypes, MPOAlgorithmTypes
import numpy as np

L = 72

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SAny, n_threads=4)

# quantum number wrapper (2Sz / X, 2Qz / Y)
driver.set_symmetry_groups("U1", "U1")
Q = driver.bw.SX
```

(continues on next page)

(continued from previous page)

```
# [Part A] Set states and matrix representation of operators in local Hilbert space
site_basis, site_ops = [], []

# Gell Mann operators
lambda_ops = {
    "L1": np.array([[0, 1, 0], [1, 0, 0], [0, 0, 0]]),              # lambda_1
    "L2": np.array([[0, -1j, 0], [1j, 0, 0], [0, 0, 0]]),           # lambda_2
    "L3": np.array([[1, 0, 0], [0, -1, 0], [0, 0, 0]]),             # lambda_3
    "L4": np.array([[0, 0, 1], [0, 0, 0], [1, 0, 0]]),              # lambda_4
    "L5": np.array([[0, 0, -1j], [0, 0, 0], [1j, 0, 0]]),           # lambda_5
    "L6": np.array([[0, 0, 0], [0, 0, 1], [0, 1, 0]]),              # lambda_6
    "L7": np.array([[0, 0, 0], [0, 0, -1j], [0, 1j, 0]]),           # lambda_7
    "L8": np.array([[1, 0, 0], [0, 1, 0], [0, 0, -2]]) / 3 ** 0.5,  # lambda_8
}

for k in range(L):
    basis = [(Q(1, 1), 1), (Q(-1, 1), 1), (Q(0, -2), 1)]
    ops = {
        "": np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]]),        # identity
        "T": (lambda_ops["L1"] + 1j * lambda_ops["L2"]).real,   # T+
        "t": (lambda_ops["L1"] - 1j * lambda_ops["L2"]).real,   # T-
        "V": (lambda_ops["L4"] + 1j * lambda_ops["L5"]).real,   # V+
        "v": (lambda_ops["L4"] - 1j * lambda_ops["L5"]).real,   # V-
        "U": (lambda_ops["L6"] + 1j * lambda_ops["L7"]).real,   # U+
        "u": (lambda_ops["L6"] - 1j * lambda_ops["L7"]).real,   # U-
        "L": lambda_ops["L3"],                                  # L3
        "l": lambda_ops["L8"],                                  # L8
    }
    site_basis.append(basis)
    site_ops.append(ops)

# [Part B] Set Hamiltonian terms
driver.initialize_system(n_sites=L, vacuum=Q(0, 0), target=Q(0, 0), hamil_init=False)
driver.ghamil = driver.get_custom_hamiltonian(site_basis, site_ops)
b = driver.expr_builder()

b.add_term("Tt", np.array([[i, i + 1, i + 1, i] for i in range(L - 1)]).ravel(), 0.5␣
↪* 0.25)
b.add_term("Vv", np.array([[i, i + 1, i + 1, i] for i in range(L - 1)]).ravel(), 0.5␣
↪* 0.25)
b.add_term("Uu", np.array([[i, i + 1, i + 1, i] for i in range(L - 1)]).ravel(), 0.5␣
↪* 0.25)
b.add_term("LL", np.array([[i, i + 1] for i in range(L - 1)]).ravel(), 0.25)
b.add_term("ll", np.array([[i, i + 1] for i in range(L - 1)]).ravel(), 0.25)
```

(continues on next page)

(continued from previous page)

```
b.iscale(1 / L) # compute energy per site instead of total energy

# [Part C] Perform DMRG
mpo = driver.get_mpo(b.finalize(adjust_order=True, fermionic_ops=""), algo_
↪type=MPOAlgorithmTypes.FastBipartite)
mps = driver.get_random_mps(tag="KET", bond_dim=250, nroots=1)

energy = driver.dmrg(mpo, mps, n_sweeps=10, bond_dims=[250] * 4 + [500] * 4,
    noises=[1e-4] * 4 + [1e-5] * 4 + [0], thrds=[1e-10] * 8, dav_max_iter=30,␣
↪iprint=1)
print("DMRG energy (per site) = %20.15f" % energy)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     11.422 | E =      -0.5128539883 | DW = 1.32e-05

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     18.071 | E =      -0.5145284723 | DE = -1.67e-03 | DW = 3.73e-08

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     23.571 | E =      -0.5145507220 | DE = -2.22e-05 | DW = 3.84e-07

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     25.998 | E =      -0.5145508502 | DE = -1.28e-07 | DW = 4.73e-07

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     33.865 | E =      -0.5145512426 | DE = -3.92e-07 | DW = 8.02e-09

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     47.131 | E =      -0.5145512486 | DE = -5.93e-09 | DW = 1.24e-08

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     60.265 | E =      -0.5145512511 | DE = -2.51e-09 | DW = 5.33e-09

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =     73.415 | E =      -0.5145512506 | DE = 4.86e-10 | DW = 1.74e-09

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
```

(continues on next page)

(continued from previous page)

```
↪Dav threshold =  1.00e-09
Time elapsed =     83.304 | E =      -0.5145512503 | DE = 3.24e-10 | DW = 2.58e-18

DMRG energy (per site) =   -0.514551250253768
```

4.7 Hubbard Model

```
[1]: !pip install block2==0.5.2rc13 -qq --progress-bar off --extra-index-url=https://
     ↪block-hczhai.github.io/block2-preview/pypi/
```

4.7.1 Introduction

In this tutorial we introduce the python interface of `block2`. This interface allows the user to design custom workflow for running DMRG. It is also more convenient and efficient for model Hamiltonians like Hubbard and Heisenberg models, and non-standard quantum chemistry models with non-Hermitian or complex number integrals.

In `block2` we support a few different symmetry modes. For fermionic models, if the Hamiltonian conserves the total spin S, the projected spin S_z, and the number of electrons N, we can label the states using quantum numbers (N, S_z) or (N, S) or simply N.

1. The symmetry group for (N, S_z) is $\mathrm{U}(1) \times \mathrm{U}(1)$, which is Abelian. This symmetry mode is activated using `symm_type=SymmetryTypes.SZ`. We will first introduce the DMRG calculation using this symmetry since it is the most convenient choice.
2. If we label states using (N, S), the symmetry group is $\mathrm{U}(1) \times \mathrm{SU}(2)$. SU(2) is a non-Abelian symmetry, which is slightly difficult to work with, but it can be two to four times faster than the `SZ` mode. This symmetry mode is activated using `symm_type=SymmetryTypes.SU2`.
3. If we label states using only N, we can remove the `alpha` and `beta` subscripts in the Hamiltonian, and the symmetry group is only $\mathrm{U}(1)$. This symmetry mode is activated using `symm_type=SymmetryTypes.SGF` (general spin fermionic). Note that the fermion statistics is considered in this mode.
4. If we label states using only N, we can also transform the Fermionic Hamiltonian to qubit Hamiltonian using JW transformation and then do DMRG on qubits without considering the fermion statistics. This symmetry mode is activated using `symm_type=SymmetryTypes.SGB` (general spin bosonic). This mode has the worst efficiency (most of the time the efficiency of `SGF` and `SGB` should be the same), but it may show a clearer connection to other JW-based methods.
5. For 2D Hubbard model with only nearest-neighbor interactions, the particle number is also a pseudo-spin (particle-hole) $\mathrm{SU}(2)$ symmetry. If we label states using (N, S_z), the symmetry group is $\mathrm{SU}(2) \times \mathrm{U}(1)$. This symmetry mode is activated using `symm_type=SymmetryTypes.SAnyPHSU2`.

6. For 2D Hubbard model with only nearest-neighbor interactions, we can use the spin and pseudo-psin symmetries simultaneously. If we label states using (N, S), the symmetry group is $\mathrm{SU}(2) \times \mathrm{SU}(2) \,/\, \mathrm{Z}_2 = \mathrm{SO}(4)$. This symmetry mode is activated using `symm_type=SymmetryTypes.SAnySO4`.

Next, we will explain the settings of the Hubbard Hamiltonian in each of the modes (1) (2) (3) (4) (5) and (6).

[2]:
```
import numpy as np
from pyblock2.driver.core import DMRGDriver, SymmetryTypes
```

4.7.2 The SZ Mode

Initialization

The SZ mode should be the most convenient for the Hubbard model. We first set a driver using this mode. `scratch` sets a folder for writing temparary scratch files. `symm_type` sets the symmetry mode (and whether the complex number should be used. Here we allow only real numbers). `n_threads` sets the number of threads for shared-memory parallelization.

Next, we use the `initialize_system` method to set the property of the target wavefunction (namely, which symmetry sector is targeted).

- `n_sites` is the number of sites L (or length of the lattice). Here each site has four local states: $|0\rangle, |\alpha\rangle, |\beta\rangle$, and $|\alpha\beta\rangle$.
- `n_elec` is the number of electrons N in the target wavefunction. For half-filling Hubbard model, we should have `n_elec == n_sites`.
- `spin` is two times the total spin $2S$ (in `SU2` mode) or two times the projected spin $2S_z$ (in `SZ` mode) quantum number of the target wavefunction. Since we want the final wavefunction to have a equal number of alpha and beta electrons (namely, 4 alpha and 4 beta electrons for $N = 8$), this number is set to zero here. Note that $2S_z = N_\alpha - N_\beta$.

[3]:
```
L = 8
N = 8
TWOSZ = 0

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SZ, n_threads=4)
driver.initialize_system(n_sites=L, n_elec=N, spin=TWOSZ)
```

Build Hamiltonian

Now we set up the Hamiltonian. We first write the nearest-neighbor Hubbard model as the following:

$$\hat{H} = -t \sum_{i=1,\sigma}^{L-1} \left(a_{i\sigma}^{\dagger} a_{i+1\sigma} + a_{i+1\sigma}^{\dagger} a_{i\sigma} \right) + U \sum_{i=1}^{L} a_{i\alpha}^{\dagger} a_{i\alpha} a_{i\beta}^{\dagger} a_{i\beta}$$

Where we have considered the open boundary condition, and $\sigma \in \{\alpha, \beta\}$. In the code we use characters `c`, `d`, `C`, `D` to represent $a_{\alpha}^{\dagger}, a_{\alpha}, a_{\beta}^{\dagger}, a_{\beta}$, respectively.

The method `expr_builder` will return a `ExprBuilder` object.

The method `add_term` of the `ExprBuilder` object has three parameters.

- The first parameter is a string of characters in `cdCD`.
- The second parameter is a list of integers indicating the site indices associated with each charactor (operator) in the first parameter
- The third parameter is a floating point number (or a list of floating point numbers), indicating the coefficient of the added term.

For example, `b.add_term("dCc", [1, 3, 0], 0.7)` will add the term: $0.7\ a_{1\alpha} a_{3\beta}^{\dagger} a_{0\alpha}^{\dagger}$.

Most of the time, there can be multiple terms differ only in the operator indices and coefficients. For this case, One can invoke `add_term` only once. So `b.add_term("CD", [1, 3, 3, 1, 2, 2, 2, 4], [0.7, 0.6, 0.5, 0.4])`

is equivalent to

```
b.add_term("CD", [1, 3], 0.7)
b.add_term("CD", [3, 1], 0.6)
b.add_term("CD", [2, 2], 0.5)
b.add_term("CD", [2, 4], 0.4)
```

`b.finalize` will do some necessary operator reordering and simplification and `driver.get_mpo` will create an optimal MPO for the given Hamiltonian. So we can set the MPO for the above Hubbard Hamiltonian using the following:

[4]:
```
t = 1
U = 2

b = driver.expr_builder()

# hopping term
b.add_term("cd", np.array([[[i, i + 1], [i + 1, i]] for i in range(L - 1)]).
↪flatten(), -t)
b.add_term("CD", np.array([[[i, i + 1], [i + 1, i]] for i in range(L - 1)]).
↪flatten(), -t)
```

(continues on next page)

(continued from previous page)

```
# onsite term
b.add_term("cdCD", np.array([[i, ] * 4 for i in range(L)]).flatten(), U)

mpo = driver.get_mpo(b.finalize(), iprint=2)
```

```
Build MPO | Nsites =     8 | Nterms =         36 | Algorithm = FastBIP | Cutoff = 1.
↪00e-14
 Site =     0 /     8 .. Mmpo =     6 DW = 0.00e+00 NNZ =        6 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.004
 Site =     1 /     8 .. Mmpo =     6 DW = 0.00e+00 NNZ =       11 SPT = 0.6944 Tmvc␣
↪= 0.000 T = 0.002
 Site =     2 /     8 .. Mmpo =     6 DW = 0.00e+00 NNZ =       11 SPT = 0.6944 Tmvc␣
↪= 0.000 T = 0.002
 Site =     3 /     8 .. Mmpo =     6 DW = 0.00e+00 NNZ =       11 SPT = 0.6944 Tmvc␣
↪= 0.000 T = 0.002
 Site =     4 /     8 .. Mmpo =     6 DW = 0.00e+00 NNZ =       11 SPT = 0.6944 Tmvc␣
↪= 0.000 T = 0.002
 Site =     5 /     8 .. Mmpo =     6 DW = 0.00e+00 NNZ =       11 SPT = 0.6944 Tmvc␣
↪= 0.000 T = 0.002
 Site =     6 /     8 .. Mmpo =     6 DW = 0.00e+00 NNZ =       11 SPT = 0.6944 Tmvc␣
↪= 0.000 T = 0.002
 Site =     7 /     8 .. Mmpo =     1 DW = 0.00e+00 NNZ =        6 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.002
Ttotal =      0.019 Tmvc-total = 0.000 MPO bond dimension =     6 MaxDW = 0.00e+00
NNZ =           78 SIZE =          228 SPT = 0.6579

Rank =     0 Ttotal =      0.047 MPO method = FastBipartite bond dimension =       6␣
↪NNZ =           78 SIZE =          228 SPT = 0.6579
```

Note that the above method should allow the user to define arbitrary second-quantized fermionic (particle-number conserving) Hamiltonians (with possibly long-range and high-order terms).

Run DMRG

Finally, we can run DMRG to find the ground state energy.

We first use `driver.get_random_mps` to create an initial guess for the wavefunction (MPS).

- `tag` is part of the scratch file name for this MPS. If there are multiple MPS objects, their tags must be all different.
- `bond_dim` is the bond dimension of the initial guess MPS. The final optimized MPS may have a larger bond dimension.
- `nroots` is the number of roots. If `nroots > 1`, it will compute the ground state and `nroots - 1` low-energy excited states.

The DMRG algorithm consists of many "sweeps". The bond dimension of the MPS can increase after several sweeps. After a few tens of the sweeps, the MPS gradually converges to the ground state if `nroots == 1`. To prevent stuck in local minima, we may add noise in most sweeps, but the last sweep will not have any noise (namely, `noises[-1] == 0`) so that the accuracy of the final printed energy is not affected. In each sweep, the DMRG algorithm will optimize L tensors in the MPS, one by one. For optimizing each tensor, an effective Hamiltonian will be created and the Davidson algorithm is used to sovle the eigenvalue problem of this effective Hamiltonian. The eigenvalue (energy) of this effective Hamiltonian is the same as the eigenvalue of the full many-body Hamiltonian (if the bond dimension of the MPS is big enough).

Therefore, to run a DMRG calculation, we need to set up a "sweep schedule", namely, we need to set for each sweep: (a) the MPS bond dimension used in this sweep; (b) the noise used in this sweep; and (c) the Davidson algorithm convergence threshold for this sweep. Typically, these parameters should be changed every 4 to 5 sweeps (for Hubbard model with several hundred sites they may be changed every 30 to 50 sweeps), and the MPS bond dimension increases, while the noise and Davidson algorithm convergence decrease. The Davidson algorithm convergence cannot be set to zero.

`bond_dims`, `noises`, and `thrds` are three lists. The first number in each list is the parameter used in the first sweep, the second number is for the second sweep, etc.

We call `driver.dmrg` to execute the DMRG algorithm.

- `mpo` is the Hamiltonian represented as an MPO.
- `ket` is the initial guess of the MPS. After DMRG finishes, the content of the MPS will be changed and it will be the optimized state.
- `n_sweeps` is the maximal number of sweeps.
- `bond_dims` is a list of integers indicating the MPS bond dimension for each sweep.
- `noises` is a list of real numbers indicating the noise for each sweep.
- `thrds` is a list of real numbers indicating the Davidson algorithm convergence threshold for each sweep.
- `cutoff` is a real number. Singular values below this number will be discarded during the optimization. Setting this to zero will not discard any small singular value in the MPS unless the MPS bond dimension becomes higher than the set value.
- `iprint` can be 0 (quiet), 1 (print energy after each sweep), or 2 (print energy after each iteration in each sweep).

[5]:
```
def run_dmrg(driver, mpo):
    ket = driver.get_random_mps(tag="KET", bond_dim=250, nroots=1)
    bond_dims = [250] * 4 + [500] * 4
    noises = [1e-4] * 4 + [1e-5] * 4 + [0]
    thrds = [1e-10] * 8
    return driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
        thrds=thrds, cutoff=0, iprint=1)
```

(continues on next page)

(continued from previous page)

```
energies = run_dmrg(driver, mpo)
print('DMRG energy = %20.15f' % energies)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.288 | E =      -6.2256341447 | DW = 5.89e-16

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.403 | E =      -6.2256341447 | DE = 2.22e-14 | DW = 7.51e-16

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.519 | E =      -6.2256341447 | DE = -7.99e-15 | DW = 1.19e-16

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.636 | E =      -6.2256341447 | DE = 8.88e-16 | DW = 1.18e-16

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.868 | E =      -6.2256341447 | DE = 5.33e-15 | DW = 9.04e-29

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.130 | E =      -6.2256341447 | DE = -1.78e-15 | DW = 1.10e-28

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.382 | E =      -6.2256341447 | DE = -8.88e-16 | DW = 1.27e-28

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.619 | E =      -6.2256341447 | DE = 8.88e-16 | DW = 1.32e-28

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      1.844 | E =      -6.2256341447 | DE = 2.66e-15 | DW = 2.76e-51

DMRG energy =   -6.225634144662398
```

Note that in the outputs, `DE` represents the energy difference between two sweeps. `DW` is the maximal discarded weight (sum of discarded eigenvalues of the density matrix) during the current sweep.

The remaining part of this docuementation introduces some other alternative symmetry to solve the same problem.

4.7.3 The SU2 Mode

Initialization

In this section, we try to solve the same problem using the SU(2) symmetry, which has a lower computational cost. Now `spin` represents the total spin (instead of the projected spin).

[6]:
```
TWOS = 0

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SU2, n_threads=4)
driver.initialize_system(n_sites=L, n_elec=N, spin=TWOS)
```

Build Hamiltonian

We need to first do some rewriting of the Hamiltonian to use it with the SU(2) symmetry. This is mainly because operators like $a^\dagger_\alpha$ do not conserve the total spin quantum number. We have to write the Hamiltonian in terms of only the total spin S preserving elementary operators.

First, note that

$$n_{i\uparrow}n_{i\downarrow} = a^\dagger_{i\alpha}a_{i\alpha}a^\dagger_{i\beta}a_{i\beta} = a^\dagger_{i\alpha}a^\dagger_{i\beta}a_{i\beta}a_{i\alpha} = a^\dagger_{i\beta}a^\dagger_{i\alpha}a_{i\alpha}a_{i\beta} = \frac{1}{2}\sum_\sigma\sum_{\sigma'\neq\sigma} a^\dagger_{i\sigma}a^\dagger_{i\sigma'}a_{i\sigma'}a_{i\sigma} = \frac{1}{2}\sum_{\sigma\sigma'} a^\dagger_{i\sigma}a^\dagger_{i\sigma'}a_{i\sigma'}a_{i\sigma}$$

We can rewrite the Hubbard Hamiltonian as

$$\hat{H} = -t\sum_{\langle ij\rangle,\sigma} a^\dagger_{i\sigma}a_{j\sigma} + \frac{U}{2}\sum_{i,\sigma\sigma'} a^\dagger_{i\sigma}a^\dagger_{i\sigma'}a_{i\sigma'}a_{i\sigma}$$

Then we introduce the spin tensor operators which are spin-adaptation of elementary operators for alpha and beta spins. The superscript or subscript $[S]$ indicates the total spin quantum number of each spin tensor operator. Define

$$\left(a^\dagger_p\right)^{[1/2]} := \begin{pmatrix} a^\dagger_{p\alpha} \\ a^\dagger_{p\beta} \end{pmatrix}^{[1/2]} \qquad \left(a_p\right)^{[1/2]} := \begin{pmatrix} -a_{p\beta} \\ a_{p\alpha} \end{pmatrix}^{[1/2]}$$

Because the total spin is a non-Abelian symmetry, we have the following notes:

- The tensor product (coupling) of two spins may generate more than one possible resulting spins. For example, the tensor product between $S = 1/2$ (doublet) and $S = 1/2$ (doublet) can generate $S = 0$ (singlet) and $S = 1$ (triplet). Therefore, when we combine two spin operators $\left(a^\dagger_p\right)^{[1/2]}$ and $\left(a_q\right)^{[1/2]}$, we have to indicate which resulting total spin is desired. So we need the following notation to diffentiate the two cases:

$$\left(a^\dagger_p\right)^{[1/2]} \otimes_{[0]} \left(a_q\right)^{[1/2]} \quad \text{(singlet product)}$$
$$\left(a^\dagger_p\right)^{[1/2]} \otimes_{[1]} \left(a_q\right)^{[1/2]} \quad \text{(triplet product)}$$

- There is no associative law. Namely, in general, $(S_1 \otimes S_2) \otimes S_3 \neq S_1 \otimes (S_2 \otimes S_3)$. The two products are connected by the spin recoupling, and the coefficients between the two products can be computed using Clebsch-Gordan factors. So when writing a string of spin tensor operators, we have to use parenthesis to indicate the coupling order of the operators.

Using spin tensor operators, we can rewrite the Hubbard Hamiltonian as (some derivation details can be found in https://block2.readthedocs.io/en/latest/theory/su2.html)

$$\hat{H}^{[0]} = -\sqrt{2}\, t \sum_{\langle ij \rangle} \left(a_i^\dagger\right)^{[1/2]} \otimes_{[0]} \left(a_j\right)^{[1/2]} + U \sum_i \left(\left(a_i^\dagger\right)^{[1/2]} \otimes_{[1/2]} \left(\left(a_i^\dagger\right)^{[1/2]} \otimes_{[0]} \left(a_i\right)^{[1/2]} \right) \right) \otimes_{[0]} \left(a_i\right)^{[1/2]}$$

Next, we have to introduce a string notation to represent the above formula. We can use `C` and `D` for $a^\dagger$ and a, repsectively. Then we use `+` to connect operators (namely, `+` represents the tensor product). We use `()` to indicate the coupling (associative) order. We add a number (2 times the total spin) after each `)` to indicate the resulting spin number of the tensor product. So, for example, `(C+D)0` represents the singlet product between $a^\dagger$ and a, and `(C+D)2` represents the triplet product.

As a result, we can set the MPO for the above SU(2) Hubbard Hamiltonian using the following:

[7]:
```
t = 1
U = 2

b = driver.expr_builder()

# hopping term
b.add_term("(C+D)0", np.array([[[i, i + 1], [i + 1, i]] for i in range(L - 1)]).
↪flatten(),
    np.sqrt(2) * -t)

# onsite term
b.add_term("((C+(C+D)0)1+D)0", np.array([[i, ] * 4 for i in range(L)]).flatten(), U)

mpo = driver.get_mpo(b.finalize(), iprint=2)
```

```
Build MPO | Nsites =     8 | Nterms =         30 | Algorithm = FastBIP | Cutoff = 1.
↪00e-14
 Site =     0 /     8 .. Mmpo =     4 DW = 0.00e+00 NNZ =        4 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.002
 Site =     1 /     8 .. Mmpo =     4 DW = 0.00e+00 NNZ =        7 SPT = 0.5625 Tmvc␣
↪= 0.000 T = 0.002
 Site =     2 /     8 .. Mmpo =     4 DW = 0.00e+00 NNZ =        7 SPT = 0.5625 Tmvc␣
↪= 0.000 T = 0.003
 Site =     3 /     8 .. Mmpo =     4 DW = 0.00e+00 NNZ =        7 SPT = 0.5625 Tmvc␣
↪= 0.001 T = 0.004
 Site =     4 /     8 .. Mmpo =     4 DW = 0.00e+00 NNZ =        7 SPT = 0.5625 Tmvc␣
↪= 0.000 T = 0.002
 Site =     5 /     8 .. Mmpo =     4 DW = 0.00e+00 NNZ =        7 SPT = 0.5625 Tmvc␣
↪= 0.000 T = 0.002
 Site =     6 /     8 .. Mmpo =     4 DW = 0.00e+00 NNZ =        7 SPT = 0.5625 Tmvc␣
↪= 0.000 T = 0.002
 Site =     7 /     8 .. Mmpo =     1 DW = 0.00e+00 NNZ =        4 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.002
Ttotal =      0.020 Tmvc-total = 0.001 MPO bond dimension =     4 MaxDW = 0.00e+00
```

(continues on next page)

(continued from previous page)

```
NNZ =           50 SIZE =          104 SPT = 0.5192

Rank =      0 Ttotal =      0.045 MPO method = FastBipartite bond dimension =       4␣
↪NNZ =           50 SIZE =          104 SPT = 0.5192
```

Run DMRG

Finally, we can run DMRG as normal:

[8]:
```
energies = run_dmrg(driver, mpo)
print('DMRG energy = %20.15f' % energies)
```

```

Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.122 | E =      -6.2256341447 | DW = 1.62e-31

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.193 | E =      -6.2256341447 | DE = -1.78e-15 | DW = 1.34e-31

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.259 | E =      -6.2256341447 | DE = 8.88e-16 | DW = 2.12e-32

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.325 | E =      -6.2256341447 | DE = 1.78e-15 | DW = 2.46e-31

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.396 | E =      -6.2256341447 | DE = -7.99e-15 | DW = 0.00e+00

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.466 | E =      -6.2256341447 | DE = 1.78e-15 | DW = 0.00e+00

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.587 | E =      -6.2256341447 | DE = 4.44e-15 | DW = 0.00e+00

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.703 | E =      -6.2256341447 | DE = -5.33e-15 | DW = 0.00e+00
```

(continues on next page)

(continued from previous page)

```
Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      0.796 | E =      -6.2256341447 | DE = 1.78e-15 | DW = 0.00e+00

DMRG energy =   -6.225634144676156
```

4.7.4 The SGF Mode

In this section, we try to solve the same Hubbard problem using only particle number symmetry. The number of sites will now be the number of spin orbitals (which is equal to the number of qubits used in the next section), which is two times the number of the original (spatial) sites.

[9]:
```
driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SGF, n_threads=4)
driver.initialize_system(n_sites=L * 2, n_elec=N, heis_twos=1)
```

Build Hamiltonian

Now we first split every Hubbard site into the alpha (odd) and beta (even) sites (next to each other). We have

$$\hat{H} = -t \sum_{i=1}^{L-1} \left(a_{2i}^\dagger a_{2i+2} + a_{2i+2}^\dagger a_{2i} + a_{2i-1}^\dagger a_{2i+1} + a_{2i+1}^\dagger a_{2i-1} \right) + U \sum_{i=1}^{L} a_{2i-1}^\dagger a_{2i-1} a_{2i}^\dagger a_{2i}$$

We use characters `C` and `D` to represent $a^\dagger$ and a, respectively.

So we can set the MPO for the above Hubbard Hamiltonian using the following:

[10]:
```
t = 1
U = 2

b = driver.expr_builder()

# hopping term
b.add_term("CD", np.array([[i, i + 2], [i + 2, i]] for i in range(2 * (L - 1))]).
↪flatten(), -t)

# onsite term
b.add_term("CDCD", np.array([[2 * i, 2 * i, 2 * i + 1, 2 * i + 1] for i in␣
↪range(L)]).flatten(), U)

mpo = driver.get_mpo(b.finalize(), iprint=2)
```

```
Build MPO | Nsites =    16 | Nterms =         36 | Algorithm = FastBIP | Cutoff = 1.
↪00e-14
```

(continues on next page)

(continued from previous page)

```
 Site =     0 /    16 .. Mmpo =     4 DW = 0.00e+00 NNZ =        4 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.004
 Site =     1 /    16 .. Mmpo =     6 DW = 0.00e+00 NNZ =        6 SPT = 0.7500 Tmvc␣
↪= 0.000 T = 0.004
 Site =     2 /    16 .. Mmpo =     7 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.003
 Site =     3 /    16 .. Mmpo =     6 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.004
 Site =     4 /    16 .. Mmpo =     7 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.002
 Site =     5 /    16 .. Mmpo =     6 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.002
 Site =     6 /    16 .. Mmpo =     7 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.002
 Site =     7 /    16 .. Mmpo =     6 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.006
 Site =     8 /    16 .. Mmpo =     7 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.002
 Site =     9 /    16 .. Mmpo =     6 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.002
 Site =    10 /    16 .. Mmpo =     7 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.002
 Site =    11 /    16 .. Mmpo =     6 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.002
 Site =    12 /    16 .. Mmpo =     7 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.002
 Site =    13 /    16 .. Mmpo =     6 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.002
 Site =    14 /    16 .. Mmpo =     4 DW = 0.00e+00 NNZ =        6 SPT = 0.7500 Tmvc␣
↪= 0.000 T = 0.002
 Site =    15 /    16 .. Mmpo =     1 DW = 0.00e+00 NNZ =        4 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.002
Ttotal =      0.044 Tmvc-total = 0.000 MPO bond dimension =     7 MaxDW = 0.00e+00
NNZ =          128 SIZE =          560 SPT = 0.7714

Rank =     0 Ttotal =      0.095 MPO method = FastBipartite bond dimension =       7␣
↪NNZ =          128 SIZE =          560 SPT = 0.7714
```

Run DMRG

Finally, we can run DMRG as normal:

[11]:
```
energies = run_dmrg(driver, mpo)
print('DMRG energy = %20.15f' % energies)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.921 | E =      -6.2256341447 | DW = 2.51e-15

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.099 | E =      -6.2256341447 | DE = -8.88e-15 | DW = 2.67e-15

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.272 | E =      -6.2256341447 | DE = 1.78e-15 | DW = 2.33e-15

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.444 | E =      -6.2256341447 | DE = 1.07e-14 | DW = 2.38e-15

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.853 | E =      -6.2256341447 | DE = -8.88e-16 | DW = 9.93e-33

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.250 | E =      -6.2256341447 | DE = -1.78e-15 | DW = 4.03e-32

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.697 | E =      -6.2256341447 | DE = -8.88e-16 | DW = 1.85e-32

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      3.094 | E =      -6.2256341447 | DE = -1.78e-15 | DW = 5.42e-32

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      3.330 | E =      -6.2256341447 | DE = 3.55e-15 | DW = 0.00e+00

DMRG energy =   -6.225634144669716
```

4.7.5 The SGB Mode

Initialization

In this section, we try to solve the same Hubbard problem by first transfroming the model into a qubit (spin) model. The number of sites will now be the number of qubits (spins), which is two times the number of the original sites.

The parameter `heis_twos` represents two times the spin in each site. Namely, for $S = 1/2$ Heisenberg model, `heis_twos = 1`, for $S = 1$ Heisenberg model, `heis_twos = 2`, etc.

[12]:
```
driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SGB, n_threads=4)
driver.initialize_system(n_sites=L * 2, n_elec=N, heis_twos=1)
```

Build Hamiltonian

Now we first need to do the Jordan-wigner transform. We split every Hubbard site into the alpha (odd) and beta (even) sites (next to each other). We have

$$\hat{H} = -t \sum_{i=1}^{L-1} \left(S_{2i}^{+} S_{2i}^{z} S_{2i+1}^{z} S_{2i+2}^{-} - S_{2i}^{-} S_{2i}^{z} S_{2i+1}^{z} S_{2i+2}^{+} + S_{2i-1}^{+} S_{2i-1}^{z} S_{2i}^{z} S_{2i+1}^{-} - S_{2i-1}^{-} S_{2i-1}^{z} S_{2i}^{z} S_{2i+1}^{+} \right) + U \sum_{i=1}^{L} S_{2i-1}^{+} S_{2i-1}^{-}$$

We use charactors `P` (plus), `M` (minus), and `Z` to represent operators S^+, S^- and $\frac{1}{2}S^z$, respectively. Note that we assume $S^z|\alpha\rangle = 1 \cdot |\alpha\rangle$ and $S^z|\beta\rangle = -1 \cdot |\beta\rangle$ here.

So we can set the MPO for the above Hubbard Hamiltonian using the following:

[13]:
```
t = 1
U = 2

b = driver.expr_builder()

# hopping term
b.add_term("PZZM",
    np.array([[i, i, i + 1, i + 2] for i in range(2 * (L - 1))]).flatten(),
    [-t * 4] * 2 * (L - 1))
b.add_term("MZZP",
    np.array([[i, i, i + 1, i + 2] for i in range(2 * (L - 1))]).flatten(),
    [+t * 4] * 2 * (L - 1))

# onsite term
b.add_term("PMPM",
    np.array([[i, i] for i in range(2 * L)]).flatten(),
    [U] * 2 * L)

mpo = driver.get_mpo(b.finalize(), iprint=2)
```

```
Build MPO | Nsites =    16 | Nterms =         36 | Algorithm = FastBIP | Cutoff = 1.
↪00e-14
 Site =     0 /    16 .. Mmpo =     4 DW = 0.00e+00 NNZ =        4 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.002
 Site =     1 /    16 .. Mmpo =     6 DW = 0.00e+00 NNZ =        6 SPT = 0.7500 Tmvc␣
↪= 0.000 T = 0.003
 Site =     2 /    16 .. Mmpo =     7 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.003
 Site =     3 /    16 .. Mmpo =     6 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.002
 Site =     4 /    16 .. Mmpo =     7 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.002
 Site =     5 /    16 .. Mmpo =     6 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.003
 Site =     6 /    16 .. Mmpo =     7 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.002
 Site =     7 /    16 .. Mmpo =     6 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.002
 Site =     8 /    16 .. Mmpo =     7 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.004
 Site =     9 /    16 .. Mmpo =     6 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.002
 Site =    10 /    16 .. Mmpo =     7 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.003
 Site =    11 /    16 .. Mmpo =     6 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.002
 Site =    12 /    16 .. Mmpo =     7 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.002
 Site =    13 /    16 .. Mmpo =     6 DW = 0.00e+00 NNZ =        9 SPT = 0.7857 Tmvc␣
↪= 0.000 T = 0.002
 Site =    14 /    16 .. Mmpo =     4 DW = 0.00e+00 NNZ =        6 SPT = 0.7500 Tmvc␣
↪= 0.000 T = 0.002
 Site =    15 /    16 .. Mmpo =     1 DW = 0.00e+00 NNZ =        4 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.002
Ttotal =      0.040 Tmvc-total = 0.000 MPO bond dimension =     7 MaxDW = 0.00e+00
NNZ =          128 SIZE =          560 SPT = 0.7714

Rank =     0 Ttotal =      0.087 MPO method = FastBipartite bond dimension =       7␣
↪NNZ =          128 SIZE =          560 SPT = 0.7714
```

Run DMRG

Finally, we can run DMRG as normal:

[14]:
```
energies = run_dmrg(driver, mpo)
print('DMRG energy = %20.15f' % energies)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.925 | E =      -6.2256341447 | DW = 4.38e-15

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.078 | E =      -6.2256341447 | DE = 2.13e-14 | DW = 4.26e-15

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.236 | E =      -6.2256341447 | DE = 0.00e+00 | DW = 4.25e-15

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.432 | E =      -6.2256341447 | DE = 5.33e-15 | DW = 3.95e-15

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      1.785 | E =      -6.2256341447 | DE = 0.00e+00 | DW = 2.54e-32

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.172 | E =      -6.2256341447 | DE = 4.44e-15 | DW = 7.02e-32

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.570 | E =      -6.2256341447 | DE = 0.00e+00 | DW = 1.86e-32

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      2.929 | E =      -6.2256341447 | DE = 1.78e-15 | DW = 5.74e-32

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      3.157 | E =      -6.2256341447 | DE = -3.55e-15 | DW = 0.00e+00

DMRG energy =   -6.225634144672112
```

4.7.6 The PHSU2 Mode

Let

$$\left(E_p\right)^{[1/2]} := \begin{pmatrix} a_{p,\uparrow}^\dagger \\ (-1)^p a_{p,\downarrow} \end{pmatrix}^{[1/2]} \quad \left(F_p\right)^{[1/2]} := \begin{pmatrix} -(-1)^p a_{p,\downarrow}^\dagger \\ a_{p,\uparrow} \end{pmatrix}^{[1/2]}$$

We have (assuming p and q differ by 1)

$$\begin{aligned} \left(E_p\right)^{[1/2]} \otimes_{[0]} \left(F_q\right)^{[1/2]} &= \begin{pmatrix} a_{p,\uparrow}^\dagger \\ (-1)^p a_{p,\downarrow} \end{pmatrix}^{[1/2]} \otimes_{[0]} \begin{pmatrix} -(-1)^q a_{q,\downarrow}^\dagger \\ a_{q,\uparrow} \end{pmatrix}^{[1/2]} \\ &= \frac{1}{\sqrt{2}} \left(a_{p,\uparrow}^\dagger a_{q,\uparrow} - a_{p,\downarrow} a_{q,\downarrow}^\dagger\right) = \frac{1}{\sqrt{2}} \left(a_{p,\uparrow}^\dagger a_{q,\uparrow} + a_{q,\downarrow}^\dagger a_{p,\downarrow}\right) \end{aligned}$$

So

$$\begin{aligned} &\left(E_p\right)^{[1/2]} \otimes_{[0]} \left(F_{p+1}\right)^{[1/2]} + \left(E_{p+1}\right)^{[1/2]} \otimes_{[0]} \left(F_p\right)^{[1/2]} \\ =& \frac{1}{\sqrt{2}} \left(a_{p,\uparrow}^\dagger a_{p+1,\uparrow} + a_{p+1,\downarrow}^\dagger a_{p,\downarrow}\right) + \frac{1}{\sqrt{2}} \left(a_{p+1,\uparrow}^\dagger a_{p,\uparrow} + a_{p,\downarrow}^\dagger a_{p+1,\downarrow}\right) \\ =& \frac{1}{\sqrt{2}} \left(a_{p,\uparrow}^\dagger a_{p+1,\uparrow} + a_{p,\downarrow}^\dagger a_{p+1,\downarrow} + a_{p+1,\uparrow}^\dagger a_{p,\uparrow} + a_{p+1,\downarrow}^\dagger a_{p,\downarrow}\right) \\ =& \frac{1}{\sqrt{2}} \sum_\sigma \left(a_{p,\sigma}^\dagger a_{p+1,\sigma} + \text{H.c.}\right) \end{aligned}$$

$$-t \sum_{\langle i,j\rangle,\sigma} \left(c_{i,\sigma}^\dagger c_{j,\sigma} + \text{H.c.}\right) = -\sqrt{2}t \sum_{\langle i,j\rangle} \left[\left(E_i\right)^{[1/2]} \otimes_{[0]} \left(F_j\right)^{[1/2]} + \left(E_j\right)^{[1/2]} \otimes_{[0]} \left(F_i\right)^{[1/2]}\right]$$

Now consider the onsite term

$$\begin{aligned} \left(E_p\right)^{[1/2]} \otimes_{[0]} \left(F_p\right)^{[1/2]} &= \begin{pmatrix} a_{p,\uparrow}^\dagger \\ (-1)^p a_{p,\downarrow} \end{pmatrix}^{[1/2]} \otimes_{[0]} \begin{pmatrix} -(-1)^p a_{p,\downarrow}^\dagger \\ a_{p,\uparrow} \end{pmatrix}^{[1/2]} \\ &= \frac{1}{\sqrt{2}} \left(a_{p,\uparrow}^\dagger a_{p,\uparrow} + a_{p,\downarrow} a_{p,\downarrow}^\dagger\right) = \frac{1}{\sqrt{2}} \left(a_{p,\uparrow}^\dagger a_{p,\uparrow} - a_{p,\downarrow}^\dagger a_{p,\downarrow} + 1\right) \\ \left(E_p\right)^{[1/2]} \otimes_{[0]} \left(E_p\right)^{[1/2]} &= \begin{pmatrix} a_{p,\uparrow}^\dagger \\ (-1)^p a_{p,\downarrow} \end{pmatrix}^{[1/2]} \otimes_{[0]} \begin{pmatrix} a_{p,\uparrow}^\dagger \\ (-1)^p a_{p,\downarrow} \end{pmatrix}^{[1/2]} \\ &= \frac{1}{\sqrt{2}} \left((-1)^p a_{p,\uparrow}^\dagger a_{p,\downarrow} - (-1)^p a_{p,\downarrow} a_{p,\uparrow}^\dagger\right) = (-1)^p \sqrt{2} a_{p,\uparrow}^\dagger a_{p,\downarrow} \\ \left(F_p\right)^{[1/2]} \otimes_{[0]} \left(F_p\right)^{[1/2]} &= \begin{pmatrix} -(-1)^p a_{p,\downarrow}^\dagger \\ a_{p,\uparrow} \end{pmatrix}^{[1/2]} \otimes_{[0]} \begin{pmatrix} -(-1)^p a_{p,\downarrow}^\dagger \\ a_{p,\uparrow} \end{pmatrix}^{[1/2]} \\ &= \frac{1}{\sqrt{2}} \left(-(-1)^p a_{p,\downarrow}^\dagger a_{p,\uparrow} + (-1)^p a_{p,\uparrow}\right) = -(-1)^p \sqrt{2} a_{p,\downarrow}^\dagger a_{p,\uparrow} \end{aligned}$$

So the product of above terms

$$\begin{aligned} &\left(\left(E_p\right)^{[1/2]} \otimes_{[0]} \left(E_p\right)^{[1/2]}\right) \otimes_{[0]} \left(\left(F_p\right)^{[1/2]} \otimes_{[0]} \left(F_p\right)^{[1/2]}\right) \\ =& -2 a_{p,\uparrow}^\dagger a_{p,\downarrow} a_{p,\downarrow}^\dagger a_{p,\uparrow} = -2 a_{p,\uparrow}^\dagger a_{p,\uparrow} a_{p,\downarrow} a_{p,\downarrow}^\dagger = 2 a_{p,\uparrow}^\dagger a_{p,\uparrow} a_{p,\downarrow}^\dagger a_{p,\downarrow} - 2 a_{p,\uparrow}^\dagger a_{p,\uparrow} = 2 n_{p\uparrow} n_{p\downarrow} - 2 n_{p\uparrow} \end{aligned}$$

[15]:
```
t = 1
U = 2
n_elec = L

driver = DMRGDriver(scratch='./tmp', symm_type=SymmetryTypes.SAnyPHSU2, n_threads=4)
driver.initialize_system(n_sites=L, n_elec=n_elec, spin=0)

b = driver.expr_builder()
b.add_term("(E+F)0", [g for i in range(L - 1) for g in [i, i + 1]], 2 ** 0.5)
b.add_term("(F+E)0", [g for i in range(L - 1) for g in [i, i + 1]], 2 ** 0.5)
b.add_term("((E+E)0+(F+F)0)0", np.array([i for i in range(L) for _ in range(4)]), 0.
↪5 * U)
b.add_const(n_elec * U / 2)
mpo = driver.get_mpo(b.finalize(adjust_order=False), iprint=2)

ket = driver.get_random_mps(tag="KET", bond_dim=250, nroots=1)
bond_dims = [250] * 4 + [500] * 4
noises = [1e-4] * 4 + [1e-5] * 4 + [0]
thrds = [1e-10] * 8
energies = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,␣
↪thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energies)
```

```
Build MPO | Nsites =     8 | Nterms =         22 | Algorithm = FastBIP | Cutoff = 1.
↪00e-14
 Site =     0 /     8 .. Mmpo =     4 DW = 0.00e+00 NNZ =        4 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.005
 Site =     1 /     8 .. Mmpo =     4 DW = 0.00e+00 NNZ =        7 SPT = 0.5625 Tmvc␣
↪= 0.000 T = 0.003
 Site =     2 /     8 .. Mmpo =     4 DW = 0.00e+00 NNZ =        7 SPT = 0.5625 Tmvc␣
↪= 0.000 T = 0.003
 Site =     3 /     8 .. Mmpo =     4 DW = 0.00e+00 NNZ =        7 SPT = 0.5625 Tmvc␣
↪= 0.000 T = 0.005
 Site =     4 /     8 .. Mmpo =     4 DW = 0.00e+00 NNZ =        7 SPT = 0.5625 Tmvc␣
↪= 0.000 T = 0.005
 Site =     5 /     8 .. Mmpo =     4 DW = 0.00e+00 NNZ =        7 SPT = 0.5625 Tmvc␣
↪= 0.000 T = 0.003
 Site =     6 /     8 .. Mmpo =     4 DW = 0.00e+00 NNZ =        7 SPT = 0.5625 Tmvc␣
↪= 0.000 T = 0.003
 Site =     7 /     8 .. Mmpo =     1 DW = 0.00e+00 NNZ =        4 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.003
Ttotal =      0.028 Tmvc-total = 0.000 MPO bond dimension =     4 MaxDW = 0.00e+00
NNZ =           50 SIZE =          104 SPT = 0.5192

Rank =     0 Ttotal =      0.069 MPO method = FastBipartite bond dimension =       4␣
↪NNZ =           50 SIZE =          104 SPT = 0.5192
```

(continues on next page)

(continued from previous page)

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.144 | E =      -6.2256341447 | DW = 2.13e-20

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.187 | E =      -6.2256341447 | DE = 4.26e-14 | DW = 2.06e-20

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.232 | E =      -6.2256341447 | DE = -8.88e-15 | DW = 1.49e-20

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.274 | E =      -6.2256341447 | DE = 1.78e-15 | DW = 2.86e-20

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.317 | E =      -6.2256341447 | DE = -3.55e-15 | DW = 1.51e-20

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.359 | E =      -6.2256341447 | DE = 0.00e+00 | DW = 1.15e-20

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.402 | E =      -6.2256341447 | DE = 1.78e-15 | DW = 1.96e-20

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.444 | E =      -6.2256341447 | DE = 7.11e-15 | DW = 1.37e-20

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      0.478 | E =      -6.2256341447 | DE = -2.84e-14 | DW = 1.94e-20

DMRG energy =   -6.225634144668852
```

4.7.7 The S04 Mode

Similarly, we can consider the $\mathrm{SU}(2) \times \mathrm{SU}(2)$ operator G, where the row represents the pseudo spin symmetry and the column represents the spin symmetry.

$$(G_p)^{[1/2,1/2]} := \begin{pmatrix} a^\dagger_{p,\uparrow} & a^\dagger_{p,\downarrow} \\ (-1)^p a_{p,\downarrow} & -(-1)^p a_{p,\uparrow} \end{pmatrix}^{[1/2,1/2]}$$

So the components can be represented as

$$\begin{pmatrix} [1/2,1/2] & [1/2,-1/2] \\ [-1/2,1/2] & [-1/2,-1/2] \end{pmatrix}^{[1/2,1/2]}$$

[16]:
```
t = 1
U = 2
n_elec = L

driver = DMRGDriver(scratch='./tmp', symm_type=SymmetryTypes.SAnySO4, n_threads=4)
driver.initialize_system(n_sites=L, n_elec=n_elec, spin=0)

b = driver.expr_builder()
b.add_term("(G[1,1]+G[1,1])[0,0]",
    [g for i in range(L - 1) for g in [i, i + 1]], [2.0 if i % 2 == 0 else -2.0 for␣
↪i in range(L - 1)])
b.add_term("((G[1,1]+G[1,1])[0,2]+(G[1,1]+G[1,1])[0,2])[0,0]",
    np.array([i for i in range(L) for _ in range(4)]), U / (2 * 3 ** 0.5))
b.add_const(n_elec * U / 2)
mpo = driver.get_mpo(b.finalize(adjust_order=False), iprint=2)

ket = driver.get_random_mps(tag="KET", bond_dim=250, nroots=1)
bond_dims = [250] * 4 + [500] * 4
noises = [1e-4] * 4 + [1e-5] * 4 + [0]
thrds = [1e-10] * 8
energies = driver.dmrg(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,␣
↪thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energies)
```

```
Build MPO | Nsites =     8 | Nterms =         15 | Algorithm = FastBIP | Cutoff = 1.
↪00e-14
 Site =     0 /     8 .. Mmpo =     3 DW = 0.00e+00 NNZ =        3 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.004
 Site =     1 /     8 .. Mmpo =     3 DW = 0.00e+00 NNZ =        5 SPT = 0.4444 Tmvc␣
↪= 0.000 T = 0.003
 Site =     2 /     8 .. Mmpo =     3 DW = 0.00e+00 NNZ =        5 SPT = 0.4444 Tmvc␣
↪= 0.000 T = 0.003
 Site =     3 /     8 .. Mmpo =     3 DW = 0.00e+00 NNZ =        5 SPT = 0.4444 Tmvc␣
```

(continues on next page)

(continued from previous page)

```
↪= 0.000 T = 0.003
 Site =     4 /     8 .. Mmpo =     3 DW = 0.00e+00 NNZ =        5 SPT = 0.4444 Tmvc␣
↪= 0.000 T = 0.003
 Site =     5 /     8 .. Mmpo =     3 DW = 0.00e+00 NNZ =        5 SPT = 0.4444 Tmvc␣
↪= 0.000 T = 0.003
 Site =     6 /     8 .. Mmpo =     3 DW = 0.00e+00 NNZ =        5 SPT = 0.4444 Tmvc␣
↪= 0.000 T = 0.003
 Site =     7 /     8 .. Mmpo =     1 DW = 0.00e+00 NNZ =        3 SPT = 0.0000 Tmvc␣
↪= 0.000 T = 0.003
Ttotal =      0.022 Tmvc-total = 0.000 MPO bond dimension =     3 MaxDW = 0.00e+00
NNZ =           36 SIZE =           60 SPT = 0.4000

Rank =     0 Ttotal =      0.047 MPO method = FastBipartite bond dimension =       3␣
↪NNZ =           36 SIZE =           60 SPT = 0.4000

Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.066 | E =     -6.2256341447 | DW = 1.22e-20

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.113 | E =     -6.2256341447 | DE = 3.02e-14 | DW = 2.80e-21

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.159 | E =     -6.2256341447 | DE = -3.55e-15 | DW = 7.31e-21

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-04 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.192 | E =     -6.2256341447 | DE = 8.88e-15 | DW = 9.99e-21

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.229 | E =     -6.2256341447 | DE = -1.95e-14 | DW = 1.22e-20

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.261 | E =     -6.2256341447 | DE = -1.78e-15 | DW = 5.33e-21

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
Time elapsed =      0.295 | E =     -6.2256341447 | DE = 3.55e-15 | DW = 8.67e-21

Sweep =    7 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-10
```

(continues on next page)

(continued from previous page)

```
Time elapsed =      0.333 | E =      -6.2256341447 | DE = -1.78e-15 | DW = 7.17e-21

Sweep =    8 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-09
Time elapsed =      0.367 | E =      -6.2256341447 | DE = -1.78e-15 | DW = 1.25e-20

DMRG energy =   -6.225634144673016
```

4.8 Heisenberg Model

[1]:
```
!pip install block2==0.5.2rc13 -qq --progress-bar off --extra-index-url=https://
↪block-hczhai.github.io/block2-preview/pypi/
```

4.8.1 Introduction

In this tutorial we explain how to solve the Heisenberg model using the python interface of `block2`.

First, we have to define the "site-spin" of the model. The parameter `heis_twos` represents two times the spin in each site. Namely, for $S = 1/2$ Heisenberg model, `heis_twos = 1`, for $S = 1$ Heisenberg model, `heis_twos = 2`, etc. Note that arbitrary non-negative half-integer and integer S can be supported in `block2`.

Second, we can solve the model using the SU2 symmetry (`SU2` mode) or U1 symmetry (`SGB` mode). The SU2 symmetry can be more efficient (for large `S`) and can generate states with well-defined total spin symmetry, but requires some additional rearrangement of the Hamiltonian terms.

4.8.2 The SGB Mode

The Hamiltonian is

$$\hat{H} = \sum_{i=1}^{L-1} \left(\frac{1}{2} S_i^+ S_{i+1}^- + \frac{1}{2} S_i^- S_{i+1}^+ + S_i^z S_{i+1}^z \right)$$

We can solve this model using the following code:

[2]:
```
import numpy as np
from pyblock2.driver.core import DMRGDriver, SymmetryTypes

L = 100
heis_twos = 1

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SGB, n_threads=4)
```

(continues on next page)

(continued from previous page)

```
driver.initialize_system(n_sites=L, heis_twos=heis_twos, heis_twosz=0)

b = driver.expr_builder()
for i in range(L - 1):
    b.add_term("PM", [i, i + 1], 0.5)
    b.add_term("MP", [i, i + 1], 0.5)
    b.add_term("ZZ", [i, i + 1], 1.0)
heis_mpo = driver.get_mpo(b.finalize(), iprint=0)

def run_dmrg(driver, mpo):
    ket = driver.get_random_mps(tag="KET", bond_dim=250, nroots=1)
    bond_dims = [250] * 4 + [500] * 4
    noises = [1e-5] * 4 + [1e-6] * 2 + [0]
    thrds = [1e-6] * 4 + [1e-8] * 4
    return driver.dmrg(
        mpo,
        ket,
        n_sweeps=8,
        bond_dims=bond_dims,
        noises=noises,
        thrds=thrds,
        cutoff=1E-24,
        iprint=1,
    )

energies = run_dmrg(driver, heis_mpo)
print('DMRG energy = %20.15f' % energies)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-06
Time elapsed =     32.844 | E =     -44.1201487994 | DW = 1.12e-09

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-06
Time elapsed =     38.582 | E =     -44.1277383773 | DE = -7.59e-03 | DW = 1.21e-13

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-06
Time elapsed =     41.882 | E =     -44.1277383773 | DE = -4.01e-12 | DW = 7.28e-14

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-06
Time elapsed =     44.918 | E =     -44.1277383773 | DE = -1.34e-12 | DW = 2.79e-14

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-06 |␣
```

(continues on next page)

(continued from previous page)

```
↪Dav threshold =  1.00e-08
Time elapsed =     60.141 | E =     -44.1277398826 | DE = -1.51e-06 | DW = 4.18e-17

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-06 |␣
↪Dav threshold =  1.00e-08
Time elapsed =     79.821 | E =     -44.1277398826 | DE = -1.92e-13 | DW = 3.84e-17

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-08
Time elapsed =     94.075 | E =     -44.1277398826 | DE = -1.35e-13 | DW = 1.61e-20

DMRG energy =  -44.127739882610513
```

4.8.3 The SU2 Mode

To solve the model in the SU2 mode, we define the following (triplet) spin tensor operator:

$$(T_p)^{[1]} := \begin{pmatrix} -S_p^+ \\ \sqrt{2} S_p^z \\ S_p^- \end{pmatrix}^{[1]}$$

Then we have

$$(T_p)^{[1]} \otimes_{[0]} (T_q)^{[1]} = \begin{pmatrix} -S_p^+ \\ \sqrt{2} S_p^z \\ S_p^- \end{pmatrix}^{[1]} \otimes_{[0]} \begin{pmatrix} -S_q^+ \\ \sqrt{2} S_q^z \\ S_q^- \end{pmatrix}^{[1]}$$

$$= \frac{1}{\sqrt{3}} \left(-S_p^+ S_q^- - S_p^- S_q^+ - 2 S_p^z S_q^z\right)^{[0]} = -\frac{2}{\sqrt{3}} \left(\tfrac{1}{2} S_p^+ S_q^- + \tfrac{1}{2} S_p^- S_q^+ + S_p^z S_q^z\right)^{[0]}$$

Note that in the above calculation, we have used the following CG factors:

[3]:
```
from block2 import SU2CG
print('<Jp=1,Jzp=+1; Jq=1,Jzq=-1|J=0,Jz=0> = ', SU2CG().cg(2, 2, 0, 2, -2, 0))
print('<Jp=1,Jzp=-1; Jq=1,Jzq=+1|J=0,Jz=0> = ', SU2CG().cg(2, 2, 0, -2, 2, 0))
print('<Jp=1,Jzp= 0; Jq=1,Jzq= 0|J=0,Jz=0> = ', SU2CG().cg(2, 2, 0, 0, 0, 0))
print(1 / 3 ** 0.5)
```

```
<Jp=1,Jzp=+1; Jq=1,Jzq=-1|J=0,Jz=0> =  0.5773502691896257
<Jp=1,Jzp=-1; Jq=1,Jzq=+1|J=0,Jz=0> =  0.5773502691896257
<Jp=1,Jzp= 0; Jq=1,Jzq= 0|J=0,Jz=0> =  -0.5773502691896257
0.5773502691896258
```

So the Hamiltonian in SU2 notation is

$$\hat{H} = -\frac{\sqrt{3}}{2} \sum_{i=1}^{L-1} (T_i)^{[1]} \otimes_{[0]} (T_{i+1})^{[1]}$$

We can solve this model using the following code:

[5]:
```
import numpy as np
from pyblock2.driver.core import DMRGDriver, SymmetryTypes

L = 100
heis_twos = 1

driver = DMRGDriver(scratch="./tmp", symm_type=SymmetryTypes.SU2, n_threads=4)
driver.initialize_system(n_sites=L, heis_twos=heis_twos, spin=0)

b = driver.expr_builder()
for i in range(L - 1):
    b.add_term("(T+T)0", [i, i + 1], -np.sqrt(3) / 2)
heis_mpo = driver.get_mpo(b.finalize(adjust_order=False), iprint=0)

energies = run_dmrg(driver, heis_mpo)
print('DMRG energy = %20.15f' % energies)
```

```
Sweep =    0 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-06
Time elapsed =     48.587 | E =     -44.1275743858 | DW = 7.23e-12

Sweep =    1 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-06
Time elapsed =     53.973 | E =     -44.1277393752 | DE = -1.65e-04 | DW = 6.90e-14

Sweep =    2 | Direction =  forward | Bond dimension =  250 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-06
Time elapsed =     59.107 | E =     -44.1277393752 | DE = -3.84e-13 | DW = 3.59e-16

Sweep =    3 | Direction = backward | Bond dimension =  250 | Noise =  1.00e-05 |␣
↪Dav threshold =  1.00e-06
Time elapsed =     63.144 | E =     -44.1277393752 | DE = -1.28e-13 | DW = 8.81e-17

Sweep =    4 | Direction =  forward | Bond dimension =  500 | Noise =  1.00e-06 |␣
↪Dav threshold =  1.00e-08
Time elapsed =     75.269 | E =     -44.1277398850 | DE = -5.10e-07 | DW = 3.06e-22

Sweep =    5 | Direction = backward | Bond dimension =  500 | Noise =  1.00e-06 |␣
↪Dav threshold =  1.00e-08
Time elapsed =     94.287 | E =     -44.1277398850 | DE = -1.99e-13 | DW = 6.38e-23

Sweep =    6 | Direction =  forward | Bond dimension =  500 | Noise =  0.00e+00 |␣
↪Dav threshold =  1.00e-08
Time elapsed =    103.865 | E =     -44.1277398850 | DE = 0.00e+00 | DW = 1.42e-23

DMRG energy =  -44.127739885005937
```

CHAPTER

FIVE

DEVELOPER GUIDE

5.1 DMRG Options

5.1.1 `me->dot`

- **Values:** 1 or 2.
- **Meaning:** Select 2-site or 1-site algorithm, unless affected by `last_site_1site`.

5.1.2 `decomp_last_site`

- **Meaning:** If `false`, decomposition for *affected* sites will be skipped.
- **Default:** `true`.
- **Affected sites:** - For `me->dot = 1`, only affect site `n - 1` for forward and site `0` for backward sweep. - For `me->dot = 2` and `last_site_1site = true`, only affect site `n - 1` for forward sweep.
- **Side effect:** *some* sites will have a canonical form different from the typical one.
- **Restriction:** Only active for `me->dot = 1` (or when `me->dot = 2` but `last_site_1site = true`).
- **Accuracy:** Should not affect accuracy.
- **Efficiency:** `decomp_last_site = false` provides faster speed.
- **Indicator:** When `decomp_last_site = false`, the affected site will print `Mmps = 0`.

5.1.3 last_site_svd

- **Meaning:** If `true`, for *affected* sites: - Davidson step will be skipped - *and* decomposition method will be changed to SVD - *and* if `noise_type = DensityMatrix`, it will be changed to `Wavefunction`.
- **Default:** `false`.
- **Affected sites:** only affect site `n - 1` for backward sweep.
- **Restriction:** Only active for `me->dot = 1` (or when `me->dot = 2` but `last_site_1site = true`).
- **Accuracy:** If `true`: - Skipping davidson step should not affect accuracy. - Using SVD instead of density matrix may decrease accuracy.
- **Efficiency:** `last_site_svd = true` provides faster speed.
- **Indicator:** When `last_site_svd = true`, the affected site will print `Ndav = 0 E = 0.0`.
- **Requirement:** Need `DMRGSCI`.

5.1.4 last_site_1site

- **Meaning:** If `true`, for *affected* sites: - In forward sweep, 2-site iteration for sites `n - 2` and `n - 1` will be changed to 1-site iteration for site `n - 1`. - In backward sweep, 2-site iteration for sites `n - 2` and `n - 1` will be changed to 1-site iteration for site `n - 1`.
- **Default:** `false`.
- **Affected sites:** only affect site `n - 2` and `n - 1` for forward and backward sweep.
- **Side effect:** MPS bond between site `n - 2` and site `n - 1` will not be updated in forward sweep.
- **Restriction:** Only active for `me->dot = 2`.
- **Accuracy:** If `true`: - Accuracy decreased because of 1-site algorithm. - Accuracy decreased because of the side effect.
- **Efficiency:** `last_site_1site = true` provides faster speed.
- **Indicator:** When `last_site_1site = true`, the affected site will print `Site = <n - 1> LAST`.
- **Requirement:** Need `DMRGSCI`.

5.1.5 Early DMRG stop

To stop a DMRG run gracefully, e.g., in case of non-convergence, create a file named `BLOCK_STOP_CALCULATION` with the text `STOP`. The DMRG run will then stop as it would be converged after the current sweep is over.

5.2 MPS Orbital Rotation

In this part we explain how to tranform an MPS with one orbital basis to another orbital basis. For the case when the new basis is the one with natural orbitals, please see *Input File: Advanced Usage* for a simple solution.

We assume that the orbital rotation only happens within each irrep. If this is not the case, you need to first transform MPS from a higher-order point group to a lower-order point group, according to *Point Group Mapping*.

5.2.1 Example

We consider, for example, the rotation from Hartree-Fock orbitals to localized orbitals within each irrep. As a first step, we construct these orbitals using `pyscf`:

```
from block2 import FCIDUMP, VectorUInt8
from pyscf import gto, scf, mcscf, lo, tools, ao2mo
from pyscf.mcscf import casci_symm
import scipy.linalg
import scipy.optimize
import numpy as np
mol = gto.M(atom='C 0 0 0; C 0 0 1.2425', basis='ccpvdz', symmetry='d2h')
mf = scf.RHF(mol).run()
mc = mcscf.CASCI(mf, 26, 8)

ncore = mc.ncore
nactorb = mc.ncas


# localize orbitals
def scdm(coeff, overlap):
    aux = lo.orth.lowdin(overlap)
    no = coeff.shape[1]
    ova = coeff.T @ overlap @ aux
    piv = scipy.linalg.qr(ova, pivoting=True)[2]
    bc = ova[:, piv[:no]]
    ova = np.dot(bc.T, bc)
    s12inv = lo.orth.lowdin(ova)
    return coeff @ bc @ s12inv
```

(continues on next page)

(continued from previous page)

```
# sort orbitals by irrep
def irrep_sort(coeff):
    optimal_reorder = [0, 6, 3, 5, 7, 1, 4, 2]  # d2h
    orb_sym = casci_symm.label_symmetry_(mc, mo_coeff_act).orbsym
    orb_opt = [optimal_reorder[x] for x in orb_sym]
    idx = np.argsort(orb_opt)
    return coeff[:, idx], orb_sym[idx]


# HF orbitals (old basis)
mo_coeff_act = mc.mo_coeff[:, mc.ncore:mc.ncore + mc.ncas].copy()
mo_coeff_act, mo_orb_sym = irrep_sort(mo_coeff_act)

# Symmetrized localized orbitals (new basis)
lmo_coeff_act = mo_coeff_act.copy()
for isym in set(mo_orb_sym):
    mask = np.array(mo_orb_sym) == isym
    lmo_coeff_act[:, mask] = scdm(
        mo_coeff_act[:, mask], mol.intor('cint1e_ovlp_sph'))
```

where `mo_coeff_act` represents the AO to MO coefficients for the old orbitals, and `lmo_coeff_act` represents the AO to MO coefficients for the new orbitals. The two sets of orbitals share the same irrep labels `mo_orb_sym`.

It is not necessary that the orbitals should be sorted according to irrep. But if orbitals with the same irrep are far from each other, the orbital rotation may be likely non-local.

Next, we construct the rotation matrix between the two sets of orbitals:

```
# orbital transform rot[old, new]
orb_rot = np.linalg.pinv(mo_coeff_act) @ lmo_coeff_act
assert np.linalg.norm(orb_rot.T - np.linalg.inv(orb_rot)) < 1E-12
assert np.linalg.norm(lmo_coeff_act - mo_coeff_act @ orb_rot) < 1E-12
```

To make the transformation as local as possible (so that the required MPS bond dimension for time evolution can be lower), we need to do some premutation and flipping of signs in the rotation matrix and consequently the new orbitals:

```
# change det sign and reorder rot within each irrep
def regularize_rot_mat(rot, orb_sym, iprint=False):
    rot = rot.copy()
    for isym in set(orb_sym):
        mask = np.array(orb_sym) == isym
        # orbital matching (reordering within irrep)
        kmidx = scipy.optimize.linear_sum_assignment(
```

(continues on next page)

(continued from previous page)

```
            1 - rot[mask, :][:, mask] ** 2)[1]
        if iprint:
            print("overlap before matching = ", np.sum(
                np.diag(rot[mask, :][:, mask]) ** 2))
        rot[:, mask] = rot[:, mask][:, kmidx]
        if iprint:
            print("overlap after matching = ", np.sum(
                np.diag(rot[mask, :][:, mask]) ** 2))
        # change sign to make it quasi-positive-definite
        for j in range(len(np.arange(len(mask))[mask])):
            mrot = rot[mask, :][:j + 1, :][:, mask][:, :j + 1]
            mrot_det = np.linalg.det(mrot)
            if iprint:
                print("ISYM = %d J = %d MDET = %15.10f" % (isym, j, mrot_det))
            if mrot_det < 0:
                mask0 = np.arange(len(mask), dtype=int)[mask][j]
                rot[:, mask0] = -rot[:, mask0]
    return rot


reg_orb_rot = regularize_rot_mat(orb_rot, mo_orb_sym)
assert np.linalg.det(reg_orb_rot) > 0

# regularized new basis
lmo_coeff_act = mo_coeff_act @ reg_orb_rot
```

Note that `reg_orb_rot` must have a $+1$ determinant, because otherwise the logarithm of it will have to be complex.

Now we can calculate the logarithm of the rotation matrix, namely, `kappa`:

```
# get logarithm of the rotation matrix
def get_kappa(rot, orb_sym):
    kappa = np.zeros_like(rot)
    for isym in set(orb_sym):
        mask = np.array(orb_sym) == isym
        mrot = rot[mask, :][:, mask]
        # scipy.linalg.logm works perfectly for
        # quasi-positive-definite matrices
        mkappa = scipy.linalg.logm(mrot)
        assert mkappa.dtype == float
        gkappa = np.zeros((kappa.shape[0], mkappa.shape[1]))
        gkappa[mask, :] = mkappa
        kappa[:, mask] = gkappa
    assert np.linalg.norm(
        scipy.linalg.expm(kappa) - rot) < 1E-10
```

(continues on next page)

(continued from previous page)

```
    assert np.linalg.norm(kappa + kappa.T) < 1E-10
    return kappa

kappa = get_kappa(reg_orb_rot, mo_orb_sym)
```

Next, The `FCIDUMP` objects for DMRG and time evolution can be constructed from the orbitals and kappa, respectively:

```
def get_fcidump(coeff, orb_sym, fname=None, tol=1E-13):
    mc.mo_coeff[:, mc.ncore:mc.ncore + mc.ncas] = coeff
    mp_orb_sym = [tools.fcidump.ORBSYM_MAP[mol.groupname][i] for i in orb_sym]
    h1e, e_core = mc.get_h1cas()
    h1e = h1e.ravel()
    g2e = ao2mo.restore(8, mc.get_h2cas(), mc.ncas)
    h1e[np.abs(h1e) < tol] = 0
    g2e[np.abs(g2e) < tol] = 0
    na, nb = mc.nelecas
    fcidump = FCIDUMP()
    fcidump.initialize_su2(mc.ncas, na + nb, na - nb, 1, e_core, h1e, g2e)
    fcidump.orb_sym = VectorUInt8(mp_orb_sym)
    assert fcidump.symmetrize(VectorUInt8(orb_sym)) < 1E-10
    if fname is not None:
        fcidump.write(fname)
    return fcidump


def get_kappa_fcidump(kappa, orb_sym, fname=None, tol=1E-13):
    mp_orb_sym = [tools.fcidump.ORBSYM_MAP[mol.groupname][i] for i in orb_sym]
    na, nb = mc.nelecas
    fcidump = FCIDUMP()
    kappa = kappa.flatten()
    kappa[np.abs(kappa) < tol] = 0
    fcidump.initialize_h1e(mc.ncas, na + nb, na - nb, 1, 0.0, kappa)
    fcidump.orb_sym = VectorUInt8(mp_orb_sym)
    assert fcidump.symmetrize(VectorUInt8(orb_sym)) < 1E-10
    if fname is not None:
        fcidump.write(fname)
    return fcidump


fd_old = get_fcidump(mo_coeff_act, mo_orb_sym)
fd_new = get_fcidump(lmo_coeff_act, mo_orb_sym)
fd_kappa = get_kappa_fcidump(kappa, mo_orb_sym)
```

where `fd_old` is for the DMRG in the old basis, and `fd_new` is for the DMRG in the new basis, and `fd_kappa` is for the orbital transform.

Now we are ready to do a DMRG in the old basis to find the ground-state MPS in this basis:

```
from block2 import *
from block2.su2 import *
import numpy as np
SX = SU2

Global.frame = DoubleDataFrame(10 * 1024 ** 2, 10 * 1024 ** 3, "nodex")
n_threads = Global.threading.n_threads_global
Global.threading = Threading(
    ThreadingTypes.OperatorBatchedGEMM | ThreadingTypes.Global,
    n_threads, n_threads, 1)
Global.threading.seq_type = SeqTypes.Tasked
Global.frame.fp_codec = DoubleFPCodec(1E-16, 1024)
Global.frame.minimal_disk_usage = True
Global.frame.use_main_stack = False
print(Global.frame)
print(Global.threading)



# Hamiltonian in old basis
fcidump = fd_old
pg = "d2h"
swap_pg = getattr(PointGroup, "swap_" + pg)
vacuum = SX(0)
target = SX(fcidump.n_elec, fcidump.twos, swap_pg(fcidump.isym))
n_sites = fcidump.n_sites
orb_sym = VectorUInt8(map(swap_pg, fcidump.orb_sym))
hamil = HamiltonianQC(vacuum, n_sites, orb_sym, fcidump)
print("D2H ORB SYM = ", hamil.orb_sym)

# MPS
mps_info = MPSInfo(n_sites, vacuum, target, hamil.basis)
mps_info.tag = 'KET'
mps_info.set_bond_dimension(250)
mps_info.save_data('./mps_info.bin')
mps = MPS(n_sites, 0, 2)
mps.initialize(mps_info)
mps.random_canonicalize()
mps.save_mutable()
mps_info.save_mutable()

# MPO
mpo = MPOQC(hamil, QCTypes.Conventional)
mpo = SimplifiedMPO(mpo, RuleQC(), True, True, OpNamesSet((OpNames.R, OpNames.RD)))
```

(continues on next page)

(continued from previous page)

```
# DMRG
me = MovingEnvironment(mpo, mps, mps, "DMRG")
me.delayed_contraction = OpNamesSet.normal_ops()
me.cached_contraction = True
me.init_environments(True)
dmrg = DMRG(me, VectorUBond([250, 500]), VectorDouble([1E-5] * 5 + [1E-6] * 5 + [0]))
dmrg.noise_type = NoiseTypes.ReducedPerturbativeCollected
dmrg.davidson_conv_thrds = VectorDouble([1E-6] * 5 + [1E-7] * 5)
ener = dmrg.solve(20, mps.center == 0, 1E-8)
print('DMRG Energy = %20.15f' % ener)
```

The following script can be used to transform the ground-state MPS to the new basis:

```
# Hamiltonain for orbital transform
hamil_kappa = HamiltonianQC(vacuum, n_sites, orb_sym, fd_kappa)

# MPO (anti-Hermitian)
mpo_kappa = MPOQC(hamil_kappa, QCTypes.Conventional)
mpo_kappa = SimplifiedMPO(mpo_kappa, AntiHermitianRuleQC(RuleQC()),
    True, True, OpNamesSet((OpNames.R, OpNames.RD)))

# Time Step
dt = 0.05
# Target time
tt = 1.0
n_steps = int(abs(tt) / abs(dt) + 0.1)
assert np.abs(abs(n_steps * dt) - abs(tt)) < 1E-10
print("Time Evolution NSTEPS = %d" % n_steps)
me_kappa = MovingEnvironment(mpo_kappa, mps, mps, "DMRG")
me_kappa.delayed_contraction = OpNamesSet.normal_ops()
me_kappa.cached_contraction = True
me_kappa.init_environments(True)

# Time Evolution (anti-Hermitian)
# te_type can be TETypes.RK4 or TETypes.TangentSpace (TDVP)
te_type = TETypes.RK4
te = TimeEvolution(me_kappa, VectorUBond([1000]), te_type)
te.hermitian = False
te.iprint = 2
te.n_sub_sweeps = 1 if te.mode == TETypes.TangentSpace else 2
te.normalize_mps = False
for i in range(n_steps):
    if te.mode == TETypes.TangentSpace:
        te.solve(2, dt / 2, mps.center == 0)
    else:
        te.solve(1, dt, mps.center == 0)
```

(continues on next page)

(continued from previous page)

```
    print("T = %10.5f <E> = %20.15f <Norm^2> = %20.15f" %
            ((i + 1) * dt, te.energies[-1], te.normsqs[-1]))
```

Note that when constructing MPO, `AntiHermitianRuleQC` has to be used. Also `te.hermitian` must be set to `False` for anti-Hermitian "Hamiltonian", otherwise it will be assumed Hermitian.

Note: `TimeEvolution` can support both one-site and two-site algorithm, but we highly recommend the two-site algorithm as there is no noise, and the one-site algorithm may have severe problem with losing quantum numbers.

Since every step in time evolution is a unitary transform, the "energy" expectation should always be zero, and the "norm" of the MPS should be close to one. Normally, a too large discarded weight or "norm" far from 1 indicates that the error during the transform is too large.

Finally, we can check the energy expectation of the transformed MPS in the new basis:

```
# Hamiltonain in new basis
hamil_new = HamiltonianQC(vacuum, n_sites, orb_sym, fd_new)

# MPO
mpo_new = MPOQC(hamil_new, QCTypes.Conventional)
mpo_new = SimplifiedMPO(mpo_new, RuleQC(), True, True, OpNamesSet((OpNames.R,␣
↪OpNames.RD)))

# Energy Expectation
me_new = MovingEnvironment(mpo_new, mps, mps, "OVL")
me_new.delayed_contraction = OpNamesSet.normal_ops()
me_new.cached_contraction = True
me_new.init_environments(True)

expect = Expect(me_new, mps.info.bond_dim, mps.info.bond_dim)
ener_new = expect.solve(False, mps.center == 0)

print('Energy expectation = %20.15f' % ener_new)
```

Some reference outputs for this example:

```
D2H ORB SYM =  VectorUInt8[ 0 0 0 0 0 0 5 5 5 5 5 5 7 7 7 2 2 2 6 6 6 3 3 3 1 4 ]
DMRG Energy =  -75.728487321653233
Time Evolution NSTEPS = 20
T =    0.05000 <E> =   -0.000000000000000 <Norm^2> =    0.999999979398520
T =    0.10000 <E> =   -0.000000000000000 <Norm^2> =    0.999999926838107
... ...
T =    0.95000 <E> =    0.000000000000000 <Norm^2> =    0.999996763879923
Time elapsed =      5.738 | E =       0.0000000000 | Norm^2 =       0.9999964412 |␣
↪DW = 3.83e-08
```

(continues on next page)

(continued from previous page)

```
T =    1.00000 <E> =    0.000000000000000 <Norm^2> =    0.999996441150652
Energy expectation =  -75.728011987963555
```

5.2.2 Distributed Parallelization

Since the "Hamiltonian" used in orbital rotation has only one-body term, it is more efficient to use a different parallelization rule. The normal two-body parallelization rule can still be used, but it will not provide any speed-up when more than one MPI processes are used.

The one-body only parallelization rule can be used in the following way:

```
MPI = MPICommunicator()
prule_one_body = ParallelRuleOneBodyQC(MPI)
mpo_kappa = ParallelMPO(mpo_kappa, prule_one_body)
```

5.2.3 MRCI (Big-Site) Example

The same procedure can be easily applied to the big-site MPO and MPS for MRCI calculation, with very little change. The above script for normal MPS can be reused without change for big-site until line `from block2 import *`.

Then, for big-site MPO/MPS, the following script can be used:

```
from block2 import *
from block2.su2 import *
import numpy as np
SX = SU2

Global.frame = DoubleDataFrame(10 * 1024 ** 2, 10 * 1024 ** 3, "nodex")
n_threads = Global.threading.n_threads_global
Global.threading = Threading(
    ThreadingTypes.OperatorBatchedGEMM | ThreadingTypes.Global,
    n_threads, n_threads, 1)
Global.threading.seq_type = SeqTypes.Nothing
Global.frame.fp_codec = DoubleFPCodec(1E-16, 1024)
Global.frame.minimal_disk_usage = True
Global.frame.use_main_stack = False
print(Global.frame)
print(Global.threading)

# create a big site in MPO
n_ext, ci_order = 5, 2
def create_big_site(hamil, mpo):
    mrci_mps_info = MRCIMPSInfo(hamil.n_sites, n_ext, ci_order, hamil.vacuum, target,
↪ hamil.basis)
```

(continues on next page)

(continued from previous page)

```
    mpo.basis = hamil.basis
    for i in range(n_ext):
        mpo = FusedMPO(mpo, mpo.basis, mpo.n_sites - 2, mpo.n_sites - 1, mrci_mps_
↪info.right_dims_fci[mpo.n_sites - 2])
    for k, op in mpo.tensors[-1].ops.items():
        smat = CSRSparseMatrix()
        if op.sparsity() > 0.75:
            smat.from_dense(op)
            op.deallocate()
        else:
            smat.wrap_dense(op)
        mpo.tensors[-1].ops[k] = smat
    mpo.sparse_form = mpo.sparse_form[:-1] + 'S'
    mpo.tf = TensorFunctions(CSROperatorFunctions(hamil.opf.cg))
    return mpo

# Hamiltonian in old basis
fcidump = fd_old
pg = "d2h"
swap_pg = getattr(PointGroup, "swap_" + pg)
vacuum = SX(0)
target = SX(fcidump.n_elec, fcidump.twos, swap_pg(fcidump.isym))
n_sites = fcidump.n_sites
orb_sym = VectorUInt8(map(swap_pg, fcidump.orb_sym))
hamil = HamiltonianQC(vacuum, n_sites, orb_sym, fcidump)
print("D2H ORB SYM = ", hamil.orb_sym)

# MPO
mpo = MPOQC(hamil, QCTypes.Conventional)
mpo = create_big_site(hamil, mpo)
mpo = SimplifiedMPO(mpo, RuleQC(), True, True, OpNamesSet((OpNames.R, OpNames.RD)))

# MPS
mps_info = MPSInfo(mpo.n_sites, vacuum, target, mpo.basis)
mps_info.tag = 'KET'
mps_info.set_bond_dimension(250)
mps_info.save_data('./mps_info.bin')
mps = MPS(mpo.n_sites, 0, 2)
mps.initialize(mps_info)
mps.random_canonicalize()
mps.save_mutable()
mps_info.save_mutable()

# DMRG
me = MovingEnvironment(mpo, mps, mps, "DMRG")
```

(continues on next page)

(continued from previous page)

```
me.delayed_contraction = OpNamesSet.normal_ops()
me.cached_contraction = True
me.init_environments(True)
dmrg = DMRG(me, VectorUBond([250, 500]), VectorDouble([1E-5] * 5 + [1E-6] * 5 + [0]))
dmrg.noise_type = NoiseTypes.ReducedPerturbativeCollected
dmrg.davidson_conv_thrds = VectorDouble([1E-6] * 5 + [1E-7] * 5)
ener = dmrg.solve(20, mps.center == 0, 1E-8)
print('MRCI DMRG Energy = %20.15f' % ener)


# Hamiltonain for orbital transform
hamil_kappa = HamiltonianQC(vacuum, n_sites, orb_sym, fd_kappa)

# MPO (anti-Hermitian)
mpo_kappa = MPOQC(hamil_kappa, QCTypes.Conventional)
mpo_kappa = create_big_site(hamil_kappa, mpo_kappa)
mpo_kappa = SimplifiedMPO(mpo_kappa, AntiHermitianRuleQC(RuleQC()), True, True,␣
↪OpNamesSet((OpNames.R, OpNames.RD)))

# Time Step
dt = 0.05
# Target time
tt = 1.0
n_steps = int(abs(tt) / abs(dt) + 0.1)
assert np.abs(abs(n_steps * dt) - abs(tt)) < 1E-10
print("Time Evolution NSTEPS = %d" % n_steps)
me_kappa = MovingEnvironment(mpo_kappa, mps, mps, "DMRG")
me_kappa.delayed_contraction = OpNamesSet.normal_ops()
me_kappa.cached_contraction = True
me_kappa.init_environments(True)

# Time Evolution (anti-Hermitian)
# te_type can be TETypes.RK4 or TETypes.TangentSpace (TDVP)
te_type = TETypes.RK4
te = TimeEvolution(me_kappa, VectorUBond([1000]), te_type)
te.hermitian = False
te.iprint = 2
te.n_sub_sweeps = 1 if te.mode == TETypes.TangentSpace else 2
te.normalize_mps = False
for i in range(n_steps):
    if te.mode == TETypes.TangentSpace:
        te.solve(2, dt / 2, mps.center == 0)
    else:
        te.solve(1, dt, mps.center == 0)
    print("T = %10.5f <E> = %20.15f <Norm^2> = %20.15f" %
```

(continues on next page)

(continued from previous page)

```
            ((i + 1) * dt, te.energies[-1], te.normsqs[-1]))


# Hamiltonain in new basis
hamil_new = HamiltonianQC(vacuum, n_sites, orb_sym, fd_new)

# MPO
mpo_new = MPOQC(hamil_new, QCTypes.Conventional)
mpo_new = create_big_site(hamil_new, mpo_new)
mpo_new = SimplifiedMPO(mpo_new, RuleQC(), True, True, OpNamesSet((OpNames.R,␣
↪OpNames.RD)))

# Energy Expectation
me_new = MovingEnvironment(mpo_new, mps, mps, "OVL")
me_new.delayed_contraction = OpNamesSet.normal_ops()
me_new.cached_contraction = True
me_new.init_environments(True)

expect = Expect(me_new, mps.info.bond_dim, mps.info.bond_dim)
ener_new = expect.solve(False, mps.center == 0)

print('Energy expectation = %20.15f' % ener_new)
```

where the big-site MPO is created using the function `create_big_site`, where the right-boundary sites in the MPO are folded to a big site using the `FusedMPO` class. Other more efficient methods for creating a big site can be used, but note that, the big site in the three MPOs `mpo`, `mpo_kappa`, and `mpo_new` must be created using the same method. This is to ensure that the quantum number fusing order is consistent among different MPOs. This is required because the same MPS is used with all these MPOs.

Also note that `SeqTypes.Nothing` (instead of `SeqTypes.Tasked`) should be used for big-site with CSR matrices.

Some reference outputs for this example:

```
D2H ORB SYM =  VectorUInt8[ 0 0 0 0 0 0 5 5 5 5 5 5 7 7 7 2 2 2 6 6 6 3 3 3 1 4 ]
MRCI DMRG Energy =  -75.727859086194130
Time Evolution NSTEPS = 20
T =    0.05000 <E> =    0.000000000000000 <Norm^2> =    0.999999980443349
T =    0.10000 <E> =   -0.000000000000000 <Norm^2> =    0.999999930992521
... ...
T =    0.95000 <E> =    0.000000000000000 <Norm^2> =    0.999996944337650
Time elapsed =      6.035 | E =      -0.0000000000 | Norm^2 =       0.9999966399 |␣
↪DW = 3.84e-08
T =    1.00000 <E> =   -0.000000000000000 <Norm^2> =    0.999996639941846
Energy expectation =  -75.727409014459965
```

5.3 Point Group Mapping

Here we discuss how to transform an MPS with a point group (PG) (such as D_{2h}) to an MPS with another PG (such as C_{2v} or C_s).

Limitations:

- Can transform from high-order PG (ket) to low-order PG (bra) or low-order PG (ket) to high-order PG (bra);
- The mapping between the two PG must be a homomorphism. As long as it is a homomorphism, any mapping can be used.
- For normal MPS, the transformation only have a tiny fitting error. For MPS with big site, the matrix elements in the big-site tensor in the MPS will be artificially reordered. (Because the "fusing order" inside the big site can have an influence on the order of states within each symmetry block. Since "fusing order" in the big site can be arbitrary, the mapping code itself cannot figure out the correct mapping for the order of states within each symmetry block. For normal site, there is only one state for each symmetry block so there is no problem.) So the transformed big-site tensor will not be accurate (but the normal site tensors in a big-site MPS will still be accurate).
- The integral (FCIDUMP) with the two PG must be exactly the same. Consider the following case: if you generate the integral from `pyscf`, you calculate one integral with `D2h` symmetry in the molecule, and another integral with `C2v` symmetry in the molecule. Then there can be small (or big) changes in the integral. Then you cannot use this feature, since point group symmetry is not the only thing that changed.

5.3.1 Example

The example integral file `C2.CAS.PVDZ.FCIDUMP` can be found in the `data` folder.

First we do a ground state calculation using D_{2h} point group:

```
from block2 import *
from block2.su2 import *
import numpy as np
SX = SU2

Global.frame = DoubleDataFrame(10 * 1024 ** 2, 10 * 1024 ** 3, "nodex")
n_threads = Global.threading.n_threads_global
Global.threading = Threading(
    ThreadingTypes.OperatorBatchedGEMM | ThreadingTypes.Global,
    n_threads, n_threads, 1)
Global.threading.seq_type = SeqTypes.Tasked
Global.frame.fp_codec = DoubleFPCodec(1E-16, 1024)
Global.frame.minimal_disk_usage = True
Global.frame.use_main_stack = False
print(Global.frame)
```

(continues on next page)

(continued from previous page)

```
print(Global.threading)

fcidump = FCIDUMP()
fcidump.read('C2.CAS.PVDZ.FCIDUMP')

# D2H Hamiltonian
pg = "d2h"
swap_pg = getattr(PointGroup, "swap_" + pg)
vacuum = SX(0)
target = SX(fcidump.n_elec, fcidump.twos, swap_pg(fcidump.isym))
n_sites = fcidump.n_sites
orb_sym = VectorUInt8(map(swap_pg, fcidump.orb_sym))
hamil = HamiltonianQC(vacuum, n_sites, orb_sym, fcidump)
print("D2H ORB SYM = ", hamil.orb_sym)

# MPS
mps_info = MPSInfo(n_sites, vacuum, target, hamil.basis)
mps_info.tag = 'KET'
mps_info.set_bond_dimension(250)
mps_info.save_data('./mps_info.bin')
mps = MPS(n_sites, 0, 2)
mps.initialize(mps_info)
mps.random_canonicalize()
mps.save_mutable()
mps_info.save_mutable()

# MPO
mpo = MPOQC(hamil, QCTypes.Conventional)
mpo = SimplifiedMPO(mpo, RuleQC(), True, True, OpNamesSet((OpNames.R, OpNames.RD)))

# DMRG
me = MovingEnvironment(mpo, mps, mps, "DMRG")
me.delayed_contraction = OpNamesSet.normal_ops()
me.cached_contraction = True
me.init_environments(True)
dmrg = DMRG(me, VectorUBond([250, 500]), VectorDouble([1E-5] * 5 + [1E-6] * 5 + [0]))
dmrg.noise_type = NoiseTypes.ReducedPerturbativeCollected
dmrg.davidson_conv_thrds = VectorDouble([1E-6] * 5 + [1E-7] * 5)
ener = dmrg.solve(20, mps.center == 0, 1E-8)
print('DMRG Energy = %20.15f' % ener)
```

Then we define a Hamiltonian with a different PG (but the same integral). The mapping should be based on the XOR notation. Here we create `hamil_c2v` by mapping D2h irreps to C2v irreps. The mapping is not unique, you may need to figure out the actual mapping based on how you will need to mix orbitals with different irreps. Note that something like `pg_map = lambda x: x & 6` or `pg_map = lambda x: x & 3` should also work.

```
# C2V Hamiltonian
pg_map = lambda x: (x & 6) >> 1
orb_sym_c2v = VectorUInt8([pg_map(x) for x in orb_sym]) # the mapping is not unique
hamil_c2v = HamiltonianQC(vacuum, n_sites, orb_sym_c2v, fcidump)
target_c2v = SX(target.n, target.twos, pg_map(target.pg))
print("C2V ORB SYM = ", hamil_c2v.orb_sym)
```

To transform MPS, we need a special identity MPO. This identity will not have bond dimension 1 since it has to mix different PG irreps. If the MPS does not have any big-site, the last two parameters `orb_sym_c2v, orb_sym` can be omitted.

```
# Identity MPO for PG mapping
delta_target = (target_c2v - target)[0]
impo = IdentityMPO(hamil_c2v.basis, hamil.basis, vacuum,
    delta_target, hamil.opf, orb_sym_c2v, orb_sym)
impo = SimplifiedMPO(impo, NoTransposeRule(RuleQC()))
```

Next, we can perform the transformation of MPS using fitting.

```
# C2V MPS
mps_info_c2v = MPSInfo(n_sites, vacuum, target_c2v, hamil_c2v.basis)
mps_info_c2v.tag = 'KET-C2V'
mps_info_c2v.set_bond_dimension(500)
mps_info_c2v.save_data('./mps_info_c2v.bin')
mps_c2v = MPS(n_sites, mps.center, 2)
mps_c2v.initialize(mps_info_c2v)
mps_c2v.random_canonicalize()
mps_c2v.save_mutable()
mps_info_c2v.save_mutable()

# Linear
me = MovingEnvironment(impo, mps_c2v, mps, "LIN")
me.delayed_contraction = OpNamesSet.normal_ops()
me.cached_contraction = True
me.init_environments(True)
cps = Linear(me, VectorUBond([500]), VectorUBond([500]))
norm = cps.solve(20, mps.center == 0, 1E-8)
print('Norm = %20.15f' % norm)
```

Finally, we can check whether the MPS gives the correct energy in the new C2v basis:

```
# C2V MPO
mpo_c2v = MPOQC(hamil_c2v, QCTypes.Conventional)
mpo_c2v = SimplifiedMPO(mpo_c2v, RuleQC(), True, True, OpNamesSet((OpNames.R,␣
↪OpNames.RD)))

# Expectation
```

(continues on next page)

(continued from previous page)

```
me = MovingEnvironment(mpo_c2v, mps_c2v, mps_c2v, "DMRG")
me.delayed_contraction = OpNamesSet.normal_ops()
me.cached_contraction = True
me.init_environments(True)
ex = Expect(me, 500, 500)
ener_c2v = ex.solve(False)
print('C2V Energy = %20.15f' % ener_c2v)
```

The printed energy should be very close to the D2h sweep energy at the last site of the last sweep. Note that this may not be the same as the DMRG energy, which is the lowest energy in the last sweep, because here the MPS is transformed from the previous D2h MPS with the center at the last site.

If the MPS contains big-site, there can be a much larger error in the energy due to the reordering of states in the big-site MPS tensor. Re-optimizing the big-site tensor may solve this problem. In addition, `me.delayed_contraction = OpNamesSet.normal_ops()` *must not* be set. Otherwise, the following assertion occurs:

```
Assertion`a->get_type() == SparseMatrixTypes::Normal && b->get_type() ==␣
↪SparseMatrixTypes::Normal && c->get_type() == SparseMatrixTypes::Normal && v->get_
↪type() == SparseMatrixTypes::Normal && da->get_type() == SparseMatrixTypes::Normal␣
↪&& db->get_type() == SparseMatrixTypes::Normal' failed.
```

Some reference output for this example:

```
D2H ORB SYM =  VectorUInt8[ 5 0 6 5 3 5 0 0 5 0 3 6 5 0 3 6 7 2 7 2 7 2 1 4 0 5 ]
<-- Site =    0-   1 .. Mmps =    3 Ndav =   1 E =    -75.7284493902 Error = 1.14e-
↪16 FLOPS = 8.66e+05 Tdav = 0.00 T = 0.01
DMRG Energy =  -75.728475543752168
C2V ORB SYM =  VectorUInt8[ 2 0 3 2 1 2 0 0 2 0 1 3 2 0 1 3 3 1 3 1 3 1 0 2 0 2 ]
Norm =    1.000000000000001
C2V Energy =  -75.728449390238850
```

5.3.2 Inverse Mapping

The inverse mapping from C_{2v} to D_{2h} is also supported. The script is basically the same (except the exchange between C_{2v} and D_{2h}):

```
from block2 import *
from block2.su2 import *
import numpy as np
SX = SU2

Global.frame = DoubleDataFrame(10 * 1024 ** 2, 10 * 1024 ** 3, "nodex")
n_threads = Global.threading.n_threads_global
Global.threading = Threading(
```

(continues on next page)

(continued from previous page)

```
    ThreadingTypes.OperatorBatchedGEMM | ThreadingTypes.Global,
    n_threads, n_threads, 1)
Global.threading.seq_type = SeqTypes.Tasked
Global.frame.fp_codec = DoubleFPCodec(1E-16, 1024)
Global.frame.minimal_disk_usage = True
Global.frame.use_main_stack = False
print(Global.frame)
print(Global.threading)

fcidump = FCIDUMP()
fcidump.read('C2.CAS.PVDZ.FCIDUMP')

# C2V Hamiltonian
pg = "d2h"
pg_map = lambda x: (x & 6) >> 1
swap_pg = getattr(PointGroup, "swap_" + pg)
vacuum = SX(0)
target_d2h = SX(fcidump.n_elec, fcidump.twos, swap_pg(fcidump.isym))
target_c2v = SX(target_d2h.n, target_d2h.twos, pg_map(target_d2h.pg))
n_sites = fcidump.n_sites
orb_sym_d2h = VectorUInt8(map(swap_pg, fcidump.orb_sym))
orb_sym_c2v = VectorUInt8([pg_map(x) for x in orb_sym_d2h]) # the mapping is not␣
↪unique
hamil = HamiltonianQC(vacuum, n_sites, orb_sym_c2v, fcidump)
print("C2V ORB SYM = ", hamil.orb_sym)

# C2V MPS
mps_info = MPSInfo(n_sites, vacuum, target_c2v, hamil.basis)
mps_info.tag = 'KET'
mps_info.set_bond_dimension(250)
mps_info.save_data('./mps_info.bin')
mps = MPS(n_sites, 0, 2)
mps.initialize(mps_info)
mps.random_canonicalize()
mps.save_mutable()
mps_info.save_mutable()

# C2V MPO
mpo = MPOQC(hamil, QCTypes.Conventional)
mpo = SimplifiedMPO(mpo, RuleQC(), True, True, OpNamesSet((OpNames.R, OpNames.RD)))

# C2V DMRG
me = MovingEnvironment(mpo, mps, mps, "DMRG")
me.delayed_contraction = OpNamesSet.normal_ops()
me.cached_contraction = True
```

(continues on next page)

(continued from previous page)

```
me.init_environments(True)
dmrg = DMRG(me, VectorUBond([250, 500]), VectorDouble([1E-5] * 5 + [1E-6] * 5 + [0]))
dmrg.noise_type = NoiseTypes.ReducedPerturbativeCollected
dmrg.davidson_conv_thrds = VectorDouble([1E-6] * 5 + [1E-7] * 5)
ener = dmrg.solve(20, mps.center == 0, 1E-8)
print('DMRG Energy = %20.15f' % ener)

# D2H Hamiltonian
hamil_d2h = HamiltonianQC(vacuum, n_sites, orb_sym_d2h, fcidump)
print("D2H ORB SYM = ", hamil_d2h.orb_sym)

# Identity MPO for PG mapping
delta_target = (target_d2h - target_c2v)[0]
impo = IdentityMPO(hamil_d2h.basis, hamil.basis, vacuum,
    delta_target, hamil.opf, orb_sym_d2h, orb_sym_c2v)
impo = SimplifiedMPO(impo, NoTransposeRule(RuleQC()))

# D2H MPS
mps_info_d2h = MPSInfo(n_sites, vacuum, target_d2h, hamil_d2h.basis)
mps_info_d2h.tag = 'KET-D2H'
mps_info_d2h.set_bond_dimension(500)
mps_info_d2h.save_data('./mps_info_d2h.bin')
mps_d2h = MPS(n_sites, mps.center, 2)
mps_d2h.initialize(mps_info_d2h)
mps_d2h.random_canonicalize()
mps_d2h.save_mutable()
mps_info_d2h.save_mutable()

# Linear
me = MovingEnvironment(impo, mps_d2h, mps, "LIN")
me.delayed_contraction = OpNamesSet.normal_ops()
me.cached_contraction = True
me.init_environments(True)
cps = Linear(me, VectorUBond([500]), VectorUBond([500]))
norm = cps.solve(20, mps.center == 0, 1E-8)
print('Norm = %20.15f' % norm)

# D2H MPO
mpo_d2h = MPOQC(hamil_d2h, QCTypes.Conventional)
mpo_d2h = SimplifiedMPO(mpo_d2h, RuleQC(), True, True, OpNamesSet((OpNames.R,␣
↪OpNames.RD)))

# D2H Expectation
me = MovingEnvironment(mpo_d2h, mps_d2h, mps_d2h, "DMRG")
me.delayed_contraction = OpNamesSet.normal_ops()
```

(continues on next page)

(continued from previous page)

```
me.cached_contraction = True
me.init_environments(True)
ex = Expect(me, 500, 500)
ener_d2h = ex.solve(False) / norm ** 2
print('D2H Energy = %20.15f' % ener_d2h)
```

Some reference outputs for this example:

```
C2V ORB SYM =  VectorUInt8[ 2 0 3 2 1 2 0 0 2 0 1 3 2 0 1 3 3 1 3 1 3 1 0 2 0 2 ]
--> Site =   24-  25 .. Mmps =    3 Ndav =   1 E =    -75.7284490538 Error = 1.62e-
↪19 FLOPS = 3.87e+05 Tdav = 0.00 T = 0.01
DMRG Energy =  -75.728475021520978
D2H ORB SYM =  VectorUInt8[ 5 0 6 5 3 5 0 0 5 0 3 6 5 0 3 6 7 2 7 2 7 2 1 4 0 5 ]
Norm =    0.999999999998821
D2H Energy =  -75.728449053829152
```

5.3.3 Initial Guess for Compression

For large systems, the initial guess for `Linear` (`mps_c2v` or `mps_d2h` in the above examples) may be too bad, and very small overlap (`F` value) with `mps` can be observed. The MPS bond dimension will be kept as 1 or a very small number (it should be at least one, since by default the random FCI initial guess is used, where at least one state is kept for each quantum number in the initial guess).

To solve this problem, one can add `cps.cutoff = 0` before the line `norm = cps.solve(...)`. Alternatively, one can add `cps.trunc_type = TruncationTypes.KeepOne * n` before the line `norm = cps.solve(...)`, where `n` is a small positive integer.

Generating initial guess using occupation numbers may also alleviate this problem, but using the above settings, better initial guess with occupation numbers is not mandatory.

5.4 MPO Reloading

For systems with large number of orbitals, it is sometimes beneficial to save/reload the MPO object to reduce memory fragmentation. The step of creation of the Hamiltonian and FCIDUMP object can also be done only once and all Hamiltonian information can be kept in the MPO object, which can be saved in disk storage. This can save computational cost (if creation of the Hamiltonian/MPO is expensive) and memory cost (if the FCIDUMP object is big) for restarting.

For even larger number of orbitals, keeping the whole MPO object during the DMRG calculation may still be memory-demanding. To solve this problem, the MPO can be reloaded in a minimal memory mode. In this mode, only the essential data in MPO is loaded in the beginning. Then, during the DMRG calculation, blocking formulae and definition of single-site operators will be loaded for each site only. After the iteration for one site, the memory consumed by the blocking formulae and single-site operators can be released. Therefore, even if the MPO object itself can be big, only a small part (for each current site) is loaded into memory (dynamically) at any instant during the DMRG calculation.

Limitations:

- If an MPO is loaded in the minimal memory mode, the MPO file must be kept in the file system (namely, not deleted / overwritten) during any subsequent algorithms using this MPO.
- If an MPO is loaded in the minimal memory mode, it is read-only. This means, you cannot simplify or parallelize such an MPO. As a result, if you use distributed parallelism, you have to save the already parallelized MPO (for each rank as separate files), and reload them in the minimal memory mode.
- Inspecting some site-related contents inside the minimal-memory MPO can be more complicated (requiring `mpo.load_*` before the operation) since these contents are not in memory by default.

5.4.1 Example

The example integral file `C2.CAS.PVDZ.FCIDUMP` can be found in the `data` folder.

Saving a Serial MPO

First we save a non-parallelized MPO using the following script:

```
from block2 import *
from block2.su2 import *
import numpy as np
import psutil
import os
SX = SU2

Global.frame = DoubleDataFrame(10 * 1024 ** 2, 10 * 1024 ** 3, "nodex")
n_threads = Global.threading.n_threads_global
Global.threading = Threading(
    ThreadingTypes.OperatorBatchedGEMM | ThreadingTypes.Global,
    n_threads, n_threads, 1)
Global.threading.seq_type = SeqTypes.Tasked
Global.frame.fp_codec = DoubleFPCodec(1E-16, 1024)
Global.frame.minimal_disk_usage = True
Global.frame.use_main_stack = False
print(Global.frame)
print(Global.threading)

fcidump = FCIDUMP()
fcidump.read('C2.CAS.PVDZ.FCIDUMP')

# D2H Hamiltonian
pg = "d2h"
swap_pg = getattr(PointGroup, "swap_" + pg)
```

(continues on next page)

(continued from previous page)

```
vacuum = SX(0)
target = SX(fcidump.n_elec, fcidump.twos, swap_pg(fcidump.isym))
n_sites = fcidump.n_sites
orb_sym = VectorUInt8(map(swap_pg, fcidump.orb_sym))
hamil = HamiltonianQC(vacuum, n_sites, orb_sym, fcidump)
print("ORB SYM = ", hamil.orb_sym)

mem = psutil.Process(os.getpid()).memory_info().rss
print(" pre-mpo memory usage = %10s" % Parsing.to_size_string(mem))

# MPO
mpo = MPOQC(hamil, QCTypes.Conventional)
mpo = SimplifiedMPO(mpo, RuleQC(), True, True, OpNamesSet((OpNames.R, OpNames.RD)))
mpo.basis = hamil.basis

mem = psutil.Process(os.getpid()).memory_info().rss
print("post-mpo memory usage = %10s" % Parsing.to_size_string(mem))

mpo.reduce_data()
mpo.save_data('mpo.bin')

fsize = os.path.getsize('mpo.bin')
print("mpo size = %10s" % Parsing.to_size_string(fsize))
```

Some reference outputs (the memory information can be different for each run):

```
$ grep 'usage\|size' dmrg-1.out
 pre-mpo memory usage =    58.5 MB
post-mpo memory usage =     126 MB
mpo size =    2.35 MB
```

So without saving and reloading the MPO, the MPO object needs roughly 67.5 MB memory.

Loading a Serial MPO

We can now load the saved `mpo.bin` to do DMRG, and skip the step for creating `HamiltonianQC` and `FCIDUMP`:

```
from block2 import *
from block2.su2 import *
import numpy as np
import psutil
import os
SX = SU2
```

(continues on next page)

(continued from previous page)

```
Global.frame = DoubleDataFrame(10 * 1024 ** 2, 10 * 1024 ** 3, "nodex")
n_threads = Global.threading.n_threads_global
Global.threading = Threading(
    ThreadingTypes.OperatorBatchedGEMM | ThreadingTypes.Global,
    n_threads, n_threads, 1)
Global.threading.seq_type = SeqTypes.Tasked
Global.frame.fp_codec = DoubleFPCodec(1E-16, 1024)
Global.frame.minimal_disk_usage = True
Global.frame.use_main_stack = False
print(Global.frame)
print(Global.threading)

mem = psutil.Process(os.getpid()).memory_info().rss
print(" pre-load-mpo memory usage = %10s" % Parsing.to_size_string(mem))

mpo = MPO(0)
mpo.load_data('mpo.bin')

mem = psutil.Process(os.getpid()).memory_info().rss
print("post-load-mpo memory usage = %10s" % Parsing.to_size_string(mem))

n_sites = mpo.n_sites
vacuum = SX(0)
target = SX(8, 0, 0)

mps_info = MPSInfo(mpo.n_sites, vacuum, target, mpo.basis)
mps_info.tag = 'KET'
mps_info.set_bond_dimension(250)
mps = MPS(n_sites, 0, 2)
mps.initialize(mps_info)
mps.random_canonicalize()
mps.save_mutable()
mps_info.save_mutable()

# DMRG
me = MovingEnvironment(mpo, mps, mps, "DMRG")
me.delayed_contraction = OpNamesSet.normal_ops()
me.cached_contraction = True
me.init_environments(True)
dmrg = DMRG(me, VectorUBond([250, 500]), VectorDouble([1E-5] * 5 + [1E-6] * 5 + [0]))
dmrg.noise_type = NoiseTypes.ReducedPerturbativeCollected
dmrg.davidson_conv_thrds = VectorDouble([1E-6] * 5 + [1E-7] * 5)
ener = dmrg.solve(20, mps.center == 0, 1E-8)
print('DMRG Energy = %20.15f' % ener)
```

Some reference outputs (the memory information can be different for each run):

```
$ grep 'usage\|Energy' dmrg-2.out
 pre-load-mpo memory usage =    42.6 MB
post-load-mpo memory usage =    53.5 MB
DMRG Energy =  -75.728475321395166
```

So the reloaded MPO object is smaller, which needs only 10.9 MB memory. The DMRG takes 70.581 seconds.

Loading a Serial MPO with Minimal Memory

One can change the line in the above script:

```
mpo.load_data('mpo.bin')
```

to:

```
mpo.load_data('mpo.bin', minimal=True)
```

Then rerun the script. Now the MPO is loaded in the minimal memory mode.

Some reference outputs (the memory information can be different for each run):

```
$ grep 'usage\|Energy' dmrg-2.out
 pre-load-mpo memory usage =    40.7 MB
post-load-mpo memory usage =    43.0 MB
DMRG Energy =  -75.728475329694518
```

Now the reloaded MPO object occupies only 2.3 MB memory before the DMRG calculation. The DMRG takes 70.688 seconds (which is not greatly affected by dynamically reloading MPO parts).

Saving Parallelized MPO

For distributed calculations, we can still reload the serial MPO and parallelize it. But this way is only compatible to the non-minimal-memory mode. To save the memory for distributed calculations, we need to save the parallelized MPO. The parallelization script for MPO does not have to be run in parallel (but you still can run it in parallel, which has a lower wall time cost but a higher memory cost).

The following script generates and saves the parallelized MPO for 7 mpi processsors (note that this script should be run in serial, namely, no `mpirun`):

```
from block2 import *
from block2.su2 import *
import numpy as np
import psutil
import os

```

(continues on next page)

(continued from previous page)

```
Global.frame = DoubleDataFrame(10 * 1024 ** 2, 10 * 1024 ** 3, "nodex")

mpo = MPO(0)
mpo.load_data('mpo.bin')

# size, rank, root
comm = ParallelCommunicator(7, 0, 0)
prule = ParallelRuleQC(comm)

for irank in range(comm.size):
    comm.rank = irank
    para_mpo = ParallelMPO(mpo, prule)
    para_mpo.save_data('mpo.bin.%d' % irank)
    fsize = os.path.getsize('mpo.bin.%d' % irank)
    print("mpo.%d size = %10s" % (irank, Parsing.to_size_string(fsize)))
```

Here we assume a serial MPO `mpo.bin` has already been saved in the disk. The `ParallelCommunicator` is a fake object for distributed parallelism. We can manually change the `rank` of `ParallelCommunicator` to generate parallelized MPOs for different ranks.

Some reference outputs:

```
mpo.0 size =    2.74 MB
mpo.1 size =    2.75 MB
mpo.2 size =    2.73 MB
mpo.3 size =    2.74 MB
mpo.4 size =    2.77 MB
mpo.5 size =    2.78 MB
mpo.6 size =    2.77 MB
```

Note that each parallelized MPO is larger than the serial MPO. Actually, each of them includes both the "local" part and "global" part. The "global" part then has the same size as the serial MPO. (For big site code the "global" part for parallelized MPO can be smaller than the full MPO).

Reloading Parallelized MPO

The following script is used for parallel DMRG with 7 mpi processsors (namely, `mpirun -n 7 --bind-to none python -u dmrg.py`, for example):

```
from block2 import *
from block2.su2 import *
import numpy as np
import psutil
import os
SX = SU2
```

(continues on next page)

(continued from previous page)

```
MPI = MPICommunicator()

Global.frame = DoubleDataFrame(10 * 1024 ** 2, 10 * 1024 ** 3, "nodex")
n_threads = Global.threading.n_threads_global // MPI.size
Global.threading = Threading(
    ThreadingTypes.OperatorBatchedGEMM | ThreadingTypes.Global,
    n_threads, n_threads, 1)
Global.threading.seq_type = SeqTypes.Tasked
Global.frame.fp_codec = DoubleFPCodec(1E-16, 1024)
Global.frame.minimal_disk_usage = True
Global.frame.use_main_stack = False
print(Global.frame)
print(Global.threading)

prule = ParallelRuleQC(MPI)

mem = psutil.Process(os.getpid()).memory_info().rss
print(" pre-load-mpo memory usage = %10s" % Parsing.to_size_string(mem))

mpo = ParallelMPO(0, prule)
mpo.load_data('mpo.bin.%d' % MPI.rank, minimal=False)

mem = psutil.Process(os.getpid()).memory_info().rss
print("post-load-mpo memory usage = %10s" % Parsing.to_size_string(mem))

n_sites = mpo.n_sites
vacuum = SX(0)
target = SX(8, 0, 0)

mps_info = MPSInfo(mpo.n_sites, vacuum, target, mpo.basis)
mps_info.tag = 'KET'
mps_info.set_bond_dimension(250)
mps = MPS(n_sites, 0, 2)
mps.initialize(mps_info)
mps.random_canonicalize()
mps.save_mutable()
mps_info.save_mutable()

# DMRG
me = MovingEnvironment(mpo, mps, mps, "DMRG")
me.delayed_contraction = OpNamesSet.normal_ops()
me.cached_contraction = True
me.init_environments(True)
dmrg = DMRG(me, VectorUBond([250, 500]), VectorDouble([1E-5] * 5 + [1E-6] * 5 + [0]))
dmrg.noise_type = NoiseTypes.ReducedPerturbativeCollected
```

(continues on next page)

(continued from previous page)

```
dmrg.davidson_conv_thrds = VectorDouble([1E-6] * 5 + [1E-7] * 5)
ener = dmrg.solve(20, mps.center == 0, 1E-8)
print('DMRG Energy = %20.15f' % ener)
```

Some reference outputs (the memory information can be different for each run):

```
$ grep 'post-\|Energy' dmrg-3.out
post-load-mpo memory usage =    59.6 MB
post-load-mpo memory usage =    61.6 MB
post-load-mpo memory usage =    59.4 MB
post-load-mpo memory usage =    63.6 MB
post-load-mpo memory usage =    59.4 MB
post-load-mpo memory usage =    59.4 MB
post-load-mpo memory usage =    59.4 MB
DMRG Energy =  -75.728475146585453
DMRG Energy =  -75.728475146585453
DMRG Energy =  -75.728475146585453
DMRG Energy =  -75.728475146585453
DMRG Energy =  -75.728475146585453
DMRG Energy =  -75.728475146585453
DMRG Energy =  -75.728475146585453
```

Reloading Parallelized MPO with Minimal Memory

One can change the line in the above script:

```
mpo.load_data('mpo.bin.%d' % MPI.rank, minimal=False)
```

to:

```
mpo.load_data('mpo.bin.%d' % MPI.rank, minimal=True)
```

Then rerun the script. Now the MPO is loaded in the minimal memory mode.

Some reference outputs (the memory information can be different for each run):

```
$ grep 'post-\|Energy' dmrg-3.out
post-load-mpo memory usage =    52.8 MB
post-load-mpo memory usage =    48.8 MB
post-load-mpo memory usage =    50.8 MB
post-load-mpo memory usage =    50.8 MB
post-load-mpo memory usage =    52.8 MB
post-load-mpo memory usage =    48.9 MB
post-load-mpo memory usage =    48.8 MB
DMRG Energy =  -75.728475151371001
DMRG Energy =  -75.728475151371001
```

(continues on next page)

(continued from previous page)

```
DMRG Energy =  -75.728475151371001
DMRG Energy =  -75.728475151371001
DMRG Energy =  -75.728475151371001
DMRG Energy =  -75.728475151371001
DMRG Energy =  -75.728475151371001
```

We can see that the memory usage after loading MPO is smaller, compared to the non-minimal-memory-usage mode.

5.5 Debugging Hints

Here we list some of common assertion failure, errors, wrong outputs, and the solutions.

5.5.1 Ground State Calculation

[2021-05-09]

Assertion:

```
block2/parallel_mpi.hpp:330: void block2::MPICommunicator<S>::reduce_sum(double*,␣
↪size_t, int) [with S = block2::SU2Long; size_t = long unsigned int]: Assertion␣
↪`ierr == 0' failed.
```

Conditions: More than one MPI processors, `QCTypes.Conventional`, `Random.rand_seed(0)`, and `gaopt`. Random assertion failure.

Reason: A different gaopt reordering was used in different mpi processors. Then the error happens during the initialization of environments. Then there will be an array-size mismatching due to the difference in integrals.

Solution: Broadcast the orbital reordering indices before reordering the integral.

[2021-05-10]

Assertion:

```
block2/operator_functions.hpp:185: void block2::OperatorFunctions<S>::tensor_
↪rotate(const std::shared_ptr<block2::SparseMatrix<S> >&, const std::shared_ptr
↪<block2::SparseMatrix<S> >&, const std::shared_ptr<block2::SparseMatrix<S> >&,␣
↪const std::shared_ptr<block2::SparseMatrix<S> >&, bool, double) const [with S =␣
↪block2::SZLong]: Assertion `a->get_type() == SparseMatrixTypes::Normal && c->get_
↪type() == SparseMatrixTypes::Normal && rot_bra->get_type() == SparseMatrixTypes::
↪Normal && rot_ket->get_type() == SparseMatrixTypes::Normal' failed.
```

Conditions: Loaded MPO, CSR.

Reason: The non-CSR `OperatorFunctions` is used for calculation requiring CSR matrices, after loading MPO.

Solution: Change `csr_opf = OperatorFunctions(cg)` to `csr_opf = CSROperatorFunctions(cg)`.

[2021-05-11]

Output:

```
Sweep =   15 | Direction = backward | Bond dimension = 2000 | Noise =  1.00e-07 | Dav threshold =  1.00e-08
<-- Site =   11-  12 .. Mmps =    3 Ndav =   1 E =    -36.8356589402 Error = 0.00e+00 FLOPS = 4.12e+06 Tdav = 0.02 T = 0.17
<-- Site =   10-  11 .. Mmps =   10 Ndav =   1 E =    -36.8356589402 Error = 0.00e+00 FLOPS = 2.91e+08 Tdav = 0.02 T = 0.18
<-- Site =    9-  10 .. Mmps =   35 Ndav =   1 E =    -36.8356589402 Error = 0.00e+00 FLOPS = 9.70e+09 Tdav = 0.02 T = 0.20
<-- Site =    8-   9 .. Mmps =  126 Ndav =   1 E =    -36.8356589402 Error = 0.00e+00 FLOPS = 6.96e+10 Tdav = 0.06 T = 0.40
<-- Site =    7-   8 .. Mmps =  462 Ndav =   1 E =    -36.8356589402 Error = 0.00e+00 FLOPS = 1.52e+11 Tdav = 0.28 T = 1.08
<-- Site =    6-   7 .. Mmps = 1454 Ndav =   1 E =    -36.8356589402 Error = 4.57e-13 FLOPS = 2.27e+11 Tdav = 0.79 T = 2.54
<-- Site =    5-   6 .. Mmps = 1679 Ndav =  12 E =    -37.0888587109 Error = 1.41e-12 FLOPS = 2.83e+11 Tdav = 7.21 T = 12.32
<-- Site =    4-   5 .. Mmps =  904 Ndav =   1 E =    -37.0888587109 Error = 1.53e-12 FLOPS = 1.95e+11 Tdav = 0.27 T = 1.91
<-- Site =    3-   4 .. Mmps =  490 Ndav =   1 E =    -37.0888587109 Error = 8.62e-13 FLOPS = 7.32e+10 Tdav = 0.05 T = 0.60
<-- Site =    2-   3 .. Mmps =  209 Ndav =   1 E =    -37.0888587109 Error = 2.47e-13 FLOPS = 9.69e+09 Tdav = 0.02 T = 0.28
<-- Site =    1-   2 .. Mmps =   64 Ndav =   1 E =    -37.0888587109 Error = 9.69e-15 FLOPS = 1.06e+09 Tdav = 0.01 T = 0.26
<-- Site =    0-   1 .. Mmps =   11 Ndav =   1 E =    -37.0888587109 Error = 5.58e-15 FLOPS = 5.78e+06 Tdav = 0.02 T = 0.16
Time elapsed =    187.772 | E =     -37.0888587109 | DE = -6.18e-12 | DW = 1.53e-12
Time sweep =       20.100 | 2.11 TFLOP/SWP
| Tcomm = 7.916 | Tidle = 3.657 | Twait = 0.000 | Dmem = 89.2 MB (11%) | Imem = 93.8 KB (96%) | Hmem = 736 MB | Pmem = 50.8 MB
| Tread = 0.505 | Twrite = 0.553 | Tfpread = 0.462 | Tfpwrite = 0.090 | Tasync = 0.000
| Trot = 0.368 | Tctr = 0.055 | Tint = 0.016 | Tmid = 2.304 | Tdctr = 0.033 | Tdiag = 0.310 | Tinfo = 0.039
| Teff = 1.591 | Tprt = 2.578 | Teig = 8.760 | Tblk = 16.722 | Tmve = 3.376 | Tdm = 0.000 | Tsplt = 0.000 | Tsvd = 1.678
```

(continues on next page)

(continued from previous page)

```
Sweep =   16 | Direction =  forward | Bond dimension = 2000 | Noise =  1.00e-07 | Dav threshold =  1.00e-08
--> Site =    0-   1 .. Mmps =    3 Ndav =   1 E =    -37.0888587109 Error = 0.00e+00 FLOPS = 4.51e+06 Tdav = 0.02 T = 0.18
--> Site =    1-   2 .. Mmps =   10 Ndav =   1 E =    -37.0888587109 Error = 0.00e+00 FLOPS = 8.65e+08 Tdav = 0.01 T = 0.09
--> Site =    2-   3 .. Mmps =   35 Ndav =   1 E =    -37.0888587109 Error = 0.00e+00 FLOPS = 1.03e+10 Tdav = 0.02 T = 0.11
--> Site =    3-   4 .. Mmps =  126 Ndav =   1 E =    -37.0888587109 Error = 0.00e+00 FLOPS = 7.65e+10 Tdav = 0.05 T = 0.35
--> Site =    4-   5 .. Mmps =  462 Ndav =   1 E =    -37.0888587109 Error = 0.00e+00 FLOPS = 1.61e+11 Tdav = 0.32 T = 1.25
--> Site =    5-   6 .. Mmps = 1511 Ndav =   1 E =    -37.0888587109 Error = 3.25e-13 FLOPS = 2.24e+11 Tdav = 0.76 T = 2.50
--> Site =    6-   7 .. Mmps = 1805 Ndav =  17 E =    -36.8356599462 Error = 1.65e-12 FLOPS = 3.10e+11 Tdav = 10.53 T = 15.73
--> Site =    7-   8 .. Mmps =  975 Ndav =   1 E =    -36.8356599462 Error = 1.51e-12 FLOPS = 1.59e+11 Tdav = 0.38 T = 2.13
--> Site =    8-   9 .. Mmps =  408 Ndav =   1 E =    -36.8356599462 Error = 8.11e-13 FLOPS = 7.52e+10 Tdav = 0.06 T = 0.53
--> Site =    9-  10 .. Mmps =  156 Ndav =   1 E =    -36.8356599462 Error = 2.57e-13 FLOPS = 1.16e+10 Tdav = 0.02 T = 0.13
--> Site =   10-  11 .. Mmps =   57 Ndav =   1 E =    -36.8356599462 Error = 1.46e-14 FLOPS = 4.29e+08 Tdav = 0.01 T = 0.18
--> Site =   11-  12 .. Mmps =   12 Ndav =   1 E =    -36.8356599462 Error = 4.83e-15 FLOPS = 4.13e+06 Tdav = 0.02 T = 0.06
Time elapsed =    211.003 | E =     -37.0888587109 | DE = 4.21e-12 | DW = 1.65e-12
Time sweep =       23.231 | 3.23 TFLOP/SWP
| Tcomm = 8.521 | Tidle = 2.996 | Twait = 0.000 | Dmem = 95.4 MB (10%) | Imem = 93.8 KB (96%) | Hmem = 736 MB | Pmem = 52.5 MB
| Tread = 0.550 | Twrite = 0.624 | Tfpread = 0.504 | Tfpwrite = 0.092 | Tasync = 0.000
| Trot = 0.385 | Tctr = 0.039 | Tint = 0.023 | Tmid = 2.480 | Tdctr = 0.052 | Tdiag = 0.323 | Tinfo = 0.035
| Teff = 1.734 | Tprt = 2.656 | Teig = 12.197 | Tblk = 19.563 | Tmve = 3.667 | Tdm = 0.000 | Tsplt = 0.000 | Tsvd = 1.508
```

Conditions: More than one MPI processors, and `QCTypes.Conventional`.

Reason: We see from the output that the energy jumps between two values even in very large bond dimension. If only one MPI is used, there is no such behavior. This is because the input integrals `h1e` and `g2e` are not synchronized. In `QCTypes.Conventional`, communication between MPI procs only happens at the middle site. After this communication, the inconsistentcy between integrals can cause an artificial change in energy. Note that inside `block2`, we do not explicitly synchronize integral. In future, for larger systems, the integral can even be distributed, such that synchronization will not be meaningful.

Solution: Synchronizing the input integrals `h1e` and `g2e` can solve this problem.

[2021-05-12 | 2021-06-07]

Error Message: (note that this problem in `block2main` has been fixed in commit 4f87784)

```
Traceback (most recent call last):
File "block2/pyblock2/driver/block2main", line 302, in <module>
    mps.load_data()
RuntimeError: MPS::load_data on '/central/scratch/.../F.MPS.KET.-1' failed.
```

or

```
Traceback (most recent call last):
File "block2/pyblock2/driver/block2main", line 313, in <module>
    mps.load_mutable()
RuntimeError: SparseMatrix:load_data on '/central/scratch/.../F.MPS.KET.14' failed.
```

or

```
Traceback (most recent call last):
File "block2/pyblock2/driver/block2main", line 313, in <module>
    mps.load_mutable()
ValueError: cannot create std::vector larger than max_size()
```

Conditions: More than one MPI processors, python driver, happening with a very low probability.

Reason A: The problematic code is:

```
mps.load_data()
if mps.dot != dot and nroots == 1:
    mps.dot = dot
    mps.save_data()
```

And the non-root MPI proc can load data before or after the root proc saves the data. The wrong loaded data can cause the subsequent `mps.load_mutable()` to fail.

Solution A: Adding `MPI.barrier()` around `mps.save_data()`.

Reason B: The problematic code is:

```
mps.load_mutable()
mps.save_mutable()
```

And the non-root MPI proc can load data before or after the root proc saves the data. This may cause simultaneously reading and writing into the same file (with a very low probability).

Solution B: Adding `MPI.barrier()` between `mps.load_mutable()` and `mps.save_mutable()`.

5.5.2 Linear

[2021-05-14]

Assertion:

```
block2/moving_environment.hpp:110: block2::MovingEnvironment<S>::
↪MovingEnvironment(const std::shared_ptr<block2::MPO<S> >&, const std::shared_ptr
↪<block2::MPS<S> >&, const std::shared_ptr<block2::MPS<S> >&, const string&) [with␣
↪S = block2::SU2Long; std::string = std::__cxx11::basic_string<char>]: Assertion␣
↪`bra->center == ket->center && bra->dot == ket->dot' failed.
```

Conditions: Different bra and ket.

Reason: The bra and ket for initialization of MovingEnvironment do not have the same center.

Solution: Initializing bra or ket with consistent center, or do a sweep to align the MPS center.

[2021-05-14]

Assertion:

```
block2/operator_functions.hpp:194: void block2::OperatorFunctions<S>::tensor_
↪rotate(const std::shared_ptr<block2::SparseMatrix<S> >&, const std::shared_ptr
↪<block2::SparseMatrix<S> >&, const std::shared_ptr<block2::SparseMatrix<S> >&,␣
↪const std::shared_ptr<block2::SparseMatrix<S> >&, bool, double) const [with S =␣
↪block2::SU2Long]: Assertion `adq == cdq && a->info->n >= c->info->n' failed.
```

Conditions 1: Different bra and ket.

Reason 1: The bra and ket has different MPSInfo, but the two MPSInfo has the same tag. When saving to/loading from disk, the information stored in the two MPSInfo can interfere with each other.

Solution 1: Use different tags for different MPSInfo.

Conditions 2: MPSInfo in MPS differs from data in MPS.

Reason 2: An MPS has been loaded in from disk with a wrong MPSInfo.

Solution 2: Load in MPSInfo as well or make sure MPSInfo is correct.

[2021-05-14]

Assertion:

```
block2/csr_matrix_functions.hpp:387: static void block2::CSRMatrixFunctions::
↪multiply(const MatrixRef&, bool, const block2::CSRMatrixRef&, bool, const␣
↪MatrixRef&, double, double): Assertion `(conja ? a.m : a.n) == (conjb ? b.n : b.m)
↪' failed.
```

Conditions: Different bra and ket, CSR, IdentityMPO with bra and ket with different bases.

Reason: Wrong basis was used in the constructor of IdentityMPO.

Solution: Change `IdentityMPO(mpo_bra.basis, mpo_bra.basis, ...)` to `IdentityMPO(mpo_bra.basis, mpo_ket.basis, ...)`.

[2021-05-18]

Assertion:

```
block2/csr_matrix_functions.hpp:396: static void block2::CSRMatrixFunctions::
↪multiply(const MatrixRef&, bool, const block2::CSRMatrixRef&, bool, const␣
↪MatrixRef&, double, double): Assertion `st == SPARSE_STATUS_SUCCESS' failed.
```

Conditions: CSR, `SeqTypes.Tasked`.

Reason: `SeqTypes.Tasked` cannot be used together with CSR.

Solution: Change `Global.threading.seq_type = SeqTypes.Tasked` to `Global.threading.seq_type = SeqTypes.Nothing`.

[2021-05-22]

Assertion:

```
block2/sparse_matrix.hpp:552: void block2::SparseMatrixInfo<S, typename std::enable_
↪if<std::integral_constant<bool, (sizeof (S) == sizeof (unsigned int))>::value>::
↪type>::save_data(std::ostream&, bool) const [with S = block2::SU2Long; typename␣
↪std::enable_if<std::integral_constant<bool, (sizeof (S) == sizeof (unsigned int))>:
↪:value>::type = void; std::ostream = std::basic_ostream<char>]: Assertion `n != -1
↪' failed.
```

Conditions: `mps.save_mutable`.

Reason: Some MPS tensors are deallocated (unloaded) after `mps.flip_fused_form(...)` or `mps.move_left(...)`.

Solution: Call `mps.load_mutable()` after using `mps.flip_fused_form(...)` or `mps.move_left(...)`, so that `mps.save_mutable()` will be successful.

[2021-05-31]

Error:

```
exceeding allowed memory
```

Conditions: `Linear` with `tme != nullptr`.

Reason: By default, no bond dimension truncation is performed for MPS in `Linear::tme`.

Solution: Set `target_bra_bond_dim` and `target_ket_bond_dim` fields in `Linear` to a suitable bond dimension.

[2021-06-07]

Assertion:

```
block2/parallel_rule.hpp:215: void block2::ParallelRule<S>::distributed_apply(T,␣
↪const std::vector<std::shared_ptr<block2::OpExpr<S> > >&, const std::vector<std::
↪shared_ptr<block2::OpExpr<S> > >&, std::vector<std::shared_ptr<block2::SparseMatrix
↪<S> > >&) const [with T = block2::ParallelTensorFunctions<S>::right_contract(const␣
↪std::shared_ptr<block2::OperatorTensor<S> >&, const std::shared_ptr<block2::
↪OperatorTensor<S> >&, std::shared_ptr<block2::OperatorTensor<S> >&, const std::
↪shared_ptr<block2::Symbolic<S> >&, block2::OpNamesSet) const [with S = block2::
↪SZLong]::<lambda(const std::vector<std::shared_ptr<block2::OpExpr<block2::SZLong> >
↪, std::allocator<std::shared_ptr<block2::OpExpr<block2::SZLong> > > >&)>; S =␣
↪block2::SZLong]: Assertion `expr->get_type() == OpTypes::ExprRef' failed.
```

Conditions: `ParallelMPO`.

Reason: The problematic code is:

```
impo = IdentityMPOSCI(hamil)
impo = ParallelMPO(impo, ParallelRuleIdentity(MPI))
```

On most cases, `ParallelMPO` may not work with unsimplified MPO. The MPO should first be simplified and then parallelized.

Solution: Use `ClassicParallelMPO` (may have bad performance) or change the code to

```
impo = IdentityMPOSCI(hamil)
impo = SimplifiedMPO(impo, Rule())
impo = ParallelMPO(impo, ParallelRuleIdentity(MPI))
```

[2021-06-08]

Assertion:

```
block2/sparse_matrix.hpp:1548:  void block2::SparseMatrix<S>::swap_to_fused_
↪left(const std::shared_ptr<block2::SparseMatrix<S> >&, const block2::StateInfo<S>&,
↪ const block2::StateInfo<S>&, const block2::StateInfo<S>&, const block2::StateInfo
↪<S>&, const block2::StateInfo<S>&, const block2::StateInfo<S>&, const block2::
↪StateInfo<S>&, const std::shared_ptr<block2::CG<S> >&) [with S = block2::SZLong]:␣
↪Assertion `mat->info->is_wavefunction' failed.
```

Conditions: IdentityMPO used in MPI simulation without `ParallelMPO`.

Reason: The problematic code is:

```
impo = IdentityMPOSCI(hamil)
me = MovingEnvironment(impo, mps1, mps2)
```

Solution: Use `ParallelMPO` (vide supra):

```
impo = IdentityMPOSCI(hamil)
impo = SimplifiedMPO(impo, Rule())
impo = ParallelMPO(impo, ParallelRuleIdentity(MPI))
```

[2021-08-20]

Assertion:

```
dmrg/mps.hpp:1547: void block2::MPS<S>::move_left(const std::shared_ptr<block2::CG<S>
↪ >&, const std::shared_ptr<block2::ParallelRule<S> >&) [with S = block2::SU2Long]:␣
↪Assertion `tensors[center - 1]->info->n != 0' failed.
```

Reason: A SZ MPS is loaded for use in a SU2 code.

[2021-12-14]

Assertion:

```
core/matrix_functions.hpp:307: static void block2::GMatrixFunctions<double>::
↪multiply(const MatrixRef&, uint8_t, const MatrixRef&, uint8_t, const MatrixRef&,␣
↪double, double): Assertion `a.n >= b.m && c.m == a.m && c.n >= b.n' failed.
```

Reason: For transition reduced density matrix, if bra and ket are the MPSs with the same tag, they must be the same object. For example, this is the case when they are the same ith root from a state-averaged MultiMPS. Therefore, one should not "extract" the same root twice with the same tag. This will cause the conflict in the disk storage. This was a bug in the main driver for onedot transition 1/2 reduced density matrix with more than one root.

5.5.3 MRCI/SCI computations

[2021-06-08]

Error:

```
find_site_op_info cant find q:< N=? SZ=? PG=? >iSite=??
```

Conditions: Issue with quantum number setup.

Reason: This can happen if symmetry is used but the integrals don't obey symmetry.

Solution: Add the following code. Attention: This will change the fcidump. Use with case and check `symmetrize_error`

```
symmetrize_error = fcidump.symmetrize(orb_sym)
```

5.5.4 Library Import

[2022-08-18]

Error:

```
*** Error in `python': double free or corruption (out)
```

Reason: This can happen if there are two pybind11 libraries, but they are built with differrent compiler versions. When import both libraries in the same python script, this error can happen.

Solution: One workaround is to write two python scripts so that the two pybind11 libraries are not imported in the same script. Otherwise, one needs to compile both extensions manually, or use the pip version for both libraries, so that they can be used together.

[2022-08-19]

Error:

```
/.../libmkl_avx2.so: undefined symbol: mkl__blas_write_lock_dgemm_hashtable
INTEL MKL ERROR: /.../libmkl_avx2.so.1: undefined symbol: mkl_sparse_optimize_bsr_
↪trsm_i8.
OSError: /.../libmkl_def.so: undefined symbol: mkl_dnn_getTtl_F32
```

Solution: This can be solved by add link flags `-Wl,--no-as-needed` with all absolute `*so` path for mkl libraries. Note that flags `-Wl,-rpath -Wl,/../lib -L/.../lib` should not be used.

A special case is when both the "so.1" (2021 MKL) and "so" (2019 ML) are used. We need to make sure the `block2.so` library is linked to only pure "so.1" or only pure "so".

[2022-10-11]

Error:

```
svd dead loop, with full CPU usage. CPU has avx512.
```

Solution: Update MKL from 2019 to 2021.4 or 2022.1.

[2022-12-26]

Error:

```
Insert ``cout`` in openMP parallel code will cause dead lock. Because it is not␣
↪thread-safe. Use ``printf`` instead.
```

[2023-10-13]

Error:

```
ImportError: /.../block2.cpython-38-x86_64-linux-gnu.so: undefined symbol: _ZNSt15__
↪exception_ptr13exception_ptr10_M_releaseEv
```

Solution: This happens when code is compiled using gcc/12.2.0 but gcc module is not loaded. If compile using gcc/9.2.0 there is no problem.

[2023-11-03]

Error:

```
INTEL MKL ERROR: /opt/hostedtoolcache/Python/3.9.18/x64/lib/libmkl_def.so.1:␣
↪undefined symbol: mkl_sparse_optimize_bsr_trsm_i8.
Intel MKL FATAL ERROR: Cannot load libmkl_def.so.1.
```

Solution: Check if this only happens in the AMD CPU but not Intel CPU. If this is the case, we can fix this by the `fakeintel` trick

```
export CPUTYPE=$(lscpu | grep 'Vendor ID' | awk '{print $3}')
if [ "$CPUTYPE" = "AuthenticAMD" ]; then
    echo "int mkl_serv_intel_cpu_true() { return 1; }" > fixcpu.c
    $CC -shared -fPIC -o libfixcpu.so fixcpu.c
    export LD_PRELOAD=$PWD/libfixcpu.so
fi
```

[2023-11-30]

Error:

```
/.../block2.cpython-38-x86_64-linux-gnu.so: undefined symbol: _ZNSt15__exception_
↪ptr13exception_ptr9_M_addrefEv
```

Solution A: This can be caused by loading the wrong `libstdc++.so.6` at runtime. For example, when using a new g++ (gcc 13.2.0) to build the code,
and there is an old `/anaconda3/envs/python3.7/lib/libstdc++.so.6` which is linked by cmake and loaded at runtime, this error can happen. Adding `LD_PRELOAD=/lib64/libstdc++.so.6` at runtime is one way to fix this.

Solution B: This can also be related to Python import order. If `numpy` or `pyscf` was built using an old version of gcc and you first
import `numpy` or `pyscf`, then `import block2` will cause error. To sovle this problem, one can `import block2` at the beginning.

5.6 Notes

5.6.1 Build block2 in `manylinux2010` docker image

The docker image named `quay.io/pypa/manylinux2010_x86_64` is used.

First we need to select one python version:

```
export PATHBAK=$PATH
export PATH=/opt/python/cp37-cp37m/bin:$PATHBAK
export PATH=/opt/python/cp38-cp38/bin:$PATHBAK
export PATH=/opt/python/cp39-cp39/bin:$PATHBAK
export PATH=/opt/python/cp310-cp310/bin:$PATHBAK
which python3
```

Clone the block2 repo:

```
git clone https://github.com/block-hczhai/block2
```

Edit the `setup.py`:

```
'-DPYTHON_EXECUTABLE={}'.format('/opt/python/cp37-cp37m/bin/python3'),
```

Instal dependencies and build:

```
python3 -m pip install pip build twine --upgrade
python3 -m pip install mkl==2019 mkl-include intel-openmp numpy 'cmake>=3.19'␣
↪pybind11
python3 -m build
```

Change linux tag and upload:

```
mv dist/block2-0.1.10-cp38-cp38-linux_x86_64.whl dist/block2-0.1.10-cp38-cp38-
↪manylinux2010_x86_64.whl
python3 -m twine upload dist/block2-0.1.10-cp38-cp38-manylinux2010_x86_64.whl
```

5.6.2 Installing block2 using python virtual environment

This guide shows how one can manually build block2 with openmpi without using Anaconda and Intel oneapi.

First we assume a suitable python3, openmpi library, and gcc compiler can be found in the system. For example, I have

```
$ which gcc
/opt/gcc/11.2.0/bin/gcc
$ which mpirun
/opt/openmpi/4.1.2/gnu/bin/mpirun
$ which python3
```

First we install the python virtualenv

```
$ python3 -m pip install --user --upgrade pip
$ python3 -m pip install --user virtualenv
```

Then we create a virtualenv called base and activate it, which will create a folder called base in the current folder

```
$ python3 -m venv base
$ source base/bin/activate
```

Then we install necessary packages in this virtualenv

```
$ pip install mkl mkl-include pybind11 numpy scipy psutil
$ pip install mpi4py --no-binary mpi4py
```

Then we can build block as the following

```
$ mkdir build
$ cd build
$ export MKLROOT=/path/to/base
$ export PATH=/path/to/base:$PATH
$ cmake .. -DUSE_MKL=ON -DBUILD_LIB=ON -DMPI=ON -DLARGE_BOND=ON -DTBB=ON -DUSE_
↪DMRG=ON -DUSE_BIG_SITE=ON -DUSE_SP_DMRG=ON -DUSE_IC=ON -DUSE_KSYMM=ON -DUSE_
↪COMPLEX=ON
```

We can test that mpi4py is working

```
$ mpirun -n 2 python -c 'from mpi4py import MPI;print(MPI.COMM_WORLD.rank)'
```

CHAPTER

SIX

API REFERENCE

6.1 Python Interface API

Python interface of block2.

class pyblock2.driver.core.**SymmetryTypes**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `IntFlag`

Enumeration of symmetry modes (symmetry groups, complex/real types, and floating-point precision).

`CPX`, `SP`, `SAny` can be combined with some other types using operator `|`. For example, `SymmetryTypes.CPX | SymmetryTypes.SU2` is the SU(2) mode with the complex number.

Nothing = 0

Real number and double precision.

SU2 = 1

Spin-adapted Fermion mode, U(1) x SU(2) x AbelianPG.

SZ = 2

Non-spin-adapted Fermion mode, U(1) x U(1)[Sz] x AbelianPG.

SGF = 4

General spin Fermion mode. U(1)[Fermion number] x AbelianPG.

SGB = 8

General spin Boson mode. U(1)[Boson number] x AbelianPG.

CPX = 16

Complex number. Can be combined with Fermion/Boson symmetry types.

SP = 32

Single precision. Can be combined with Fermion/Boson symmetry types.

SGFCPX = 20

Complex number general spin Fermion mode, for relativistic DMRG.

SPCPX = 48

Single precision complex number.

SAny = 64

General symmetry type.

SAnySU2 = 65

Equivalent to 'SU2', implemented using general symmetry types.

SAnySZ = 66

Equivalent to 'SZ', implemented using general symmetry types.

SAnySGF = 68

Equivalent to 'SGF', implemented using general symmetry types.

SAnySGFCPX = 84

Equivalent to 'SGFCPX', implemented using general symmetry types.

SO4 = 128

PHSU2 = 256

SO3 = 512

LZ = 1024

SAnySO4 = 192

SO(4) symmetry for 2-D Hubbard model with NN interactions.

SAnyPHSU2 = 320

Particle-hole symmetry for 2-D Hubbard model with NN interactions.

SAnySO3 = 576

SO(3) spatial symmetry for atoms.

SAnySU2LZ = 1089

Spin-adapted mode for diatomic molecules, U(1) x SU(2) x U(1)[Lz].

SAnySZLZ = 1090

Non-spin-adapted mode for diatomic molecules, U(1) x U(1)[Sz] x U(1)[Lz].

SAnySGFLZ = 1092

General spin mode for diatomic molecules, U(1) x U(1)[Lz].

class pyblock2.driver.core.**ParallelTypes**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `IntFlag`

Enumeration of strategies for distributing the quantum chemistry `h1e` and `g2e` integrals. See Eq. (2) in *J. Chem. Phys.* **154**, 224116 (2021).

Nothing = 0

Serial computation.

I = 1

Distribute integrals over the first index.

J = 2

Distribute integrals over the second index.

SI = 3

Distribute integrals over the smallest index.

SJ = 4

Distribute integrals over the second smallest index.

SIJ = 5

Distribute integrals over the unordered tuple of the two smallest indices (IJ) of `g2e`. When `J == K`, distribute over J. Same as 'SI' for `h1e`. See Eq. (5) in *J. Chem. Phys.* **154**, 224116 (2021).

SKL = 6

Distribute integrals over the unordered tuple of the two largest indices (KL) of `g2e`. When `J == K`, distribute over J. Same as 'SJ' for `h1e`. See Eq. (6) in *J. Chem. Phys.* **154**, 224116 (2021).

UIJ = 7

Distribute integrals over the unordered tuple of the two smallest indices (IJ) of `g2e`. When `J == K`, distribute over IJ. Same as 'SI' for `h1e`.

UKL = 8

Distribute integrals over the unordered tuple of the two largest indices (KL) of `g2e`. When `J == K`, distribute over KL. Same as 'SJ' for `h1e`.

UIJM2 = 9

Similar to 'UIJ', but distribute over IJ / 2.

UKLM2 = 10

Similar to 'UKL', but distribute over KL / 2.

UIJM4 = 11

Similar to 'UIJ', but distribute over IJ / 4.

UKLM4 = 12

Similar to 'UKL', but distribute over KL / 4.

MixUIJUKL = 1000

Two-layer nested UIJ (outer) and UKL (inner). Not a good strategy.

MixUKLUIJ = 1001

Two-layer nested UKL (outer) and UIJ (inner). Not a good strategy.

MixUIJSI = 1002

Two-layer nested UIJ (outer) and SI (inner).

class pyblock2.driver.core.**MPOAlgorithmTypes**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `IntFlag`

Enumeration of strategies for building MPO from symbolic expression of second quantized operators. Algorithms (such as `Bipartite`) and modifiers (such as `Fast`) can be combined using operator `|`.

Nothing = 0

No algorithm specified (invalid).

Bipartite = 1

Bipartite algorithm. See *J. Chem. Phys.* **153**, 084118 (2020).

SVD = 2

SVD algorithm. See *J. Chem. Phys.* **145**, 014102 (2016).

Rescaled = 4

Use rescaled singular values for truncation. Optional for 'SVD', cannot be used alone.

Fast = 8

Fast implementation for dense integrals. Optional for 'SVD' and 'Bipartite', cannot be used alone.

Blocked = 16

Separate SVD for blocks. Optional for 'SVD' and 'Bipartite', cannot be used alone.

Sum = 32

Sum of MPO approach. See *J. Chem. Phys.* **145**, 014102 (2016). Optional for 'SVD' and 'Bipartite', cannot be used alone.

Constrained = 64

SVD with constraints on sparsity. See *PLoS ONE* **14** e0211463 (2019). Optional for 'SVD', cannot be used alone.

Disjoint = 128

SVD with block-diagonal sparsity preserved. Optional for 'SVD', cannot be used alone.

Length = 8192

Separate SVD for different operator widths. Optional for 'SVD' and 'Bipartite', cannot be used alone.

DisjointSVD = 130

BlockedSumDisjointSVD = 178

FastBlockedSumDisjointSVD = 186

```
BlockedRescaledSumDisjointSVD = 182

FastBlockedRescaledSumDisjointSVD = 190

BlockedDisjointSVD = 146

FastBlockedDisjointSVD = 154

BlockedRescaledDisjointSVD = 150

FastBlockedRescaledDisjointSVD = 158

RescaledDisjointSVD = 134

FastDisjointSVD = 138

FastRescaledDisjointSVD = 142

ConstrainedSVD = 66

BlockedSumConstrainedSVD = 114

FastBlockedSumConstrainedSVD = 122

BlockedRescaledSumConstrainedSVD = 118

FastBlockedRescaledSumConstrainedSVD = 126

BlockedConstrainedSVD = 82

FastBlockedConstrainedSVD = 90

BlockedRescaledConstrainedSVD = 86

FastBlockedRescaledConstrainedSVD = 94

RescaledConstrainedSVD = 70

FastConstrainedSVD = 74

FastRescaledConstrainedSVD = 78

BlockedSumSVD = 50

FastBlockedSumSVD = 58

BlockedRescaledSumSVD = 54

FastBlockedRescaledSumSVD = 62

BlockedSumBipartite = 49

FastBlockedSumBipartite = 57

BlockedSVD = 18
```

FastBlockedSVD = 26

Fast algorithm for blocked SVD.

BlockedRescaledSVD = 22

FastBlockedRescaledSVD = 30

BlockedLengthSVD = 8210

FastBlockedLengthSVD = 8218

BlockedBipartite = 17

FastBlockedBipartite = 25

RescaledSVD = 6

FastSVD = 10

FastRescaledSVD = 14

FastBipartite = 9

Fast algorithm for bipartite (recommended).

NC = 256

Normal-Complementary (NC) partition in conventional quantum chemistry DMRG. Optional for 'Conventional', cannot be used alone.

CN = 512

Complementary-Normal (CN) partition in conventional quantum chemistry DMRG. Optional for 'Conventional', cannot be used alone.

Conventional = 1024

Mixed NC/CN partition in conventional quantum chemistry DMRG. See Sec. III.A in *J. Chem. Phys.* **154**, 224116 (2021).

ConventionalNC = 1280

Normal-Complementary (NC) partition in conventional quantum chemistry DMRG. See Eq. (B5) in *J. Chem. Phys.* **154**, 224116 (2021).

ConventionalCN = 1536

Complementary-Normal (CN) partition in conventional quantum chemistry DMRG. See Eq. (B6) in *J. Chem. Phys.* **154**, 224116 (2021).

NoTranspose = 2048

Allow different bra and ket in conventional quantum chemistry MPO. Optional for 'Conventional', cannot be used alone.

NoRIntermed = 4096

Not use R complementary operator intermediates in conventional quantum chemistry MPO. Optional for 'Conventional', cannot be used alone.

NoTransConventional = 3072

NoTransConventionalNC = 3328

NoTransConventionalCN = 3584

NoRIntermedConventional = 5120

NoTransNoRIntermedConventional = 7168

class pyblock2.driver.core.**NPDMAlgorithmTypes**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: IntFlag

Enumeration of strategies for computing N-particle density matrices (NPDM). See Sec. II.E.1 in *J. Chem. Phys.* **159**, 234801 (2023). Items can be combined using operator |.

Nothing = 0

No algorithm specified (invalid).

SymbolFree = 1

Symbol-free NPDM algorithm.

Normal = 2

Normal algorithm with explicit symbols. Conflict with 'SymbolFree'.

Fast = 4

Fast algorithm with explicit symbols. Conflict with 'SymbolFree'.

Compressed = 8

Reduced disk storage. Optional for 'SymbolFree', cannot be used alone.

LowMem = 16

Reduced memory usage. Optional for 'SymbolFree', cannot be used alone.

Default = 9

Same as NPDMAlgorithmTypes.SymbolFree | NPDMAlgorithmTypes.Compressed (recommended).

Conventional = 32

Old manual implementation for 1pdm and 2pdm (less efficient).

class pyblock2.driver.core.**STTypes**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: IntFlag

Enumeration of truncation of expression in DMRG for similarity-transformed Hamiltonians. See Sec. II.D.3.iii in *J. Chem. Phys.* **159**, 234801 (2023).

H = 1

Terms in H.

HT = 2
Terms in [H, T].

HT2T2 = 4
Terms in 1/2 [[H, T2], T2].

HT1T2 = 8
Terms in 1/2 [[H, T1], T1] + [[H, T2], T1].

HT1T3 = 16
Terms in [[H, T3], T1].

HT2T3 = 32
Terms in [[H, T3], T2].

H_HT = 3
Sum of 'H' and 'HT'.

H_HT_HT2T2 = 7
Sum of 'H', 'HT' and 'HT2T2'.

H_HT_HTT = 15
Sum of 'H', 'HT', 'HT2T2', and 'HT1T2'.

H_HT_HTT_HT1T3 = 31
Sum of 'H_HT_HTT', and 'HT1T3'.

H_HT_HTT_HT2T3 = 47
Sum of 'H_HT_HTT', and 'HT2T3'.

H_HT_HTT_HT3 = 63
Sum of 'H_HT_HTT', 'HT1T3', and 'HT2T3'.

H_HT_HT2T2_HT1T3 = 23
Sum of 'H_HT_HT2T2', and 'HT1T3'.

H_HT_HT2T2_HT2T3 = 39
Sum of 'H_HT_HT2T2', and 'HT2T3'.

H_HT_HT2T2_HT3 = 55
Sum of 'H_HT_HT2T2', 'HT1T3', and 'HT2T3'.

class pyblock2.driver.core.**Block2Wrapper**(*symm_type=SymmetryTypes.SU2*)

Bases: object

Wrapper for low-level block2 C++ type bindings for different symmetries.

Attributes:

symm_type
[SymmetryTypes] The symmetry/floating point number mode.

b
[module] The block2 module.

bx
[module] Sub-module of `block2` for the floating point number type.

bc
[module] Sub-module of `block2` for the complex variant of the floating point number type. For example, for `bx = block2 or block2.cpx`, `bc = block2.cpx`.

bs
[module] Sub-module of `block2` for the symmetry mode and floating point number types. For example, when `symm_type = SymmetryTypes.SU2`, `bs = block2.su2`.

brs
[module] Sub-module of `block2` for the symmetry mode and the real variant of the floating point number types. For example, when `symm_type = SymmetryTypes.SU2`, `brs = block2.su2`.

bcs
[module] Sub-module of `block2` for the symmetry mode and the complex variant of the floating point number types. For example, when `symm_type = SymmetryTypes.SU2`, `brs = block2.cpx.su2`.

SX
[type] The quantum number type. For example, when `symm_type = SymmetryTypes.SU2`, `SX = block2.SU2`.

VectorFL
[type] Vector of the floating point number type.

VectorFP
[type] Vector of the real variant of the floating point number type. For example, for `FL = complex<double>/double`, `FP = double`.

VectorPG
[type] Vector of the point group irrep integer type.

VectorSX
[type] Vector of the quantum number type. For example, when `symm_type = SymmetryTypes.SU2`, `VectorSX = block2.VectorSU2`.

VectorVectorSX
[type] Vector of Vector of the quantum number type. For example, when `symm_type = SymmetryTypes.SU2`, `VectorSX = block2.VectorVectorSU2`.

__init__(*symm_type=SymmetryTypes.SU2*)

set_symmetry_groups(**args*)

Set the combination of symmetry sub-groups for `symm_type = SAny`.

Args:

args
[list[str]] List of names of (Abelian) symmetry groups. `0 <= len(args) <= 6` is required. Possible sub-group names are “U1”, “Z1”, “Z2”, “Z3”, . . . , “Z2055”,

"U1Fermi", "Z1Fermi", "Z2Fermi", "Z3Fermi", ..., "Z2055Fermi", "LZ", and "AbelianPG".

class pyblock2.driver.core.**DMRGDriver**(*stack_mem=1073741824*, *scratch='./nodex'*, *clean_scratch=True*, *restart_dir=None*, *n_threads=None*, *n_mkl_threads=1*, *symm_type=SymmetryTypes.SU2*, *mpi=None*, *stack_mem_ratio=0.4*, *fp_codec_cutoff=1e-16*)

Bases: `object`

Simple Python interface for DMRG calculations.

Attributes:

symm_type
[`SymmetryTypes`] The symmetry/floating point number mode.

bw
[`Block2Wrapper`] Wrapper for low-level type bindings.

mpi
[MPICommunicator or None] MPI Communicator.

__init__(*stack_mem=1073741824*, *scratch='./nodex'*, *clean_scratch=True*, *restart_dir=None*, *n_threads=None*, *n_mkl_threads=1*, *symm_type=SymmetryTypes.SU2*, *mpi=None*, *stack_mem_ratio=0.4*, *fp_codec_cutoff=1e-16*)

Initialize `DMRGDriver`.

Args:

symm_type
[`SymmetryTypes`] The symmetry/floating point number mode. Default: `SymmetryTypes.SU2`.

scratch
[str] The working directory (scratch space). Default is "./nodex".

clean_scratch
[bool] If True, large temporary files in the scratch space will be removed once the DMRG finishes successfully. MPS files will not be removed. Default is True.

restart_dir
[None or str] If not None, MPS will be copied to the given directory after each DMRG sweep. Default is None (MPS will not be copied).

n_threads
[None or int] Number of threads. When MPI is used, this is the number of threads for each MPI processor. Default is None, and the max number of threads available on this node will be used.

n_mkl_threads
[int] Number of threads for parallelization inside MKL (for dense matrix

multiplication). `n_mkl_threads` should be a factor of `n_threads`. When `n_mkl_threads` is not 1, nested threading will be used. Default is 1.

mpi
[None or bool] If True, MPI parallelization is used. If False or None, serial implementation is used. Default is None.

stack_mem
[int] The memory used for storing renormalized operators (in bytes). Default is 1 GB. When MPI is used, this is the number per MPI processor. Note that this argument is only responsible for part of the memory consumption in `block2`. The other part will be dynamically determined.

stack_mem_ratio
[float] The fraction of stack space occupied by the main stacks. Default is 0.4.

fp_codec_cutoff
[float] Floating-point number (absolute) precision for compressed storage of renormalized operators. Default is 1E-16.

property symm_type

The symmetry/floating point number mode.

property scratch

The working directory (scratch space).

property restart_dir

If not None, MPS will be copied to the given directory after each DMRG sweep.

set_symm_type(*symm_type*, *reset_frame=True*)

Change the symmetry type of this `DMRGDriver`.

Args:

symm_type
[`SymmetryTypes`] The symmetry/floating point number mode. Default: `SymmetryTypes.SU2`.

reset_frame
[bool] Whether the data frame should be reset. This is required to be True after switching between single precision and double precision. Default is True.

set_symmetry_groups(**args*)

Set the combination of symmetry sub-groups for `symm_type = SAny`.

Args:

args
[list[str]] List of names of (Abelian) symmetry groups. `0 <= len(args) <= 6` is required. Possible sub-group names are "U1", "Z1", "Z2", "Z3", ..., "Z2055", "U1Fermi", "Z1Fermi", "Z2Fermi", "Z3Fermi", ..., "Z2055Fermi", "LZ", and "AbelianPG".

initialize_system(*n_sites*, *n_elec=0*, *spin=0*, *pg_irrep=None*, *orb_sym=None*, *heis_twos=-1*, *heis_twosz=0*, *singlet_embedding=True*, *pauli_mode=False*, *vacuum=None*, *left_vacuum=None*, *target=None*, *hamil_init=True*)

Set the basic information of the system. Required before invoking `self.get_mpo`, `self.get_random_mps`, etc. Note that one can directly set the `target` and `vacuum`, so that `n_elec`, `spin`, `pg_irrep`, `heis_twos`, and `heis_twosz` are not required.

Args:

n_sites
[int] Number of sites (orbitals).

n_elec
[int] Number of electrons int the target state.

spin
[int] Two times total spin (in SU2 mode) or two times projected spin (in SZ mode) int the target state.

pg_irrep
[None or int] Point group irreducible representation in the target state. 0 is the trivial point group irrep.

orb_sym
[None or list[int]] The point group irreducible representation of each site (orbital). If None, this is `[0] * n_sites`.

heis_twos
[int] For non-Heisenberg model, this should be -1 (default). For Heisenberg model, this is two times the total spin of each site. For example, `heis_twos = 1` for the most common spin-1/2 Heisenberg model.

heis_twosz
[int] For Heisenberg model in SGB mode, `spin` is not used and `heis_twosz` is used to specify two times the projected spin of the target state. Default is zero. Not used for non-Heisenberg model.

singlet_embedding
[bool] Whether singlet embedding is used for non-singlet target state in SU2 mode. Default is True. Note that when this is True, `DMRGDriver.target` will always be the singlet even if the actual target state is not singlet.

pauli_mode
[bool] Whether Pauli mode should be activated. Default is False. When Pauli mode is activated, quantum numbers are not used. Only `n_sites` is required in `initialize_system`. The `SGB` or `SGB|CPX` symmetry types are required for this case. In Pauli mode one can use `DMRGDriver.get_mpo_any_pauli` to construct MPO from XYZ Pauli operators.

vacuum
[SX or None] The quantum number of the reference vacuum state. If None, will be set according to other parameters. Default is None.

target
[SX or None] The quantum number of the target state. If None, will be set according to other parameters. Default is None.

left_vacuum
[SX or None] For non-singlet states in non-Abelian symmetry modes (such as SU2), and when `singlet_embedding = True`, `left_vacuum` is an adjusted vacuum to represent the non-singlet fictitious non-singlet spin. For most cases, this can be automatically determined and stored as `DMRGDriver.left_vacuum`. For Abelian symmetry mode or singlet states in non-Abelian symmetry modes, this should be equal to `vacuum`.

hamil_init
[bool] Whether the Hamiltonian object `DMRGDriver.ghamil` should be initialized. Default is True. When a custom symmetry type is used and `symm_type = SymmetryTypes.SAny`, one can set this to False and manually initialize `DMRGDriver.ghamil` later using the return value of `DMRGDriver.get_custom_hamiltonian`.

divide_nprocs(*n*)

Helper method for two-level MPI parallelization. This purpose of this method is to almost evenly divide n procs to two levels. Faster than a pure sqrt method when `n >= 20000000`.

Args:

n
[int] Number of MPI processors.

Returns:

(a, b)
[tuple[int, int]] Number of MPI processors at two levels such that `a * b == n` and `abs(a - b)` is minimized.

parallelize_integrals(*para_type*, *h1e*, *g2e*, *const*, *msize=None*, *mrank=None*)

Prepare integrals for parallel quantum chemistry DMRG.

Args:

para_type
[ParallelTypes] The strategy for distributing the integrals (when `self.mpi` is not None).

h1e
[np.ndarray[float|complex]] `ndim = 2` one-electron integral (serial/complete version).

g2e
[np.ndarray[float|complex]] `ndim = 4` unpacked two-electron integral (serial/complete version).

const
[float|complex] Constant term.

msize
[None or int] Total number of MPI processors. If None, will be determined from `self.mpi.size`. Default is None.

mrank
[None or int] The rank of current MPI processor. If None, will be determined from `self.mpi.rank`. Default is None.

Returns:

h1e
[np.ndarray[float|complex]] One-electron integral with elements belonging to other processors set to zero.

g2e
[np.ndarray[float|complex]] Two-electron integral with elements belonging to other processors set to zero.

const
[float or complex] Constant energy (non-zero only at the root processor).

get_pauli_hamiltonian()
Internal method for setting Hamiltonian in the Pauli/SGB mode.

get_so4_hamiltonian()
Internal method for setting Hamiltonian in the SAnySO4 mode.

get_phsu2_hamiltonian()
Internal method for setting Hamiltonian in the SAnyPHSU2 mode.

get_so3_hamiltonian()
Internal method for setting Hamiltonian in the SAnySO3 mode.

get_lz_hamiltonian()
Internal method for setting Hamiltonian in the SAnySU2LZ/SAnySZLZ/SAnySGFLZ modes.

get_custom_hamiltonian(*site_basis*, *site_ops*, *orb_dependent_ops='cdCD'*)
Setting Hamiltonian in the general symmetry mode. `SZ` or `SAny` symmetry mode is required.

Args:

site_basis
[list[list[tuple[SX, int]]]] The set of quantum numbers (SX) and number of states (int) in the local Hilbert space at each site.

site_ops
[list[dict[str, np.ndarray[float|complex]]]] The matrix representation of elementary operators in the local Hilbert space at each site. Matrices must have `ndim == 2`. The indices of rows and columns correspond to the list given in `site_basis`. For example, When `site_basis[0] = [(Q1, 2), (Q2, 3), (Q3, 1)]`, each matrix in `site_ops[0]` should have shape `(6, 6)` where the first 2

rows/columns correspond to the Q1 block, the next 3 rows/columns correspond to the Q2 block, etc. The operator name can only have one character.

orb_dependent_ops
[str] List of operator names that can have point group irrep. If point group or `orb_sym` is not used, this can be empty. Default is "cdCD".

Returns:

ghamil
[CustomHamiltonian] The Hamiltonian object, implicitly required for MPO and MPS construction.

write_fcidump(*h1e*, *g2e*, *ecore=0*, *filename=None*, *h1e_symm=False*, *pg='d2h'*)

Write quantum chemistry integrals as a FCIDUMP format file. Supports SZ, SU2, and SGF modes.

Args:

h1e
[np.ndarray[float|complex] or list[np.ndarray[float|complex]]] `ndim = 2` one-electron integral. For SZ mode, this is a list/tuple of two `np.ndarray`, for the aa and bb components, respectively.

g2e
[np.ndarray[float|complex] or list[np.ndarray[float|complex]] or None] `ndim = 4 or 2 or 1` unpacked/packed two-electron integral. For SZ mode, this is a list/tuple of three `np.ndarray`, for the aa, ab, and bb components, respectively.

ecore
[float or complex] Constant energy. Default is 0.

filename
[str or None] If not None, will write the integrals in the FCIDUMP format to a file named `filename`. Otherwise, no files will be written. Default is None.

h1e_symm
[bool] If True, the `h1e` is assumed symmetric/Hermitian and only the triangular part of it will be stored. This is the standard FCIDUMP format. If False, the full h1e will be stored, which will ensure the correctness for non-Hermitian/anti-Hermitian integrals. Default is False. Set to True if this FCIDUMP format will be read by other programs.

pg
[str] Point group name. The MOLPRO convention for orb_sym is required for FCIDUMP files. This point group name is used to translate from the XOR convention (used by block2) to MOLPRO convention.

Returns:

fcidump
[FCIDUMP] The block2 fcidump object.

read_fcidump(*filename*, *pg='d2h'*, *rescale=None*, *iprint=1*)

Read the quantum chemistry integrals from a FCIDUMP format file. Supports SZ, SU2, and SGF modes. When this method returns, `self.h1e`, `self.g2e`, `self.ecore`, `self.n_sites`, `self.n_elec`, `self.spin`, `self.pg_irrep`, and `self.pg` will be set.

Args:

filename
[str] The name of the FCIDUMP format file.

pg
[str] Point group name, used to translate from the MOLPRO convention (used in FCIDUMP format) to the XOR convention (used by block2).

rescale
[None or float or True] If None, will not rescale (default). If zero or True, will adjust `h1e` and the const energy so that the average diagonal element of `h1e` is zero. If non-zero float, will adjust `h1e` and the const energy so that the const energy becomes the given `rescale` number. After rescale, the integrals will only be correct for the given `n_elec`.

iprint
[int] Verbosity. Default is 1.

Returns:

fcidump
[FCIDUMP] The block2 fcidump object.

su2_to_sgf()

Transform the spin-restricted integrals `h1e` and `g2e` to general spin orbital integrals `h1e` and `g2e`. Assuming `self.h1e` and `self.g2e` available and unpacked. The transformed integrals will be stored as `self.h1e` and `self.g2e`. This will also change `self.n_sites` and `self.orb_sym` (if available) accordingly.

integral_symmetrize(*orb_sym*, *h1e=None*, *g2e=None*, *hxe=None*, *k_symm=None*, *iprint=1*)

Symmetrize quantum chemistry integrals so that all elements violating point group restrictions are set to zero. Integrals (if not None) will be changed inplace.

Args:

orb_sym
[list[int]] The point group irreducible representation of each site (orbital), or the K-space irreducible representation of each site (orbital) if `k_symm` is not None.

h1e
[np.ndarray[float|complex] or None] `ndim = 2` one-electron integral.

g2e
[np.ndarray[float|complex] or None] `ndim = 4` unpacked two-electron integral.

hxe
[np.ndarray[float|complex] or None] Arbitrary `ndim` integrals or amplitudes.

k_symm
[None or True] If None, `orb_sym` is understood as point group irreps. Otherwise, `orb_sym` is understood as K-space irreps.

iprint
[int] Verbosity. Default is 1.

get_conventional_qc_mpo(*fcidump*, *algo_type=None*, *iprint=1*)

Construct MPO for quantum chemistry Hamiltonian, using conventional methods. This method cannot take care of MPI parallelization (one may use `self.parallelize_integrals` to parallelize the integrals before invoking this method).

Args:

fcidump
[FCIDUMP] The block2 fcidump object.

algo_type
[MPOAlgorithmTypes or None] Strategies for building MPO from symbolic expression of second quantized operators. Only the `Conventional|NC|CN` based algorithms are accepted in this method. If None (default), `MPOAlgorithmTypes.Conventional` will be used.

iprint
[int] Verbosity. Default is 1.

Returns:

mpo
[MPO] The block2 MPO object.

get_identity_mpo(*ancilla=False*)

Construct MPO for the identity operator.

Args:

ancilla
[bool] If True, will generate identity MPO including ancilla sites (used in finite temperature DMRG). Default is False.

Returns:

mpo
[MPO] The block2 MPO object.

unpack_g2e(*g2e*, *n_sites=None*)

Unfold the 8-fold or 4-fold or 1-fold two-electron integral to the 1-fold (unpacked) two-electron integral.

Args:

g2e
[np.ndarray[float|complex]] `ndim = 4 or 2 or 1` packed/unpacked two-electron integral.

n_sites
[int or None] Number of sites (orbitals). If None, will look at `self.n_sites`.

Returns:

g2e
[np.ndarray[float|complex]] `ndim = 4` unpacked two-electron integral.

get_qc_mpo(*h1e*, *g2e*, *ecore=0.0*, *para_type=None*, *reorder=None*, *cutoff=1e-20*, *integral_cutoff=1e-20*, *post_integral_cutoff=1e-20*, *fast_cutoff=1e-20*, *unpack_g2e=True*, *algo_type=None*, *normal_order_ref=None*, *normal_order_single_ref=None*, *normal_order_wick=True*, *symmetrize=True*, *sum_mpo_mod=-1*, *compute_accurate_svd_error=True*, *csvd_sparsity=0.0*, *csvd_eps=1e-10*, *csvd_max_iter=1000*, *disjoint_levels=None*, *disjoint_all_blocks=False*, *disjoint_multiplier=1.0*, *block_max_length=False*, *add_ident=True*, *esptein_nesbet_partition=False*, *ancilla=False*, *reorder_imat=None*, *gaopt_opts=None*, *iprint=1*)

Construct MPO from integrals in a quantum chemistry Hamiltonian.

For quantum chemistry Hamiltonians, the unpacked 2-electron integral `g2e` uses chemists' notation.

In SU2 symmetry (spin restricted) mode, the quantum chemistry Hamiltonian is given by

$$H = \sum_{\sigma,ij} [\mathrm{h1e}]_{ij}\ a^\dagger_{i\sigma} a_{j\sigma} + \frac{1}{2} \sum_{\sigma\sigma',ijkl} [\mathrm{g2e}]_{ijkl}\ a^\dagger_{i\sigma} a^\dagger_{k\sigma'} a_{l\sigma'} a_{j\sigma} + \mathrm{ecore}$$

In SZ symmetry (spin unrestricted) mode, the quantum chemistry Hamiltonian is given by

$$H = \sum_{\sigma,ij} [\mathrm{h1e}]_{\sigma,ij}\ a^\dagger_{i\sigma} a_{j\sigma} + \frac{1}{2} \sum_{\sigma\sigma',ijkl} [\mathrm{g2e}]_{\sigma\sigma',ijkl}\ a^\dagger_{i\sigma} a^\dagger_{k\sigma'} a_{l\sigma'} a_{j\sigma} + \mathrm{ecore}$$

with

$$\begin{aligned}
[\mathrm{h1e}][0] &= [\mathrm{h1e}]_\alpha \\
[\mathrm{h1e}][1] &= [\mathrm{h1e}]_\beta \\
[\mathrm{g2e}][0] &= [\mathrm{g2e}]_{\alpha\alpha} \\
[\mathrm{g2e}][1] &= [\mathrm{g2e}]_{\alpha\beta} \\
[\mathrm{g2e}][2] &= [\mathrm{g2e}]_{\beta\beta}
\end{aligned}$$

In SGF symmetry (general spin) mode, the quantum chemistry Hamiltonian is given by

$$H = \sum_{ij} [\mathrm{h1e}]_{ij}\ a^\dagger_i a_j + \frac{1}{2} \sum_{ijkl} [\mathrm{g2e}]_{ijkl}\ a^\dagger_i a^\dagger_k a_l a_j + \mathrm{ecore}.$$

Args:

h1e
[np.ndarray[float|complex] or list[np.ndarray[float|complex]]] `ndim = 2` one-electron integral. For SZ mode, this can be a list/tuple of two `np.ndarray`, for

the aa and bb components, respectively. For SZ mode, if only one np.ndarray is given, the two components will be assumed the same.

g2e
[np.ndarray[float|complex] or list[np.ndarray[float|complex]]] ndim = 4 or 2 or 1 unpacked/packed two-electron integral. For SZ mode, this can be a list/tuple of three np.ndarray, for the aa, ab, and bb components, respectively. For SZ mode, if only one np.ndarray is given, the three components will be assumed the same.

ecore
[float or complex] Constant term. Default is 0.

para_type
[ParallelTypes or None] The strategy for distributing the integrals (when self.mpi is not None). If None, the strategy ParallelTypes.SIJ will be used. Default is None. If the input integrals are already manually distributed, one should set this to ParallelTypes.Nothing. This argument has no effect if MPI is not activated (namely, when self.mpi is None).

reorder
[None or str or np.ndarray[int] or True] The strategy for site/orbital reordering. If None, orbital will not be reordered. If np.ndarray[int], the orbital will be reordered using the given permutation. If this is "irrep", orbitals with the same point group irrep will be put together. If this is "fiedler" or True, the fiedler method will be used. See also reorder_imat. If this is "gaopt", the genetic algorithm will be used to find the optimal orbital reordering, using the "fiedler" ordering as the initial guess. See also reorder_imat and gaopt_opts. Note that this argument will perform the orbital reordering implicitly. This implicit ordering can be recognized by DMRGDriver.dmrg and DMRGDriver.get_npdm but it may not be compatible to some other operations in DMRGDriver. It is recommended to manually to perform the reordering on the integrals h1e, g2e and orb_sym before invoking this method to prevent any ambiguity.

cutoff
[float] Cutoff of singular values when MPOAlgorithmTypes.SVD is used for MPO construction. Default is 1E-20.

integral_cutoff
[float] Cutoff of individual elements in the integrals. Default is 1E-20.

post_integral_cutoff
[float] Cutoff of individual elements in the transformed integrals. Default is 1E-20. Only have effect if normal_order_ref is not None.

fast_cutoff
[float] Cutoff of individual elements in the integrals, implemented using C++ code. Default is 1E-20. This is intended to be used when the integrals are very large and any copying or unpacking of the integrals should be avoided to save memory. This should be used together with unpack_g2e = False, integral_cutoff = 0 and symmetrize = False to avoid copying or unpacking. Default is 1E-20. Only have effect if normal_order_ref is None.

unpack_g2e
[bool] Whether the `g2e` should be unpacked (using the Python code). Setting this to False may save some amount of memory, but many other operations on the integrals (including symmetrization and distributed parallelization) may fail. Default is True.

algo_type
[None or MPOAlgorithmTypes] Strategies for building MPO from symbolic expression of second quantized operators. If None, `MPOAlgorithmTypes.FastBipartite` will be used (default).

normal_order_ref
[None or np.ndarray[bool]] If not None, the integrals will be normal ordered, using `normal_order_ref` as the reference. Elements in `normal_order_ref` indicate whether the orbital is doubly occupied (True) or empty/singly occupied (False) in the reference state. Default is None (the integrals will not be normal ordered).

normal_order_single_ref
[None or np.ndarray[bool]] Only have effect if `normal_order_ref is not None`. Elements in `normal_order_single_ref` indicate whether the orbital is singly occupied (True) or empty/doubly occupied (False) in the reference state.

normal_order_wick
[bool] Only have effect if `normal_order_ref is not None`. If True, will use `WickNormalOrder` implementation (via automatic symbolic derivation). Otherwise, will use the manual implementation. Default is True.

symmetrize
[bool] Only have effect if `self.orb_sym is not None` (when point group symmetry is used). If True, will symmetrize integrals so that integral elements violating point group restrictions are set to zero. Default is True. May generate runtime error during DMRG when point group is used but the integrals are not symmetrized.

sum_mpo_mod
[int] Only have effect if `MPOAlgorithmTypes.Sum` modifier appears in `algo_type`. Set the denominator for grouping indices in the sum of MPO approach. When this is -1, indices will not be grouped. Default is -1.

compute_accurate_svd_error
[bool] Only have effect if `MPOAlgorithmTypes.SVD` appears in `algo_type`. If True, will compute and print the accurate error due to SVD truncation by comparing the difference between the original tensor and the contraction of its decomposed parts. Setting this to False may reduce some time cost of the SVD approach. Default is True.

csvd_sparsity
[float] Only have effect if `MPOAlgorithmTypes.Constrained` modifier appears in `algo_type`. Sparsity for constrained SVD. Default is 0.0.

csvd_eps

[float] Only have effect if `MPOAlgorithmTypes.Constrained` modifier appears in `algo_type`. Threshold for constrained SVD. Default is 1E-10.

csvd_max_iter
[int] Only have effect if `MPOAlgorithmTypes.Constrained` modifier appears in `algo_type`. Maximal iteration number for constrained SVD. Default is 1000.

disjoint_levels
[None or list[float]] Only have effect if `MPOAlgorithmTypes.Disjoint` modifier appears in `algo_type`. Threshold for finding connected elements at each level. Default is None.

disjoint_all_blocks
[bool] Only have effect if `MPOAlgorithmTypes.Disjoint` modifier appears in `algo_type`. If False, only SVD for part of blocks will be done using disjoint SVD. Default is False.

disjoint_multiplier
[float] Only have effect if `MPOAlgorithmTypes.Disjoint` modifier appears in `algo_type`. Allowing the number of singular values to exceed the maximal number, but no more than `disjoint_multiplier` times the maximal number. Default is 1.0.

block_max_length
[bool] Only have effect if `MPOAlgorithmTypes.SVD` or `MPOAlgorithmTypes.Bipartite` appears in `algo_type`. If True, will separate the SVD or Bipartite for one- and two-electron integrals. Default is False.

add_ident
[bool] If True, the hidden identity operator will be added into the MPO. This is required when `ecore` is not zero and `DMRGDriver.expectation` will be invoked using this MPO. Default is True. One needs to set this to False to allow the MPO to be transformed into the Python format.

esptein_nesbet_partition
[bool] If True, will only keep the "diagonal" part of the integrals for building MPO. This can be used to build the MPO for the zeroth-order Hamiltonian in the Esptein-Nesbet perturbation theory (used in perturbative DMRG). Default is False.

ancilla
[bool] If True, will insert ancilla sites in the MPO, which can then be used for finite-temperature DMRG. Default is False.

reorder_imat
[None or np.ndarray[float]] Only have effect when `reorder == "fiedler"` or `reorder == "gaopt"`. The orbital interaction matrix (`ndim == 2`) used for computing the cost function for orbital reordering. If None, the exchange integral `Kij` will be used.

gaopt_opts
[dict or None] Only have effect when `reorder == "gaopt"`. Custom op-

tions for the genetic orbital ordering algorithm. Possible keys are `n_tasks`, `n_generations`, `n_configs`, `n_elite`, `clone_rate`, and `mutate_rate`.

iprint
[int] Verbosity. Default is 1.

Returns:

mpo
[MPO] The block2 MPO object.

get_mpo(*expr*, *iprint=0*, *cutoff=1e-14*, *left_vacuum=None*, *algo_type=None*, *sum_mpo_mod=-1*, *compute_accurate_svd_error=True*, *csvd_sparsity=0.0*, *csvd_eps=1e-10*, *csvd_max_iter=1000*, *disjoint_levels=None*, *disjoint_all_blocks=False*, *disjoint_multiplier=1.0*, *block_max_length=False*, *add_ident=True*, *ancilla=False*)

Construct MPO from arbitrary symbolic expression of second quantized operators.

Args:

expr
[GeneralFCIDUMP] The block2 GeneralFCIDUMP object. This is often obtained from the `ExprBuilder.finalize` method.

iprint
[int] Verbosity. Default is 0 (quiet).

cutoff
[float] Cutoff of singular values when `MPOAlgorithmTypes.SVD` is used for MPO construction. Default is 1E-20.

left_vacuum
[SX or None] The `left_vacuum` of MPO. If None, this will be automatically determined.

algo_type
[None or MPOAlgorithmTypes] Strategies for building MPO from symbolic expression of second quantized operators. If None, `MPOAlgorithmTypes.FastBipartite` will be used (default).

sum_mpo_mod
[int] Only have effect if `MPOAlgorithmTypes.Sum` modifier appears in `algo_type`. Set the denominator for grouping indices in the sum of MPO approach. When this is -1, indices will not be grouped. Default is -1.

compute_accurate_svd_error
[bool] Only have effect if `MPOAlgorithmTypes.SVD` appears in `algo_type`. If True, will compute and print the accurate error due to SVD truncation by comparing the difference between the original tensor and the contraction of its decomposed parts. Setting this to False may reduce some time cost of the SVD approach. Default is True.

csvd_sparsity
[float] Only have effect if `MPOAlgorithmTypes.Constrained` modifier appears in

algo_type. Sparsity for constrained SVD. Default is 0.0.

csvd_eps
[float] Only have effect if MPOAlgorithmTypes.Constrained modifier appears in algo_type. Threshold for constrained SVD. Default is 1E-10.

csvd_max_iter
[int] Only have effect if MPOAlgorithmTypes.Constrained modifier appears in algo_type. Maximal iteration number for constrained SVD. Default is 1000.

disjoint_levels
[None or list[float]] Only have effect if MPOAlgorithmTypes.Disjoint modifier appears in algo_type. Threshold for finding connected elements at each level. Default is None.

disjoint_all_blocks
[bool] Only have effect if MPOAlgorithmTypes.Disjoint modifier appears in algo_type. If False, only SVD for part of blocks will be done using disjoint SVD. Default is False.

disjoint_multiplier
[float] Only have effect if MPOAlgorithmTypes.Disjoint modifier appears in algo_type. Allowing the number of singular values to exceed the maximal number, but no more than disjoint_multiplier times the maximal number. Default is 1.0.

block_max_length
[bool] Only have effect if MPOAlgorithmTypes.SVD or MPOAlgorithmTypes.Bipartite appears in algo_type. If True, will separate the SVD or Bipartite for one- and two-electron integrals. Default is False.

add_ident
[bool] If True, the hidden identity operator will be added into the MPO. This is required when ecore is not zero and DMRGDriver.expectation will be invoked using this MPO. Default is True. One needs to set this to False to allow the MPO to be transformed into the Python format.

ancilla
[bool] If True, will insert ancilla sites in the MPO, which can then be used for finite-temperature DMRG. Default is False.

Returns:

mpo
[MPO] The block2 MPO object.

get_site_mpo(*op*, *site_index*, *iprint=1*)

Construct MPO from the creation (C) or destroy (D) operator on a single site. Supports SU2, SZ, and SGF modes.

Args:

op
[str] The name of the operator.

site_index
[int] The index (subscript) of the operator.

iprint
[int] Verbosity. Default is 1.

Returns:

mpo
[MPO] The block2 MPO object.

get_spin_square_mpo(*iprint=1*)

Construct MPO for the S^2 operator where S is the total spin operator. Supports SU2, SZ, and SGF modes.

Args:

iprint
[int] Verbosity. Default is 1.

Returns:

mpo
[MPO] The block2 MPO object.

get_mpo_any_fermionic(*op_list*, *ecore=None*, ***kwargs*)

Construct MPO from a list of second quantized fermionic operators and coefficients. Supports the SGF mode only.

Args:

op_list
[list[tuple[str, float]]] A list of second quantized fermionic operators and coefficients. `+` is creation, `-` is destroy. For example,

```
[ ('+_3 +_4 -_1 -_3', 0.0068705380508780715),
  ('+_3 +_4 -_4 -_3', -0.009852150878546906) ]
```

ecore
[float|complex or None] Constant term. Default is None (0.0).

kwargs
[dict] Other options that should be passed to `DMRGDriver.get_mpo`.

Returns:

mpo
[MPO] The block2 MPO object.

get_mpo_any_pauli(*op_list*, *ecore=None*, ***kwargs*)

Construct MPO from a list of Pauli strings and coefficients. Supports the SGB mode with the `pauli_mode = True` in `DMRGDriver.initialize_system` only.

Args:

op_list
[list[tuple[str, float]]] A list of Pauli strings and coefficients. Characters in the string can be IXYZ. For example,

```
[ ('IIXXXIIX', 0.0559742284070319),
  ('IIIXIIXZ', 0.0018380565450674) ]
```

ecore
[float|complex or None] Constant term. Default is None (0.0).

kwargs
[dict] Other options that should be passed to `DMRGDriver.get_mpo`.

Returns:

mpo
[MPO] The block2 MPO object.

orbital_reordering(*h1e*, *g2e*, *method='fiedler'*, ***kwargs*)

Find optimal orbital ordering for integrals `h1e` and `g2e`. Note that this method will not actually perform any orbital ordering. The exchange integral `Kij` (constructed from the given `h1e` and `g2e`) will be used for computing the cost function.

Args:

h1e
[np.ndarray[float|complex]] `ndim = 2` one-electron integral.

g2e
[np.ndarray[float|complex]] `ndim = 4` unpacked two-electron integral.

method
[str] The algorithm name for orbital reordering. Can be "gaopt" or "fiedler" (default).

kwargs
[dict] Only have effect when `method == "gaopt"`. Custom options for the genetic orbital ordering algorithm. Possible keys are `n_tasks`, `n_generations`, `n_configs`, `n_elite`, `clone_rate`, and `mutate_rate`.

Returns:

idx
[np.ndarray[int]] Optimal orbital ordering (permutation array).

orbital_reordering_interaction_matrix(*imat*, *method='fiedler'*, ***kwargs*)

Find optimal orbital ordering using the orbital interaction matrix to compute the cost function. Note that this method will not actually perform any orbital ordering.

Args:

imat
[np.ndarray[float]] The orbital interaction matrix (`ndim == 2`) used for computing the cost function.

method
[str] The algorithm name for orbital reordering. Can be “gaopt” or “fiedler” (default).

kwargs
[dict] Only have effect when `method == "gaopt"`. Custom options for the genetic orbital ordering algorithm. Possible keys are `n_tasks`, `n_generations`, `n_configs`, `n_elite`, `clone_rate`, and `mutate_rate`.

Returns:

idx
[np.ndarray[int]] Optimal orbital ordering (permutation array).

dmrg(*mpo*, *ket*, *n_sweeps=10*, *tol=1e-08*, *bond_dims=None*, *noises=None*, *thrds=None*, *iprint=0*, *dav_type=None*, *cutoff=1e-20*, *twosite_to_onesite=None*, *dav_max_iter=4000*, *dav_def_max_size=50*, *proj_mpss=None*, *proj_weights=None*, *store_wfn_spectra=True*, *spectra_with_multiplicity=False*, *lowmem_noise=False*, *sweep_start=0*, *forward=None*)

Perform the ground state and/or excited state Density Matrix Renormalization Group (DMRG) algorithm, which finds the solution of the following optimization problem:

$$E_0 = \min_{\text{ket}} \frac{\langle \text{ket}|\text{mpo}|\text{ket}\rangle}{\langle \text{ket}|\text{ket}\rangle}.$$

Statistics during the sweeps can be obtained from `self._dmrg` after the method returns.

Args:

mpo
[MPO] The block2 MPO object.

ket
[MPS] The block2 MPS object. The given MPS `ket` will be used as the initial guess for DMRG. When this method returns, the MPS `ket` will contain the optimized (ground and/or excited) state. If `ket.nroots != 1`, state-averaged DMRG will be done to find the ground and excited states. If `ket.dot == 2`, will perform 2-site DMRG algorithm. If `ket.dot == 1`, will perform 1-site DMRG algorithm. The initial input `ket` is not required to be normalized. The output `ket` will always be normalized.

n_sweeps
[int] Maximal number of DMRG sweeps. Default is 10.

tol
[float] Energy converge threshold. If the absolute value of the total energy difference between two consecutive sweeps is below `tol`, and the `noise` for the current sweep is zero, the algorithm will terminate. Default is 1E-8.

bond_dims
[None or list[int]] List of MPS bond dimensions for each sweep. Default is None. If None, the bond dimension of the initial MPS will be used for all sweeps.

noises
[None or list[float]] List of prefactor of the noise for each sweep. Default is None. If None, this is set to `[1e-5] * 5 + [0]`. Typically, this should be zero for the last few sweeps.

thrds
[None or list[float]] List of the convergence threshold (square of the residual) of the Davidson algorithms each sweep. Default is None. If None, this is set to `[1e-6] * 4 + [1e-7] * 1` for double precision and `[1e-5] * 4 + [5e-6] * 1` for single precision.

iprint
[int] Verbosity. Default is 0 (quiet).

dav_type
[None or str] The type of the Davidson algorithm. If None, this is set to "Normal". Possible other options are "NonHermitian" (required for non-Hermitian Hamiltonian), "Exact" (constructing the full dense effective Hamiltonian and find all eigenvalues), "ExactNonHermitian", "DavidsonPrecond", and "NoPrecond". "Normal" will use the Olsen preconditioning, "DavidsonPrecond" will use the Davidson preconditioning, and "NoPrecond" will not use any preconditioning. Default is None.

cutoff
[float] States with eigenvalue below this number will be discarded, even when the bond dimension is large enough to keep this state. Default is 1E-20.

twosite_to_onesite
[None or int] If not None and `ket.dot == 2` in the initial MPS, will perform `twosite_to_onesite` 2-site sweeps and then switch to the 1-site algorithm for the remaining sweeps. Default is None (no switching).

dav_max_iter
[int] Maximal number of Davidson iteration. Default is 4000.

dav_def_max_size
[int] Maximal size of the Davidson deflation space (Krylov space). Default is 50. For very large MPS bond dimension and very small MPO bond dimension, one may reduce this number to save memory.

proj_mpss
[None or list[MPS]] If not None, the MPS given in `proj_mpss` will be projected out during sweeps. Can be used for performing state-specific excited state DMRG. Default is None.

proj_weights
[None or list[float]] The weights of the MPS projection. This should be larger than the energy gap between the targeted state and the projected state. But if this is too large, the error in the projected state will affect the quality of the targeted state.

store_wfn_spectra
[bool] If True, the MPS singular value spectra will be stored as `self.`

`_sweep_wfn_spectra` which can be later used to compute the bipartite entropy. If False, the spectra will not be computed. Default is True.

spectra_with_multiplicity
[bool] If True, in SU2 mode, the MPS singular value will be multiplied by the multiplicity of the spin quantum number. Default is False.

lowmem_noise
[bool] If True, the noise step will cost less memory. Default is False.

sweep_start
[int] The starting sweep index in `bond_dims`, `noises`, and `thrds`. Default is 0. This may be useful in restarting, when one wants to skip the sweep parameters for a few already finished sweeps.

forward
[None or bool] The direction of the first sweep (`forward == True` means the left-to-right direction). If None, will use the canonical center of MPS to determine the direction. Default is None. This may be useful in restarting.

Returns:

energy
[float|complex or list[float|complex]] When `ket.nroots == 1`, this is the ground state energy. When `ket.nroots != 1`, this is a list of ground and excited state energies.

td_dmrg(*mpo*, *ket*, *delta_t=None*, *target_t=None*, *final_mps_tag=None*, *te_type='rk4'*, *n_steps=None*, *bond_dims=None*, *n_sub_sweeps=2*, *normalize_mps=False*, *hermitian=False*, *iprint=0*, *cutoff=1e-20*, *krylov_conv_thrd=5e-06*, *krylov_subspace_size=20*)

Perform the time-dependent DMRG algorithm, which computes:

$$|\mathrm{ket}'\rangle = \exp(-t \cdot \mathrm{mpo})|\mathrm{ket}\rangle.$$

Statistics during the sweeps can be obtained from `self._te` after the method returns.

Args:

mpo
[MPO] The block2 MPO object.

ket
[MPS] The block2 MPS object at time zero. When this MPS has a very low bond dimension, this implementation of td-DMRG may be inaccurate. When this method returns, this MPS will not be modified.

delta_t
[None or float or complex] The time step. When this is complex, `SymmetryTypes.CPX` must be used. If None, will be determined using `delta_t = target_t / n_steps`.

target_t
[None or float or complex] The target time (the time parameter in the final state). If None, will be determined using `target_t = delta_t * n_steps`.

final_mps_tag
[None or str] The tag of the output MPS. If None, the output MPS tag will be `"TD-" + ket.info.tag`. One can set this to be same as the `tag` of `ket`, then the `ket` object should not be used after calling this method.

te_type
[str] The time evolution algorithm. Can be "rk4" (the time-step-targeting method) or "tdvp" (the time-dependent variational principle method).

n_steps
[None or int] The number of time steps. If None, will be determined using `n_steps = int(abs(target_t) / abs(delta_t) + 0.1)`.

bond_dims
[None or list[int]] MPS bond dimension for each time step. If None, will be set to the bond dimension of the initial MPS for all steps.

n_sub_sweeps
[int] For `te_type = "rk4"`, this is the number of sweeps performed for each time step. Default is 2.

normalize_mps
[bool] If True, MPS will be normalized after each time step. Default is False.

hermitian
[bool] If True, the `mpo` operator will be assumed to be Hermitian. Default is False. Only have effects when `te_type = "tdvp"`.

iprint
[int] Verbosity. Default is 0 (quiet).

cutoff
[float] States with eigenvalue below this number will be discarded, even when the bond dimension is large enough to keep this state. Default is 1E-20.

krylov_conv_thrd
[float] Convergence threshold (square of the residual) of the Matrix exponentiation algorithm. Default is 5E-6. Only have effects when `te_type = "tdvp"`.

krylov_subspace_size
[int] Maximal size of the Krylov space of the Matrix exponentiation algorithm. Default is 20. Only have effects when `te_type = "tdvp"`.

Returns:

final_mps
[MPS] The time evolved MPS.

get_dmrg_results()

Obtain the statistics of DMRG sweeps. This should be called after `DMRGDriver.dmrg`.

Returns:

bond_dims
[np.ndarray[int]] The MPS bond dimension for each sweep.

dws
[np.ndarray[float]] The maximal discarded weight (sum of discarded eigenvalues) for each sweep.

energies
[np.ndarray[list[float]]] The list of ground (and possibly excited) state energies at each sweep.

get_bipartite_entanglement(*ket=None*)

Compute the bipartite entanglement of the MPS.

Args:

ket
[MPS or None] The MPS. If None, will use `self._sweep_wfn_spectra` computed during the DMRG/expectation sweeps if this is called after calling `DMRGDriver.dmrg` or `DMRGDriver.expectation`. Default is None.

Returns:

bip_ent
[np.ndarray[float]] The bipartite entanglement of the MPS after each non-terminating site.

get_n_orb_rdm_mpos(*orb_type=1*, *ij_symm=True*, *iprint=0*)

Internal method for MPO construction for computing the 1- or 2-orbital reduced density matrices.

Args:

orb_type
[int] 1 or 2. Indicating whether the 1- or 2-orbital reduced density matrices should be computed. Default is 1.

ij_symm
[bool] Whether the `ij` index symmetry should be used. Default is True.

iprint
[int] Verbosity. Default is 0 (quiet).

Returns:

mpos
[dict[tuple[int...], list[MPO]]] The MPOs for computing the 1- or 2-orbital reduced density matrices.

get_orbital_entropies(*ket*, *orb_type=1*, *ij_symm=True*, *use_npdm=True*, *iprint=0*)

Compute the 1- or 2- orbital entropies for the given MPS.

Args:

ket
[MPS] The given MPS for computing orbital entropies.

orb_type
[int] 1 or 2. Indicating whether the 1- or 2-orbital reduced density matrices should be computed. Default is 1.

ij_symm
[bool] Whether the `ij` index symmetry should be used. Default is True.

use_npdm
[bool] Whether the more efficient algorithm based on N-PDM should be used. Default is True.

iprint
[int] Verbosity. Default is 0 (quiet).

Returns:

ents
[np.ndarray[float]] When `orb_type == 1`, this is `ndim == 1` vector containing the 1-orbital entropies. When `orb_type == 2`, this is `ndim == 2` matrix containing the 2-orbital entropies.

get_orbital_entropies_use_npdm(*ket*, *orb_type=1*, *iprint=0*)

Compute the 1- or 2- orbital entropies for the given MPS using the efficient algorithm based on N-PDM.

Args:

ket
[MPS] The given MPS for computing orbital entropies.

orb_type
[int] 1 or 2. Indicating whether the 1- or 2-orbital reduced density matrices should be computed. Default is 1.

iprint
[int] Verbosity. Default is 0 (quiet).

Returns:

ents
[np.ndarray[float]] When `orb_type == 1`, this is `ndim == 1` vector containing the 1-orbital entropies. When `orb_type == 2`, this is `ndim == 2` matrix containing the 2-orbital entropies.

get_orbital_interaction_matrix(*ket*, *use_npdm=True*, *iprint=0*)

Compute orbital interaction matrix based on the 1- or 2- orbital entropies for the given MPS.

Args:

ket
[MPS] The given MPS for computing orbital interaction matrix.

use_npdm
[bool] Whether the more efficient algorithm based on N-PDM should be used. Default is True.

iprint
[int] Verbosity. Default is 0 (quiet).

Returns:

imat
[np.ndarray[float]] The `ndim == 2` orbital interaction matrix.

get_conventional_npdm(*ket*, *pdm_type=1*, *bra=None*, *soc=False*, *site_type=1*, *iprint=0*, *max_bond_dim=None*)

Compute the N-Particle Density Matrix (NPDM) for the given MPS using the conventional method.

Args:

ket
[MPS] The given MPS for computing NPDM.

pdm_type
[int] 1 or 2. Whether the 1PDM or 2PDM should be computed. Default is 1. For `pdm_type == 2`, the SGF mode is not supported.

bra
[MPS or None] If None, will compute the normal NPDM. If not None, will compute the transition NPDM between `bra` and `ket`.

soc
[bool] When `pdm_type == 1` this indicates whether the 1 particle transition triplet density matrix (for spin-orbit coupling) should be computed instead of the normal 1PDM. Only have effects in the SU2 mode. Default is False.

site_type
[int] 0 or 1 or 2. Indicates whether the NPDM should be computed using the 0- or 1- or 2-site algorithms. Default is 1. 0-site and 1-site are faster than the 2-site algorithm for this purpose.

iprint
[int] Verbosity. Default is 0 (quiet).

max_bond_dim
[None or int] If not None, the MPS bond dimension will be restricted during the sweeps. Default is None.

Returns:

dm

[np.ndarray[float|complex]] When `pdm_type == 1`:

$$\begin{cases} \mathrm{dm}[\sigma, i, j] = \langle a^\dagger_{i\sigma} a_{j\sigma} \rangle & \text{(SZ)} \\ \mathrm{dm}[i, j] = \sum_\sigma \langle a^\dagger_{i\sigma} a_{j\sigma} \rangle & \text{(SU2)} \\ \mathrm{dm}[i, j] = \langle T_{ij} \rangle & \text{(SU2/ soc)} \\ \mathrm{dm}[i, j] = \langle a^\dagger_i a_j \rangle & \text{(SGF)} \end{cases}$$

When `pdm_type == 2` (for SU2/SZ only):

$$\begin{cases} \mathrm{dm}[0, i, j, k, l] = \langle a^\dagger_{i\alpha} a^\dagger_{j\alpha} a_{k\alpha} a_{l\alpha} \rangle \\ \mathrm{dm}[1, i, j, k, l] = \langle a^\dagger_{i\alpha} a^\dagger_{j\beta} a_{k\beta} a_{l\alpha} \rangle \\ \mathrm{dm}[2, i, j, k, l] = \langle a^\dagger_{i\beta} a^\dagger_{j\beta} a_{k\beta} a_{l\beta} \rangle \end{cases}$$

get_conventional_1pdm(*ket*, **args*, ***kwargs*)

Compute the 1-Particle Density Matrix (1PDM) for the given MPS using the conventional method. See `DMRGDriver.get_conventional_npdm`.

get_conventional_2pdm(*ket*, **args*, ***kwargs*)

Compute the 2-Particle Density Matrix (2PDM) for the given MPS using the conventional method. See `DMRGDriver.get_conventional_npdm`.

get_conventional_trans_1pdm(*bra*, *ket*, **args*, ***kwargs*)

Compute the Transition 1-Particle Density Matrix (T-1PDM) between the given bra and ket MPSs using the conventional method. See `DMRGDriver.get_conventional_npdm`.

get_conventional_trans_2pdm(*bra*, *ket*, **args*, ***kwargs*)

Compute the Transition 2-Particle Density Matrix (T-2PDM) between the given bra and ket MPSs using the conventional method. See `DMRGDriver.get_conventional_npdm`.

get_npdm(*ket*, *pdm_type=1*, *bra=None*, *soc=False*, *site_type=0*, *algo_type=None*, *npdm_expr=None*, *mask=None*, *simulated_parallel=0*, *fused_contraction_rotation=True*, *cutoff=1e-24*, *iprint=0*, *max_bond_dim=None*)

Compute the N-Particle Density Matrix (NPDM) for the given MPS. Supports SU2, SZ, and SGF modes.

Args:

ket
[MPS] The given MPS for computing NPDM.

pdm_type
[int] Integer >= 1. The order of the PDM. Default is 1.

bra
[MPS or None] If None, will compute the normal NPDM. If not None, will compute the transition NPDM between `bra` and `ket`.

soc
[bool] When `pdm_type == 1` this indicates whether the 1 particle transition triplet density matrix (for spin-orbit coupling) should be computed instead of

the normal 1PDM. Only have effects in the SU2 mode. Default is False. If True, `NPDMAlgorithmTypes.Conventional` is required in `algo_type`.

site_type
[int] 0 or 1 or 2. Indicates whether the NPDM should be computed using the 0- or 1- or 2-site algorithms. Default is 0. 0-site and 1-site are faster than the 2-site algorithm for this purpose.

algo_type
[None or NPDMAlgorithmTypes] Strategies for computing N-particle density matrices. If None, this is set to `NPDMAlgorithmTypes.Default`. Default is None.

npdm_expr
[None or str or list[str]] The operator expression for the NPDM. If None, this will be determined automatically. Multiple operator expressions are allowed in the SZ and SGF modes. Default is None.

mask
[None or list[int] or list[list[int]]] The mask for setting repeated indices for the operator expression. Default is None, meaning that all indices can be different.

simulated_parallel
[int] Number of processors for simulating parallel algorithm serially. Default is zero, meaning that the serial algorithm is used if `self.mpi is None` or the parallel algorithm is used if `self.mpi is not None`. When `self.mpi is None`, one can optionally set this to a positive number to simulate the parallel algorithm which will be computationally less efficient but requiring less amount of memory.

fused_contraction_rotation
[bool] Indicating whether contraction and rotation should be done within one-step, without using large memory for blocking (only saving memory when no explicit left/right_contact is invoked, which is the case for `site_type == 0`). Default is True.

cutoff
[float] States with eigenvalue below this number will be discarded, even when the bond dimension is large enough to keep this state. Default is 1E-24.

iprint
[int] Verbosity. Default is 0 (quiet).

max_bond_dim
[None or int] If not None, the MPS bond dimension will be restricted during the sweeps. Default is None.

Returns:

dms
[np.ndarray[flat|complex] or list[np.ndarray[flat|complex]]] A list of density matrices for different spin components in the SZ mode, or the spin-traced density matrix in the SU2 mode, or the spin-orbit density matrix in the SGF mode.

In the SU2 mode (when `npdm_expr is None`):

$$\mathrm{dm}[i,j,\cdots,b,a] = \sum_{\sigma,\sigma',\cdots} \langle a^\dagger_{i\sigma} a^\dagger_{j\sigma'} \cdots a_{b\sigma'} a_{a\sigma} \rangle$$

In the SZ mode (when `npdm_expr is None`):

$$\begin{aligned}
\mathrm{dms}[0][i,\cdots,j,k,c,b,\cdots,a] &= \langle a^\dagger_{i\alpha} \cdots a^\dagger_{j\alpha} a_{k\alpha} a_{c\alpha} a_{b\alpha} \cdots a_{a\alpha} \rangle \\
\mathrm{dms}[1][i,\cdots,j,k,c,b,\cdots,a] &= \langle a^\dagger_{i\alpha} \cdots a^\dagger_{j\alpha} a_{k\beta} a_{c\beta} a_{b\alpha} \cdots a_{a\alpha} \rangle \\
\mathrm{dms}[2][i,\cdots,j,k,c,b,\cdots,a] &= \langle a^\dagger_{i\alpha} \cdots a^\dagger_{j\beta} a_{k\beta} a_{c\beta} a_{b\beta} \cdots a_{a\alpha} \rangle \\
&\cdots \\
\mathrm{dms}[\mathrm{pdm_type}][i,\cdots,j,k,c,b,\cdots,a] &= \langle a^\dagger_{i\beta} \cdots a^\dagger_{j\beta} a_{k\beta} a_{c\beta} a_{b\beta} \cdots a_{a\beta} \rangle
\end{aligned}$$

In the SGF mode (when `npdm_expr is None`):

$$\mathrm{dm}[i,j,\cdots,b,a] = \langle a^\dagger_i a^\dagger_j \cdots a_b a_a \rangle$$

get_1pdm(*ket*, **args*, ***kwargs*)

Compute the 1-Particle Density Matrix (1PDM) for the given MPS. See `DMRGDriver.get_npdm`.

get_2pdm(*ket*, **args*, ***kwargs*)

Compute the 2-Particle Density Matrix (2PDM) for the given MPS. See `DMRGDriver.get_npdm`.

get_3pdm(*ket*, **args*, ***kwargs*)

Compute the 3-Particle Density Matrix (3PDM) for the given MPS. See `DMRGDriver.get_npdm`.

get_4pdm(*ket*, **args*, ***kwargs*)

Compute the 4-Particle Density Matrix (4PDM) for the given MPS. See `DMRGDriver.get_npdm`.

get_trans_1pdm(*bra*, *ket*, **args*, ***kwargs*)

Compute the Transition 1-Particle Density Matrix (T-1PDM) between the given bra and ket MPSs. Note that there can be an overall phase uncertainty for transition NPDMs. See `DMRGDriver.get_npdm`.

get_trans_2pdm(*bra*, *ket*, **args*, ***kwargs*)

Compute the Transition 2-Particle Density Matrix (T-2PDM) between the given bra and ket MPSs. Note that there can be an overall phase uncertainty for transition NPDMs. See `DMRGDriver.get_npdm`.

get_trans_3pdm(*bra*, *ket*, **args*, ***kwargs*)

Compute the Transition 3-Particle Density Matrix (T-3PDM) between the given bra and ket MPSs. Note that there can be an overall phase uncertainty for transition NPDMs. See `DMRGDriver.get_npdm`.

get_trans_4pdm(*bra*, *ket*, **args*, ***kwargs*)

Compute the Transition 4-Particle Density Matrix (T-4PDM) between the given bra and ket MPSs. Note that there can be an overall phase uncertainty for transition NPDMs. See `DMRGDriver.get_npdm`.

get_csf_coefficients(*ket*, *cutoff=0.1*, *given_dets=None*, *max_print=200*, *iprint=1*)

Find the dominant Configuration State Functions (CSFs, in SU2 mode) or determinants (DETs, in SZ/SGF mode) and their coefficients in the given MPS.

Args:

ket
[MPS] The given MPS for computing CSF/DET coefficients. If targeting non-singlet state in the SU2 mode, the MPS should be in the singlet embedding format.

cutoff
[float] The lower bound for the absolute value of the CSF/DET coefficients that should be searched. Default is 0.1. If `cutoff == 0.0`, will compute coefficients for all CSF/DET, which may take an exponential amount of time.

given_dets
[None or list[str]] If not None, will compute the coefficients for the given CSF/DET set. If `cutoff != 0.0 and (given_dets == [] or given_dets is None)`, will consider all possible CSF/DET with the absolute value of the coefficients above `cutoff`. If `cutoff != 0.0 and given_dets is not None and len(given_dets) != 0`, the `cutoff` and `given_dets` will apply simultaneously, namely, a subset of `given_dets` which satisfies the `cutoff` constraint will be computed.

max_print
[int] The max number of CSF/DET and their coefficients that should be print. When the number of computed CSF/DET is larger than this number, only the dominant ones will be printed. When `iprint == 0`, this argument has no effect and nothing will be printed.

iprint
[int] Verbosity. Default is 1.

Returns:

dets
[np.ndarray[np.uint8]] Array of CSF/DET, represented as a matrix with shape `(n_dets, n_sites)`. The occupancy value 0, 1, 2, 3 represents "0" (empty), "+" (spin-up coupling), "-" (spin-down coupling), and "2" (doubly occupied) in the SU2 mode, or "0" (empty), "a" (alpha occupied), "b" (beta occupied), and "2" (doubly occupied) in the SZ/SGF mode.

dvals
[np.ndarray[float|complex]] Array of coefficients for each CSF/DET with size `n_dets`. Note that the weight is the square of coefficient. There can be an overall phase uncertainty for all coefficients.

compress_mps(*ket*, *max_bond_dim=None*)

Compress the MPS to change its maximal bond dimension (inplace).

Args:

ket
[MPS] The block2 MPS object.

max_bond_dim
[None or int] If not None, will restrict the maximal bond dimension of the MPS to the given number. Default is None.

Returns:

ket
[MPS] The MPS with changed bond dimension. Note that this operation can change the canonical center of the MPS as a side effect.

align_mps_center(*ket*, *ref*, *max_bond_dim=None*)

Change the canonical center of the given MPS, or align the canonical center for two MPSs (inplace).

Args:

ket
[MPS] The block2 MPS object.

ref
[MPS or int] If this is MPS, will change the canonical center of `ket` so that its center is the same as that of `ref`. If this is int, will set the canonical center of `ket` to the given number. Only the left/right canonical forms are allowed, namely, `ref` cannot correspond to a canonical center in the middle of MPS.

max_bond_dim
[None or int] If not None, will restrict the maximal bond dimension of the resulting MPS to the given number. Default is None.

adjust_mps(*ket*, *dot=None*)

Adjust the MPS (inplace) for performing 1-site or 2-site sweep algorithms, and find the direction of the next sweep. When restarting from the middle, this method should not be called.

Args:

ket
[MPS] The block2 MPS object.

dot
[None or int] If not None, should be 1 or 2 and the "site type" of the MPS will be changed to "1-site" or "2-site". If None, the "site type" of the MPS will not be changed. Default is None.

Returns:

ket
[MPS] The MPS with the desired "site type".

forward
[bool] Indicating whether the direction of the next sweep should be forward or not.

split_mps(*ket*, *iroot*, *tag*)

Split a state-averaged MPS into individual MPSs.

Args:

ket
[MPS] The state-averaged MPS object with `ket.nroots > 1`.

iroot
[int] The root index to extract from the state-averaged MPS. Counting from zero. `0 <= iroot < ket.nroots`.

tag
[str] The tag of the extracted MPS.

Returns:

iket
[MPS] The extracted MPS.

multiply(*bra*, *mpo*, *ket*, *n_sweeps=10*, *tol=1e-08*, *bond_dims=None*, *bra_bond_dims=None*, *noises=None*, *noise_mpo=None*, *thrds=None*, *left_mpo=None*, *cutoff=1e-24*, *linear_max_iter=4000*, *iprint=0*)

Apply the MPO to the MPS to get a new MPS (when `left_mpo is None`), which is

$$|\mathrm{bra}\rangle = \mathrm{mpo}|\mathrm{ket}\rangle$$

or apply the inverse of an MPO to the MPS to get a new MPS,

$$|\mathrm{bra}\rangle = \frac{\mathrm{mpo}|\mathrm{ket}\rangle}{\mathrm{left_mpo}}$$

or fit the MPS to an MPS with a different bond dimension (when `left_mpo is None` and `mpo` is the identity MPO).

Args:

bra
[MPS] The block2 MPS object. The given MPS `bra` will be used as the initial guess for the left-hand side MPS. When this method returns, the MPS `bra` will contain the optimized state fitted to the value of the right-hand side.

mpo
[MPO] The block2 MPO object (the right-hand side operator).

ket
[MPS] The block2 MPS object (the right-hand side state).

n_sweeps
[int] Maximal number of sweeps. Default is 10.

tol
[float] converge threshold for the norm of `bra` (when `left_mpo is None`) or the overlap `<bra|ket>` (when `left_mpo is not None`). Default is 1E-8.

bond_dims
[None or list[int]] List of `ket` bond dimensions for each sweep. Default is None. If None, the bond dimension of `ket` will be used.

bra_bond_dims
[None or list[int]] List of `bra` bond dimensions for each sweep. Default is None. If None, the bond dimension of initial `bra` will be used.

noises
[None or list[float]] List of prefactor of the noise for each sweep. Default is None. If None or `[0]`, will not add any noise.

noise_mpo
[None or MPO] If not None and `noises` is not zero or None, This MPO will be used for computing noise and `left_mpo` will be ignored. This should be used for the MPS compression with perturbative noise task.

thrds
[None or list[float]] List of the convergence threshold (square of the residual) of the linear solver. Default is None. If None, this is set to `[1e-6] * 4 + [1e-7] * 1` for double precision and `[1e-5] * 4 + [5e-6] * 1` for single precision.

left_mpo
[None or MPO] If not None and `noise_mpo is None`, will apply the inverse of this MPO to the right-hand side.

cutoff
[float] States with eigenvalue below this number will be discarded, even when the bond dimension is large enough to keep this state. Default is 1E-24.

linear_max_iter
[int] Maximal number of iteration in the linear solver. Default is 4000.

iprint
[int] Verbosity. Default is 0 (quiet).

Returns:

norm
[float|complex] The norm of `bra` (when `left_mpo is None`) or the overlap `<bra|ket>` (when `left_mpo is not None`).

addition(*bra*, *ket_a*, *ket_b*, *mpo_a=None*, *mpo_b=None*, *n_sweeps=10*, *tol=1e-08*, *bra_bond_dims=None*, *ket_a_bond_dims=None*, *ket_b_bond_dims=None*, *noises=None*, *noise_mpo=None*, *cutoff=1e-24*, *iprint=0*)

Perform the addition of two MPSs to generate a new MPS (using fitting):

$$|\mathrm{bra}\rangle = \mathrm{mpo_a}|\mathrm{ket_a}\rangle + \mathrm{mpo_b}|\mathrm{ket_b}\rangle.$$

Args:

bra
[MPS] The block2 MPS object. The given MPS `bra` will be used as the initial guess for the left-hand side MPS. When this method returns, the MPS `bra` will contain the optimized state fitted to the value of the right-hand side.

ket_a
[MPS] The first input block2 MPS object.

ket_b
[MPS] The second input block2 MPS object.

mpo_a
[None or int or float or complex or MPO] The first input block2 MPO object. If None or a scalar, this will be the identity MPO or a scalar times the identity MPO. Default is None.

mpo_b
[None or int or float or complex or MPO] The second input block2 MPO object. If None or a scalar, this will be the identity MPO or a scalar times the identity MPO. Default is None.

n_sweeps
[int] Maximal number of sweeps. Default is 10.

tol
[float] converge threshold for the norm of `bra`. Default is 1E-8.

bra_bond_dims
[None or list[int]] List of `bra` bond dimensions for each sweep. Default is None. If None, the bond dimension of initial `bra` will be used.

ket_a_bond_dims
[None or list[int]] List of `ket_a` bond dimensions for each sweep. Default is None. If None, the bond dimension of `ket_a` will be used.

ket_b_bond_dims
[None or list[int]] List of `ket_b` bond dimensions for each sweep. Default is None. If None, the bond dimension of `ket_b` will be used.

noises
[None or list[float]] List of prefactor of the noise for each sweep. Default is None. If None or `[0]`, will not add any noise.

noise_mpo
[None or MPO] If not None and `noises` is not zero or None, This MPO will be used for computing noise.

cutoff
[float] States with eigenvalue below this number will be discarded, even when the bond dimension is large enough to keep this state. Default is 1E-24.

iprint
[int] Verbosity. Default is 0 (quiet).

Returns:

norm
[float|complex] The norm of `bra`.

expectation(*bra*, *mpo*, *ket*, *store_bra_spectra=False*, *store_ket_spectra=False*, *iprint=0*)
Compute the expectation value between MPO and bra and ket MPSs:

$$X = \langle \mathrm{bra} | \mathrm{mpo} | \mathrm{ket} \rangle.$$

Args:

bra
[MPS] The "bra" MPS. In complex mode, the complex conjugate of `bra` is implied.

mpo
[MPO] The block2 MPO object, representing the operator.

ket
[MPS] The "ket" MPS.

store_bra_spectra
[bool] If True, the `bra` MPS singular value spectra will be stored as `self._sweep_wfn_spectra` which can be later used to compute the bipartite entropy. If False, the spectra will not be computed. Default is False. Only one of `store_bra_spectra` and `store_ket_spectra` can be True.

store_ket_spectra
[bool] If True, the `ket` MPS singular value spectra will be stored as `self._sweep_wfn_spectra` which can be later used to compute the bipartite entropy. If False, the spectra will not be computed. Default is False. Only one of `store_bra_spectra` and `store_ket_spectra` can be True.

iprint
[int] Verbosity. Default is 0 (quiet).

Returns:

ex
[float|complex] The expectation value.

greens_function(*bra*, *mpo*, *rmpo*, *ket*, *omega*, *eta*, *n_sweeps=10*, *tol=1e-08*, *bra_bond_dims=None*, *ket_bond_dims=None*, *noises=None*, *thrds=None*, *cutoff=1e-24*, *linear_max_iter=4000*, *iprint=0*)
Compute the correction vector MPS for Green's function:

$$|\mathrm{bra}\rangle = \frac{\mathrm{rmpo}|\mathrm{ket}\rangle}{\mathrm{mpo} + \omega + \eta \mathrm{i}}$$

and returns the diagonal element of Green's function:

$$G = \langle \mathrm{bra} | \mathrm{rmpo} | \mathrm{ket} \rangle.$$

Args:

bra
[MPS] The block2 MPS object. The given MPS `bra` will be used as the initial guess for the correction vector MPS. When this method returns, the MPS `bra` will contain the optimized state fitted to the value of the right-hand side of the correction vector equation.

mpo
[MPO] The input block2 MPO object in the denominator.

rmpo
[MPO] The input block2 MPO object in the numerator.

ket
[MPS] The input block2 MPS object in the numerator.

omega
[float] The frequency.

eta
[float] The broadening factor.

n_sweeps
[int] Maximal number of sweeps. Default is 10.

tol
[float] Converge threshold for the absolute value of the diagonal Green's function value <bra|rmpo|ket>. Default is 1E-8.

bra_bond_dims
[None or list[int]] List of `bra` bond dimensions for each sweep. Default is None. If None, the bond dimension of initial `bra` will be used.

ket_bond_dims
[None or list[int]] List of `ket` bond dimensions for each sweep. Default is None. If None, the bond dimension of `ket` will be used.

noises
[None or list[float]] List of prefactor of the noise for each sweep. Default is None. If None or `[0]`, will not add any noise.

thrds
[None or list[float]] List of the convergence threshold (square of the residual) of the linear solver. Default is None. If None, this is set to `[1e-6] * 4 + [1e-7] * 1` for double precision and `[1e-5] * 4 + [5e-6] * 1` for single precision.

cutoff
[float] States with eigenvalue below this number will be discarded, even when the bond dimension is large enough to keep this state. Default is 1E-24.

linear_max_iter
[int] Maximal number of iteration in the linear solver. Default is 4000.

iprint
[int] Verbosity. Default is 0 (quiet).

Returns:

gf
[complex] The diagonal element of Green's function <bra|rmpo|ket>.

fix_restarting_mps(*mps*)

Internal method for fixing the canonical form of MPS loaded from disk when the calculation was interrupted in the middle of a sweep.

Args:

mps
[MPS] The MPS loaded from disk.

copy_mps(*mps*, *tag*)

Make a deep copy of an MPS.

Args:

mps
[MPS] The input MPS.

tag
[str] The tag of the output MPS (for disk storage).

Returns:

ket
[MPS] The copy of MPS.

load_mps(*tag*, *nroots=1*)

Load an MPS from disk (from the DMRGDriver.scratch folder).

Args:

tag
[str] The tag of the MPS to be loaded.

nroots
[int] Number of roots in the MPS. Default is 1.

Returns:

mps
[MPS] The loaded MPS.

mps_change_singlet_embedding(*mps*, *tag*, *forward*, *left_vacuum=None*)

Change MPS between the Singlet-Embedding (SE) and Non-Singlet-Embedding (NSE) formats.

Args:

mps
[MPS] The input MPS.

tag
[str] The tag of the output MPS.

forward
[bool] If True, will change from NSE to SE. Otherwise, will change from SE to NSE.

left_vacuum
[None or SX] If `forward == True and left_vacuum is not None`, this is the left vacuum to be used in SE MPS. If None, the left vacuum quantum number will be automatically determined. Default is None.

Returns:

cp_mps
[MPS] The output MPS.

mps_change_to_singlet_embedding(*mps*, *tag*, *left_vacuum=None*)

Change MPS from Non-Singlet-Embedding (NSE) to Singlet-Embedding (SE).

Args:

mps
[MPS] The input MPS.

tag
[str] The tag of the output MPS.

left_vacuum
[None or SX] If not None, this is the left vacuum to be used in SE MPS. If None, the left vacuum quantum number will be automatically determined. Default is None.

Returns:

cp_mps
[MPS] The output MPS.

mps_change_from_singlet_embedding(*mps*, *tag*, *left_vacuum=None*)

Change MPS from Singlet-Embedding (SE) to Non-Singlet-Embedding (NSE).

Args:

mps
[MPS] The input MPS.

tag
[str] The tag of the output MPS.

left_vacuum
[None or SX] Not used. Default is None.

Returns:

cp_mps
[MPS] The output MPS.

mps_change_precision(*mps*, *tag*)

Change MPS from single to double precision (when `symm_type` has the `SymmetryTypes.SP` modifier) or from double to single precision (when `symm_type` does not have the `SymmetryTypes.SP` modifier).

Args:

mps
[MPS] The input MPS.

tag
[str] The tag of the output MPS.

Returns:

r
[MPS] The output MPS.

mps_change_complex(*mps*, *tag*)

Change MPS from complex to real (when `symm_type` has the `SymmetryTypes.CPX` modifier) or from real to complex (when `symm_type` does not have the `SymmetryTypes.CPX` modifier). For complex to real transformation, the imaginary part will be discarded.

Args:

mps
[MPS] The input MPS.

tag
[str] The tag of the output MPS.

Returns:

r
[MPS] The output MPS.

mps_change_to_sz(*mps*, *tag*, *sz=None*)

Change MPS from spin-adapted to non-spin-adapted. Only works in SU2 mode. The resulting MPS should be used in SZ mode.

Args:

mps
[MPS] The input MPS.

tag
[str] The tag of the output MPS.

sz
[None or int] If not None, will restrict two times the project spin of the output MPS to be the given number. If None, the output MPS will contain all possible project spin components. Default is None.

Returns:

zmps
[MPS] The output MPS.

get_random_mps(*tag*, *bond_dim=500*, *center=0*, *dot=2*, *target=None*, *nroots=1*, *occs=None*, *full_fci=True*, *left_vacuum=None*, *casci_ncore=0*, *casci_nvirt=0*, *mrci_order=0*, *orig_dot=False*)

Create a random MPS, which can be used as initial guess for various sweep algorithms, such as DMRG.

Args:

tag
[str] The tag of the output MPS, for disk storage.

bond_dim
[int] The maximal bond dimension hint of the MPS. Default is 500. Note that the output MPS may not have exactly the given bond dimension.

center
[int] The canonical center of the MPS. Default is zero.

dot
[int] Can be 1 or 2. The "site type" of the MPS. Default is 2.

target
[None or SX.] The target quantum number of the MPS. If None, will use `self.target`. Default is None.

nroots
[int] Number of roots in the MPS. Default is 1. `nroots > 1` indicates the state-averaged MPS (which may be used for excited state DMRG).

occs
[None or list[float]] If not None, the hint of the occupancy information at each site of the MPS. Default is None, and a uniform probability of occupancy will be assumed.

full_fci
[bool] If True, the full fci space is used (including block quantum numbers outside the space of the target quantum number). Default is True.

left_vacuum
[None or SX] If not None, this is the left vacuum to be used in SE MPS. If None, `self.left_vacuum` will be used. Only has effects in SU2 mode for SE MPS with non-singlet target.

casci_ncore
[int] If not zero, The number of core orbitals in a CASCI MPS. These orbitals will always be kept doubly occupied (if `mrci_order == 0`). Default is zero.

casci_nvirt
[int] If not zero, The number of virtual orbitals in a CASCI MPS. These orbitals

will always be kept empty (if `mrci_order == 0`). Default is zero.

mrci_order
[int] If not zero, the core and virtual orbitals will have at most `mrci_order` holes and electrons, respectively. Default is zero.

orig_dot
[bool] If False, will always create "1-site" MPS and then change to the suitable "site type". Otherwise, the "1-site" or "2-site" MPS will be directly created. Default is False.

Returns:

mps
[MPS] The output MPS (normalized).

get_ancilla_mps(*tag*, *center=0*, *dot=2*, *target=None*, *full_fci=True*)

Create the MPS with ancilla sites for finite-temperature algorithms. This is the MPS at infinite temperature.

Args:

tag
[str] The tag of the output MPS, for disk storage.

center
[int] The canonical center of the MPS. Default is zero.

dot
[int] Can be 1 or 2. The "site type" of the MPS. Default is 2.

target
[None or SX.] The target quantum number of the MPS. If None, will use `self.target`. Default is None.

full_fci
[bool] If True, the full FCI space is used (including block quantum numbers outside the space of the target quantum number). Default is True.

Returns:

mps
[MPS] The output MPS.

get_mps_from_csf_coefficients(*dets*, *dvals*, *tag*, *dot=2*, *target=None*, *full_fci=True*, *left_vacuum=None*)

Construct an MPS from the given linear combination of Configuration State Functions (CSFs, in SU2 mode) or determinants (DETs, in SZ/SGF mode).

Args:

dets
[np.ndarray[np.uint8] or list[str]] Array of CSF/DET, represented as a matrix with shape `(n_dets, n_sites)`. The occupancy value 0, 1, 2, 3 represents "0" (empty), "+" (spin-up coupling), "-" (spin-down coupling), and "2" (doubly

occupied) in the SU2 mode, or "0" (empty), "a" (alpha occupied), "b" (beta occupied), and "2" (doubly occupied) in the SZ/SGF mode.

dvals
[np.ndarray[float|complex]] Array of coefficients for each CSF/DET with size `n_dets`.

tag
[str] The tag of the output MPS, for disk storage.

dot
[int] Can be 1 or 2. The "site type" of the MPS. Default is 2.

target
[None or SX.] The target quantum number of the MPS. If None, will use `self.target`. Default is None.

full_fci
[bool] If True, the full fci space is used (including block quantum numbers outside the space of the target quantum number). Default is True.

left_vacuum
[None or SX] If not None, this is the left vacuum to be used in SE MPS. If None, `self.left_vacuum` will be used. Only has effects in SU2 mode for SE MPS with non-singlet target.

Returns:

mps
[MPS] The output MPS.

expr_builder()
Get the ExprBuilder object for setting terms in second quantized operators.

Returns:

builder
[ExprBuilder] The ExprBuilder object.

finalize()
Release stack memory allocated for this `DMRGDriver` object. Once finalized, this object should not be used.

class pyblock2.driver.core.**SOCDMRGDriver**(**args*, ***kwargs*)
Bases: DMRGDriver

Simple Python interface for DMRG calculations with Spin-Orbit-Coupling (SOC).

__init__(**args*, ***kwargs*)
Initialize `DMRGDriver`.

Args:

symm_type
[SymmetryTypes] The symmetry/floating point number mode. Default: SymmetryTypes.SU2.

scratch
[str] The working directory (scratch space). Default is "./nodex".

clean_scratch
[bool] If True, large temporary files in the scratch space will be removed once the DMRG finishes successfully. MPS files will not be removed. Default is True.

restart_dir
[None or str] If not None, MPS will be copied to the given directory after each DMRG sweep. Default is None (MPS will not be copied).

n_threads
[None or int] Number of threads. When MPI is used, this is the number of threads for each MPI processor. Default is None, and the max number of threads available on this node will be used.

n_mkl_threads
[int] Number of threads for parallelization inside MKL (for dense matrix multiplication). n_mkl_threads should be a factor of n_threads. When n_mkl_threads is not 1, nested threading will be used. Default is 1.

mpi
[None or bool] If True, MPI parallelization is used. If False or None, serial implementation is used. Default is None.

stack_mem
[int] The memory used for storing renormalized operators (in bytes). Default is 1 GB. When MPI is used, this is the number per MPI processor. Note that this argument is only responsible for part of the memory consumption in block2. The other part will be dynamically determined.

stack_mem_ratio
[float] The fraction of stack space occupied by the main stacks. Default is 0.4.

fp_codec_cutoff
[float] Floating-point number (absolute) precision for compressed storage of renormalized operators. Default is 1E-16.

hybrid_mpo_dmrg(*mpo*, *mpo_cpx*, *ket*, *n_sweeps=10*, *iprint=0*, *tol=1e-08*, ***kwargs*)

Perform the ground state and excited state Density Matrix Renormalization Group (DMRG) algorithm using the sum of a real MPO and a complex MPO.

Args:

mpo
[MPO] The real MPO.

mpo_cpx
[MPO] The complex MPO.

ket
[MPS] The block2 MPS object. The given MPS `ket` will be used as the initial guess for DMRG. When this method returns, the MPS `ket` will contain the optimized (ground and/or excited) state. If `ket.nroots != 1`, state-averaged DMRG will be done to find the ground and excited states. If `ket.dot == 2`, will perform 2-site DMRG algorithm. If `ket.dot == 1`, will perform 1-site DMRG algorithm. The initial input `ket` is not required to be normalized. The output `ket` will always be normalized.

n_sweeps
[int] Maximal number of DMRG sweeps. Default is 10.

tol
[float] Energy converge threshold. If the absolute value of the total energy difference between two consecutive sweeps is below `tol`, and the `noise` for the current sweep is zero, the algorithm will terminate. Default is 1E-8.

iprint
[int] Verbosity. Default is 0 (quiet).

kwargs
[dict] Other options that should be passed to `DMRGDriver.dmrg`.

Returns:

energy
[float|complex or list[float|complex]] When `ket.nroots == 1`, this is the ground state energy. When `ket.nroots != 1`, this is a list of ground and excited state energies.

soc_two_step(*energies*, *twoss*, *pdms_dict*, *hsomo*, *iprint=1*)

The second step in the two-step SOC-DMRG.

Args:

energies
[list[float|complex]] Energy of the spin-free states.

twoss
[list[int]] Two times the total spin for each spin-free state.

pdms_dict
[dict[tuple[int, int], np.ndarray[float|complex]]] The 1-particle triplet transition density matrix for each pair of states.

hsomo
[np.ndarray[complex]] The spin-orbit coupling integral in molecular orbitals.

iprint
[int] Verbosity. Default is 1.

Returns:

heig

[list[float|complex]] The ground and excited state energies including SOC effects.

class pyblock2.driver.core.**NormalOrder**

Bases: object

Static methods for normal ordering for quantum chemistry integrals.

static def_ix(*cidx*)

Internal method for index slicing.

static def_gctr(*cidx*, *h1e*, *g2e*)

Internal method for contraction with integrals.

static def_gctr_sz(*cidx*, *h1e*, *g2e*)

Internal method for contraction with integrals in the SZ mode.

static make_su2(*h1e*, *g2e*, *const_e*, *cidx*, *use_wick*)

Perform normal ordering of quantum chemistry integrals in the SU2 mode.

static make_sz(*h1e*, *g2e*, *const_e*, *cidx*, *use_wick*)

Perform normal ordering of quantum chemistry integrals in the SZ mode.

static make_sgf(*h1e*, *g2e*, *const_e*, *cidx*, *use_wick*)

Perform normal ordering of quantum chemistry integrals in the SGF mode.

class pyblock2.driver.core.**WickNormalOrder**

Bases: object

Static methods for normal ordering for quantum chemistry integrals implemented using automatic symbolic derivation.

static make_su2(*h1e*, *g2e*, *const_e*, *cidx*, *iprint=1*)

Perform normal ordering of quantum chemistry integrals in the SU2 mode.

static make_su2_open_shell(*h1e*, *g2e*, *const_e*, *cidx*, *midx*, *iprint=1*)

Perform normal ordering of quantum chemistry integrals in the SU2 mode for non-singlet reference states.

static make_sz(*h1e*, *g2e*, *const_e*, *cidx*, *iprint=1*)

Perform normal ordering of quantum chemistry integrals in the SZ mode.

static make_sgf(*h1e*, *g2e*, *const_e*, *cidx*, *iprint=1*)

Perform normal ordering of quantum chemistry integrals in the SGF mode.

class pyblock2.driver.core.**ExprBuilder**(*bw=None*)

Bases: object

Helper class for setting terms in second quantized operators.

Attributes:

bw

[Block2Wrapper] The wrapper for low-level block2 modules.

data
[GeneralFCIDUMP] The block2 GeneralFCIDUMP object.

__init__(*bw=None*)

Initialize `ExprBuilder`.

Args:

bw
[Block2Wrapper or None] The wrapper for low-level block2 modules. If None, will assume the SU2 symmetry mode.

add_const(*x*)

Add a constant term.

Args:

x
[int or float or complex] A scalar constant.

Returns:

self
[ExprBuilder] The ExprBuilder object.

add_term(*expr*, *idx*, *val*)

Add a string of elementary operators with the given coefficient (when the length of `idx` matches the length of `expr`), or a sum of strings of elementary operators with the same name, but different site indices and coefficients (when the length of `idx` is multiple of the length of `expr`).

Args:

expr
[str] The names of elementary operators, such as “cdCD”.

idx
[list[int]] The site index of each elementary operator.

val
[list[float|complex] or float or complex] The coefficient of the term or the coefficients of multiple terms.

Returns:

self
[ExprBuilder] The ExprBuilder object.

add_sum_term(*expr*, *arr*, *cutoff=1e-12*, *fast=True*, *factor=1.0*, *perm=None*)

Add terms with coefficients as a tensor.

Args:

expr
[str] The names of elementary operators, such as “cdCD”.

arr
[np.ndarray[float|complex]] The coefficients as a tensor. The `ndim` of this tensor should match the length of `expr`.

cutoff
[float] If the absolute value of any coefficient is below this threshold, the term will not be included. Default is 1E-12.

fast
[bool] If True, will use the fast C++ implementation. Default is True.

factor
[float] The scale factor for all terms.

perm
[None or list[int]] If not None, a permutation will be applied on the coefficient tensor indices. Default is None.

Returns:

self
[ExprBuilder] The ExprBuilder object.

add_terms(*expr*, *arr*, *idx*, *cutoff=1e-12*)

Add a sum of strings of elementary operators with the same name, but different site indices and coefficients.

Args:

expr
[str] The names of elementary operators, such as "cdCD".

arr
[list[float|complex]] The list of coefficients.

idx
[list[list[int]]] The list of operator indices.

cutoff
[float] If the absolute value of any coefficient is below this threshold, the term will not be included. Default is 1E-12.

Returns:

self
[ExprBuilder] The ExprBuilder object.

iscale(*d*)

Scale the coefficients of all terms by a scalar factor.

Args:

d
[float or complex] The scalar factor.

Returns:

self
[ExprBuilder] The ExprBuilder object.

finalize(*adjust_order=True*, *merge=True*, *is_drt=False*, *fermionic_ops=None*)

Finalize the symbolic expression.

Args:

adjust_order
[bool] If True, the order of operator indices will be automatically adjusted. This is normally required for MPO construction, unless the operator indices have already been sorted. Default is True.

merge
[bool] If True, will merge terms whenever possible. Default is True.

is_drt
[bool] If True, the DRT rule will be used. Only have effects in SU2 mode. Default is False.

fermionic_ops
[None or str] If not None, the given set of operator names will be treated as Fermion operators (for computing signs for swapping operators). Default is None, and operators like "cdCD" will be treated as Fermion operators.

Returns:

gfd
[GeneralFCIDUMP] The block2 GeneralFCIDUMP object.

class pyblock2.driver.core.**FermionTransform**

Bases: object

Static methods for fermion to spin operator transforms.

static jordan_wigner(*h1e*, *g2e*)

Jordan-Wigner transform of quantum chemistry integrals.

class pyblock2.driver.core.**OrbitalEntropy**

Bases: object

Static methods for orbital entropy computations.

ops = ['1-n-N+nN', 'D-nD', 'd-Nd', 'Dd', 'C-nC', 'N-nN', 'Cd', '-Nd', 'c-Nc', 'Dc', 'n-nN', 'nD', 'Cc', '-Nc', 'nC', 'nN']

Entropy operators in SZ mode. See Table 4. in *J. Chem. Theory Comput.* **9**, 2959-2973 (2013).

ops_ghf = ['1-N', 'D', 'C', 'N']

Entropy operators in SGF mode.

static parse_expr(*x*)

Internal method for paring entropy operator expressions.

static get_one_orb_rdm_h_terms(*n_sites*, *is_sgf=False*)

Internal method for computing symbolic terms in one-orbital RDM.

static get_two_orb_rdm_h_terms(*n_sites*, *ij_symm=True*, *block_symm=True*, *is_sgf=False*)

Internal method for computing symbolic terms in two-orbital RDM.

static get_one_orb_rdm_exprs(*is_sgf=False*)

Internal method for computing one-orbital RDM expressions (for NPDM engine).

static get_two_orb_rdm_exprs(*is_sgf=False*)

Internal method for computing two-orbital RDM expressions (for NPDM engine).

static get_two_orb_rdm_eigvals(*ld*, *diag_only=False*)

Internal method for solving eigenvalue problem for two-orbital RDM.

class pyblock2.driver.core.**WickSpinAdaptation**

Bases: `object`

Static methods for symbolic expression processing for normal ordering.

static spin_tag_to_pattern(*x*)

Internal method for transforming `[1, 2, 2, 1] -> ((.+(.+.)0)1+.)0`.

static adjust_spin_coupling(*eq*)

Internal method for the adjustment of spin coupling. Correct up to 4-body terms.

static get_eq_exprs(*eq*)

Internal method for transforming from symbolic equations to second quantized operator expressions that can be used in the SU2 mode.

class pyblock2.driver.core.**SimilarityTransform**

Bases: `object`

Static methods for DMRG with similarity transformed Hamiltonians.

static make_su2(*h1e*, *g2e*, *ecore*, *t1*, *t2*, *scratch*, *n_elec*, *t3=None*, *ncore=0*, *ncas=-1*, *st_type=STTypes.H_HT_HT2T2*, *iprint=1*)

Construct expression for the similarity transformed Hamiltonians in the SU2 mode.

static make_sz(*h1e*, *g2e*, *ecore*, *t1*, *t2*, *scratch*, *n_elec*, *ncore=0*, *ncas=-1*, *st_type=STTypes.H_HT_HT2T2*, *iprint=1*)

Construct expression for the similarity transformed Hamiltonians in the SZ mode.

static make_sgf(*h1e*, *g2e*, *ecore*, *t1*, *t2*, *scratch*, *n_elec*, *t3=None*, *ncore=0*, *ncas=-1*, *st_type=STTypes.H_HT_HT2T2*, *iprint=1*)

Construct expression for the similarity transformed Hamiltonians in the SGF mode.

6.2 Global Settings

In **block2**, we try to minimize the use of global variables. Two global variables (`frame_()` and `threading_()`) have been used for controlling global settings such as stack memory, scartch folder and threading schemes.

Note that in `block2` the distributed parallelization scheme is handled locally.

6.2.1 Threading

enum class `block2::`**`ThreadingTypes`** : uint8_t

An indicator for where the openMP shared-memory threading should be activated. In the case of nested openMP, the total number of nested threading layers is determined from this enumeration.

For each enumerator, the number in brackets is the total number of threading layers.

Values:

enumerator **`SequentialGEMM`**

[0] seq mkl

enumerator **`BatchedGEMM`**

[1] parallel mkl

enumerator **`Quanta`**

[1] openmp quanta + seq mkl

enumerator **`QuantaBatchedGEMM`**

[2] openmp quanta + parallel mkl

enumerator **`Operator`**

[1] openmp operator

enumerator **`OperatorBatchedGEMM`**

[2] openmp operator + parallel mkl

enumerator **`OperatorQuanta`**

[2] openmp operator + openmp quanta

enumerator **`OperatorQuantaBatchedGEMM`**

[3] openmp operator + openmp quanta + parallel mkl

enumerator **Global**

[1] openmp for general non-core-algorithm tasks

enum class block2::**SeqTypes** : uint8_t

Method of GEMM (dense matrix multiplication) parallelism. For CSR matrix multiplication, the only possbile case is SeqTypes::None, but one can still use SeqTypes::Simple and it will only parallelize dense matrix multiplication.

Values:

enumerator **None**

GEMM are not parallelized. Parallelism may happen inside each GEMM, if a threaded version of MKL is linked.

enumerator **Simple**

GEMM written to the different outputs are parallelized, otherwise they are executed in sequential. With this mode, the code will sort and divide GEMM to several groups (batches). Inside each batch, the output addresses are guarenteed to be different. The cblas_dgemm_batch is invoked to compute each batch.

enumerator **Auto**

DGEMM automatically divided into several batches only when there are data dependency. Conflicts of output are automatically resolved by introducing temporary arrays. The cblas_dgemm_batch is invoked to compute each batch. This option normally requires a large amount of time for preprocessing and it will introduce a large number of temporary arrays, which is not memory friendly.

enumerator **Tasked**

GEMM will be evenly divided into n_threads groups, Different groups are executed in different threads. Since different threads may write into the same output array, there is an additional reduction step after all GEMM finishes. This mode is mainly implemented for Davidson matrix-vector step (tensor_product_multiply), where the size of the output array (wavefunction) is small compared to that of all input arrays. For blocking/rotation step, SeqTypes::Tasked has no effect and it is equivalent to SeqTypes::None. The cblas_dgemm_batch is not used in this mode.

enumerator **SimpleTasked**

This is the same as SeqTypes::Tasked for the Davidson matrix-vector step, and the same as SeqTypes::Simple for other steps.

struct **Threading**

Global information for threading schemes.

Public Functions

inline bool **openmp_available**() const
Whether openmp compiler option is set.

inline bool **tbb_available**() const
Whether tbb memory allocator is used.

inline bool **mkl_available**() const
Whether MKL math library is used.

inline bool **blis_available**() const
Whether BLIS math library is used.

inline bool **complex_available**() const
Whether complex number extension is used.

inline bool **single_precision_available**() const
Whether single precision extension is used.

inline bool **ksymm_available**() const
Whether K symmetry extension is used.

inline string **get_mkl_version**() const
Check version of the linked MKL library.

Returns
A version string of the linked MKL library if MKL is linked, or an empty string otherwise.

inline string **get_mkl_threading_type**() const
Return a string indicating which threaded MKL library is linked.

inline string **get_seq_type**() const
Return a string indicating which `SeqTypes` is used.

inline int **get_thread_id**() const
If inside a openMP parallel region, return the id of the current thread.

inline int **activate_global**() const
Set number of threads for a general task. Parallelism inside MKL will be deactivated for a general task.

Returns
Number of threads for general tasks. Returns 1 if openMP should not be used for a general task.

inline int **activate_global_mkl**() const
Set number of threads for a general task with parallelism inside MKL. Parallelism outside MKL will be deactivated.

Returns
Number of threads for general tasks. Returns 1 if MKL is not supported.

inline int **activate_normal**() const

Set number of threads for a normal (parallelism over renormalized operators) task.

Returns
Number of threads for parallelism over renormalized operators.

inline int **activate_operator**() const

Set number of threads for parallelism over renormalized operators.

Returns
Number of threads for parallelism over renormalized operators.

inline int **activate_quanta**() const

Set number of threads for parallelism over symmetry sectors.

Returns
Number of threads for parallelism over symmetry sectors.

inline **Threading**()

Default constructor. Uses `ThreadingTypes::Global | ThreadingTypes::BatchedGEMM` with maximal available number of threads, and `SeqTypes::None` for dense matrix multiplication.

inline **Threading**(*ThreadingTypes* type, int nta = -1, int ntb = -1, int ntc = -1, int ntd = -1)

Constructor.

Parameters

- **type** – Type of the threading scheme.
- **nta** – Number of threads for a general task (if `ThreadingTypes::Global` is set) or number of threads in the first threading layer.
- **ntb** – Number of threads in the first threading layer for a non-general threaded task (if `ThreadingTypes::Global` is set) or number of threads in the second threading layer.
- **ntc** – Number of threads in the second threading layer for a non-general threaded task (if `ThreadingTypes::Global` is set) or number of threads in the third threading layer.
- **ntd** – Number of threads in the third threading layer for a non-general threaded task (if `ThreadingTypes::Global` is set).

Public Members

ThreadingTypes **type**

Type of the threading scheme.

SeqTypes **seq_type** = *SeqTypes*::*None*

Method of dense matrix multiplication parallelism.

int **n_threads_op** = 0

Number of threads for parallelism over renormalized operators.

int **n_threads_quanta** = 0

Number of threads for parallelism over symmetry sectors.

int **n_threads_mkl** = 0

Number of threads for parallelism within dense matrix multiplications.

int **n_threads_global** = 0

Number of threads for general tasks.

int **n_levels** = 0

Number of nested threading layers.

Friends

inline friend ostream &**operator<<**(ostream &os, const *Threading* &th)

Print threading information.

inline shared_ptr<*Threading*> &block2::**threading_**()

Implementation of the `threading` global variable.

threading

Global variable containing information for shared-memory parallelism schemes and number of threads used for each threading layer.

6.2.2 Allocators

template<typename **T**>

struct **Allocator**

Abstract memory allocator.

Template Parameters

T – The type of the element in the array.

Subclassed by *block2::StackAllocator< T >*, *block2::VectorAllocator< T >*

Public Functions

inline **Allocator**()

Default constructor.

virtual **~Allocator**() = default

Default destructor.

inline virtual *T* ***allocate**(size_t n)

Allocate a length n array.

Parameters

n – Number of elements in the array.

Returns

The allocated pointer.

inline virtual complex<*T*> ***complex_allocate**(size_t n)

Allocate a length n complex array.

Parameters

n – Number of elements in the array.

Returns

The allocated pointer.

inline virtual void **deallocate**(void *ptr, size_t n)

Deallocate a length n array.

Parameters

- **ptr** – The pointer to be deallocated.
- **n** – Number of elements in the array.

inline virtual void **complex_deallocate**(void *ptr, size_t n)

Deallocate a length n complex array.

Parameters

- **ptr** – The pointer to be deallocated.
- **n** – Number of elements in the array.

inline virtual *T* ***reallocate**(*T* *ptr, size_t n, size_t new_n)

Adjust the size an allocated pointer. No data copying will happen.

Parameters

- **ptr** – The allocated pointer.
- **n** – Number of elements in original allocation.
- **new_n** – Number of elements in the new allocation.

Returns

The new pointer.

inline virtual shared_ptr<*Allocator*<*T*>> **copy**() const

Return a copy of the allocator.

Returns
ptr The copy of this allocator.

template<typename **T**>

struct **StackAllocator** : public block2::*Allocator*<*T*>

Stack memory allocator.

Template Parameters
T – The type of the element in the array.

Subclassed by block2::TemporaryAllocator< T >

Public Functions

inline **StackAllocator**(*T* *ptr, size_t max_size)

Constructor.

Parameters

- **ptr** – Pointer to the first elemenet in the stack. The stack should be pre-allocated.
- **max_size** – Total size of the stack (in number of elements).

inline **StackAllocator**()

Default constructor.

inline virtual *T* ***allocate**(size_t n) override

Allocate a length n array.

Parameters
n – Number of elements in the array.

Returns
The allocated pointer.

inline virtual void **deallocate**(void *ptr, size_t n) override

Deallocate a length n array. Must be invoked in the reverse order of allocation.

Parameters

- **ptr** – The pointer to be deallocated.
- **n** – Number of elements in the array.

inline virtual *T* ***reallocate**(*T* *ptr, size_t n, size_t new_n) override

Change the allocated size in middle of stack memory and introduce a shift for moving memory after it.

Parameters

- **ptr** – The allocated pointer.
- **n** – Number of elements in original allocation.
- **new_n** – Number of elements in the new allocation.

Returns
The new pointer.

Public Members

size_t **size**

Total size of the stack (in number of elements).

size_t **used**

Occupied size of the stack (in number of elements).

size_t **shift**

Temporary shift introduced due to deallocation in the middle of the stack.

T ***data**

Pointer to the first elemenet in the stack.

Friends

inline friend ostream &**operator<<**(ostream &os, const *StackAllocator* &c)

Print the status of the allocator.

Parameters

- **os** – The output stream.
- **c** – The object to be printed.

Returns
The output stream.

template<typename **T**>

struct **VectorAllocator** : public block2::*Allocator*<*T*>

Vector memory allocator.

Template Parameters
T – The type of the element in the array.

Public Functions

inline **VectorAllocator**()

Default constructor.

inline virtual *T* ***allocate**(size_t n) override

Allocate a length n array.

Parameters

n – Number of elements in the array.

Returns

The allocated pointer.

inline virtual void **deallocate**(void *ptr, size_t n) override

Deallocate a length n array. Note that explicit deallocation is not required for vector allocator. Can be invoked in arbitrary order.

Parameters

- **ptr** – The pointer to be deallocated.
- **n** – Number of elements in the array.

inline virtual *T* ***reallocate**(*T* *ptr, size_t n, size_t new_n) override

Change the allocated size for one allocated block.

Parameters

- **ptr** – The allocated pointer.
- **n** – Number of elements in original allocation.
- **new_n** – Number of elements in the new allocation.

Returns

The new pointer.

inline virtual shared_ptr<*Allocator*<*T*>> **copy**() const override

Return a copy of the allocator. When deep-copying objects using *VectorAllocator*, the other object should have an independent allocator, since *VectorAllocator* is not global.

Returns

The copy of this allocator.

Public Members

vector<vector<*T*>> **data**

The allocated data blocks.

Friends

inline friend ostream &**operator<<**(ostream &os, const *VectorAllocator* &c)

Print the status of the allocator.

Parameters

- **os** – The output stream.
- **c** – The object to be printed.

Returns

The output stream.

inline shared_ptr<*StackAllocator*<uint32_t>> &block2::**ialloc_**()

Implementation of the ialloc global variable.

template<typename **FL**>
inline shared_ptr<*StackAllocator*<*FL*>> &block2::**dalloc_**()

Implementation of the dalloc global variable.

ialloc

Global variable for the integer stack memory allocator.

6.2.3 Data Frame

template<typename **FL**>

struct **DataFrame**

DataFrame includes several (n_frames = 2) frames. Each frame includes one integer stack memory and one double stack memory. The two frames are used alternatively to avoid data copying.

Public Functions

inline **DataFrame**(size_t isize = 1 << 28, size_t dsize = 1 << 30, const string &save_dir = "node0", double dmain_ratio = 0.7, double imain_ratio = 0.7, int n_frames = 2)

Constructor.

Parameters

- **isize** – Max size (in bytes) of all integer stacks.
- **dsize** – Max size (in bytes) of all double stacks.
- **save_dir** – Scartch folder for renormalized operators.
- **dmain_ratio** – The fraction of stack space occupied by the main double stacks.
- **imain_ratio** – The fraction of stack space occupied by the main integer stacks.
- **n_frames** – Number of data frames.

inline virtual **~DataFrame**()

Destructor.

inline void **activate**(int i)

Activate one data frame.

Parameters

i – The index of the data frame to be activated.

inline void **reset**(int i)

Reset one data frame, marking all stack memory as unused.

Parameters

i – The index of the data frame to be reset.

inline void **reset_buffer**(int i)

Reset saving and loading buffers for one data frame. Contents in the loading buffer will be deleted. Unsaved contents in the saving buffer will be saved in disk.

Parameters

i – The index of the data frame.

inline void **rename_data**(const string &old_filename, const string &new_filename) const

Rename one scratch file.

Parameters

- **old_filename** – original filename.
- **new_filename** – new filename.

inline void **load_data_from**(int i, istream &ifs) const

Load one data frame from input stream.

Parameters

- **i** – The index of the data frame.
- **ifs** – The input stream.

inline void **load_data**(int i, const string &filename) const

Load one data frame from disk.

Parameters

- **i** – The index of the data frame.
- **filename** – The filename for the data frame.

inline void **save_data_to**(int i, ostream &ofs) const

Save one data frame into output stream.

Parameters

- **i** – The index of the data frame.
- **ofs** – The output stream.

inline void **save_data**(int i, const string &filename) const

Save one data frame to disk.

Parameters

- **i** – The index of the data frame.
- **filename** – The filename for the data frame.

inline void **deallocate**()

Deallocate the memory allocated for all stacks. Note that this method is automatically invoked at deconstruction.

inline size_t **memory_used**() const

Return the current used memory in all stacks.

Returns

The current used memory in Bytes.

inline void **update_peak_used_memory**() const

Update prak used memory statistics.

inline void **reset_peak_used_memory**() const

Reset prak used memory statistics to zero.

Public Members

string **save_dir**

Scartch folder for renormalized operators.

string **mps_dir**

Scartch folder for MPS (default is the same as save_dir).

string **mpo_dir**

Scartch folder for MPO (default is the same as save_dir, only used when minimal_memory_usage is true).

string **restart_dir** = ""

If not empty, save MPS to this dir after each sweep.

string **restart_dir_per_sweep** = ""

if not empty, save MPS to this dir with sweep index as suffix, so that MPS from all sweeps will be kept in individual dirs.

string **restart_dir_optimal_mps** = ""

If not empty, save MPS to this dir whenever an optimal solution is reached in one sweep. For DMRG, this is the MPS with the lowest energy. Note that if the best solution from the current sweep is worse than the best solution from the previous sweep (for example in a reverse schedule), the best solution from the current sweep is saved.

string **restart_dir_optimal_mps_per_sweep** = ""

If not empty, save the optimal MPS from each sweep to this dir with sweep index as suffix.

string **prefix** = "F"

Filename prefix for common scratch files (such as MPS tensors).

string **prefix_distri** = "F0"

Filename prefix for distributed scratch files (such as renormalized operators). When distributed parallelization is used, different procs will have different values for this data.

bool **prefix_can_write** = true

Whether this proc should be able to write common scratch files (such as MPS tensors).

bool **partition_can_write** = true

Whether this proc should be able to write renormalized operators.

size_t **isize**

Max number of elements in all integer stacks.

size_t **dsize**

Max number of elements in all double stacks.

int **n_frames**

Total number of data frames.

int **i_frame**

The index of Current activated data frame.

mutable double **tread** = 0

IO Time cost for reading scratch files.

double **twrite** = 0

IO Time cost for writing scratch files.

double **tasync** = 0

IO Time cost for async writing scratch files.

mutable double **fpread** = 0

IO Time cost for reading scratch files with floating-point decompression.

double **fpwrite** = 0

IO Time cost for writing scratch files with floating-point compression.

mutable Timer **_t**

Temporary timer.

Timer **_t2**

Auxiliary temporary timer.

vector<shared_ptr<*StackAllocator*<uint32_t>>> **iallocs**

Integer stacks allocators.

vector<shared_ptr<*StackAllocator*<*FL*>>> **dallocs**

Double stacks allocators.

mutable vector<size_t> **peak_used_memory**

Peak used memory by stacks (in Bytes). Even indices are for double stacks. Odd indices are for interger stacks.

mutable vector<string> **present_filenames**

The filename for the current stack memory content for each data frame. Used for tracking loading and saving buffering to avoid loading the same data into memory.

mutable vector<pair<string, shared_ptr<stringstream>>> **load_buffers**

Buffers for loading. Skpping reading a file with certain filename, if the contents of the file with that filename is in the loading buffer.

mutable vector<pair<string, shared_ptr<stringstream>>> **save_buffers**

Buffers for async saving.

mutable vector<shared_future<void>> **save_futures**

Async saving files.

bool **load_buffering** = false

Whether load buffering should be used. If true, memory usage will increase.

bool **save_buffering** = false

Whether async saving and saving buffering should be used. If true, memory usage will increase.

bool **use_main_stack** = true

Whether main stack should be used for storing blocked operators in enlarged blocks. If false, these blocked operators will be stored in dynamically allocated memory.

bool **minimal_disk_usage** = false

Whether temporary renormalized operator files should be deleted as soon as possible. If true, will save roughly half of required storage for renormalized operators.

bool **minimal_memory_usage** = false

Whether MPO should be build in minimal memory mode by saving intermediates to disk. In this mode, MPO should have different tags.

shared_ptr<*FPCodec*<*FL*>> **fp_codec** = nullptr

Floating-point compression codec. If nullptr, floating-point compression will not be used.

Public Static Functions

static inline void **buffer_save_data**(const string &filename, const shared_ptr<stringstream> &ss, double *tasync)

Save the data in buffer stream into disk.

Parameters

- **filename** – The filename for saving data.
- **ss** – The buffer stream.
- **tasync** – Pointer to the time recorder for async saving.

Friends

inline friend ostream &**operator<<**(ostream &os, const *DataFrame* &df)

Print the status of the data frame.

Parameters

- **os** – The output stream.
- **df** – The object to be printed.

Returns

The output stream.

template<typename **FL**>
inline shared_ptr<*DataFrame*<*FL*>> &block2::**frame_**()

Global variable for accessing global stack memory and file I/O in scratch space.

6.2.4 Miscellanies

inline auto block2::**check_signal_**() -> void (*&)()

Function pointer for signal checking.

inline void block2::**print_trace**()

Print calling stack when an error happens. Not working for non-unix systems.

6.3 Sparse Matrix

In **block2**, we support a few different representations of the block-sparse matrix.

6.3.1 Archived Sparse Matrix

template<typename **S**, typename **FL**>

struct **ArchivedSparseMatrix** : public block2::SparseMatrix<*S*, *FL*>

Block-sparse Matrix associated with disk storage, representing sparse operator.

Template Parameters

- **S** – Quantum label type.
- **FL** – float point type.

Public Functions

inline **ArchivedSparseMatrix**(const string &filename, int64_t offset, const shared_ptr<*Allocator*<FP>> &alloc = nullptr)

Constructor.

Parameters

- **filename** – The name of the associated disk file.
- **offset** – Byte offset in the file (where to read/write the content).
- **alloc** – Memory allocator.

inline virtual SparseMatrixTypes **get_type**() const override

Get the type of this sparse matrix.

Returns
Type of this sparse matrix.

inline virtual void **allocate**(const shared_ptr<SparseMatrixInfo<*S*>> &info, *FL* *ptr = 0) override

Allocate memory for the sparse matrix non-zero elements. This method is not allowed here. Will cause assertion failure.

Parameters

- **info** – The quantum label information for the sparse matrix.
- **ptr** – If not zero, the given pointer is used as the data pointer (no allocation will happen).

inline virtual void **deallocate**() override

Release the allocated memory. This method does nothing here, since no memory is used by this object.

inline shared_ptr<SparseMatrix<*S*, *FL*>> **load_archive**()

Load the sparse matrix data from disk.

Returns
A normal or CSR sparse matrix (with data in memory).

inline void **save_archive**(const shared_ptr<SparseMatrix<*S*, *FL*>> &mat)

Write the sparse matrix data to disk.

Parameters
mat – A normal or CSR sparse matrix (with data in memory).

Public Members

string **filename**

The name of the associated disk file.

int64_t **offset** = 0

Byte offset in the file.

SparseMatrixTypes **sparse_type**

Type of the archived sparse matrix. Note that this is not the type of this sparse matrix.

6.4 Tensor Functions

Tensor functions are drivers for performing operations for operator tensors. For operator tensors with different internal data representations or distribed parallelization schemes, a few different implementations of the tensor function drivers are defined. They all have the same interface.

6.4.1 Archived Tensor Functions

template<typename **S**, typename **FL**>

struct **ArchivedTensorFunctions** : public block2::TensorFunctions<*S*, *FL*>

Operations for operator tensors with internal data stored in disk file.

Template Parameters

- **S** – Quantum label type.
- **FL** – float point type.

Public Functions

inline **ArchivedTensorFunctions**(const shared_ptr<OperatorFunctions<*S*, *FL*>> &opf)

Constructor.

Parameters

opf – Sparse matrix algebra driver.

inline virtual TensorFunctionsTypes **get_type**() const override

Get the type of this driver for tensor functions.

Returns

Type of this driver for tensor functions.

inline void **archive_tensor**(const shared_ptr<OperatorTensor<*S*, *FL*>> &a) const

Save the content of an operator tensor into disk, transforming its internal representation to sparse matrices with internal data stored in disk file, and deallocating its memory data.

Parameters

a – Operator tensor with ordinary memory storage.

inline virtual void **left_assign**(const shared_ptr<OperatorTensor<*S*, *FL*>> &a, shared_ptr<OperatorTensor<*S*, *FL*>> &c) const override

Left assignment (copy) operation: c = a. This is the edge case for the left blocking step. Left assignment means that the operator tensor is a row vector of symbols.

Parameters

- **a** – Operator a (input).
- **c** – Operator c (output).

inline virtual void **right_assign**(const shared_ptr<OperatorTensor<*S*, *FL*>> &a, shared_ptr<OperatorTensor<*S*, *FL*>> &c) const override

Right assignment (copy) operation: c = a. This is the edge case for the right blocking step. Right assignment means that the operator tensor is a column vector of symbols.

Parameters

- **a** – Operator a (input).
- **c** – Operator c (output).

inline virtual void **tensor_product_partial_multiply**(const shared_ptr<OpExpr<*S*>> &expr, const shared_ptr<OperatorTensor<*S*, *FL*>> &lopt, const shared_ptr<OperatorTensor<*S*, *FL*>> &ropt, bool trace_right, const shared_ptr<SparseMatrix<*S*, *FL*>> &cmat, const vector<pair<uint8_t, *S*>> &psubsl, const vector<vector<shared_ptr<typename SparseMatrixInfo<*S*>::ConnectionInfo>>> &cinfos, const vector<*S*> &vdqs, const shared_ptr<SparseMatrixGroup<*S*, *FL*>> &vmats, int &vidx, int tvidx, bool do_reduce) const override

Partial tensor product multiplication operation: vmat = expr[L part | R part] x cmat. This is used only for perturbative noise, where only left or right block part of the Hamiltonian expression is multiplied by the wavefunction cmat, to get a perurbed wavefunction vmat.

Parameters

- **expr** – Symbolic expression in form of sum of tensor products.
- **lopt** – Symbol lookup table for left operands in the tensor products.
- **ropt** – Symbol lookup table for right operands in the tensor products.
- **trace_right** – If true, the left operands in the tensor products are used. The right operands are treated as identity. Otherwise, the right operands in the tensor products are used.
- **cmat** – Input "vector" operand (wavefunction).
- **psubsl** – Vector of transpose pattern and delta quantum of the "matrix" operand (in the same order as cinfos).
- **cinfos** – Vector of sparse matrix connection info (in the same order as cinfos).
- **vdqs** – Vector of quantum number of each vmat (for lookup).
- **vmats** – Vector of output "vectors" (perurbed wavefunctions).
- **vidx** – If -1, there is only one perurbed wavefunction for each target quantum number (used in NoiseTypes::ReducedPerturbative). Otherwise, one vmat is created for each tensor product (used in NoiseTypes::Perturbative), and vidx is used as an incremental index in vmats.
- **tvidx** – If -1, vmats is copied to every thread, which is the high memory mode but may be more load-balanced, the multi-thread parallelization is over different terms in expr for this case. If -2, every thread works on a single vmat, which is the low memory mode (used in NoiseTypes:: NoiseTypes::LowMem), the multi-thread parallelization is over different vmats for this case. Otherwise, if >= 0, only the specified vmat is handled, which is used only internally.
- **do_reduce** – If true, the output vmats are accumulated to the root processor in the distributed parallel case.

inline virtual void **tensor_product_multi_multiply**(const shared_ptr<OpExpr<*S*>> &expr, const shared_ptr<OperatorTensor<*S*, *FL*>> &lopt, const shared_ptr<OperatorTensor<*S*, *FL*>> &ropt, const shared_ptr<SparseMatrixGroup<*S*, *FL*>> &cmats, const shared_ptr<SparseMatrixGroup<*S*, *FL*>> &vmats, const unordered_map<*S*, shared_ptr<typename SparseMatrixInfo<*S*>::ConnectionInfo>> &cinfos, *S* opdq, *FL* factor, bool all_reduce) const override

Tensor product multiplication operation (multi-root case): vmats = expr x cmats. Both cmats and vmats are wavefunctions with multi-target components.

Parameters

- **expr** – Symbolic expression in form of sum of tensor products.
- **lopt** – Symbol lookup table for left operands in the tensor products.
- **ropt** – Symbol lookup table for right operands in the tensor products.
- **cmats** – Input "vector" operand (multi-target wavefunction).
- **vmats** – Output "vector" result (multi-target wavefunction).
- **cinfos** – Lookup table of sparse matrix connection info, where the key is the combined quantum number of the vmat and cmat.
- **opdq** – The delta quantum number of expr.
- **factor** – Sacling factor applied to the results.
- **all_reduce** – If true, the output result is accumulated and broadcast to all processors.

inline virtual void **tensor_product_multiply**(const shared_ptr<OpExpr<*S*>> &expr, const shared_ptr<OperatorTensor<*S*, *FL*>> &lopt, const shared_ptr<OperatorTensor<*S*, *FL*>> &ropt, const shared_ptr<SparseMatrix<*S*, *FL*>> &cmat, const shared_ptr<SparseMatrix<*S*, *FL*>> &vmat, *S* opdq, bool all_reduce) const override

Tensor product multiplication operation (single-root case): vmat = expr x cmat.

Parameters

- **expr** – Symbolic expression in form of sum of tensor products.
- **lopt** – Symbol lookup table for left operands in the tensor products.
- **ropt** – Symbol lookup table for right operands in the tensor products.
- **cmat** – Input "vector" operand (wavefunction), assuming the cinfo is already attached.
- **vmat** – Output "vector" result (wavefunction).
- **opdq** – The delta quantum number of expr.
- **all_reduce** – If true, the output result is accumulated and broadcast to all processors.

inline virtual void **tensor_product_diagonal**(const shared_ptr<OpExpr<*S*>> &expr, const shared_ptr<OperatorTensor<*S*, *FL*>> &lopt, const shared_ptr<OperatorTensor<*S*, *FL*>> &ropt, const shared_ptr<SparseMatrix<*S*, *FL*>> &mat, *S* opdq) const override

Extraction of diagonal of a tensor product expression: mat = diag(expr).

Parameters

- **expr** – Symbolic expression in form of sum of tensor products.
- **lopt** – Symbol lookup table for left operands in the tensor products.
- **ropt** – Symbol lookup table for right operands in the tensor products.
- **mat** – Output "vector" result (diagonal part).
- **opdq** – The delta quantum number of expr.

inline virtual void **tensor_product**(const shared_ptr<OpExpr<*S*>> &expr, const unordered_map<shared_ptr<OpExpr<*S*>>, shared_ptr<SparseMatrix<*S*, *FL*>>> &lop, const unordered_map<shared_ptr<OpExpr<*S*>>, shared_ptr<SparseMatrix<*S*, *FL*>>> &rop, shared_ptr<SparseMatrix<*S*, *FL*>> &mat) const override

Direct evaluation of a tensor product expression: mat = eval(expr).

Parameters

- **expr** – Symbolic expression in form of sum of tensor products.
- **lop** – Symbol lookup table for left operands in the tensor products.
- **rop** – Symbol lookup table for right operands in the tensor products.
- **mat** – Output "vector" result (the sparse matrix value of the expression).

inline virtual void **left_rotate**(const shared_ptr<OperatorTensor<*S*, *FL*>> &a, const shared_ptr<SparseMatrix<*S*, *FL*>> &mpst_bra, const shared_ptr<SparseMatrix<*S*, *FL*>> &mpst_ket, shared_ptr<OperatorTensor<*S*, *FL*>> &c) const override

Rotation (renormalization) of a left-block operator tensor: c = mpst_bra.T x a x mpst_ket. In the above expression, [x] means multiplication. Note that the row and column of mpst_bra and mpst_ket are for system and environment indices, respectively. Left rotation means that system indices are contracted.

Parameters

- **a** – Input operator tensor a (as a row vector of symbols).
- **mpst_bra** – Rotation matrix (bra MPS tensor) for row of sparse matrices in a.
- **mpst_ket** – Rotation matrix (ket MPS tensor) for column of sparse matrices in a.
- **c** – Output operator tensor c (as a row vector of symbols).

inline virtual void **right_rotate**(const shared_ptr<OperatorTensor<*S*, *FL*>> &a, const shared_ptr<SparseMatrix<*S*, *FL*>> &mpst_bra, const shared_ptr<SparseMatrix<*S*, *FL*>> &mpst_ket, shared_ptr<OperatorTensor<*S*, *FL*>> &c) const override

Rotation (renormalization) of a right-block operator tensor: c = mpst_bra x a x mpst_ket.T. In the above expression, [x] means multiplication. Note that the row and column of mpst_bra and mpst_ket are for system and environment indices, respectively. Right rotation means that environment indices are contracted.

Parameters

- **a** – Input operator tensor a (as a column vector of symbols).
- **mpst_bra** – Rotation matrix (bra MPS tensor) for row of sparse matrices in a.
- **mpst_ket** – Rotation matrix (ket MPS tensor) for column of sparse matrices in a.
- **c** – Output operator tensor c (as a column vector of symbols).

inline virtual void **intermediates**(const shared_ptr<Symbolic<*S*>> &names, const shared_ptr<Symbolic<*S*>> &exprs, const shared_ptr<OperatorTensor<*S*, *FL*>> &a, bool left) const override

Compute the intermediates to speed up the tensor product operations in the next blocking step. Intermediates are formed by collecting terms sharing the same left or right operands during the MPO simplification step.

Parameters

- **names** – Operator names (symbols, only used in distributed parallelization).
- **exprs** – Tensor product expressions (expressions of symbols).
- **a** – Symbol lookup table for symbols in the tensor product expressions (updated).
- **left** – Whether this is for left-blocking or right-blocking.

inline virtual void **numerical_transform**(const shared_ptr<OperatorTensor<*S*, *FL*>> &a, const shared_ptr<Symbolic<*S*>> &names, const shared_ptr<Symbolic<*S*>> &exprs) const override

Numerical transform from normal operators to complementary operators near the middle site.

Parameters

- **a** – Symbol lookup table for symbols in the tensor product expressions (updated).
- **names** – List of complementary operator names (symbols).

- **exprs** – List of symbolic expression of complementary operators as linear combination of normal operators.

inline virtual void **left_contract**(const shared_ptr<OperatorTensor<*S*, *FL*>> &a, const shared_ptr<OperatorTensor<*S*, *FL*>> &b, shared_ptr<OperatorTensor<*S*, *FL*>> &c, const shared_ptr<Symbolic<*S*>> &cexprs = nullptr, OpNamesSet delayed = OpNamesSet()) const override

Tensor product operation in left blocking: c = a x b.

Parameters

- **a** – Operator a (left block tensor).
- **b** – Operator b (dot block single-site tensor).
- **c** – Operator c (enlarged left block tensor).
- **cexprs** – Symbolic expression for the tensor product operation. If nullptr, this is automatically contructed from expressions in a and b.
- **delayed** – The set of operator names for which the tensor product operation should be delayed. The delayed tensor product will not be performed here. Instead, it will be evaluated as three-tensor operations later.

inline virtual void **right_contract**(const shared_ptr<OperatorTensor<*S*, *FL*>> &a, const shared_ptr<OperatorTensor<*S*, *FL*>> &b, shared_ptr<OperatorTensor<*S*, *FL*>> &c, const shared_ptr<Symbolic<*S*>> &cexprs = nullptr, OpNamesSet delayed = OpNamesSet()) const override

Tensor product operation in right blocking: c = b x a.

Parameters

- **a** – Operator a (right block tensor).
- **b** – Operator b (dot block single-site tensor).
- **c** – Operator c (enlarged right block tensor).
- **cexprs** – Symbolic expression for the tensor product operation. If nullptr, this is automatically contructed from expressions in b and a.
- **delayed** – The set of operator names for which the tensor product operation should be delayed. The delayed tensor product will not be performed here. Instead, it will be evaluated as three-tensor operations later.

Public Members

string **filename** = ""

The name of the associated disk file.

mutable int64_t **offset** = 0

Byte offset in the file (where to read/write the content).

6.5 Tools

6.5.1 Floating-Point Number Compression

For data like `FCIDUMP` integrals, the array elements do not need to be stored in its full binary form. Storage or memory can be saved by using flexible bit representation for numbers of different magnitudes. The `FPCodec` class implements the lossless and lossy compression and decompression of floating-point number arrays. Lossless compression is possible when all numebrs are close in their order of magnitudes.

template<typename **T**>

struct **FPtraits**

Floating-point number representation details.

Template Parameters
T – The floating type to implement.

Public Types

typedef *T* **U**

The representation integer type.

Public Static Attributes

static const int **mbits** = sizeof(*T*) * 8

Number of bits in significand.

static const int **ebits** = 0

Number of bits in exponent.

template<>

struct **FPtraits**<float>

Representation details for single precision numbers.

Public Types

typedef uint32_t **U**

The representation integer type.

Public Static Attributes

static const int **mbits** = 23

Number of bits in significand.

static const int **ebits** = 8

Number of bits in exponent.

template<>

struct **FPtraits**<double>

Representation details for double precision numbers.

Public Types

typedef uint64_t **U**

The representation integer type.

Public Static Attributes

static const int **mbits** = 52

Number of bits in significand.

static const int **ebits** = 11

Number of bits in exponent.

template<typename **T**, typename **U** = typename *FPtraits*<*T*>::U, int **mbits** = *FPtraits*<*T*>::mbits, int **ebits** = *FPtraits*<*T*>::ebits>
inline string `block2::`**binary_repr**(*T* d)

Represent the binary form of a floating-point or integer number as a string.

Template Parameters

- **T** – The floating type to implement.
- **U** – The representation integer type.
- **mbits** – Number of bits in significand.

- **ebits** – Number of bits in exponent.

Parameters
d – The number to be represented.

Returns
The binary form of the number as a string.

template<typename **T**, typename **U**>

struct **BitsCodec**

Read/write bits from/into a floating-point number array.

Template Parameters

- **T** – The floating type for the elemenet of the array.
- **U** – The corresponding integer/representation type of T.

Public Functions

inline **BitsCodec**(*T* *op_data)

Constructor.

Parameters
op_data – The floating-point number array for storing the bits.

inline void **begin_decode**()

Prepare for decoding.

template<typename **X**>
inline void **encode**(*X* x, int l)

Encode data and write into the array.

Template Parameters
X – The type of the data.

Parameters

- **x** – The data to encode.
- **l** – The number of bits of the data.

template<typename **X**>
inline void **decode**(*X* &x, int l)

Read from the array and decode data.

Template Parameters
X – The type of the data.

Parameters

- **x** – The output decoded data.
- **l** – The number of bits of the data.

inline size_t **finish_encode**()

Finalize encoding.

Public Members

U **buf**

Current buffer.

T ***op_data**

The floating-point number array for storing the bits.

size_t **d_offset**

The position in the array of the current buffer.

int **i_offset**

Number of bits already used in the current buffer.

Public Static Attributes

static const int **i_length** = sizeof(*U*) * 8

Number of bits in single array element.

template<typename **T**, typename **U** = typename *FPtraits*<*T*>::U, int **mbits** = *FPtraits*<*T*>::mbits, int **ebits** = *FPtraits*<*T*>::ebits>
struct **FPCodec**

Codec for compressing/decompressing array of floating-point numbers.

Template Parameters

- **T** – Floating type to implement.
- **U** – The corresponding integer type.
- **mbits** – Number of bits in significand.
- **ebits** – Number of bits in exponent.

Public Functions

inline **FPCodec**()

Default constructor.

inline **FPCodec**(*T* prec)

Constructor.

Parameters

prec – Floating-point number precision.

inline **FPCodec**(*T* prec, size_t chunk_size)

Constructor.

Parameters

- **prec** – Floating-point number precision.
- **chunk_size** – Length of the array elements that should be processed at one time.

inline size_t **decode**(*T* *ip_data, size_t len, *T* *op_data) const

Decode data.

Parameters

- **ip_data** – The compressed floating-point array.
- **len** – Length of the original floating-point array.
- **op_data** – Output array for storing original data. Memory should be pre-allocated with length = len.

Returns

Length of the array for the compressed data.

inline size_t **encode**(*T* *ip_data, size_t len, *T* *op_data) const

Encode data.

Parameters

- **ip_data** – The original floating-point array.
- **len** – Length of the original floating-point array.
- **op_data** – Output array for storing compressed data. Memory should be pre-allocated with length >= len + 1.

Returns

Length of the array for the compressed data.

inline void **write_array**(ostream &ofs, *T* *data, size_t len) const

Compress array of floating-point data and write into file stream.

Parameters

- **ofs** – Output stream.

- **data** – The original floating-point array.
- **len** – The length of the original floating-point array.

inline void **read_array**(istream &ifs, *T* *data, size_t len) const

Read from file stream and deompress the data.

Parameters

- **ifs** – Input stream.
- **data** – The floating-point array for storing the original data.
- **len** – The length of the original floating-point array.

inline void **read_chunks**(istream &ifs, size_t len, vector<vector<*T*>> &chunks, size_t &chunk_size) const

Read from file stream (but not decompress the data).

Parameters

- **ifs** – Input stream.
- **len** – The length of the original floating-point array.
- **chunks** – For storing chunks of the compressed data.
- **chunk_size** – Number of original array elements in each chunk (output).

Public Members

T **prec**

Precision for compression.

U **prec_u**

Integer representation of the precision.

mutable size_t **ndata** = 0

Length of the array of the data that has been compressed.

size_t **ncpsd** = 0

Length of the array for the compressed data.

size_t **chunk_size** = 4096

Length of the array elements that should be processed at one time.

size_t **n_parallel_chunks** = 4096

Number of chunks to be processed in the same batch.

Public Static Attributes

static const int **m** = *mbits*

Number of bits in significand.

static const *U* **e** = *U*(1) << *mbits*

Exponent least significant bit mask.

static const *U* **s** = *e* << *ebits*

Sign bit mask.

static const *U* **x** = ~(*e* + *s* - 1)

Exponent mask.

template<typename **T**>

struct **CompressedVector**

An array with data stored in compressed form. Only support single thread.

Template Parameters
T – Elemenet type.

Subclassed by *block2::CompressedVectorMT< T >*

Public Functions

inline **CompressedVector**(size_t arr_len, *T* prec, size_t chunk_size, int ncache = 4)

Constructor.

Parameters

- **arr_len** – Size of the array.
- **prec** – Precision for compression.
- **chunk_size** – Number of array elements in each chunk.
- **ncache** – Number of cached decompressed chunks.

inline **CompressedVector**(istream &ifs, size_t arr_len, *T* prec, int ncache = 4)

Constructor.

Parameters

- **ifs** – Input stream, from which the compressed data is obtained.
- **arr_len** – Size of the array.
- **prec** – Precision for compression.
- **ncache** – Number of cached decompressed chunks.

inline **CompressedVector**(*T* *arr, size_t arr_len, *T* prec, int ncache = 4)

Constructor.

Parameters

- **arr** – The compressed data as an array.
- **arr_len** – Size of the array.
- **prec** – Precision for compression.
- **ncache** – Number of cached decompressed chunks.

virtual **~CompressedVector**() = default

Deconstructor.

inline void **clear**()

Set all array elements to zero.

inline void **shrink_to_fit**()

Minimize the storage cost of the compressed data (after the data has been changed).

inline void **finalize**()

Write all cached data into compressed form.

inline virtual *T* &**operator[]**(size_t i)

Write one element into the array.

Parameters
i – Array index.

Returns
The array elemenet.

inline virtual *T* **operator[]**(size_t i) const

Read one element from the array.

Parameters
i – Array index.

Returns
The array elemenet.

inline size_t **size**() const

Get the size of the array.

Public Members

size_t **arr_len**
: Size of the array.

size_t **chunk_size**
: Number of array elements in each chunk.

int **ncache**
: Number of cached decompressed chunks.

mutable int **icache**
: Index of next available cache (head of the circular queue).

mutable vector<vector<*T*>> **cp_data**
: Chunks of compressed data.

mutable vector<pair<size_t, vector<*T*>>> **cache_data**
: Cached data of decompressed chunks.

mutable vector<bool> **cache_dirty**
: Whether each cached chunk has been changed.

FPCodec<*T*> **fpc**
: Floating-point number compression driver.

template<typename **T**>

struct **CompressedVectorMT** : public block2::*CompressedVector*<*T*>

A read-only array with data stored in compressed form, with multi-threading support.

Template Parameters
: **T** – Elemenet type.

Public Functions

inline **CompressedVectorMT**(const shared_ptr<*CompressedVector*<*T*>> &ref_cv, int ntg)
: Constructor.

Parameters

- **ref_cv** – Pointer to the single-thread compressed array.
- **ntg** – Number of threads.

inline virtual *T* &**operator[]**(size_t i)

Write one element into the array (not supported, will cause assertion failure).

Parameters
i – Array index.

Returns
The array elemenet.

inline virtual *T* **operator[]**(size_t i) const

Read one element from the array.

Parameters
i – Array index.

Returns
The array elemenet.

inline size_t **size**() const

Get the size of the array.

Public Members

mutable vector<int> **icaches**

Index of next available cache (head of the circular queue) for each thread.

mutable vector<vector<pair<size_t, vector<*T*>>>> **cache_datas**

Cached data of decompressed chunks for each thread.

shared_ptr<*CompressedVector*<*T*>> **ref_cv**

Pointer to the single-thread compressed array.

6.5.2 Kuhn-Munkres Algorithm

The `KuhnMunkres` class implements the Kuhn-Munkres algorithm for finding the best matching in a bipartite with the lowest cost.

struct **KuhnMunkres**

Kuhn-Munkres algorithm for finding the best matching with lowest cost. Complexity: O(n ^ 3).

Public Functions

inline **KuhnMunkres**(const vector<double> &cost_matrix, int m, int n = 0)

Constructor.

Parameters

- **cost_matrix** – Flattened matrix of cost.
- **m** – Number of rows.
- **n** – Number of columns.

inline bool **match**(vector<int> &x, int u)

Find an augmenting alternating path in the equality subgraph. If an equality subgraph has a perfect matching, the it is a maximum-weight matching in the graph.

Parameters

- **x** – Current matching.
- **u** – Starting vertex.

Returns

`true` if an augmenting alternating path is found.

inline pair<double, vector<int>> **solve**()

Find the lowest cost and a matching.

Returns

the lowest cost and an index array x, with x[i] = j meaning that column index i should be matched to row index j.

Public Members

vector<double> **cost**

Flattened n x n matrix of cost.

int **n**

Number of rows or columns.

double **inf**

Infinity (constant).

double **eps**

Machine precision (constant).

vector<double> **lx**

Left feasible labeling (working array).

vector<double> **ly**

Right feasible labeling (working array).

vector<double> **slack**

Slack working array.

vector<char> **st**

Candidate augmenting alternating path (working array).

6.5.3 Number Theory Algorithms

The `Prime` class includes some number theory algorithms necessary for integer factorization and finding primitive roots, which is used in some Fast Fourier Transform algorithms.

struct **Prime**

Number theory algorithms for prime numbers and primitive root.

Public Functions

inline **Prime**()

Default constructor.

inline void **init_primes**(int np = 50000)

Initialize set of small primes, using sieve of Eratosthenes.

Parameters

np – Maximal number considered for finding primes.

inline void **factors**(LL x, vector<pair<LL, int>> &pp)

Integer factorization.

Parameters

- **x** – Integer x.
- **pp** – (output) Set of prime factors and their occurrence.

inline LL **euler**(LL n)

Euler's totient function.

Parameters

n – Integer n (not necessarily prime).

Returns

phi(n), which counts the positive integers up to the given integer n that are relatively prime to n.

inline bool **is_prime**(LL n)

Prime testing.

Parameters
n – Integer n to be tested (n < 2^63 - 1).

Returns
`true` if n is a prime, `false` if n is not a prime.

inline int **primitive_root**(LL p)

Find one of primitive roots modulo n. A number g is a primitive root modulo n if every number a coprime to n is congruent to a power of g modulo n.

Parameters
p – Modulus p.

Returns
a primitive root g.

inline void **primitive_roots**(LL p, vector<LL> &gg)

Find all primitive roots modulo n.

Parameters

- **p** – Modulus p.
- **gg** – (output) All primitive roots modulo n.

Public Members

int **np**

Maximal number considered for generation of set of small primes.

vector<int> **primes**

Set of small primes.

Public Static Functions

static inline int **pmod**(LL x, LL n)

Return positive x mod n.

static inline void **exgcd**(LL a, LL b, LL &d, LL &x, LL &y)

Extended Euclidean algorithm. Find integers x and y such that a x + b y = gcd(a, b).

Parameters

- **a** – Integer a.
- **b** – Integer b.
- **d** – (output) Greatest Common Divisor gcd(a, b).

- **x** – (output) Integer x.
- **y** – (output) Integer y.

static inline LL **gcd**(LL a, LL b)

Euclidean algorithm. Find Greatest Common Divisor gcd(a, b).

Parameters

- **a** – Integer a.
- **b** – Integer b.

Returns

Greatest Common Divisor gcd(a, b).

static inline LL **inv**(LL a, LL n)

Find modular multiplicative inverse of a (mod n). Using extended Euclidean algorithm.

Parameters

- **a** – Integer a.
- **n** – Modulus n.

Returns

a ^ {-1} (mod n) if it exists, otherwise -1.

static inline LL **power**(LL n, int i)

Find n to the power of i using binary algorithm.

Parameters

- **n** – Integer n.
- **i** – Integer i (i >= 0).

Returns

n ^ i.

static inline LL **sqrt**(LL x)

Find the largest number r such that r * r <= x. Using binary algorithm.

Parameters

x – Integer x.

Returns

floor(sqrt(x)).

static inline LL **quick_multiply**(LL x, LL y, LL p)

Find (x * y) mod p. Note that the expression x * y % p may overflow if the intermediate x * y > 2 ^ 63 - 1. This function will not overflow (if p <= 2 ^ 62 - 1)

Parameters

- **x** – Integer x.
- **y** – Integer y.

- **p** – Modulus p.

Returns
(x * y) mod p.

static inline LL **quick_power**(LL n, LL i, LL p)

Find (n ^ i) mod p using binary algorithm.

Parameters

- **n** – Integer n.
- **i** – Integer i (i >= 0).
- **p** – Modulus p.

Returns
(n ^ i) mod p.

static inline bool **miller_rabin**(LL a, LL n)

Miller-Rabin primality test.

Note: If n < 3215031751, it is enough to test a = 2, 3, 5, and 7; if n < 341550071728321, it is enough to test a = 2, 3, 5, 7, 11, 13, and 17.

Parameters

- **a** – Base number a.
- **n** – The number to be tested for prime (n >= 3).

Returns
`true` if n is likely to be a prime. `false` if n is not a prime.

static inline LL **pollard_rho**(LL n)

Pollard's rho algorithm. Find a factor of integer n.

Parameters
n – Integer n.

Returns
A (not necessarily prime) factor of n.

6.5.4 Fast Fourier Transform (FFT)

A collection of FFT algorithms is implemented here. The implementation focuses more on flexibility and readability, rather than optimal performance (but the performance should be acceptable).

The `block2` implementation is faster than `numpy` implementation for some special array lengths, such as `5929741 = 181 ** 3` and `5929742 = 2 * 7 * 13 * 31 * 1051`.

template<int **base**>

struct **BasicFFT**

Fast Fourier Transform (FFT) with array length of base^k (k >= 0). The complexity is O(n log_base n * base^2).

Template Parameters
base – The radix (base = 2 is specialized)

Public Functions

inline **BasicFFT**()

Constructor.

inline void **init**(size_t n)

Precompute for array length n for both forward and backward FFT.

Parameters
n – The array length, must be a power of base.

inline void **fft**(complex<double> *arr, size_t n, bool forth)

Perform inplace FFT.

Parameters

- **arr** – A pointer to the array of complex numbers.
- **n** – Number of elements in the array, must be a power of base.
- **forth** – Whether this is forward transform.

Public Members

vector<size_t> **r**

The index permutation array.

vector<complex<double>> **wb**

The precomputed primitive nth root of 1 exp(i2pi k/n) for backward FFT.

vector<complex<double>> **wf**

The precomputed primitive nth root of 1 exp(-i2pi k/n) for forward FFT.

array<array<complex<double>, *base*>, *base*> **xw**[2]

The precomputed primitive base-th root of 1 exp(-/+ i2pi jk/base) for forward/backward FFT.

Public Static Functions

static inline size_t **pad**(size_t n)

Find smallest number x such that x = base^k >= n.

Parameters
n – The array length.

Returns
The padded array length.

template<>

struct **BasicFFT**<2>

Radix-2 Fast Fourier Transform (FFT) with complexity O(nlog n).

Public Functions

inline **BasicFFT**()

Constructor.

inline void **init**(size_t n)

Precompute for array length n for both forward and backward FFT.

Parameters
n – The array length must be a power of 2.

inline void **fft**(complex<double> *arr, size_t n, bool forth)

Perform inplace FFT.

Parameters

- **arr** – A pointer to the array of complex numbers.
- **n** – Number of elements in the array, must be a power of 2.
- **forth** – Whether this is forward transform.

Public Members

vector<size_t> **r**

The index permutation array.

vector<complex<double>> **wb**

The precomputed primitive nth root of 1 exp(i2pi k/n) for backward FFT.

vector<complex<double>> **wf**

The precomputed primitive nth root of 1 exp(-i2pi k/n) for forward FFT.

Public Static Functions

static inline size_t **pad**(size_t n)

Find smallest number x such that x = 2 ^ k >= n.

Parameters

n – The array length.

Returns

The padded array length.

template<typename **B** = *BasicFFT*<2>>

struct **RaderFFT**

Rader's FFT algorithm for prime array length. This algorithm transforms a FFT of length n to two (forward and backward) FFTs of length m, where m is the padded array length for 2 * n - 3 for the backend FFT.

Template Parameters

B – The backend FFT for computing the padded FFT.

Public Functions

inline **RaderFFT**()

Default constructor.

inline **RaderFFT**(const shared_ptr<*Prime*> &prime)

Constructor.

Parameters

prime – Instance for prime number algorithms.

inline void **init**(size_t n)

Precompute for array length n for both forward and backward FFT.

Parameters

n – The array length, which must be a prime number.

inline void **fft**(complex<double> *arr, size_t n, bool forth)

Perform inplace FFT.

Parameters

- **arr** – A pointer to the array of complex numbers.
- **n** – Number of elements in the array, which must be a prime number.
- **forth** – Whether this is forward transform.

Public Members

vector<LL> **wb**

Precomputed inverse of the power of primitive root g ^ {-k} mod n for k = 0, 1, . . . , n - 1.

vector<LL> **wf**

Precomputed power of primitive root g ^ k mod n for k = 0, 1, . . . , n - 1.

vector<complex<double>> **cb**

FFT transformed exp(i2pi g ^ {-k}/n).

vector<complex<double>> **cf**

FFT transformed exp(-i2pi g ^ {-k}/n).

vector<complex<double>> **arx**

Working space for padded array.

size_t **nn**

Padded array length.

B **b**

The backend FFT instance.

shared_ptr<*Prime*> **prime**

Instance for prime number algorithms.

template<typename **B** = *BasicFFT*<2>>

struct **BluesteinFFT**

Bluestein's FFT algorithm for arbitrary array length. This algorithm transforms a FFT of length n to two (forward and backward) FFTs of length m, where m is the padded array length for 2 * n for the backend FFT.

Template Parameters
B – The backend FFT for computing the padded FFT.

Public Functions

inline **BluesteinFFT**()
Default constructor.

inline **BluesteinFFT**(const shared_ptr<*Prime*> &prime)
Constructor.

Parameters
prime – Instance for prime number algorithms (ignored).

inline void **init**(size_t n)
Precompute for array length n for both forward and backward FFT.

Parameters
n – The array length.

inline void **fft**(complex<double> *arr, size_t n, bool forth)
Perform inplace FFT.

Parameters

- **arr** – A pointer to the array of complex numbers.
- **n** – Number of elements in the array.
- **forth** – Whether this is forward transform.

Public Members

vector<complex<double>> **wb**
The precomputed primitive nth root of 1 exp(i2pi k/n) for backward FFT.

vector<complex<double>> **wf**
The precomputed primitive nth root of 1 exp(-i2pi k/n) for forward FFT.

vector<complex<double>> **cb**
FFT transformed exp(i2pi [((k - n) - (k - n)(k - n)) / 2]/n).

vector<complex<double>> **cf**
FFT transformed exp(-i2pi [((k - n) - (k - n)(k - n)) / 2]/n).

vector<complex<double>> **arx**
Working space for padded array.

size_t **nn**

Padded array length.

B **b**

The backend FFT instance.

struct **DFT**

Naive Discrete Fourier Transform (*DFT*) algorithm with complexity O(n ^ 2).

Public Functions

inline **DFT**()

Default constructor.

inline **DFT**(const shared_ptr<*Prime*> &prime)

Constructor.

Parameters

prime – Instance for prime number algorithms (ignored).

inline void **init**(size_t n)

Precompute for array length n for both forward and backward FFT. For *DFT* this method does nothing.

Parameters

n – The array length.

inline void **fft**(complex<double> *arr, size_t n, bool forth)

Perform inplace DFT.

Forward DFT: $X[k] = \sum_{j=0}^{n-1} x[j] \exp(-2\pi \mathrm{i} jk/n)$

Backward DFT: $X[k] = \frac{1}{n} \sum_{j=0}^{n-1} x[j] \exp(2\pi \mathrm{i} jk/n)$

Parameters

- **arr** – A pointer to the array of complex numbers.
- **n** – Number of elements in the array.
- **forth** – Whether this is forward transform.

template<typename **F**, int **P**, int... **Q**>

struct **FactorizedFFT**

FFT algorithm using different radix FFT backends. The array length is first factorized, then different FFT backends will be used and then the results is merged using the Cooley-Tukey FFT algorithm.

Template Parameters

- **F** – The prime number FFT backend.
- **P** – Using Radix-P FFT backend.
- **Q** – Using Radix-(Q1, Q2, . . .) FFT backend.

Public Functions

inline **FactorizedFFT**()

Default constructor.

inline **FactorizedFFT**(int max_factor)

Constructor.

Parameters

max_factor – Maximal radix that should be checked for radix based FFT.

inline void **fft_internal**(complex<double> *arr, size_t p, size_t q, bool forth, int b) override

Perform independent FFTs for p arrays, each with length q.

Parameters

- **arr** – A pointer to the array of complex numbers (as a matrix).
- **p** – Number of rows (FFTs).
- **q** – Number of columns (length of each FFT).
- **forth** – Whether this is forward transform.
- **b** – Radix. Zero if radix based FFT should not be used.

template<typename **F**, int **P**>

struct **FactorizedFFT**<*F*, *P*>

FFT algorithm using different radix FFT backends. The array length is first factorized, then different FFT backends will be used and then the results is merged using the Cooley-Tukey FFT algorithm.

Template Parameters

- **F** – The prime number FFT backend.
- **P** – Using Radix-P FFT backend.

Public Functions

inline **FactorizedFFT**()

Default constructor.

inline **FactorizedFFT**(int max_factor)

Constructor.

Parameters

max_factor – Maximal radix that should be checked for radix based FFT.

inline void **init**(size_t n)

Precompute for array length n for both forward and backward FFT. For *FactorizedFFT* this method does nothing.

Parameters

n – The array length.

inline virtual void **fft_internal**(complex<double> *arr, size_t p, size_t q, bool forth, int b)

Perform independent FFTs for p arrays, each with length q.

Parameters

- **arr** – A pointer to the array of complex numbers (as a matrix).
- **p** – Number of rows (FFTs).
- **q** – Number of columns (length of each FFT).
- **forth** – Whether this is forward transform.
- **b** – Radix. Zero if radix based FFT should not be used.

inline void **cooley_tukey**(complex<double> *arr, size_t n, bool forth, const size_t *pr, const int *b, size_t np)

Cooley-Tukey FFT algorithm for FFT with array length being a composite number.

Parameters

- **arr** – A pointer to the array of complex numbers.
- **n** – Number of elements in the array.
- **forth** – Whether this is forward transform.
- **pr** – A pointer to the array of factors in n;
- **b** – A pointer to the array of radices for each factor. pr should be multiple of b, if b is not zero.
- **np** – Number of factors.

inline void **fft**(complex<double> *arr, size_t n, bool forth)

Perform inplace FFT.

Forward FFT: $X[k] = \sum_{j=0}^{n-1} x[j] \exp(-2\pi \mathrm{i} jk/n)$

Backward FFT: $X[k] = \frac{1}{n} \sum_{j=0}^{n-1} x[j] \exp(2\pi \mathrm{i} jk/n)$

Parameters

- **arr** – A pointer to the array of complex numbers.
- **n** – Number of elements in the array.
- **forth** – Whether this is forward transform.

Public Members

const int **max_factor** = *P*

Maximal radix number.

shared_ptr<*Prime*> **prime**

Instance for prime number algorithms.

typedef *FactorizedFFT*<*RaderFFT*<>, 2, 3, 5, 7, 11> block2::**FFT2**

FFT with small prime factorization implemeted using Rader's algorithm.

typedef *FactorizedFFT*<*BluesteinFFT*<>, 2, 3, 5, 7, 11> block2::**FFT**

FFT with small prime factorization implemeted using Bluestein's algorithm.

CHAPTER

SEVEN

THEORY

7.1 DMRG Hamiltonian

7.1.1 DMRG Quantum Chemistry Hamiltonian in Spatial Orbitals

Hamiltonian

The quantum chemistry Hamiltonian is written as follows

$$\hat{H} = \sum_{ij,\sigma} t_{ij}\ a^\dagger_{i\sigma} a_{j\sigma} + \frac{1}{2} \sum_{ijkl,\sigma\sigma'} v_{ijkl}\ a^\dagger_{i\sigma} a^\dagger_{k\sigma'} a_{l\sigma'} a_{j\sigma}$$

where

$$t_{ij} = t_{(ij)} = \int \mathrm{d}\mathbf{x}\ \phi^*_i(\mathbf{x}) \left(-\frac{1}{2}\nabla^2 - \sum_a \frac{Z_a}{r_a} \right) \phi_j(\mathbf{x})$$

$$v_{ijkl} = v_{(ij)(kl)} = v_{(kl)(ij)} = \int \mathrm{d}\mathbf{x}_1 \mathrm{d}\mathbf{x}_2\ \frac{\phi^*_i(\mathbf{x}_1)\phi^*_k(\mathbf{x}_2)\phi_l(\mathbf{x}_2)\phi_j(\mathbf{x}_1)}{r_{12}}$$

Note that here the order of $ijkl$ is the same as that in `FCIDUMP` (chemist's notation $[ij|kl]$).

Partitioning in Spatial Orbitals

The partitioning of Hamiltonian in left (L) and right (R) blocks is given by

$$\begin{aligned}
\hat{H} = {} & \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R \\
& + \Big(\sum_{i\in L,\sigma} a^\dagger_{i\sigma} \hat{S}^R_{i\sigma} + h.c. \Big) + \Big(\sum_{i\in L,\sigma} a^\dagger_{i\sigma} \hat{R}^R_{i\sigma} + h.c. + \sum_{i\in R,\sigma} a^\dagger_{i\sigma} \hat{R}^L_{i\sigma} + h.c. \Big) \\
& + \frac{1}{2} \Big(\sum_{ik\in L,\sigma\sigma'} \hat{A}^L_{ik,\sigma\sigma'} \hat{P}^R_{ik,\sigma\sigma'} + h.c. \Big) + \sum_{ij\in L} \hat{B}_{ij} \hat{Q}^R_{ij} - \sum_{il\in L,\sigma\sigma'} \hat{B}'_{il\sigma\sigma'} \hat{Q}'^R_{il\sigma\sigma'}
\end{aligned}$$

where the normal and complementary operators are defined by

$$\begin{aligned}
\hat{S}^{L/R}_{i\sigma} &= \sum_{j\in L/R} t_{ij} a_{j\sigma}, \\
\hat{R}^{L/R}_{i\sigma} &= \sum_{jkl\in L/R,\sigma'} v_{ijkl} a^\dagger_{k\sigma'} a_{l\sigma'} a_{j\sigma}, \\
\hat{A}_{ik,\sigma\sigma'} &= a^\dagger_{i\sigma} a^\dagger_{k\sigma'}, \\
\hat{B}_{ij} &= \sum_{\sigma} a^\dagger_{i\sigma} a_{j\sigma}, \\
\hat{B}'_{il,\sigma\sigma'} &= a^\dagger_{i\sigma} a_{l\sigma'}, \\
\hat{P}^{R}_{ik,\sigma\sigma'} &= \sum_{jl\in R} v_{ijkl} a_{l\sigma'} a_{j\sigma}, \\
\hat{Q}^{R}_{ij} &= \sum_{kl\in R,\sigma'} v_{ijkl} a^\dagger_{k\sigma'} a_{l\sigma'}, \\
\hat{Q}'^{R}_{il,\sigma\sigma'} &= \sum_{jk\in R} v_{ijkl} a^\dagger_{k\sigma'} a_{j\sigma}
\end{aligned}$$

Note that we need to move all on-site interaction into local Hamiltonian, so that when construction interaction terms in Hamiltonian, operators anticommute (without giving extra constant terms).

Derivation

First consider one-electron term. ij indices have only two possibilities: i left, j right, or i right, j left. Index i must be associated with creation operator. So the second case is the Hermitian conjugate of the first case. Namely,

$$\sum_{i\in L,\sigma} a^\dagger_{i\sigma} \hat{S}^R_{i\sigma} + h.c. = \sum_{i\in L,\sigma} a^\dagger_{i\sigma} \hat{S}^R_{i\sigma} + \sum_{j\in L,\sigma} \hat{S}^{R\dagger}_{j\sigma} a_{j\sigma} = \sum_{i\in L/R, j\in R/L,\sigma} t_{ij} a^\dagger_{i\sigma} a_{j\sigma}$$

Next consider one of $ijkl$ in left, and three of them in right. These terms are

$$\begin{aligned}
\hat{H}_{1L,3R} &= \frac{1}{2} \sum_{i\in L,jkl\in R,\sigma\sigma'} v_{ijkl} a^\dagger_{i\sigma} a^\dagger_{k\sigma'} a_{l\sigma'} a_{j\sigma} + \frac{1}{2} \sum_{j\in L,ikl\in R,\sigma\sigma'} v_{ijkl} a^\dagger_{i\sigma} a^\dagger_{k\sigma'} a_{l\sigma'} a_{j\sigma} + \frac{1}{2} \sum_{k\in L,ijl\in R,\sigma\sigma'} v_{ijkl} a^\dagger_{i\sigma} a^\dagger_{k\sigma'} a_{l\sigma'} a_{j\sigma} + \\
&= \left[\frac{1}{2} \sum_{i\in L,jkl\in R,\sigma\sigma'} v_{ijkl} a^\dagger_{i\sigma} a^\dagger_{k\sigma'} a_{l\sigma'} a_{j\sigma} + \frac{1}{2} \sum_{k\in L,ijl\in R,\sigma\sigma'} v_{ijkl} a^\dagger_{i\sigma} a^\dagger_{k\sigma'} a_{l\sigma'} a_{j\sigma} \right] + \frac{1}{2} \sum_{j\in L,ikl\in R,\sigma\sigma'} v_{ijkl} a^\dagger_{i\sigma} a^\dagger_{k\sigma'} a_{l\sigma'} a_{j
\end{aligned}$$

where the terms in bracket equal to first and third terms in left-hand-side. Outside the bracket are second, forth terms.

The conjugate of third term in rhs is second term in rhs

$$\frac{1}{2} \sum_{j\in L,ikl\in R,\sigma\sigma'} v_{ijkl} a^\dagger_{j\sigma} a^\dagger_{l\sigma'} a_{k\sigma'} a_{i\sigma} = \frac{1}{2} \sum_{k\in L,ijl\in R,\sigma\sigma'} v_{lkji} a^\dagger_{k\sigma} a^\dagger_{i\sigma'} a_{j\sigma'} a_{l\sigma} = \frac{1}{2} \sum_{k\in L,ijl\in R,\sigma\sigma'} v_{ijkl} a^\dagger_{i\sigma'} a^\dagger_{k\sigma} a_{l\sigma} a_{j\sigma'}$$

The conjugate of forth term in rhs is first term in rhs

$$\frac{1}{2} \sum_{l\in L,ijk\in R,\sigma\sigma'} v_{ijkl} a^\dagger_{j\sigma} a^\dagger_{l\sigma'} a_{k\sigma'} a_{i\sigma} = \frac{1}{2} \sum_{i\in L,jkl\in R,\sigma\sigma'} v_{lkji} a^\dagger_{k\sigma} a^\dagger_{i\sigma'} a_{j\sigma'} a_{l\sigma} = \frac{1}{2} \sum_{i\in L,jkl\in R,\sigma\sigma'} v_{ijkl} a^\dagger_{i\sigma'} a^\dagger_{k\sigma} a_{l\sigma} a_{j\sigma'}$$

Therefore, using $v_{ijkl} = v_{klij}$

$$\begin{aligned}
\hat{H}_{1L,3R} &= \left[\frac{1}{2}\sum_{i\in L,jkl\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a^\dagger_{k\sigma'}a_{l\sigma'}a_{j\sigma} + \frac{1}{2}\sum_{k\in L,ijl\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a^\dagger_{k\sigma'}a_{l\sigma'}a_{j\sigma}\right] + h.c. \\
&= \left[\frac{1}{2}\sum_{i\in L,jkl\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a^\dagger_{k\sigma'}a_{l\sigma'}a_{j\sigma} + \frac{1}{2}\sum_{k\in L,ijl\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{k\sigma'}a^\dagger_{i\sigma}a_{j\sigma}a_{l\sigma'}\right] + h.c. \\
&= \left[\frac{1}{2}\sum_{i\in L,jkl\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a^\dagger_{k\sigma'}a_{l\sigma'}a_{j\sigma} + \frac{1}{2}\sum_{i\in L,jkl\in R,\sigma\sigma'} v_{klij}a^\dagger_{i\sigma'}a^\dagger_{k\sigma}a_{l\sigma}a_{j\sigma'}\right] + h.c. \\
&= \sum_{i\in L,jkl\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a^\dagger_{k\sigma'}a_{l\sigma'}a_{j\sigma} + h.c. \\
&= \sum_{i\in L,\sigma} a^\dagger_{i\sigma}\sum_{jkl\in R,\sigma'} v_{ijkl}a^\dagger_{k\sigma'}a_{l\sigma'}a_{j\sigma} + h.c. = \sum_{i\in L,\sigma} a^\dagger_{i\sigma}R^R_{i\sigma} + h.c.
\end{aligned}$$

Next consider the two creation operators together in left or in together in right. There are two cases. The second case is the conjugate of the first case, namely,

$$\sum_{ik\in R,jl\in L,\sigma\sigma'} a^\dagger_{i\sigma}a^\dagger_{k\sigma'}v_{ijkl}a_{l\sigma'}a_{j\sigma} = \sum_{jl\in R,ik\in L,\sigma\sigma'} a^\dagger_{j\sigma}a^\dagger_{l\sigma'}v_{jilk}a_{k\sigma'}a_{i\sigma} = \sum_{ik\in L,jl\in R,\sigma\sigma'} v_{jilk}a^\dagger_{j\sigma}a^\dagger_{l\sigma'}a_{k\sigma'}a_{i\sigma} = \sum_{ik\in L,jl\in R,\sigma\sigma'} v_{ij}$$

This explains the $\hat{A}\hat{P}$ term. The last situation is, one creation in left and one creation in right. Note that when exchange two elementary operators, one creation and one annihilation, one in left and one in right, they must anticommute.

$$\begin{aligned}
\hat{H}_{2L,2R} &= \frac{1}{2}\sum_{il\in L,jk\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a^\dagger_{k\sigma'}a_{l\sigma'}a_{j\sigma} + \frac{1}{2}\sum_{ij\in L,kl\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a^\dagger_{k\sigma'}a_{l\sigma'}a_{j\sigma} + \frac{1}{2}\sum_{kl\in L,ij\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a^\dagger_{k\sigma'}a_{l\sigma'}a_{j\sigma} + \\
&= -\frac{1}{2}\sum_{il\in L,jk\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a_{l\sigma'}a^\dagger_{k\sigma'}a_{j\sigma} + \frac{1}{2}\sum_{ij\in L,kl\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a_{j\sigma}a^\dagger_{k\sigma'}a_{l\sigma'} + \frac{1}{2}\sum_{kl\in L,ij\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a_{j\sigma}a^\dagger_{k\sigma'}a_{l\sigma'}
\end{aligned}$$

where the first, forth terms are combing different spins. The second, third terms are for the same spin. First consider the same-spin case

$$\begin{aligned}
&\frac{1}{2}\sum_{ij\in L,kl\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a_{j\sigma}a^\dagger_{k\sigma'}a_{l\sigma'} + \frac{1}{2}\sum_{kl\in L,ij\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a_{j\sigma}a^\dagger_{k\sigma'}a_{l\sigma'} \\
=& \frac{1}{2}\sum_{ij\in L,kl\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a_{j\sigma}a^\dagger_{k\sigma'}a_{l\sigma'} + \frac{1}{2}\sum_{kl\in L,ij\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{k\sigma'}a_{l\sigma'}a^\dagger_{i\sigma}a_{j\sigma} \\
=& \frac{1}{2}\sum_{ij\in L,kl\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a_{j\sigma}a^\dagger_{k\sigma'}a_{l\sigma'} + \frac{1}{2}\sum_{ij\in L,kl\in R,\sigma\sigma'} v_{klij}a^\dagger_{i\sigma'}a_{j\sigma'}a^\dagger_{k\sigma}a_{l\sigma} \\
=& \sum_{ij\in L,kl\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a_{j\sigma}a^\dagger_{k\sigma'}a_{l\sigma'} = \sum_{ij\in L}\sum_{\sigma} a^\dagger_{i\sigma}a_{j\sigma}\sum_{kl\in R_k}\sum_{\sigma'} v_{ijkl}a^\dagger_{k\sigma'}a_{l\sigma'} = \sum_{ij\in L}\hat{B}_{ij}\hat{Q}^R_{ij}
\end{aligned}$$

For the different-spin case,

$$\begin{aligned}
&-\frac{1}{2}\sum_{il\in L,jk\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a_{l\sigma'}a^\dagger_{k\sigma'}a_{j\sigma} - \frac{1}{2}\sum_{jk\in L,il\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a_{l\sigma'}a^\dagger_{k\sigma'}a_{j\sigma} = -\sum_{il\in L,jk\in R,\sigma\sigma'} v_{ijkl}a^\dagger_{i\sigma}a_{l\sigma'}a^\dagger_{k\sigma'}a_{j\sigma} \\
=& -\sum_{il\in L\sigma\sigma'} a^\dagger_{i\sigma}a_{l\sigma'}\sum_{jk\in R} v_{ijkl}a^\dagger_{k\sigma'}a_{j\sigma} = -\sum_{il\in L\sigma\sigma'} \hat{B}'_{il\sigma\sigma'}{\hat{Q}'_{il\sigma\sigma'}}^R
\end{aligned}$$

Normal/Complementary Partitioning

The above version is used when left block is short in length. Note that all terms should be written in a way that operators for particles in left block should appear in the left side of operator string, and operators for particles in right block should appear in the right side of operator string. To write the Hermitian conjugate explicitly, we have

$$\begin{aligned}\hat{H}^{NC} =\ & \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R \\ & + \sum_{i\in L,\sigma} \left(a_{i\sigma}^\dagger \hat{S}_{i\sigma}^R - a_{i\sigma} \hat{S}_{i\sigma}^{R\dagger}\right) + \sum_{i\in L,\sigma} \left(a_{i\sigma}^\dagger \hat{R}_{i\sigma}^R - a_{i\sigma} \hat{R}_{i\sigma}^{R\dagger}\right) + \sum_{i\in R,\sigma} \left(\hat{R}_{i\sigma}^{L\dagger} a_{i\sigma} - \hat{R}_{i\sigma}^L a_{i\sigma}^\dagger\right) \\ & + \frac{1}{2} \sum_{ik\in L,\sigma\sigma'} \left(\hat{A}_{ik,\sigma\sigma'} \hat{P}_{ik,\sigma\sigma'}^R + \hat{A}_{ik,\sigma\sigma'}^\dagger \hat{P}_{ik,\sigma\sigma'}^{R\dagger}\right) + \sum_{ij\in L} \hat{B}_{ij} \hat{Q}_{ij}^R - \sum_{il\in L,\sigma\sigma'} \hat{B}'_{il\sigma\sigma'} \hat{Q}'^R_{il\sigma\sigma'}\end{aligned}$$

Note that no minus sign for Hermitian conjugate terms with A, P because these are not Fermion operators.

Also note that

$$\sum_{i\in L,\sigma} a_{i\sigma}^\dagger \hat{S}_{i\sigma}^R = \sum_{i\in L, j\in R,\sigma} t_{ij} a_{i\sigma}^\dagger a_{j\sigma} = \sum_{j\in R,\sigma} S_{j\sigma}^{L\dagger} a_{j\sigma}$$

Define

$$\hat{R}'^{L/R}_{i\sigma} = \frac{1}{2} \hat{S}_{i\sigma}^{L/R} + \hat{R}_{i\sigma}^{L/R} = \frac{1}{2} \sum_{j\in L/R} t_{ij} a_{j\sigma} + \sum_{jkl\in L/R,\sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma}$$

we have

$$\begin{aligned}\hat{H}^{NC} =\ & \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R + \sum_{i\in L,\sigma} \left(a_{i\sigma}^\dagger \hat{R}'^R_{i\sigma} - a_{i\sigma} \hat{R}'^{R\dagger}_{i\sigma}\right) + \sum_{i\in R,\sigma} \left(\hat{R}'^{L\dagger}_{i\sigma} a_{i\sigma} - \hat{R}'^L_{i\sigma} a_{i\sigma}^\dagger\right) \\ & + \frac{1}{2} \sum_{ik\in L,\sigma\sigma'} \left(\hat{A}_{ik,\sigma\sigma'} \hat{P}_{ik,\sigma\sigma'}^R + \hat{A}_{ik,\sigma\sigma'}^\dagger \hat{P}_{ik,\sigma\sigma'}^{R\dagger}\right) + \sum_{ij\in L} \hat{B}_{ij} \hat{Q}_{ij}^R - \sum_{il\in L,\sigma\sigma'} \hat{B}'_{il\sigma\sigma'} \hat{Q}'^R_{il\sigma\sigma'}\end{aligned}$$

With this normal/complementary partitioning, the operators required in left block are

$$\left\{\hat{H}^L, \hat{1}^L, a_{i\sigma}^\dagger, a_{i\sigma}, \hat{R}'^{L\dagger}_{k\sigma}, \hat{R}'^L_{k\sigma}, \hat{A}_{ij,\sigma\sigma'}, \hat{A}_{ij,\sigma\sigma'}^\dagger, \hat{B}_{ij}, \hat{B}'_{ij,\sigma\sigma'}\right\} \quad (i,j \in L,\ k \in R)$$

The operators required in right block are

$$\left\{\hat{1}^R, \hat{H}^R, \hat{R}'^R_{i\sigma}, \hat{R}'^{R\dagger}_{i\sigma}, a_{k\sigma}, a_{k\sigma}^\dagger, \hat{P}_{ij,\sigma\sigma'}^R, \hat{P}_{ij,\sigma\sigma'}^{R\dagger}, \hat{Q}_{ij}^R, \hat{Q}'^R_{ij,\sigma\sigma'}\right\} \quad (i,j \in L,\ k \in R)$$

Assuming that there are K sites in total, and K_L/K_R sites in left/right block (optimally, $K_L \le K_R$), the total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$N_{NC} = 1 + 1 + 4K_L + 4K_R + 8K_L^2 + K_L^2 + 4K_L^2 = 13K_L^2 + 4K + 2$$

Complementary/Normal Partitioning

$$\hat{H}^{CN} = \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R + \sum_{i\in L,\sigma} \left(a^\dagger_{i\sigma} \hat{R}'^R_{i\sigma} - a_{i\sigma} \hat{R}'^{R\dagger}_{i\sigma} \right) + \sum_{i\in R,\sigma} \left(\hat{R}'^{L\dagger}_{i\sigma} a_{i\sigma} - \hat{R}'^L_{i\sigma} a^\dagger_{i\sigma} \right)$$
$$+ \frac{1}{2} \sum_{jl\in R,\sigma\sigma'} \left(\hat{P}^L_{jl,\sigma\sigma'} \hat{A}_{jl,\sigma\sigma'} + \hat{P}^{L\dagger}_{jl,\sigma\sigma'} \hat{A}^\dagger_{jl,\sigma\sigma'} \right) + \sum_{kl\in R} \hat{Q}^L_{kl} \hat{B}_{kl} - \sum_{jk\in R,\sigma\sigma'} \hat{Q}'^L_{jk\sigma\sigma'} \hat{B}'_{jk\sigma\sigma'}$$

Now the operators required in left block are

$$\left\{ \hat{H}^L, \hat{1}^L, a^\dagger_{i\sigma}, a_{i\sigma}, \hat{R}'^{L\dagger}_{k\sigma}, \hat{R}'^L_{k\sigma}, \hat{P}^L_{kl,\sigma\sigma'}, \hat{P}^{L\dagger}_{kl,\sigma\sigma'}, \hat{Q}^L_{kl}, \hat{Q}'^L_{kl,\sigma\sigma'} \right\} \quad (k,l \in R,\ i \in L)$$

The operators required in right block are

$$\left\{ \hat{1}^R, \hat{H}^R, \hat{R}'^R_{i\sigma}, \hat{R}'^{R\dagger}_{i\sigma}, a_{k\sigma}, a^\dagger_{k\sigma}, \hat{A}_{kl,\sigma\sigma'}, \hat{A}^\dagger_{kl,\sigma\sigma'}, \hat{B}_{kl}, \hat{B}'_{kl,\sigma\sigma'} \right\} \quad (k,l \in R,\ i \in L)$$

The total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$N_{CN} = 1 + 1 + 4K_R + 4K_L + 8K_R^2 + K_R^2 + 4K_R^2 = 13K_R^2 + 4K + 2$$

Blocking

The enlarged left/right block is denoted as $L*/R*$. Make sure that all L operators are to the left of $*$ operators.

$$\hat{R}'^{L*}_{i\sigma} = \hat{R}'^L_{i\sigma} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}'^*_{i\sigma} + \sum_{j\in L} \left(\sum_{kl\in *,\sigma'} v_{ijkl} a^\dagger_{k\sigma'} a_{l\sigma'} \right) a_{j\sigma} + \sum_{j\in *} \left(\sum_{kl\in L,\sigma'} v_{ijkl} a^\dagger_{k\sigma'} a_{l\sigma'} \right) a_{j\sigma}$$
$$+ \sum_{k\in L,\sigma'} a^\dagger_{k\sigma'} \left(\sum_{jl\in *} v_{ijkl} a_{l\sigma'} a_{j\sigma} \right) + \sum_{k\in *,\sigma'} a^\dagger_{k\sigma'} \left(\sum_{jl\in L} v_{ijkl} a_{l\sigma'} a_{j\sigma} \right) - \sum_{l\in L,\sigma'} a_{l\sigma'} \left(\sum_{jk\in *} v_{ijkl} a^\dagger_{k\sigma'} a_{j\sigma} \right) - \sum_{l\in *,\sigma'} a_{l\sigma}$$
$$= \hat{R}'^L_{i\sigma} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}'^*_{i\sigma} + \sum_{j\in L} a_{j\sigma} \left(\sum_{kl\in *,\sigma'} v_{ijkl} a^\dagger_{k\sigma'} a_{l\sigma'} \right) + \sum_{j\in *} \left(\sum_{kl\in L,\sigma'} v_{ijkl} a^\dagger_{k\sigma'} a_{l\sigma'} \right) a_{j\sigma}$$
$$+ \sum_{k\in L,\sigma'} a^\dagger_{k\sigma'} \left(\sum_{jl\in *} v_{ijkl} a_{l\sigma'} a_{j\sigma} \right) + \sum_{k\in *,\sigma'} \left(\sum_{jl\in L} v_{ijkl} a_{l\sigma'} a_{j\sigma} \right) a^\dagger_{k\sigma'} - \sum_{l\in L,\sigma'} a_{l\sigma'} \left(\sum_{jk\in *} v_{ijkl} a^\dagger_{k\sigma'} a_{j\sigma} \right) - \sum_{l\in *,\sigma'} \left(\sum_{j} \right.$$

Now there are two possibilities. In NC partition, in L we have $A, A^\dagger, B, B'$ and in $*$ we have $P, P^\dagger, Q, Q'$. In CN partition, the opposite is true. Therefore, we have

$$\hat{R}'^{L*,NC}_{i\sigma} = \hat{R}'^L_{i\sigma} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}'^*_{i\sigma} + \sum_{j\in L} a_{j\sigma} \hat{Q}^*_{ij} + \sum_{j\in *,kl\in L} v_{ijkl} \hat{B}_{kl} a_{j\sigma}$$
$$+ \sum_{k\in L,\sigma'} a^\dagger_{k\sigma'} \hat{P}^*_{ik,\sigma\sigma'} + \sum_{k\in *,jl\in L,\sigma'} v_{ijkl} \hat{A}^\dagger_{jl,\sigma\sigma'} a^\dagger_{k\sigma'} - \sum_{l\in L,\sigma'} a_{l\sigma'} \hat{Q}'^*_{il,\sigma\sigma'} - \sum_{l\in *,jk\in L,\sigma'} v_{ijkl} \hat{B}'_{kj,\sigma'\sigma} a_{l\sigma'}$$
$$= \hat{R}'^L_{i\sigma} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}'^*_{i\sigma} + \sum_{k\in L,\sigma'} a^\dagger_{k\sigma'} \hat{P}^*_{ik,\sigma\sigma'} + \sum_{j\in L} a_{j\sigma} \hat{Q}^*_{ij} - \sum_{l\in L,\sigma'} a_{l\sigma'} \hat{Q}'^*_{il,\sigma\sigma'}$$
$$+ \sum_{k\in *,jl\in L,\sigma'} v_{ijkl} \hat{A}^\dagger_{jl,\sigma\sigma'} a^\dagger_{k\sigma'} + \sum_{j\in *,kl\in L} v_{ijkl} \hat{B}_{kl} a_{j\sigma} - \sum_{l\in *,jk\in L,\sigma'} v_{ijkl} \hat{B}'_{kj,\sigma'\sigma} a_{l\sigma'}$$

$$\begin{aligned}
\hat{R}'^{L*,CN}_{i\sigma} &= \hat{R}'^{L}_{i\sigma} \otimes \hat{1}^{*} + \hat{1}^{L} \otimes \hat{R}'^{*}_{i\sigma} + \sum_{j\in L,kl\in *} v_{ijkl} a_{j\sigma} \hat{B}_{kl} + \sum_{j\in *} \hat{Q}^{L}_{ij} a_{j\sigma} \\
&+ \sum_{k\in L,jl\in *,\sigma'} v_{ijkl} a^{\dagger}_{k\sigma'} \hat{A}^{\dagger}_{jl,\sigma\sigma'} + \sum_{k\in *,\sigma'} \hat{P}^{L}_{ik,\sigma\sigma'} a^{\dagger}_{k\sigma'} - \sum_{l\in L,jk\in *,\sigma'} v_{ijkl} a_{l\sigma'} \hat{B}'_{kj,\sigma'\sigma} - \sum_{l\in *,\sigma'} \hat{Q}'^{L}_{il,\sigma\sigma'} a_{l\sigma'} \\
&= \hat{R}'^{L}_{i\sigma} \otimes \hat{1}^{*} + \hat{1}^{L} \otimes \hat{R}'^{*}_{i\sigma} + \sum_{k\in L,jl\in *,\sigma'} v_{ijkl} a^{\dagger}_{k\sigma'} \hat{A}^{\dagger}_{jl,\sigma\sigma'} + \sum_{j\in L,kl\in *} v_{ijkl} a_{j\sigma} \hat{B}_{kl} - \sum_{l\in L,jk\in *,\sigma'} v_{ijkl} a_{l\sigma'} \hat{B}'_{kj,\sigma'\sigma} \\
&+ \sum_{k\in *,\sigma'} \hat{P}^{L}_{ik,\sigma\sigma'} a^{\dagger}_{k\sigma'} + \sum_{j\in *} \hat{Q}^{L}_{ij} a_{j\sigma} - \sum_{l\in *,\sigma'} \hat{Q}'^{L}_{il,\sigma\sigma'} a_{l\sigma'}
\end{aligned}$$

Similarly,

$$\begin{aligned}
\hat{R}'^{R*,NC}_{i\sigma} &= \hat{R}'^{*}_{i\sigma} \otimes \hat{1}^{R} + \hat{1}^{*} \otimes \hat{R}'^{R}_{i\sigma} + \sum_{k\in *,\sigma'} a^{\dagger}_{k\sigma'} \hat{P}^{R}_{ik,\sigma\sigma'} + \sum_{j\in *} a_{j\sigma} \hat{Q}^{R}_{ij} - \sum_{l\in *,\sigma'} a_{l\sigma'} \hat{Q}'^{R}_{il,\sigma\sigma'} \\
&+ \sum_{k\in R,jl\in *,\sigma'} v_{ijkl} \hat{A}^{\dagger}_{jl,\sigma\sigma'} a^{\dagger}_{k\sigma'} + \sum_{j\in R,kl\in *} v_{ijkl} \hat{B}_{kl} a_{j\sigma} - \sum_{l\in R,jk\in *,\sigma'} v_{ijkl} \hat{B}'_{kj,\sigma'\sigma} a_{l\sigma'} \\
\hat{R}'^{R*,CN}_{i\sigma} &= \hat{R}'^{*}_{i\sigma} \otimes \hat{1}^{R} + \hat{1}^{*} \otimes \hat{R}'^{R}_{i\sigma} + \sum_{k\in *,jl\in R,\sigma'} v_{ijkl} a^{\dagger}_{k\sigma'} \hat{A}^{\dagger}_{jl,\sigma\sigma'} + \sum_{j\in *,kl\in R} v_{ijkl} a_{j\sigma} \hat{B}_{kl} - \sum_{l\in *,jk\in R,\sigma'} v_{ijkl} a_{l\sigma'} \hat{B}'_{kj,\sigma'\sigma} \\
&+ \sum_{k\in R,\sigma'} \hat{P}^{*}_{ik,\sigma\sigma'} a^{\dagger}_{k\sigma'} + \sum_{j\in R} \hat{Q}^{*}_{ij} a_{j\sigma} - \sum_{l\in R,\sigma'} \hat{Q}'^{*}_{il,\sigma\sigma'} a_{l\sigma'}
\end{aligned}$$

Number of terms

$$\begin{aligned}
N_{R',NC} &= (2 + 5K_L + 5K_L^2)K_R + (2 + 5 + 5K_R)K_L = 5K_L^2 K_R + 10K_L K_R + 2K + 5K_L \\
N_{R',CN} &= (2 + 5K_L + 5)K_R + (2 + 5K_R^2 + 5K_R)K_L = 5K_R^2 K_L + 10K_R K_L + 2K + 5K_R
\end{aligned}$$

Blocking of other complementary operators is straightforward

$$\begin{aligned}
\hat{P}^{L*,CN}_{ik,\sigma\sigma'} &= \hat{P}^{L}_{ik,\sigma\sigma'} \otimes \hat{1}^{*} + \hat{1}^{L} \otimes \hat{P}^{*}_{ik,\sigma\sigma'} + \sum_{j\in L,l\in *} v_{ijkl} a_{l\sigma'} a_{j\sigma} + \sum_{j\in *,l\in L} v_{ijkl} a_{l\sigma'} a_{j\sigma} \\
&= \hat{P}^{L}_{ik,\sigma\sigma'} \otimes \hat{1}^{*} + \hat{1}^{L} \otimes \hat{P}^{*}_{ik,\sigma\sigma'} - \sum_{j\in L,l\in *} v_{ijkl} a_{j\sigma} a_{l\sigma'} + \sum_{j\in *,l\in L} v_{ijkl} a_{l\sigma'} a_{j\sigma} \\
\hat{P}^{R*,NC}_{ik,\sigma\sigma'} &= \hat{P}^{*}_{ik,\sigma\sigma'} \otimes \hat{1}^{R} + \hat{1}^{*} \otimes \hat{P}^{R}_{ik,\sigma\sigma'} + \sum_{j\in *,l\in R} v_{ijkl} a_{l\sigma'} a_{j\sigma} + \sum_{j\in R,l\in *} v_{ijkl} a_{l\sigma'} a_{j\sigma} \\
&= \hat{P}^{*}_{ik,\sigma\sigma'} \otimes \hat{1}^{R} + \hat{1}^{*} \otimes \hat{P}^{R}_{ik,\sigma\sigma'} - \sum_{j\in *,l\in R} v_{ijkl} a_{j\sigma} a_{l\sigma'} + \sum_{j\in R,l\in *} v_{ijkl} a_{l\sigma'} a_{j\sigma}
\end{aligned}$$

and

$$\begin{aligned}
\hat{Q}^{L*,CN}_{ij} &= \hat{Q}^{L}_{ij} \otimes \hat{1}^{*} + \hat{1}^{L} \otimes \hat{Q}^{*}_{ij} + \sum_{k\in L,l\in *,\sigma'} v_{ijkl} a^{\dagger}_{k\sigma'} a_{l\sigma'} + \sum_{k\in *,l\in L,\sigma'} v_{ijkl} a^{\dagger}_{k\sigma'} a_{l\sigma'} \\
&= \hat{Q}^{L}_{ij} \otimes \hat{1}^{*} + \hat{1}^{L} \otimes \hat{Q}^{*}_{ij} + \sum_{k\in L,l\in *,\sigma'} v_{ijkl} a^{\dagger}_{k\sigma'} a_{l\sigma'} - \sum_{k\in *,l\in L,\sigma'} v_{ijkl} a_{l\sigma'} a^{\dagger}_{k\sigma'} \\
\hat{Q}^{R*,NC}_{ij} &= \hat{Q}^{*}_{ij} \otimes \hat{1}^{R} + \hat{1}^{*} \otimes \hat{Q}^{R}_{ij} + \sum_{k\in *,l\in R,\sigma'} v_{ijkl} a^{\dagger}_{k\sigma'} a_{l\sigma'} + \sum_{k\in R,l\in *,\sigma'} v_{ijkl} a^{\dagger}_{k\sigma'} a_{l\sigma'} \\
&= \hat{Q}^{*}_{ij} \otimes \hat{1}^{R} + \hat{1}^{*} \otimes \hat{Q}^{R}_{ij} + \sum_{k\in *,l\in R,\sigma'} v_{ijkl} a^{\dagger}_{k\sigma'} a_{l\sigma'} - \sum_{k\in R,l\in *,\sigma'} v_{ijkl} a_{l\sigma'} a^{\dagger}_{k\sigma'}
\end{aligned}$$

and

$$\begin{aligned}
\hat{Q}'^{L*,CN}_{il,\sigma\sigma'} &= \hat{Q}'^{L}_{il,\sigma\sigma'} \otimes \hat{1}^{*} + \hat{1}^{L} \otimes \hat{Q}'^{*}_{il,\sigma\sigma'} + \sum_{j\in L,k\in *} v_{ijkl} a^{\dagger}_{k\sigma'} a_{j\sigma} + \sum_{j\in *,k\in L} v_{ijkl} a^{\dagger}_{k\sigma'} a_{j\sigma} \\
&= \hat{Q}'^{L}_{il,\sigma\sigma'} \otimes \hat{1}^{*} + \hat{1}^{L} \otimes \hat{Q}'^{*}_{il,\sigma\sigma'} - \sum_{j\in L,k\in *} v_{ijkl} a_{j\sigma} a^{\dagger}_{k\sigma'} + \sum_{j\in *,k\in L} v_{ijkl} a^{\dagger}_{k\sigma'} a_{j\sigma} \\
\hat{Q}'^{R*,NC}_{il,\sigma\sigma'} &= \hat{Q}'^{*}_{il,\sigma\sigma'} \otimes \hat{1}^{R} + \hat{1}^{*} \otimes \hat{Q}'^{R}_{il,\sigma\sigma'} + \sum_{j\in *,k\in R} v_{ijkl} a^{\dagger}_{k\sigma'} a_{j\sigma} + \sum_{j\in R,k\in *} v_{ijkl} a^{\dagger}_{k\sigma'} a_{j\sigma} \\
&= \hat{Q}'^{*}_{il,\sigma\sigma'} \otimes \hat{1}^{R} + \hat{1}^{*} \otimes \hat{Q}'^{R}_{il,\sigma\sigma'} - \sum_{j\in *,k\in R} v_{ijkl} a_{j\sigma} a^{\dagger}_{k\sigma'} + \sum_{j\in R,k\in *} v_{ijkl} a^{\dagger}_{k\sigma'} a_{j\sigma}
\end{aligned}$$

Middle-Site Transformation

When the sweep is performed from left to right, passing the middle site, we need to switch from NC partition to CN partition. The cost is $O(K^4/16)$. This happens only once in the sweep. The cost of one blocking procedure is $O(K_<^2 K_>)$, but there are K blocking steps in one sweep. So the cost for blocking in one sweep is $O(K K_<^2 K_>)$. Note that the most expensive part in the program should be the Hamiltonian step in Davidson, which scales as $O(K_<^2)$.

$$\begin{aligned}
\hat{P}^{L,NC\to CN}_{ik,\sigma\sigma'} &= \sum_{jl\in L} v_{ijkl} a_{l\sigma'} a_{j\sigma} = \sum_{jl\in L} v_{ijkl} \hat{A}^{\dagger}_{jl,\sigma\sigma'} \\
\hat{Q}^{L,NC\to CN}_{ij} &= \sum_{kl\in L,\sigma'} v_{ijkl} a^{\dagger}_{k\sigma'} a_{l\sigma'} = \sum_{kl\in L} v_{ijkl} \hat{B}_{kl} \\
\hat{Q}'^{L,NC\to CN}_{il,\sigma\sigma'} &= \sum_{jk\in L} v_{ijkl} a^{\dagger}_{k\sigma'} a_{j\sigma} = \sum_{jk\in L} v_{ijkl} \hat{B}'_{kj,\sigma'\sigma}
\end{aligned}$$

7.1.2 Spin-Adapted DMRG Quantum Chemistry Hamiltonian

Partitioning in SU(2)

The partitioning of Hamiltonian in left (L) and right (R) blocks is given by

$$\begin{aligned}
(\hat{H})^{[0]} = &\left(\hat{H}^{L}\right)^{[0]} \otimes_{[0]} \left(\hat{1}^{R}\right)^{[0]} + \left(\hat{1}^{L}\right)^{[0]} \otimes_{[0]} \left(\hat{H}^{R}\right)^{[0]} \\
&+ \sqrt{2} \sum_{i\in L} \left[\left(a^{\dagger}_{i}\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(\hat{S}^{R}_{i}\right)^{[\frac{1}{2}]} + h.c.\right] \\
&+ 2 \sum_{i\in L} \left[\left(a^{\dagger}_{i}\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(\hat{R}^{R}_{i}\right)^{[\frac{1}{2}]} + h.c.\right] + 2 \sum_{i\in R} \left[\left(a^{\dagger}_{i}\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(\hat{R}^{L}_{i}\right)^{[\frac{1}{2}]} + h.c.\right] \\
&- \frac{1}{2} \sum_{ik\in L} \left[\sqrt{3}\left(\hat{A}_{ik}\right)^{[1]} \otimes_{[0]} \left(\hat{P}^{R}_{ik}\right)^{[1]} + \left(\hat{A}_{ik}\right)^{[0]} \otimes_{[0]} \left(\hat{P}^{R}_{ik}\right)^{[0]} + h.c.\right] \\
&+ \sum_{ij\in L} \left[\left(\hat{B}_{ij}\right)^{[0]} \otimes_{[0]} \left(2\left(\hat{Q}^{R}_{ij}\right)^{[0]} - \left(\hat{Q}'^{R}_{ij}\right)^{[0]}\right) + \sqrt{3}\left(\hat{B}'_{ij}\right)^{[1]} \otimes_{[0]} \left(\hat{Q}'^{R}_{ij}\right)^{[1]}\right]
\end{aligned}$$

where the normal and complementary operators are defined by

$$
\begin{aligned}
\big(\hat{S}_i^{L/R}\big)^{[\frac{1}{2}]} &= \sum_{j\in L/R} t_{ij} \big(a_j\big)^{[\frac{1}{2}]} \\
\big(\hat{R}_i^{L/R}\big)^{[\frac{1}{2}]} &= \sum_{jkl\in L/R} v_{ijkl} \left[\left(a_k^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(a_l\right)^{[\frac{1}{2}]}\right] \otimes_{[\frac{1}{2}]} \left(a_j\right)^{[\frac{1}{2}]} \\
\big(\hat{A}_{ik}\big)^{[0/1]} &= \big(a_i^\dagger\big)^{[\frac{1}{2}]} \otimes_{[0/1]} \big(a_k^\dagger\big)^{[\frac{1}{2}]} \\
\big(\hat{P}_{ik}^R\big)^{[0/1]} &= \sum_{jl\in R} v_{ijkl} \big(a_j\big)^{[\frac{1}{2}]} \otimes_{[0/1]} \big(a_l\big)^{[\frac{1}{2}]} \\
\big(\hat{B}_{ij}\big)^{[0]} &= \big(a_i^\dagger\big)^{[\frac{1}{2}]} \otimes_{[0]} \big(a_j\big)^{[\frac{1}{2}]} \\
\big(\hat{B}'_{ij}\big)^{[1]} &= \big(a_i^\dagger\big)^{[\frac{1}{2}]} \otimes_{[1]} \big(a_j\big)^{[\frac{1}{2}]} \\
\big(\hat{Q}_{ij}^R\big)^{[0]} &= \sum_{kl\in R} v_{ijkl} \big(a_k^\dagger\big)^{[\frac{1}{2}]} \otimes_{[0]} \big(a_l\big)^{[\frac{1}{2}]} \\
\big(\hat{Q}'^R_{ij}\big)^{[0/1]} &= \sum_{kl\in R} v_{ilkj} \big(a_k^\dagger\big)^{[\frac{1}{2}]} \otimes_{[0/1]} \big(a_l\big)^{[\frac{1}{2}]} \\
\big(\hat{Q}''^R_{ij}\big)^{[0]} &:= 2\big(\hat{Q}_{ij}^R\big)^{[0]} - \big(\hat{Q}'^R_{ij}\big)^{[0]} = \sum_{kl\in R} (2v_{ijkl} - v_{ilkj}) \big(a_k^\dagger\big)^{[\frac{1}{2}]} \otimes_{[0]} \big(a_l\big)^{[\frac{1}{2}]}
\end{aligned}
$$

Derivation

CG Factors

From $j_2 = 1/2$ CG factors

$$
\left\langle j_1 \ \left(M - \frac{1}{2}\right) \ \frac{1}{2} \ \frac{1}{2} \middle| \left(j_1 \pm \frac{1}{2}\right) \ M \right\rangle = \pm \sqrt{\frac{1}{2}\left(1 \pm \frac{M}{j_1 + \frac{1}{2}}\right)}
$$

$$
\left\langle j_1 \ \left(M + \frac{1}{2}\right) \ \frac{1}{2} \ \left(-\frac{1}{2}\right) \middle| \left(j_1 \pm \frac{1}{2}\right) \ M \right\rangle = \sqrt{\frac{1}{2}\left(1 \mp \frac{M}{j_1 + \frac{1}{2}}\right)}
$$

and symmetry relation

$$
\langle j_1 \ m_1 \ j_2 \ m_2 | J \ M \rangle = (-1)^{j_1 + j_2 - J} \langle j_2 \ m_2 \ j_1 \ m_1 | J \ M \rangle
$$

and

$$
(-1)^{j_1 + \frac{1}{2} - j_1 \mp \frac{1}{2}} = (-1)^{\frac{1}{2} \mp \frac{1}{2}} = \pm 1
$$

we have

$$
\left\langle \frac{1}{2} \ \frac{1}{2} \ j_1 \ \left(M - \frac{1}{2}\right) \middle| \left(j_1 \pm \frac{1}{2}\right) \ M \right\rangle = \sqrt{\frac{1}{2}\left(1 \pm \frac{M}{j_1 + \frac{1}{2}}\right)}
$$

$$
\left\langle \frac{1}{2} \ \left(-\frac{1}{2}\right) \ j_1 \ \left(M + \frac{1}{2}\right) \middle| \left(j_1 \pm \frac{1}{2}\right) \ M \right\rangle = \pm \sqrt{\frac{1}{2}\left(1 \mp \frac{M}{j_1 + \frac{1}{2}}\right)}
$$

let $j_1 = 1$, we have

$$\langle \tfrac{1}{2}\ \tfrac{1}{2}\ 1\ (M-\tfrac{1}{2})|\tfrac{1}{2}\ M\rangle = \sqrt{\tfrac{1}{2}(1-\frac{M}{\frac{3}{2}})}$$

$$\langle \tfrac{1}{2}\ (-\tfrac{1}{2})\ 1\ (M+\tfrac{1}{2})|\tfrac{1}{2}\ M\rangle = -\sqrt{\tfrac{1}{2}(1+\frac{M}{\frac{3}{2}})}$$

So the coefficients for $[\frac{1}{2}] \otimes_{[\frac{1}{2}]} [1]$ are

$$[\tfrac{1}{2}+0=\tfrac{1}{2}] = \sqrt{\tfrac{1}{3}}, \quad [-\tfrac{1}{2}+1=\tfrac{1}{2}] = -\sqrt{\tfrac{2}{3}}$$

$$[\tfrac{1}{2}+(-1)=-\tfrac{1}{2}] = \sqrt{\tfrac{2}{3}}, \quad [-\tfrac{1}{2}+0=-\tfrac{1}{2}] = -\sqrt{\tfrac{1}{3}}$$

The coefficients for $[1] \otimes_{[\frac{1}{2}]} [\frac{1}{2}]$ are

$$[0+\tfrac{1}{2}=\tfrac{1}{2}] = -\sqrt{\tfrac{1}{3}}, \quad [1-\tfrac{1}{2}=\tfrac{1}{2}] = \sqrt{\tfrac{2}{3}}$$

$$[(-1)+\tfrac{1}{2}=-\tfrac{1}{2}] = -\sqrt{\tfrac{2}{3}}, \quad [0-\tfrac{1}{2}=-\tfrac{1}{2}] = \sqrt{\tfrac{1}{3}}$$

This means that the SU(2) operator exchange factor for $[\frac{1}{2}] \otimes_{[\frac{1}{2}]} [1] \to [1] \otimes_{[\frac{1}{2}]} [\frac{1}{2}]$ is -1. The fermion factor is $+1$. So the overall exchange factor for this case is -1.

Tensor Product Formulas

Singlet

$$\left(a_p^\dagger\right)^{[1/2]} \otimes_{[0]} \left(a_q^\dagger\right)^{[1/2]} = \begin{pmatrix} a_{p\alpha}^\dagger \\ a_{p\beta}^\dagger \end{pmatrix}^{[1/2]} \otimes_{[0]} \begin{pmatrix} a_{q\alpha}^\dagger \\ a_{q\beta}^\dagger \end{pmatrix}^{[1/2]} = \frac{1}{\sqrt{2}} \left(a_{p\alpha}^\dagger a_{q\beta}^\dagger - a_{p\beta}^\dagger a_{q\alpha}^\dagger\right)^{[0]}$$

$$\left(a_p^\dagger\right)^{[1/2]} \otimes_{[0]} \left(a_q\right)^{[1/2]} = \begin{pmatrix} a_{p\alpha}^\dagger \\ a_{p\beta}^\dagger \end{pmatrix}^{[1/2]} \otimes_{[0]} \begin{pmatrix} -a_{q\beta} \\ a_{q\alpha} \end{pmatrix}^{[1/2]} = \frac{1}{\sqrt{2}} \left(a_{p\alpha}^\dagger a_{q\alpha} + a_{p\beta}^\dagger a_{q\beta}\right)^{[0]}$$

$$\left(a_p\right)^{[1/2]} \otimes_{[0]} \left(a_q\right)^{[1/2]} = \begin{pmatrix} -a_{p\beta} \\ a_{p\alpha} \end{pmatrix}^{[1/2]} \otimes_{[0]} \begin{pmatrix} -a_{q\beta} \\ a_{q\alpha} \end{pmatrix}^{[1/2]} = \frac{1}{\sqrt{2}} \left(-a_{p\beta} a_{q\alpha} + a_{p\alpha} a_{q\beta}\right)^{[0]}$$

Triplet

$$\left(a_p^\dagger\right)^{[1/2]} \otimes_{[1]} \left(a_q^\dagger\right)^{[1/2]} = \begin{pmatrix} a_{p\alpha}^\dagger \\ a_{p\beta}^\dagger \end{pmatrix}^{[1/2]} \otimes_{[1]} \begin{pmatrix} a_{q\alpha}^\dagger \\ a_{q\beta}^\dagger \end{pmatrix}^{[1/2]} = \begin{pmatrix} a_{p\alpha}^\dagger a_{q\alpha}^\dagger \\ \frac{1}{\sqrt{2}}\left(a_{p\alpha}^\dagger a_{q\beta}^\dagger + a_{p\beta}^\dagger a_{q\alpha}^\dagger\right) \\ a_{p\beta}^\dagger a_{q\beta}^\dagger \end{pmatrix}^{[1]}$$

$$\left(a_p^\dagger\right)^{[1/2]} \otimes_{[1]} \left(a_q\right)^{[1/2]} = \begin{pmatrix} a_{p\alpha}^\dagger \\ a_{p\beta}^\dagger \end{pmatrix}^{[1/2]} \otimes_{[1]} \begin{pmatrix} -a_{q\beta} \\ a_{q\alpha} \end{pmatrix}^{[1/2]} = \begin{pmatrix} -a_{p\alpha}^\dagger a_{q\beta} \\ \frac{1}{\sqrt{2}}\left(a_{p\alpha}^\dagger a_{q\alpha} - a_{p\beta}^\dagger a_{q\beta}\right) \\ a_{p\beta}^\dagger a_{q\alpha} \end{pmatrix}^{[1]}$$

$$\left(a_p\right)^{[1/2]} \otimes_{[1]} \left(a_q\right)^{[1/2]} = \begin{pmatrix} -a_{p\beta} \\ a_{p\alpha} \end{pmatrix}^{[1/2]} \otimes_{[1]} \begin{pmatrix} -a_{q\beta} \\ a_{q\alpha} \end{pmatrix}^{[1/2]} = \begin{pmatrix} a_{p\beta} a_{q\beta} \\ -\frac{1}{\sqrt{2}}\left(a_{p\beta} a_{q\alpha} + a_{p\alpha} a_{q\beta}\right) \\ a_{p\alpha} a_{q\alpha} \end{pmatrix}^{[1]}$$

Doublet times singlet/triplet

$$U^{[1/2]} = \left(a_p^\dagger\right)^{[1/2]} \otimes_{[1/2]} \left[\left(a_r\right)^{[1/2]} \otimes_{[1]} \left(a_s\right)^{[1/2]}\right] = \begin{pmatrix} a_{p\alpha}^\dagger \\ a_{p\beta}^\dagger \end{pmatrix}^{[1/2]} \otimes_{[1/2]} \begin{pmatrix} a_{r\beta}a_{s\beta} \\ -\frac{1}{\sqrt{2}}\left(a_{r\beta}a_{s\alpha} + a_{r\alpha}a_{s\beta}\right) \\ a_{r\alpha}a_{s\alpha} \end{pmatrix}^{[1]}$$

$$= \begin{pmatrix} -\frac{1}{\sqrt{2}}\frac{1}{\sqrt{3}}a_{p\alpha}^\dagger\left(a_{r\beta}a_{s\alpha} + a_{r\alpha}a_{s\beta}\right) - \frac{\sqrt{2}}{\sqrt{3}}a_{p\beta}^\dagger a_{r\beta}a_{s\beta} \\ \frac{\sqrt{2}}{\sqrt{3}}a_{p\alpha}^\dagger a_{r\alpha}a_{s\alpha} + \left(-\frac{1}{\sqrt{3}}\right)\left(-\frac{1}{\sqrt{2}}\right)a_{p\beta}^\dagger\left(a_{r\beta}a_{s\alpha} + a_{r\alpha}a_{s\beta}\right) \end{pmatrix}^{[1/2]} = \frac{1}{\sqrt{6}}\begin{pmatrix} -a_{p\alpha}^\dagger a_{r\beta}a_{s\alpha} - a_{p\alpha}^\dagger a_{r\alpha}a_{s\beta} - 2a_{p\beta}^\dagger a_{r\beta} \\ 2a_{p\alpha}^\dagger a_{r\alpha}a_{s\alpha} + a_{p\beta}^\dagger a_{r\beta}a_{s\alpha} + a_{p\beta}^\dagger a_{r\alpha}a \end{pmatrix}$$

$$V^{[1/2]} = \left(a_p^\dagger\right)^{[1/2]} \otimes_{[1/2]} \left[\left(a_r\right)^{[1/2]} \otimes_{[0]} \left(a_s\right)^{[1/2]}\right] = \frac{1}{\sqrt{2}}\begin{pmatrix} a_{p\alpha}^\dagger \\ a_{p\beta}^\dagger \end{pmatrix}^{[1/2]} \otimes_{[1/2]} \left(-a_{r\beta}a_{s\alpha} + a_{r\alpha}a_{s\beta}\right)^{[0]}$$

$$= \frac{1}{\sqrt{2}}\begin{pmatrix} -a_{p\alpha}^\dagger a_{r\beta}a_{s\alpha} + a_{p\alpha}^\dagger a_{r\alpha}a_{s\beta} \\ -a_{p\beta}^\dagger a_{r\beta}a_{s\alpha} + a_{p\beta}^\dagger a_{r\alpha}a_{s\beta} \end{pmatrix}^{[1/2]}$$

Therefore,

$$\sqrt{3}U^{[1/2]} - V^{[1/2]} = \frac{1}{\sqrt{2}}\begin{pmatrix} -a_{p\alpha}^\dagger a_{r\beta}a_{s\alpha} - a_{p\alpha}^\dagger a_{r\alpha}a_{s\beta} - 2a_{p\beta}^\dagger a_{r\beta}a_{s\beta} \\ 2a_{p\alpha}^\dagger a_{r\alpha}a_{s\alpha} + a_{p\beta}^\dagger a_{r\beta}a_{s\alpha} + a_{p\beta}^\dagger a_{r\alpha}a_{s\beta} \end{pmatrix}^{[1/2]} - \frac{1}{\sqrt{2}}\begin{pmatrix} -a_{p\alpha}^\dagger a_{r\beta}a_{s\alpha} + a_{p\alpha}^\dagger a_{r\alpha}a_{s\beta} \\ -a_{p\beta}^\dagger a_{r\beta}a_{s\alpha} + a_{p\beta}^\dagger a_{r\alpha}a_{s\beta} \end{pmatrix}^{[1/2]}$$

$$= \frac{1}{\sqrt{2}}\begin{pmatrix} -a_{p\alpha}^\dagger a_{r\beta}a_{s\alpha} - a_{p\alpha}^\dagger a_{r\alpha}a_{s\beta} - 2a_{p\beta}^\dagger a_{r\beta}a_{s\beta} + a_{p\alpha}^\dagger a_{r\beta}a_{s\alpha} - a_{p\alpha}^\dagger a_{r\alpha}a_{s\beta} \\ 2a_{p\alpha}^\dagger a_{r\alpha}a_{s\alpha} + a_{p\beta}^\dagger a_{r\beta}a_{s\alpha} + a_{p\beta}^\dagger a_{r\alpha}a_{s\beta} + a_{p\beta}^\dagger a_{r\beta}a_{s\alpha} - a_{p\beta}^\dagger a_{r\alpha}a_{s\beta} \end{pmatrix}^{[1/2]}$$

$$= \sqrt{2}\begin{pmatrix} -a_{p\alpha}^\dagger a_{r\alpha}a_{s\beta} - a_{p\beta}^\dagger a_{r\beta}a_{s\beta} \\ a_{p\alpha}^\dagger a_{r\alpha}a_{s\alpha} + a_{p\beta}^\dagger a_{r\beta}a_{s\alpha} \end{pmatrix}^{[1/2]}$$

Another case

$$S^{[1/2]} = \left(a_r\right)^{[1/2]} \otimes_{[1/2]} \left[\left(a_p^\dagger\right)^{[1/2]} \otimes_{[1]} \left(a_q\right)^{[1/2]}\right] = \begin{pmatrix} -a_{r\beta} \\ a_{r\alpha} \end{pmatrix}^{[1/2]} \otimes_{[1/2]} \begin{pmatrix} -a_{p\alpha}^\dagger a_{q\beta} \\ \frac{1}{\sqrt{2}}\left(a_{p\alpha}^\dagger a_{q\alpha} - a_{p\beta}^\dagger a_{q\beta}\right) \\ a_{p\beta}^\dagger a_{q\alpha} \end{pmatrix}^{[1]}$$

$$= \begin{pmatrix} \frac{1}{\sqrt{2}}\frac{1}{\sqrt{3}}(-a_{r\beta})\left(a_{p\alpha}^\dagger a_{q\alpha} - a_{p\beta}^\dagger a_{q\beta}\right) + \frac{\sqrt{2}}{\sqrt{3}}a_{r\alpha}a_{p\alpha}^\dagger a_{q\beta} \\ -\frac{\sqrt{2}}{\sqrt{3}}a_{r\beta}a_{p\beta}^\dagger a_{q\alpha} - \frac{1}{\sqrt{2}}\frac{1}{\sqrt{3}}a_{r\alpha}\left(a_{p\alpha}^\dagger a_{q\alpha} - a_{p\beta}^\dagger a_{q\beta}\right) \end{pmatrix}^{[1/2]} = \frac{1}{\sqrt{6}}\begin{pmatrix} -a_{r\beta}a_{p\alpha}^\dagger a_{q\alpha} + a_{r\beta}a_{p\beta}^\dagger a_{q\beta} + 2a_{r\alpha}a_{p\alpha}^\dagger a_{q\beta} \\ -2a_{r\beta}a_{p\beta}^\dagger a_{q\alpha} - a_{r\alpha}a_{p\alpha}^\dagger a_{q\alpha} + a_{r\alpha}a_{p\beta}^\dagger a_{q\beta} \end{pmatrix}^{[1}$$

$$T^{[1/2]} = \left(a_r\right)^{[1/2]} \otimes_{[1/2]} \left[\left(a_p^\dagger\right)^{[1/2]} \otimes_{[0]} \left(a_q\right)^{[1/2]}\right] = \frac{1}{\sqrt{2}}\begin{pmatrix} -a_{r\beta} \\ a_{r\alpha} \end{pmatrix}^{[1/2]} \otimes_{[1/2]} \left(a_{p\alpha}^\dagger a_{q\alpha} + a_{p\beta}^\dagger a_{q\beta}\right)^{[0]}$$

$$= \frac{1}{\sqrt{2}}\begin{pmatrix} -a_{r\beta}a_{p\alpha}^\dagger a_{q\alpha} - a_{r\beta}a_{p\beta}^\dagger a_{q\beta} \\ a_{r\alpha}a_{p\alpha}^\dagger a_{q\alpha} + a_{r\alpha}a_{p\beta}^\dagger a_{q\beta} \end{pmatrix}^{[1/2]}$$

Therefore,

$$\begin{aligned}
\sqrt{3}S^{[1/2]} - T^{[1/2]} &= \frac{1}{\sqrt{6}} \begin{pmatrix} -a_{r\beta}a_{p\alpha}^\dagger a_{q\alpha} + a_{r\beta}a_{p\beta}^\dagger a_{q\beta} + 2a_{r\alpha}a_{p\alpha}^\dagger a_{q\beta} \\ -2a_{r\beta}a_{p\beta}^\dagger a_{q\alpha} - a_{r\alpha}a_{p\alpha}^\dagger a_{q\alpha} + a_{r\alpha}a_{p\beta}^\dagger a_{q\beta} \end{pmatrix}^{[1/2]} - \frac{1}{\sqrt{2}} \begin{pmatrix} -a_{r\beta}a_{p\alpha}^\dagger a_{q\alpha} - a_{r\beta}a_{p\beta}^\dagger a_{q\beta} \\ a_{r\alpha}a_{p\alpha}^\dagger a_{q\alpha} + a_{r\alpha}a_{p\beta}^\dagger a_{q\beta} \end{pmatrix}^{[1/2]} \\
&= \frac{1}{\sqrt{2}} \begin{pmatrix} -a_{r\beta}a_{p\alpha}^\dagger a_{q\alpha} + a_{r\beta}a_{p\beta}^\dagger a_{q\beta} + 2a_{r\alpha}a_{p\alpha}^\dagger a_{q\beta} + a_{r\beta}a_{p\alpha}^\dagger a_{q\alpha} + a_{r\beta}a_{p\beta}^\dagger a_{q\beta} \\ -2a_{r\beta}a_{p\beta}^\dagger a_{q\alpha} - a_{r\alpha}a_{p\alpha}^\dagger a_{q\alpha} + a_{r\alpha}a_{p\beta}^\dagger a_{q\beta} - a_{r\alpha}a_{p\alpha}^\dagger a_{q\alpha} - a_{r\alpha}a_{p\beta}^\dagger a_{q\beta} \end{pmatrix}^{[1/2]} \\
&= \sqrt{2} \begin{pmatrix} a_{r\beta}a_{p\beta}^\dagger a_{q\beta} + a_{r\alpha}a_{p\alpha}^\dagger a_{q\beta} \\ -a_{r\beta}a_{p\beta}^\dagger a_{q\alpha} - a_{r\alpha}a_{p\alpha}^\dagger a_{q\alpha} \end{pmatrix}^{[1/2]}
\end{aligned}$$

Triplet times triplet

$$\begin{aligned}
X^{[0]} &= \left[\left(a_p^\dagger\right)^{[1/2]} \otimes_{[1]} \left(a_q^\dagger\right)^{[1/2]} \right] \otimes_{[0]} \left[\left(a_r\right)^{[1/2]} \otimes_{[1]} \left(a_s\right)^{[1/2]} \right] \\
&= \begin{pmatrix} a_{p\alpha}^\dagger a_{q\alpha}^\dagger \\ \frac{1}{\sqrt{2}}\left(a_{p\alpha}^\dagger a_{q\beta}^\dagger + a_{p\beta}^\dagger a_{q\alpha}^\dagger\right) \\ a_{p\beta}^\dagger a_{q\beta}^\dagger \end{pmatrix}^{[1]} \otimes_{[0]} \begin{pmatrix} a_{r\beta}a_{s\beta} \\ -\frac{1}{\sqrt{2}}\left(a_{r\beta}a_{s\alpha} + a_{r\alpha}a_{s\beta}\right) \\ a_{r\alpha}a_{s\alpha} \end{pmatrix}^{[1]} \\
&= \frac{1}{\sqrt{3}} \left(a_{p\alpha}^\dagger a_{q\alpha}^\dagger a_{r\alpha} s_{s\alpha} + \tfrac{1}{2}\left(a_{p\alpha}^\dagger a_{q\beta}^\dagger + a_{p\beta}^\dagger a_{q\alpha}^\dagger\right)\left(a_{r\beta}a_{s\alpha} + a_{r\alpha}a_{s\beta}\right) + a_{p\beta}^\dagger a_{q\beta}^\dagger a_{r\beta}a_{s\beta} \right) \\
Y^{[0]} &= \left[\left(a_p^\dagger\right)^{[1/2]} \otimes_{[0]} \left(a_q^\dagger\right)^{[1/2]} \right] \otimes_{[0]} \left[\left(a_r\right)^{[1/2]} \otimes_{[0]} \left(a_s\right)^{[1/2]} \right] \\
&= \frac{1}{\sqrt{2}} \left(a_{p\alpha}^\dagger a_{q\beta}^\dagger - a_{p\beta}^\dagger a_{q\alpha}^\dagger \right)^{[0]} \otimes_{[0]} \frac{1}{\sqrt{2}} \left(-a_{r\beta}a_{s\alpha} + a_{r\alpha}a_{s\beta} \right)^{[0]} \\
&= \frac{1}{2}\left(a_{p\alpha}^\dagger a_{q\beta}^\dagger - a_{p\beta}^\dagger a_{q\alpha}^\dagger\right)\left(- a_{r\beta}a_{s\alpha} + a_{r\alpha}a_{s\beta}\right)
\end{aligned}$$

Using

$$(a+b)(c+d) + (a-b)(-c+d) = (a+b)(2d) - 2b(-c+d) = 2(ad+bc)$$

we have

$$\begin{aligned}
\sqrt{3}X^{[0]} + Y^{[0]} &= a_{p\alpha}^\dagger a_{q\alpha}^\dagger a_{r\alpha} s_{s\alpha} + a_{p\beta}^\dagger a_{q\beta}^\dagger a_{r\beta}a_{s\beta} + a_{p\alpha}^\dagger a_{q\beta}^\dagger a_{r\alpha}a_{s\beta} + a_{p\beta}^\dagger a_{q\alpha}^\dagger a_{r\beta}a_{s\alpha} \\
&= \sum_{\sigma\sigma'} a_{p\sigma}^\dagger a_{q\sigma'}^\dagger a_{r\sigma} s_{s\sigma'}
\end{aligned}$$

Another case

$$\begin{aligned}
Z^{[0]} &= \left[\left(a_p^\dagger\right)^{[1/2]} \otimes_{[1]} \left(a_q\right)^{[1/2]} \right] \otimes_{[0]} \left[\left(a_r^\dagger\right)^{[1/2]} \otimes_{[1]} \left(a_s\right)^{[1/2]} \right] \\
&= \begin{pmatrix} -a_{p\alpha}^\dagger a_{q\beta} \\ \frac{1}{\sqrt{2}}\left(a_{p\alpha}^\dagger a_{q\alpha} - a_{p\beta}^\dagger a_{q\beta}\right) \\ a_{p\beta}^\dagger a_{q\alpha} \end{pmatrix}^{[1]} \otimes_{[0]} \begin{pmatrix} -a_{r\alpha}^\dagger a_{s\beta} \\ \frac{1}{\sqrt{2}}\left(a_{r\alpha}^\dagger a_{s\alpha} - a_{r\beta}^\dagger a_{s\beta}\right) \\ a_{r\beta}^\dagger a_{s\alpha} \end{pmatrix}^{[1]} \\
&= \frac{1}{\sqrt{3}} \left(-a_{p\alpha}^\dagger a_{q\beta} a_{r\beta}^\dagger a_{s\alpha} - \tfrac{1}{2}\left(a_{p\alpha}^\dagger a_{q\alpha} - a_{p\beta}^\dagger a_{q\beta}\right)\left(a_{r\alpha}^\dagger a_{s\alpha} - a_{r\beta}^\dagger a_{s\beta}\right) - a_{p\beta}^\dagger a_{q\alpha} a_{r\alpha}^\dagger a_{s\beta} \right) \\
W^{[0]} &= \left[\left(a_p^\dagger\right)^{[1/2]} \otimes_{[0]} \left(a_q\right)^{[1/2]} \right] \otimes_{[0]} \left[\left(a_r^\dagger\right)^{[1/2]} \otimes_{[0]} \left(a_s\right)^{[1/2]} \right] \\
&= \frac{1}{\sqrt{2}} \left(a_{p\alpha}^\dagger a_{q\alpha} + a_{p\beta}^\dagger a_{q\beta} \right)^{[0]} \otimes_{[0]} \frac{1}{\sqrt{2}} \left(a_{r\alpha}^\dagger a_{s\alpha} + a_{r\beta}^\dagger a_{s\beta} \right)^{[0]} \\
&= \frac{1}{2}\left(a_{p\alpha}^\dagger a_{q\alpha} + a_{p\beta}^\dagger a_{q\beta}\right)\left(a_{r\alpha}^\dagger a_{s\alpha} + a_{r\beta}^\dagger a_{s\beta}\right)
\end{aligned}$$

Using

$$(a-b)(c-d)+(a+b)(c+d)=(a+b)(2c)-(2b)(c-d)=2(ac+bd)$$

we have

$$\begin{aligned}-\sqrt{3}Z^{[0]}+W^{[0]} &= a^\dagger_{p\alpha}a_{q\beta}a^\dagger_{r\beta}a_{s\alpha}+a^\dagger_{p\beta}a_{q\alpha}a^\dagger_{r\alpha}a_{s\beta}+a^\dagger_{p\alpha}a_{q\alpha}a^\dagger_{r\alpha}a_{s\alpha}+a^\dagger_{p\beta}a_{q\beta}a^\dagger_{r\beta}a_{s\beta}\\ &= \sum_{\sigma\sigma'}a^\dagger_{p\sigma}a_{q\sigma'}a^\dagger_{r\sigma'}a_{s\sigma}\end{aligned}$$

S Term

From second singlet formula we have

$$\sqrt{2}\sum_{i\in L}\left(a_i^\dagger\right)^{[\frac{1}{2}]}\otimes_{[0]}\left(\hat{S}_i^R\right)^{[\frac{1}{2}]}=\sum_{i\in L}\left(t_{ij}a^\dagger_{i\alpha}a_{j\alpha}+t_{ij}a^\dagger_{i\beta}a_{j\beta}\right)$$

R Term

This is the same as the S term. Note that in the expression for $\hat{R}$, we have a $\otimes_{[0]}$, this is because in the original spatial expression there is a summation over σ. Then there is a $[0]\otimes_{[1/2]}[1/2]$, which will not produce any extra coefficients.

AP Term

Using definition

$$\begin{aligned}\left(\hat{A}_{ik}\right)^{[0/1]} &= \left(a_i^\dagger\right)^{[\frac{1}{2}]}\otimes_{[0/1]}\left(a_k^\dagger\right)^{[\frac{1}{2}]}\\ \left(\hat{P}^R_{ik}\right)^{[0/1]} &= -\sum_{jl\in R}v_{ijkl}\left(a_j\right)^{[\frac{1}{2}]}\otimes_{[0/1]}\left(a_l\right)^{[\frac{1}{2}]}\end{aligned}$$

We have

$$\begin{aligned}&\sum_{ik\in L}\left[\sqrt{3}\left(\hat{A}_{ik}\right)^{[1]}\otimes_{[0]}\left(\hat{P}^R_{ik}\right)^{[1]}+\left(\hat{A}_{ik}\right)^{[0]}\otimes_{[0]}\left(\hat{P}^R_{ik}\right)^{[0]}\right]\\ =&\sum_{ik\in L,jl\in R}v_{ijkl}\left[\sqrt{3}\left[\left(a_i^\dagger\right)^{[\frac{1}{2}]}\otimes_{[1]}\left(a_k^\dagger\right)^{[\frac{1}{2}]}\right]\otimes_{[0]}\left[\left(a_j\right)^{[\frac{1}{2}]}\otimes_{[1]}\left(a_l\right)^{[\frac{1}{2}]}\right]+\left[\left(a_i^\dagger\right)^{[\frac{1}{2}]}\otimes_{[0]}\left(a_k^\dagger\right)^{[\frac{1}{2}]}\right]\otimes_{[0]}\left[\left(a_j\right)^{[\frac{1}{2}]}\otimes_{[0]}\left(a_l\right)^{[\frac{1}{2}]}\right]\right.\\ =&\sum_{ik\in L,jl\in R}v_{ijkl}\left[\sum_{\sigma\sigma'}a^\dagger_{i\sigma}a^\dagger_{k\sigma'}a_{j\sigma}a_{l\sigma'}\right]=-\sum_{ik\in L,jl\in R,\sigma\sigma'}v_{ijkl}a^\dagger_{i\sigma}a^\dagger_{k\sigma'}a_{l\sigma'}a_{j\sigma}\end{aligned}$$

Note that in last step, we can anticommute $a_{l\sigma'},a_{j\sigma}$ because it's assumed that in the σ summation, when $j=l$, $\sigma\neq\sigma'$. Otherwise there will be two a operators acting on the same site and the contribution is zero.

BQ Term

In spatial expression, this term is $BQ - B'Q'$. Now $-\sqrt{3}Z^{[0]} + W^{[0]}$ gives $B'Q'$. And $2W^{[0]}$ gives BQ. Therefore,

$$2W^{[0]} - \left(-\sqrt{3}Z^{[0]} + W^{[0]} \right) = \sqrt{3}Z^{[0]} + W^{[0]}$$

This looks like $\hat{A}\hat{P}$ term, but without $\frac{1}{2}$ and $h.c.$. But this is not correct, because the definition of Q, Q' is not equivalent due to the index order in v_{ijkl}. So they will give different $W^{[0]}$. Instead we have (note that $\left(\hat{B}_{ij}\right)^{[0]} = \left(\hat{B}'_{ij}\right)^{[0]}$)

$$\begin{aligned}
&\sum_{ij\in L} \left[2\left(\hat{B}_{ij}\right)^{[0]} \otimes_{[0]} \left(\hat{Q}^R_{ij}\right)^{[0]} - \left(\hat{B}'_{ij}\right)^{[0]} \otimes_{[0]} \left(\hat{Q'}^R_{ij}\right)^{[0]} + \sqrt{3}\left(\hat{B}'_{ij}\right)^{[1]} \otimes_{[0]} \left(\hat{Q'}^R_{ij}\right)^{[1]} \right] \\
= &\sum_{ij\in L} \left[\left(\hat{B}_{ij}\right)^{[0]} \otimes_{[0]} \left(\left(2\hat{Q}^R_{ij}\right)^{[0]} - \left(\hat{Q'}^R_{ij}\right)^{[0]} \right) + \sqrt{3}\left(\hat{B}'_{ij}\right)^{[1]} \otimes_{[0]} \left(\hat{Q'}^R_{ij}\right)^{[1]} \right]
\end{aligned}$$

Note that B, Q do not have $[1]$ form.

Normal/Complementary Partitioning

Note that

$$\sqrt{2}\sum_{i\in L} \left[\left(a_i^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(\hat{S}_i^R\right)^{[\frac{1}{2}]} + h.c. \right] = \sqrt{2}\sum_{i\in R} \left[\left(a_i^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(\hat{S}_i^L\right)^{[\frac{1}{2}]} + h.c. \right]$$

Therefore,

$$\begin{aligned}
&\sqrt{2}\sum_{i\in L} \left[\left(a_i^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(\hat{S}_i^R\right)^{[\frac{1}{2}]} + h.c. \right] + 2\sum_{i\in L} \left[\left(a_i^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(\hat{R}_i^R\right)^{[\frac{1}{2}]} + h.c. \right] + 2\sum_{i\in R} \left[\left(a_i^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(\hat{R}_i^L\right)^{[\frac{1}{2}]} + h.c. \right] \\
= &\frac{\sqrt{2}}{2}\sum_{i\in L} \left[\left(a_i^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(\hat{S}_i^R\right)^{[\frac{1}{2}]} + h.c. \right] + \frac{\sqrt{2}}{2}\sum_{i\in R} \left[\left(a_i^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(\hat{S}_i^L\right)^{[\frac{1}{2}]} + h.c. \right] \\
&+ 2\sum_{i\in L} \left[\left(a_i^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(\hat{R}_i^R\right)^{[\frac{1}{2}]} + h.c. \right] + 2\sum_{i\in R} \left[\left(a_i^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(\hat{R}_i^L\right)^{[\frac{1}{2}]} + h.c. \right] \\
= &2\sum_{i\in L} \left[\left(a_i^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left[\left(\hat{R}_i^R\right)^{[\frac{1}{2}]} + \frac{\sqrt{2}}{4}\left(\hat{S}_i^R\right)^{[\frac{1}{2}]} \right] + h.c. \right] + 2\sum_{i\in R} \left[\left(a_i^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left[\left(\hat{R}_i^L\right)^{[\frac{1}{2}]} + \frac{\sqrt{2}}{4}\left(\hat{S}_i^L\right)^{[\frac{1}{2}]} \right] + h.c. \right]
\end{aligned}$$

So define

$$\left(\hat{R}'^{L/R}_i\right)^{[\frac{1}{2}]} := \frac{\sqrt{2}}{4}\left(\hat{S}_i^L\right)^{[\frac{1}{2}]} + \left(\hat{R}_i^L\right)^{[\frac{1}{2}]} = \frac{\sqrt{2}}{4}\sum_{j\in L/R} t_{ij}\left(a_j\right)^{[\frac{1}{2}]} + \sum_{jkl\in L/R} v_{ijkl} \left[\left(a_k^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(a_l\right)^{[\frac{1}{2}]} \right] \otimes_{[\frac{1}{2}]} \left(a_j\right)^{[\frac{1}{2}]}$$

Here $\frac{\sqrt{2}}{4}$ should be understood as $\frac{1}{2} \cdot \frac{1}{\sqrt{2}}$. The $\frac{1}{2}$ is the same as spatial case, and $\frac{1}{\sqrt{2}}$ is because the expected $\sqrt{2}$ factor is not added for the $\hat{R}$ term.

Operator Exchange factors

Here we consider fermion and SU(2) exchange factors together. From $j_2 = 1/2$ CG factors

$$\left\langle j_1 \ \left(M - \frac{1}{2}\right) \ \frac{1}{2} \ \frac{1}{2} \middle| \left(j_1 \pm \frac{1}{2}\right) \ M \right\rangle = \pm \sqrt{\frac{1}{2}\left(1 \pm \frac{M}{j_1 + \frac{1}{2}}\right)}$$

$$\left\langle j_1 \ \left(M + \frac{1}{2}\right) \ \frac{1}{2} \ \left(-\frac{1}{2}\right) \middle| \left(j_1 \pm \frac{1}{2}\right) \ M \right\rangle = \sqrt{\frac{1}{2}\left(1 \mp \frac{M}{j_1 + \frac{1}{2}}\right)}$$

Let $j_1 = \frac{1}{2}$ we have

$$\left\langle \frac{1}{2} \ \left(-\frac{1}{2}\right) \ \frac{1}{2} \ \frac{1}{2} \middle| \left(\frac{1}{2} \pm \frac{1}{2}\right) \ 0 \right\rangle = \pm \sqrt{\frac{1}{2}}$$

$$\left\langle \frac{1}{2} \ \frac{1}{2} \ \frac{1}{2} \ \left(-\frac{1}{2}\right) \middle| \left(\frac{1}{2} \pm \frac{1}{2}\right) \ 0 \right\rangle = \sqrt{\frac{1}{2}}$$

The exchange factor formula is

$$\begin{aligned}
\left(\hat{X}_1^{[S_1]} \otimes_{[S]} \hat{X}_2^{[S_2]}\right)^{[S_z]} &= \sum_{S_{1z}, S_{2z}} \hat{X}_1^{[S_1][S_{1z}]} \hat{X}_2^{[S_2][S_{2z}]} \langle S S_z | S_1 S_{1z}, \ S_2 S_{2z} \rangle \\
&= \mathrm{P}_{\mathrm{fermi}}^{\mathrm{exchange}}(N_1, N_2) \sum_{S_{1z}, S_{2z}} \hat{X}_2^{[S_2][S_{2z}]} \hat{X}_1^{[S_1][S_{1z}]} \langle S S_z | S_1 S_{1z}, \ S_2 S_{2z} \rangle \\
&= \mathrm{P}_{\mathrm{fermi}}^{\mathrm{exchange}}(N_1, N_2) \frac{\langle S S_z | S_1 S_{1z}, \ S_2 S_{2z} \rangle}{\langle S S_z | S_2 S_{2z}, \ S_1 S_{1z} \rangle} \left(\hat{X}_2^{[S_2]} \otimes_{[S]} \hat{X}_1^{[S_1]}\right)^{[S_z]} \\
\hat{X}_1^{[S_1]} \otimes_{[S]} \hat{X}_2^{[S_2]} &= \mathrm{P}_{\mathrm{fermi}}^{\mathrm{exchange}}(N_1, N_2) \mathrm{P}_{\mathrm{SU(2)}}^{\mathrm{exchange}}(S_1, S_2, S) \hat{X}_2^{[S_2]} \otimes_{[S]} \hat{X}_1^{[S_1]}
\end{aligned}$$

For $[1/2] \otimes_{[0]} [1/2]$, this is

$$\mathrm{P}^{\mathrm{exchange}}(\tfrac{1}{2}, \tfrac{1}{2}, 0) = (-1) \frac{\left\langle \frac{1}{2} \ \frac{1}{2} \ \frac{1}{2} \ \left(-\frac{1}{2}\right) \middle| 0 \ 0 \right\rangle}{\left\langle \frac{1}{2} \ \left(-\frac{1}{2}\right) \ \frac{1}{2} \ \frac{1}{2} \middle| 0 \ 0 \right\rangle} = (-1) \frac{\sqrt{\frac{1}{2}}}{-\sqrt{\frac{1}{2}}} = 1$$

For $[1/2] \otimes_{[1]} [1/2]$, this is

$$\mathrm{P}^{\mathrm{exchange}}(\tfrac{1}{2}, \tfrac{1}{2}, 1) = (-1) \frac{\left\langle \frac{1}{2} \ \frac{1}{2} \ \frac{1}{2} \ \left(-\frac{1}{2}\right) \middle| 1 \ 0 \right\rangle}{\left\langle \frac{1}{2} \ \left(-\frac{1}{2}\right) \ \frac{1}{2} \ \frac{1}{2} \middle| 1 \ 0 \right\rangle} = (-1) \frac{\sqrt{\frac{1}{2}}}{\sqrt{\frac{1}{2}}} = -1$$

From CG factors

$$\langle 1 \ m_1 \ 1 \ (-m_1) | 0 \ 0 \rangle = \frac{(-1)^{1 - m_1}}{\sqrt{3}}$$

we have

$$\mathrm{P}^{\mathrm{exchange}}(1, 1, 0) = (+1) \frac{\left\langle 1 \ 1 \ \ 1 \ \ -1 \middle| 0 \ 0 \right\rangle}{\left\langle 1 \ \ -1 \ 1 \ 1 \middle| 0 \ 0 \right\rangle} = (+1) \frac{\frac{(-1)^0}{\sqrt{3}}}{\frac{(-1)^2}{\sqrt{3}}} = 1$$

we have

$$
\begin{aligned}
(\hat{H})^{[0],NC} = &\left(\hat{H}^L\right)^{[0]} \otimes_{[0]} \left(\hat{1}^R\right)^{[0]} + \left(\hat{1}^L\right)^{[0]} \otimes_{[0]} \left(\hat{H}^R\right)^{[0]} \\
&+ 2\sum_{i\in L}\left[\left(a_i^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(\hat{R}_i^{\prime R}\right)^{[\frac{1}{2}]} + \left(a_i\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(\hat{R}_i^{\prime R\dagger}\right)^{[\frac{1}{2}]}\right] + 2\sum_{i\in R}\left[\left(\hat{R}_i^{\prime L\dagger}\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(a_i\right)^{[\frac{1}{2}]} + \left(\hat{R}_i^{\prime L}\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(a_i^\dagger\right)\right. \\
&- \frac{1}{2}\sum_{ik\in L}\left[\left(\hat{A}_{ik}\right)^{[0]} \otimes_{[0]} \left(\hat{P}_{ik}^R\right)^{[0]} + \sqrt{3}\left(\hat{A}_{ik}\right)^{[1]} \otimes_{[0]} \left(\hat{P}_{ik}^R\right)^{[1]} + \left(\hat{A}_{ik}^\dagger\right)^{[0]} \otimes_{[0]} \left(\hat{P}_{ik}^{R\dagger}\right)^{[0]} + \sqrt{3}\left(\hat{A}_{ik}^\dagger\right)^{[1]} \otimes_{[0]} \left(\hat{P}_{ik}^R\right.\right. \\
&+ \sum_{ij\in L}\left[\left(\hat{B}_{ij}\right)^{[0]} \otimes_{[0]} \left(\hat{Q}_{ij}^{\prime\prime R}\right)^{[0]} + \sqrt{3}\left(\hat{B}_{ij}^\prime\right)^{[1]} \otimes_{[0]} \left(\hat{Q}_{ij}^{\prime R}\right)^{[1]}\right]
\end{aligned}
$$

With this normal/complementary partitioning, the operators required in left block are

$$
\left\{\left(\hat{H}^L\right)^{[0]}, \left(\hat{1}^L\right)^{[0]}, \left(a_i^\dagger\right)^{[\frac{1}{2}]}, \left(a_i\right)^{[\frac{1}{2}]}, \left(\hat{R}_k^{\prime L\dagger}\right)^{[\frac{1}{2}]}, \left(\hat{R}_k^{\prime L}\right)^{[\frac{1}{2}]}, \left(\hat{A}_{ij}\right)^{[0]}, \left(\hat{A}_{ij}\right)^{[1]}, \left(\hat{A}_{ij}^\dagger\right)^{[0]}, \left(\hat{A}_{ij}^\dagger\right)^{[1]}, \left(\hat{B}_{ij}\right)^{[0]}, \left(\hat{B}_{ij}^\prime\right)^{[1]}\right\} \quad (i,j \in L,
$$

The operators required in right block are

$$
\left\{\left(\hat{1}^R\right)^{[0]}, \left(\hat{H}^R\right)^{[0]}, \left(\hat{R}_i^{\prime R}\right)^{[\frac{1}{2}]}, \left(\hat{R}_i^{\prime R\dagger}\right)^{[\frac{1}{2}]}, \left(a_k\right)^{[\frac{1}{2}]}, \left(a_k^\dagger\right)^{[\frac{1}{2}]}, \left(\hat{P}_{ij}^R\right)^{[0]}, \left(\hat{P}_{ij}^R\right)^{[1]}, \left(\hat{P}_{ij}^{R\dagger}\right)^{[0]}, \left(\hat{P}_{ij}^{R\dagger}\right)^{[1]}, \left(\hat{Q}_{ij}^{\prime\prime R}\right)^{[0]}, \left(\hat{Q}_{ij}^{\prime R}\right)^{[1]}\right\} \quad (i,j \in
$$

Assuming that there are K sites in total, and K_L/K_R sites in left/right block (optimally, $K_L \le K_R$), the total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$
N_{NC} = 1 + 1 + 2K_L + 2K_R + 4K_L^2 + 2K_L^2 = 6K_L^2 + 2K + 2
$$

Complementary/Normal Partitioning

Note that due the CG factors, exchange any $\otimes_{[0]}$ product will not produce extra sign.

$$
\begin{aligned}
(\hat{H})^{[0],CN} = &\left(\hat{H}^L\right)^{[0]} \otimes_{[0]} \left(\hat{1}^R\right)^{[0]} + \left(\hat{1}^L\right)^{[0]} \otimes_{[0]} \left(\hat{H}^R\right)^{[0]} \\
&+ 2\sum_{i\in L}\left[\left(a_i^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(\hat{R}_i^{\prime R}\right)^{[\frac{1}{2}]} + \left(a_i\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(\hat{R}_i^{\prime R\dagger}\right)^{[\frac{1}{2}]}\right] + 2\sum_{i\in R}\left[\left(\hat{R}_i^{\prime L\dagger}\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(a_i\right)^{[\frac{1}{2}]} + \left(\hat{R}_i^{\prime L}\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(a_i^\dagger\right)\right. \\
&- \frac{1}{2}\sum_{jl\in R}\left[\left(\hat{P}_{jl}^L\right)^{[0]} \otimes_{[0]} \left(\hat{A}_{jl}\right)^{[0]} + \sqrt{3}\left(\hat{P}_{jl}^L\right)^{[1]} \otimes_{[0]} \left(\hat{A}_{jl}\right)^{[1]} + \left(\hat{P}_{jl}^{L\dagger}\right)^{[0]} \otimes_{[0]} \left(\hat{A}_{jl}^\dagger\right)^{[0]} + \sqrt{3}\left(\hat{P}_{jl}^{L\dagger}\right)^{[1]} \otimes_{[0]} \left(\hat{A}_{jl}^\dagger\right)\right. \\
&+ \sum_{kl\in R}\left[\left(\hat{Q}_{kl}^{\prime\prime L}\right)^{[0]} \otimes_{[0]} \left(\hat{B}_{kl}\right)^{[0]} + \sqrt{3}\left(\hat{Q}_{kl}^{\prime L}\right)^{[1]} \otimes_{[0]} \left(\hat{B}_{kl}^\prime\right)^{[1]}\right]
\end{aligned}
$$

Now the operators required in left block are

$$
\left\{\left(\hat{H}^L\right)^{[0]}, \left(\hat{1}^L\right)^{[0]}, \left(a_i^\dagger\right)^{[\frac{1}{2}]}, \left(a_i\right)^{[\frac{1}{2}]}, \left(\hat{R}_k^{\prime L\dagger}\right)^{[\frac{1}{2}]}, \left(\hat{R}_k^{\prime L}\right)^{[\frac{1}{2}]}, \left(\hat{P}_{kl}^L\right)^{[0]}, \left(\hat{P}_{kl}^L\right)^{[1]}, \left(\hat{P}_{kl}^{L\dagger}\right)^{[0]}, \left(\hat{P}_{kl}^{L\dagger}\right)^{[1]}, \left(\hat{Q}_{kl}^{\prime\prime L}\right)^{[0]}, \left(\hat{Q}_{kl}^{\prime L}\right)^{[1]}\right\} \quad (k,l \in
$$

The operators required in right block are

$$
\left\{\left(\hat{1}^R\right)^{[0]}, \left(\hat{H}^R\right)^{[0]}, \left(\hat{R}_i^{\prime R}\right)^{[\frac{1}{2}]}, \left(\hat{R}_i^{\prime R\dagger}\right)^{[\frac{1}{2}]}, \left(a_k\right)^{[\frac{1}{2}]}, \left(a_k^\dagger\right)^{[\frac{1}{2}]}, \left(\hat{A}_{kl}\right)^{[0]}, \left(\hat{A}_{kl}\right)^{[1]}, \left(\hat{A}_{kl}^\dagger\right)^{[0]}, \left(\hat{A}_{kl}^\dagger\right)^{[1]}, \left(\hat{B}_{kl}\right)^{[0]}, \left(\hat{B}_{kl}^\prime\right)^{[1]}\right\} \quad (k,l \in
$$

The total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$
N_{CN} = 1 + 1 + 2K_L + 2K_R + 4K_R^2 + 2K_R^2 = 6K_R^2 + 2K + 2
$$

Blocking

The enlarged left/right block is denoted as $L*/R*$. Make sure that all L operators are to the left of $*$ operators. (The exchange factor for this is -1 for doublet $\otimes$ triplet and $+1$ doublet $\otimes$ singlet.)

First we have

$$\begin{aligned}
\left(\hat{R}_i^{L/R}\right)^{[1/2]} &= \sum_{jkl \in L/R} v_{ijkl} \left[\left(a_k^\dagger\right)^{[1/2]} \otimes_{[0]} \left(a_l\right)^{[1/2]}\right] \otimes_{[1/2]} \left(a_j\right)^{[1/2]} \\
&= \frac{1}{\sqrt{2}} \sum_{jkl \in L/R} v_{ijkl} \left(a_{k\alpha}^\dagger a_{l\alpha} + a_{k\beta}^\dagger a_{l\beta}\right)^{[0]} \otimes_{[1/2]} \left(a_j\right)^{[1/2]} \\
&= \frac{1}{\sqrt{2}} \sum_{jkl \in L/R} v_{ijkl} \begin{pmatrix} -a_{k\alpha}^\dagger a_{l\alpha} a_{j\beta} - a_{k\beta}^\dagger a_{l\beta} a_{j\beta} \\ a_{k\alpha}^\dagger a_{l\alpha} a_{j\alpha} + a_{k\beta}^\dagger a_{l\beta} a_{j\alpha} \end{pmatrix}^{[1/2]}
\end{aligned}$$

From the formula $\sqrt{3}U^{[1/2]} - V^{[1/2]}$ we have

$$\left(\hat{R}_i^{L/R}\right)^{[1/2]} = \frac{\sqrt{3}}{2} \sum_{jkl \in L/R} v_{ijkl} \left(a_k^\dagger\right)^{[1/2]} \otimes_{[1/2]} \left[\left(a_l\right)^{[1/2]} \otimes_{[1]} \left(a_j\right)^{[1/2]}\right] - \frac{1}{2} \sum_{jkl \in L/R} v_{ijkl} \left(a_k^\dagger\right)^{[1/2]} \otimes_{[1/2]} \left[\left(a_l\right)^{[1/2]} \otimes_{[0]} \left(a\right.\right.$$

From the formula $\sqrt{3}S^{[1/2]} - T^{[1/2]}$ we have (for $k \neq l$)

$$\left(\hat{R}_i^{L/R}\right)^{[1/2]} = \frac{\sqrt{3}}{2} \sum_{jkl \in L/R} v_{ijkl} \left(a_l\right)^{[1/2]} \otimes_{[1/2]} \left[\left(a_k^\dagger\right)^{[1/2]} \otimes_{[1]} \left(a_j\right)^{[1/2]}\right] - \frac{1}{2} \sum_{jkl \in L/R} v_{ijkl} \left(a_l\right)^{[1/2]} \otimes_{[1/2]} \left[\left(a_k^\dagger\right)^{[1/2]} \otimes_{[0]} \left(a\right.\right.$$

We have

$$
\begin{aligned}
\left(\hat{R}_i'^{L*}\right)^{[1/2]} =& \left(\hat{R}_i'^{L}\right)^{[1/2]} \otimes_{[1/2]} \left(\hat{1}^{*}\right)^{[0]} + \left(\hat{1}^{L}\right)^{[0]} \otimes_{[1/2]} \left(\hat{R}_i'^{*}\right)^{[1/2]} \\
&+ \sum_{j\in L} \left[\sum_{kl\in *} v_{ijkl} \left(a_k^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(a_l\right)^{[\frac{1}{2}]}\right] \otimes_{[\frac{1}{2}]} \left(a_j\right)^{[\frac{1}{2}]} + \sum_{j\in *} \left[\sum_{kl\in L} v_{ijkl} \left(a_k^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(a_l\right)^{[\frac{1}{2}]}\right] \otimes_{[\frac{1}{2}]} \left(a_j\right)^{[\frac{1}{2}]} \\
&- \frac{1}{2} \sum_{k\in L} \left(a_k^\dagger\right)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl\in *} v_{ijkl} \left(a_l\right)^{[1/2]} \otimes_{[0]} \left(a_j\right)^{[1/2]}\right] + \frac{\sqrt{3}}{2} \sum_{k\in L} \left(a_k^\dagger\right)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl\in *} v_{ijkl} \left(a_l\right)^{[1/2]} \otimes_{[} \right. \\
&- \frac{1}{2} \sum_{k\in *} \left(a_k^\dagger\right)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl\in L} v_{ijkl} \left(a_l\right)^{[1/2]} \otimes_{[0]} \left(a_j\right)^{[1/2]}\right] + \frac{\sqrt{3}}{2} \sum_{k\in *} \left(a_k^\dagger\right)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl\in L} v_{ijkl} \left(a_l\right)^{[1/2]} \otimes \right. \\
&- \frac{1}{2} \sum_{l\in L} \left(a_l\right)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk\in *} v_{ijkl} \left(a_k^\dagger\right)^{[1/2]} \otimes_{[0]} \left(a_j\right)^{[1/2]}\right] + \frac{\sqrt{3}}{2} \sum_{l\in L} \left(a_l\right)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk\in *} v_{ijkl} \left(a_k^\dagger\right)^{[1/2]} \otimes \right. \\
&- \frac{1}{2} \sum_{l\in *} \left(a_l\right)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk\in L} v_{ijkl} \left(a_k^\dagger\right)^{[1/2]} \otimes_{[0]} \left(a_j\right)^{[1/2]}\right] + \frac{\sqrt{3}}{2} \sum_{l\in *} \left(a_l\right)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk\in L} v_{ijkl} \left(a_k^\dagger\right)^{[1/2]} \otimes \right. \\
=& \left(\hat{R}_i'^{L}\right)^{[1/2]} \otimes_{[1/2]} \left(\hat{1}^{*}\right)^{[0]} + \left(\hat{1}^{L}\right)^{[0]} \otimes_{[1/2]} \left(\hat{R}_i'^{*}\right)^{[1/2]} \\
&+ \sum_{j\in L} \left(a_j\right)^{[\frac{1}{2}]} \otimes_{[\frac{1}{2}]} \left[\sum_{kl\in *} v_{ijkl} \left(a_k^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(a_l\right)^{[\frac{1}{2}]}\right] + \sum_{j\in *} \left[\sum_{kl\in L} v_{ijkl} \left(a_k^\dagger\right)^{[\frac{1}{2}]} \otimes_{[0]} \left(a_l\right)^{[\frac{1}{2}]}\right] \otimes_{[\frac{1}{2}]} \left(a_j\right)^{[\frac{1}{2}]} \\
&- \frac{1}{2} \sum_{k\in L} \left(a_k^\dagger\right)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl\in *} v_{ijkl} \left(a_l\right)^{[1/2]} \otimes_{[0]} \left(a_j\right)^{[1/2]}\right] + \frac{\sqrt{3}}{2} \sum_{k\in L} \left(a_k^\dagger\right)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl\in *} v_{ijkl} \left(a_l\right)^{[1/2]} \otimes_{[} \right. \\
&- \frac{1}{2} \sum_{k\in *} \left[\sum_{jl\in L} v_{ijkl} \left(a_l\right)^{[1/2]} \otimes_{[0]} \left(a_j\right)^{[1/2]}\right] \otimes_{[1/2]} \left(a_k^\dagger\right)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{k\in *} \left[\sum_{jl\in L} v_{ijkl} \left(a_l\right)^{[1/2]} \otimes_{[1]} \left(a_j\right)^{[1/2]}\right] \otimes_{[} \\
&- \frac{1}{2} \sum_{l\in L} \left(a_l\right)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk\in *} v_{ijkl} \left(a_k^\dagger\right)^{[1/2]} \otimes_{[0]} \left(a_j\right)^{[1/2]}\right] + \frac{\sqrt{3}}{2} \sum_{l\in L} \left(a_l\right)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk\in *} v_{ijkl} \left(a_k^\dagger\right)^{[1/2]} \otimes \right. \\
&- \frac{1}{2} \sum_{l\in *} \left[\sum_{jk\in L} v_{ijkl} \left(a_k^\dagger\right)^{[1/2]} \otimes_{[0]} \left(a_j\right)^{[1/2]}\right] \otimes_{[1/2]} \left(a_l\right)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{l\in *} \left[\sum_{jk\in L} v_{ijkl} \left(a_k^\dagger\right)^{[1/2]} \otimes_{[1]} \left(a_j\right)^{[1/2]}\right] \otimes
\end{aligned}
$$

After reordering of terms

$$
\begin{aligned}
\big(\hat{R}_i'^{L*}\big)^{[1/2]} &= \big(\hat{R}_i'^{L}\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{1}^{*}\big)^{[0]} + \big(\hat{1}^{L}\big)^{[0]} \otimes_{[1/2]} \big(\hat{R}_i'^{*}\big)^{[1/2]} \\
&\quad - \frac{1}{2}\sum_{k\in L} \big(a_k^\dagger\big)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl\in *} v_{ijkl} \big(a_l\big)^{[1/2]} \otimes_{[0]} \big(a_j\big)^{[1/2]}\right] + \frac{\sqrt{3}}{2}\sum_{k\in L} \big(a_k^\dagger\big)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl\in *} v_{ijkl} \big(a_l\big)^{[1/2]} \otimes_{[} \right. \\
&\quad + \sum_{j\in L} \big(a_j\big)^{[\frac{1}{2}]} \otimes_{[\frac{1}{2}]} \left[\sum_{kl\in *} v_{ijkl} \big(a_k^\dagger\big)^{[\frac{1}{2}]} \otimes_{[0]} \big(a_l\big)^{[\frac{1}{2}]}\right] \\
&\quad - \frac{1}{2}\sum_{l\in L} \big(a_l\big)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk\in *} v_{ijkl} \big(a_k^\dagger\big)^{[1/2]} \otimes_{[0]} \big(a_j\big)^{[1/2]}\right] + \frac{\sqrt{3}}{2}\sum_{l\in L} \big(a_l\big)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk\in *} v_{ijkl} \big(a_k^\dagger\big)^{[1/2]} \otimes \right. \\
&\quad - \frac{1}{2}\sum_{k\in *} \left[\sum_{jl\in L} v_{ijkl} \big(a_l\big)^{[1/2]} \otimes_{[0]} \big(a_j\big)^{[1/2]}\right] \otimes_{[1/2]} \big(a_k^\dagger\big)^{[1/2]} - \frac{\sqrt{3}}{2}\sum_{k\in *} \left[\sum_{jl\in L} v_{ijkl} \big(a_l\big)^{[1/2]} \otimes_{[1]} \big(a_j\big)^{[1/2]}\right] \otimes_{[} \\
&\quad + \sum_{j\in *} \left[\sum_{kl\in L} v_{ijkl} \big(a_k^\dagger\big)^{[\frac{1}{2}]} \otimes_{[0]} \big(a_l\big)^{[\frac{1}{2}]}\right] \otimes_{[\frac{1}{2}]} \big(a_j\big)^{[\frac{1}{2}]} \\
&\quad - \frac{1}{2}\sum_{l\in *} \left[\sum_{jk\in L} v_{ijkl} \big(a_k^\dagger\big)^{[1/2]} \otimes_{[0]} \big(a_j\big)^{[1/2]}\right] \otimes_{[1/2]} \big(a_l\big)^{[1/2]} - \frac{\sqrt{3}}{2}\sum_{l\in *} \left[\sum_{jk\in L} v_{ijkl} \big(a_k^\dagger\big)^{[1/2]} \otimes_{[1]} \big(a_j\big)^{[1/2]}\right] \otimes \\
&= \big(\hat{R}_i'^{L}\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{1}^{*}\big)^{[0]} + \big(\hat{1}^{L}\big)^{[0]} \otimes_{[1/2]} \big(\hat{R}_i'^{*}\big)^{[1/2]} \\
&\quad - \frac{1}{2}\sum_{k\in L} \big(a_k^\dagger\big)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl\in *} v_{ijkl} \big(a_l\big)^{[1/2]} \otimes_{[0]} \big(a_j\big)^{[1/2]}\right] + \frac{\sqrt{3}}{2}\sum_{k\in L} \big(a_k^\dagger\big)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl\in *} v_{ijkl} \big(a_l\big)^{[1/2]} \otimes_{[} \right. \\
&\quad + \frac{1}{2}\sum_{j\in L} \big(a_j\big)^{[1/2]} \otimes_{[1/2]} \left[\sum_{kl\in *} (2v_{ijkl} - v_{ilkj}) \big(a_k^\dagger\big)^{[1/2]} \otimes_{[0]} \big(a_l\big)^{[1/2]}\right] + \frac{\sqrt{3}}{2}\sum_{l\in L} \big(a_l\big)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk\in *} v_{ijkl} \big(\right. \\
&\quad - \frac{1}{2}\sum_{k\in *} \left[\sum_{jl\in L} v_{ijkl} \big(a_l\big)^{[1/2]} \otimes_{[0]} \big(a_j\big)^{[1/2]}\right] \otimes_{[1/2]} \big(a_k^\dagger\big)^{[1/2]} - \frac{\sqrt{3}}{2}\sum_{k\in *} \left[\sum_{jl\in L} v_{ijkl} \big(a_l\big)^{[1/2]} \otimes_{[1]} \big(a_j\big)^{[1/2]}\right] \otimes_{[} \\
&\quad + \frac{1}{2}\sum_{j\in *} \left[\sum_{kl\in L} (2v_{ijkl} - v_{ilkj}) \big(a_k^\dagger\big)^{[1/2]} \otimes_{[0]} \big(a_l\big)^{[1/2]}\right] \otimes_{[1/2]} \big(a_j\big)^{[1/2]} - \frac{\sqrt{3}}{2}\sum_{l\in *} \left[\sum_{jk\in L} v_{ijkl} \big(a_k^\dagger\big)^{[1/2]} \otimes_{[1]} \big(a \right.
\end{aligned}
$$

By definition (The overall exchange factor for $[1/2] \otimes_{[0]} [1/2]$ is 1, and for $[1/2] \otimes_{[1]} [1/2]$ is -1)

$$
\begin{aligned}
\big(\hat{A}_{ik}\big)^{[0/1]} &= \big(a_i^\dagger\big)^{[\frac{1}{2}]} \otimes_{[0/1]} \big(a_k^\dagger\big)^{[\frac{1}{2}]} \\
\big(\hat{A}_{ik}^\dagger\big)^{[0]} &= \big(a_i\big)^{[\frac{1}{2}]} \otimes_{[0]} \big(a_k\big)^{[\frac{1}{2}]} = \big(a_k\big)^{[\frac{1}{2}]} \otimes_{[0]} \big(a_i\big)^{[\frac{1}{2}]} \\
\big(\hat{A}_{ik}^\dagger\big)^{[1]} &= -\big(a_i\big)^{[\frac{1}{2}]} \otimes_{[1]} \big(a_k\big)^{[\frac{1}{2}]} = \big(a_k\big)^{[\frac{1}{2}]} \otimes_{[1]} \big(a_i\big)^{[\frac{1}{2}]} \\
\big(\hat{P}_{ik}^R\big)^{[0/1]} &= \sum_{jl \in R} v_{ijkl} \big(a_l\big)^{[\frac{1}{2}]} \otimes_{[0/1]} \big(a_j\big)^{[\frac{1}{2}]} \\
\big(\hat{B}_{ij}\big)^{[0]} &= \big(a_i^\dagger\big)^{[\frac{1}{2}]} \otimes_{[0]} \big(a_j\big)^{[\frac{1}{2}]} \\
\big(\hat{B}'_{ij}\big)^{[1]} &= \big(a_i^\dagger\big)^{[\frac{1}{2}]} \otimes_{[1]} \big(a_j\big)^{[\frac{1}{2}]} \\
\big(\hat{Q}'^R_{ij}\big)^{[1]} &= \sum_{kl \in R} v_{ilkj} \big(a_k^\dagger\big)^{[\frac{1}{2}]} \otimes_{[1]} \big(a_l\big)^{[\frac{1}{2}]} \\
\big(\hat{Q}''^R_{ij}\big)^{[0]} &= \sum_{kl \in R} (2v_{ijkl} - v_{ilkj}) \big(a_k^\dagger\big)^{[\frac{1}{2}]} \otimes_{[0]} \big(a_l\big)^{[\frac{1}{2}]}
\end{aligned}
$$

we have

$$
\begin{aligned}
\big(\hat{R}_i'^{L*,NC}\big)^{[1/2]} =\ & \big(\hat{R}_i'^{L}\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{1}^*\big)^{[0]} + \big(\hat{1}^L\big)^{[0]} \otimes_{[1/2]} \big(\hat{R}_i'^*\big)^{[1/2]} \\
& - \frac{1}{2} \sum_{k \in L} \big(a_k^\dagger\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{P}_{ik}^*\big)^{[0]} + \frac{\sqrt{3}}{2} \sum_{k \in L} \big(a_k^\dagger\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{P}_{ik}^*\big)^{[1]} \\
& + \frac{1}{2} \sum_{j \in L} \big(a_j\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{Q}''^*_{ij}\big)^{[0]} + \frac{\sqrt{3}}{2} \sum_{l \in L} \big(a_l\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{Q}'^*_{il}\big)^{[1]} \\
& - \frac{1}{2} \sum_{k \in *, jl \in L} v_{ijkl} \big(\hat{A}_{jl}^\dagger\big)^{[0]} \otimes_{[1/2]} \big(a_k^\dagger\big)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{k \in *, jl \in L} v_{ijkl} \big(\hat{A}_{jl}^\dagger\big)^{[1]} \otimes_{[1/2]} \big(a_k^\dagger\big)^{[1/2]} \\
& + \frac{1}{2} \sum_{j \in *, kl \in L} (2v_{ijkl} - v_{ilkj}) \big(\hat{B}_{kl}\big)^{[0]} \otimes_{[1/2]} \big(a_j\big)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{l \in *, jk \in L} v_{ijkl} \big(\hat{B}'_{kj}\big)^{[1]} \otimes_{[1/2]} \big(a_l\big)^{[1/2]} \\
\big(\hat{R}_i'^{L*,CN}\big)^{[1/2]} =\ & \big(\hat{R}_i'^{L}\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{1}^*\big)^{[0]} + \big(\hat{1}^L\big)^{[0]} \otimes_{[1/2]} \big(\hat{R}_i'^*\big)^{[1/2]} \\
& - \frac{1}{2} \sum_{k \in L, jl \in *} v_{ijkl} \big(a_k^\dagger\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{A}_{jl}^\dagger\big)^{[0]} + \frac{\sqrt{3}}{2} \sum_{k \in L, jl \in *} v_{ijkl} \big(a_k^\dagger\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{A}_{jl}^\dagger\big)^{[1]} \\
& + \frac{1}{2} \sum_{j \in L, kl \in *} (2v_{ijkl} - v_{ilkj}) \big(a_j\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{B}_{kl}\big)^{[0]} + \frac{\sqrt{3}}{2} \sum_{l \in L, jk \in *} v_{ijkl} \big(a_l\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{B}'_{kj}\big)^{[1]} \\
& - \frac{1}{2} \sum_{k \in *} \big(\hat{P}_{ik}^L\big)^{[0]} \otimes_{[1/2]} \big(a_k^\dagger\big)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{k \in *} \big(\hat{P}_{ik}^L\big)^{[1]} \otimes_{[1/2]} \big(a_k^\dagger\big)^{[1/2]} \\
& + \frac{1}{2} \sum_{j \in *} \big(\hat{Q}''^L_{ij}\big)^{[0]} \otimes_{[1/2]} \big(a_j\big)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{l \in *} \big(\hat{Q}'^L_{il}\big)^{[1]} \otimes_{[1/2]} \big(a_l\big)^{[1/2]}
\end{aligned}
$$

To generate symmetrized P, we need to change the A line to the following

$$
-\frac{1}{4} \sum_{k \in *, jl \in L} (v_{ijkl} + v_{ilkj}) \big(\hat{A}_{jl}^\dagger\big)^{[0]} \otimes_{[1/2]} \big(a_k^\dagger\big)^{[1/2]} - \frac{\sqrt{3}}{4} \sum_{k \in *, jl \in L} (v_{ijkl} - v_{ilkj}) \big(\hat{A}_{jl}^\dagger\big)^{[1]} \otimes_{[1/2]} \big(a_k^\dagger\big)^{[1/2]}
$$

Similarly,

$$\begin{aligned}
\big(\hat{R}_i'^{R*,NC}\big)^{[1/2]} &= \big(\hat{R}_i'^{*}\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{1}^{R}\big)^{[0]} + \big(\hat{1}^{*}\big)^{[0]} \otimes_{[1/2]} \big(\hat{R}_i'^{R}\big)^{[1/2]} \\
&- \frac{1}{2}\sum_{k\in *} \big(a_k^\dagger\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{P}_{ik}^{R}\big)^{[0]} + \frac{\sqrt{3}}{2}\sum_{k\in *} \big(a_k^\dagger\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{P}_{ik}^{R}\big)^{[1]} \\
&+ \frac{1}{2}\sum_{j\in *} \big(a_j\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{Q}_{ij}''^{R}\big)^{[0]} + \frac{\sqrt{3}}{2}\sum_{l\in *} \big(a_l\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{Q}_{il}'^{R}\big)^{[1]} \\
&- \frac{1}{2}\sum_{k\in R, jl\in *} v_{ijkl}\big(\hat{A}_{jl}^\dagger\big)^{[0]} \otimes_{[1/2]} \big(a_k^\dagger\big)^{[1/2]} - \frac{\sqrt{3}}{2}\sum_{k\in R, jl\in *} v_{ijkl}\big(\hat{A}_{jl}^\dagger\big)^{[1]} \otimes_{[1/2]} \big(a_k^\dagger\big)^{[1/2]} \\
&+ \frac{1}{2}\sum_{j\in R, kl\in *} (2v_{ijkl} - v_{ilkj})\big(\hat{B}_{kl}\big)^{[0]} \otimes_{[1/2]} \big(a_j\big)^{[1/2]} - \frac{\sqrt{3}}{2}\sum_{l\in R, jk\in *} v_{ijkl}\big(\hat{B}'_{kj}\big)^{[1]} \otimes_{[1/2]} \big(a_l\big)^{[1/2]} \\
\big(\hat{R}_i'^{R*,CN}\big)^{[1/2]} &= \big(\hat{R}_i'^{*}\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{1}^{R}\big)^{[0]} + \big(\hat{1}^{*}\big)^{[0]} \otimes_{[1/2]} \big(\hat{R}_i'^{R}\big)^{[1/2]} \\
&- \frac{1}{2}\sum_{k\in *, jl\in R} v_{ijkl}\big(a_k^\dagger\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{A}_{jl}^\dagger\big)^{[0]} + \frac{\sqrt{3}}{2}\sum_{k\in *, jl\in R} v_{ijkl}\big(a_k^\dagger\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{A}_{jl}^\dagger\big)^{[1]} \\
&+ \frac{1}{2}\sum_{j\in *, kl\in R} (2v_{ijkl} - v_{ilkj})\big(a_j\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{B}_{kl}\big)^{[0]} + \frac{\sqrt{3}}{2}\sum_{l\in *, jk\in R} v_{ijkl}\big(a_l\big)^{[1/2]} \otimes_{[1/2]} \big(\hat{B}'_{kj}\big)^{[1]} \\
&- \frac{1}{2}\sum_{k\in R} \big(\hat{P}_{ik}^{*}\big)^{[0]} \otimes_{[1/2]} \big(a_k^\dagger\big)^{[1/2]} - \frac{\sqrt{3}}{2}\sum_{k\in R} \big(\hat{P}_{ik}^{*}\big)^{[1]} \otimes_{[1/2]} \big(a_k^\dagger\big)^{[1/2]} \\
&+ \frac{1}{2}\sum_{j\in R} \big(\hat{Q}_{ij}''^{*}\big)^{[0]} \otimes_{[1/2]} \big(a_j\big)^{[1/2]} - \frac{\sqrt{3}}{2}\sum_{l\in R} \big(\hat{Q}_{il}'^{*}\big)^{[1]} \otimes_{[1/2]} \big(a_l\big)^{[1/2]}
\end{aligned}$$

Number of terms

$$\begin{aligned}
N_{R',NC} &= (2 + 4K_L + 4K_L^2)K_R + (2 + 4 + 4K_R)K_L = 4K_L^2K_R + 8K_LK_R + 2K + 4K_L \\
N_{R',CN} &= (2 + 4K_L + 4)K_R + (2 + 4K_R^2 + 4K_R)K_L = 4K_R^2K_L + 8K_RK_L + 2K + 4K_R
\end{aligned}$$

Blocking of other complementary operators is straightforward

$$\begin{aligned}
\big(\hat{P}_{ik}^{L*,CN}\big)^{[0/1]} &= \big(\hat{P}_{ik}^{L}\big)^{[0/1]} \otimes_{[0/1]} \big(\hat{1}^{*}\big)^{[0]} + \big(\hat{1}^{L}\big)^{[0]} \otimes_{[0/1]} \big(\hat{P}_{ik}^{*}\big)^{[0/1]} + \sum_{j\in L, l\in *} v_{ijkl}\big(a_l\big)^{[\frac{1}{2}]} \otimes_{[0/1]} \big(a_j\big)^{[\frac{1}{2}]} + \sum_{j\in *, l\in L} v_{ijkl}\big(a_l\big) \\
&= \big(\hat{P}_{ik}^{L}\big)^{[0/1]} \otimes_{[0/1]} \big(\hat{1}^{*}\big)^{[0]} + \big(\hat{1}^{L}\big)^{[0]} \otimes_{[0/1]} \big(\hat{P}_{ik}^{*}\big)^{[0/1]} \pm \sum_{j\in L, l\in *} v_{ijkl}\big(a_j\big)^{[\frac{1}{2}]} \otimes_{[0/1]} \big(a_l\big)^{[\frac{1}{2}]} + \sum_{j\in *, l\in L} v_{ijkl}\big(a_l\big) \\
\big(\hat{P}_{ik}^{R*,NC}\big)^{[0/1]} &= \big(\hat{P}_{ik}^{*}\big)^{[0/1]} \otimes_{[0/1]} \big(\hat{1}^{R}\big)^{[0]} + \big(\hat{1}^{*}\big)^{[0]} \otimes_{[0/1]} \big(\hat{P}_{ik}^{R}\big)^{[0/1]} \pm \sum_{j\in *, l\in R} v_{ijkl}\big(a_j\big)^{[\frac{1}{2}]} \otimes_{[0/1]} \big(a_l\big)^{[\frac{1}{2}]} + \sum_{j\in R, l\in *} v_{ijkl}\big(a_l
\end{aligned}$$

and

$$\begin{aligned}
\big(\hat{Q}_{ij}''^{L*,CN}\big)^{[0]} &= \big(\hat{Q}_{ij}''^{L}\big)^{[0]} \otimes_{[0]} \big(\hat{1}^{*}\big)^{[0]} + \big(\hat{1}^{L}\big)^{[0]} \otimes_{[0]} \big(\hat{Q}_{ij}''^{*}\big)^{[0]} + \sum_{k\in L, l\in *} (2v_{ijkl} - v_{ilkj})\big(a_k^\dagger\big)^{[\frac{1}{2}]} \otimes_{[0]} \big(a_l\big)^{[\frac{1}{2}]} + \sum_{k\in *, l\in L} (2v_{ijkl} \\
&= \big(\hat{Q}_{ij}''^{L}\big)^{[0]} \otimes_{[0]} \big(\hat{1}^{*}\big)^{[0]} + \big(\hat{1}^{L}\big)^{[0]} \otimes_{[0]} \big(\hat{Q}_{ij}''^{*}\big)^{[0]} + \sum_{k\in L, l\in *} (2v_{ijkl} - v_{ilkj})\big(a_k^\dagger\big)^{[\frac{1}{2}]} \otimes_{[0]} \big(a_l\big)^{[\frac{1}{2}]} + \sum_{k\in *, l\in L} (2v_{ijkl} \\
\big(\hat{Q}_{ij}''^{R*,NC}\big)^{[0]} &= \big(\hat{Q}_{ij}''^{*}\big)^{[0]} \otimes_{[0]} \big(\hat{1}^{R}\big)^{[0]} + \big(\hat{1}^{*}\big)^{[0]} \otimes_{[0]} \big(\hat{Q}_{ij}''^{R}\big)^{[0]} + \sum_{k\in *, l\in R} (2v_{ijkl} - v_{ilkj})\big(a_k^\dagger\big)^{[\frac{1}{2}]} \otimes_{[0]} \big(a_l\big)^{[\frac{1}{2}]} + \sum_{k\in R, l\in *} (2v_{ijk
\end{aligned}$$

and

$$\begin{aligned}
(\hat{Q}_{ij}^{\prime L*,CN})^{[1]} &= (\hat{Q}_{ij}^{\prime L})^{[1]} \otimes_{[1]} (\hat{1}^{*})^{[0]} + (\hat{1}^{L})^{[0]} \otimes_{[1]} (\hat{Q}_{ij}^{\prime *})^{[1]} + \sum_{k\in L,l\in *} v_{ilkj} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[1]} (a_l)^{[\frac{1}{2}]} + \sum_{k\in *,l\in L} v_{ilkj} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[1]} (\\
&= (\hat{Q}_{ij}^{\prime L})^{[1]} \otimes_{[1]} (\hat{1}^{*})^{[0]} + (\hat{1}^{L})^{[0]} \otimes_{[1]} (\hat{Q}_{ij}^{\prime *})^{[1]} + \sum_{k\in L,l\in *} v_{ilkj} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[1]} (a_l)^{[\frac{1}{2}]} - \sum_{k\in *,l\in L} v_{ilkj} (a_l)^{[\frac{1}{2}]} \otimes_{[1]} (\\
(\hat{Q}_{ij}^{\prime R*,CN})^{[1]} &= (\hat{Q}_{ij}^{\prime *})^{[1]} \otimes_{[1]} (\hat{1}^{R})^{[0]} + (\hat{1}^{*})^{[0]} \otimes_{[1]} (\hat{Q}_{ij}^{\prime R})^{[1]} + \sum_{k\in *,l\in R} v_{ilkj} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[1]} (a_l)^{[\frac{1}{2}]} - \sum_{k\in R,l\in *} v_{ilkj} (a_l)^{[\frac{1}{2}]} \otimes_{[1]} (
\end{aligned}$$

Middle-Site Transformation

$$\begin{aligned}
(\hat{P}_{ik}^{L,NC\rightarrow CN})^{[0/1]} &= \sum_{jl\in L} v_{ijkl} (a_l)^{[\frac{1}{2}]} \otimes_{[0/1]} (a_j)^{[\frac{1}{2}]} = \sum_{jl\in L} v_{ijkl} (\hat{A}_{jl}^\dagger)^{[0/1]} \\
(\hat{Q}_{ij}^{\prime\prime L,NC\rightarrow CN})^{[0]} &= \sum_{kl\in R} (2v_{ijkl} - v_{ilkj}) (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]} = \sum_{kl\in R} (2v_{ijkl} - v_{ilkj}) (\hat{B}_{kl})^{[0]} \\
(\hat{Q}_{ij}^{\prime L,NC\rightarrow CN})^{[1]} &= \sum_{kl\in R} v_{ilkj} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[1]} (a_l)^{[\frac{1}{2}]} = \sum_{kl\in R} v_{ilkj} (\hat{B}_{kl}^{\prime})^{[1]}
\end{aligned}$$

7.1.3 DMRG Quantum Chemistry Hamiltonian in Unrestricted Spatial Orbitals

Hamiltonian

The quantum chemistry Hamiltonian is written as follows

$$\hat{H} = \sum_{ij,\sigma} t_{ij,\sigma}\ a_{i\sigma}^\dagger a_{j\sigma} + \frac{1}{2} \sum_{ijkl,\sigma\sigma'} v_{ijkl,\sigma\sigma'}\ a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma}$$

where

$$\begin{aligned}
t_{ij,\sigma} = t_{(ij),\sigma} &= \int \mathrm{d}\mathbf{x}\ \phi_{i\sigma}^*(\mathbf{x}) \left(-\frac{1}{2}\nabla^2 - \sum_a \frac{Z_a}{r_a} \right) \phi_{j\sigma}(\mathbf{x}) \\
v_{ijkl,\sigma\sigma'} = v_{(ij)(kl),\sigma\sigma'} = v_{(kl)(ij),\sigma\sigma'} &= \int \mathrm{d}\mathbf{x}_1 \mathrm{d}\mathbf{x}_2\ \frac{\phi_{i\sigma}^*(\mathbf{x}_1)\phi_{k\sigma'}^*(\mathbf{x}_2)\phi_{l\sigma'}(\mathbf{x}_2)\phi_{j\sigma}(\mathbf{x}_1)}{r_{12}}
\end{aligned}$$

Note that here the order of $ijkl$ is the same as that in `FCIDUMP` (chemist's notation $[ij|kl]$).

Partitioning in Spatial Orbitals

The partitioning of Hamiltonian in left (L) and right (R) blocks is given by

$$\begin{aligned}
\hat{H} = &\ \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R \\
&+ \Big(\sum_{i\in L,\sigma} a_{i\sigma}^\dagger \hat{S}_{i\sigma}^R + h.c. \Big) + \Big(\sum_{i\in L,\sigma} a_{i\sigma}^\dagger \hat{R}_{i\sigma}^R + h.c. + \sum_{i\in R,\sigma} a_{i\sigma}^\dagger \hat{R}_{i\sigma}^L + h.c. \Big) \\
&+ \frac{1}{2} \Big(\sum_{ik\in L,\sigma\sigma'} \hat{A}_{ik,\sigma\sigma'}^L \hat{P}_{ik,\sigma\sigma'}^R + h.c. \Big) + \sum_{ij\in L,\sigma} \hat{B}_{ij\sigma} \hat{Q}_{ij\sigma}^R - \sum_{il\in L,\sigma\sigma'} \hat{B}_{il\sigma\sigma'}^{\prime} \hat{Q}_{il\sigma\sigma'}^{\prime R}
\end{aligned}$$

where the normal and complementary operators are defined by

$$\begin{aligned}
\hat{S}^{L/R}_{i\sigma} &= \sum_{j\in L/R} t_{ij,\sigma} a_{j\sigma}, \\
\hat{R}^{L/R}_{i\sigma} &= \sum_{jkl\in L/R,\sigma'} v_{ijkl,\sigma\sigma'} a^\dagger_{k\sigma'} a_{l\sigma'} a_{j\sigma}, \\
\hat{A}_{ik,\sigma\sigma'} &= a^\dagger_{i\sigma} a^\dagger_{k\sigma'}, \\
\hat{B}_{ij,\sigma} &= a^\dagger_{i\sigma} a_{j\sigma}, \\
\hat{B}'_{il,\sigma\sigma'} &= a^\dagger_{i\sigma} a_{l\sigma'}, \\
\hat{P}^R_{ik,\sigma\sigma'} &= \sum_{jl\in R} v_{ijkl,\sigma\sigma'} a_{l\sigma'} a_{j\sigma}, \\
\hat{Q}^R_{ij,\sigma} &= \sum_{kl\in R,\sigma'} v_{ijkl,\sigma\sigma'} a^\dagger_{k\sigma'} a_{l\sigma'}, \\
\hat{Q}'^R_{il,\sigma\sigma'} &= \sum_{jk\in R} v_{ijkl,\sigma\sigma'} a^\dagger_{k\sigma'} a_{j\sigma}
\end{aligned}$$

Note that we need to move all on-site interaction into local Hamiltonian, so that when construction interaction terms in Hamiltonian, operators anticommute (without giving extra constant terms).

Define

$$\hat{R}'^{L/R}_{i\sigma} = \frac{1}{2}\hat{S}^{L/R}_{i\sigma} + \hat{R}^{L/R}_{i\sigma} = \frac{1}{2}\sum_{j\in L/R} t_{ij,\sigma} a_{j\sigma} + \sum_{jkl\in L/R,\sigma'} v_{ijkl,\sigma\sigma'} a^\dagger_{k\sigma'} a_{l\sigma'} a_{j\sigma}$$

Then we have

$$\begin{aligned}
\hat{H}^{NC} = \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R + \sum_{i\in L,\sigma} \left(a^\dagger_{i\sigma} \hat{R}'^R_{i\sigma} - a_{i\sigma} \hat{R}'^{R\dagger}_{i\sigma} \right) + \sum_{i\in R,\sigma} \left(\hat{R}'^{L\dagger}_{i\sigma} a_{i\sigma} - \hat{R}'^L_{i\sigma} a^\dagger_{i\sigma} \right) \\
+ \frac{1}{2} \sum_{ik\in L,\sigma\sigma'} \left(\hat{A}_{ik,\sigma\sigma'} \hat{P}^R_{ik,\sigma\sigma'} + \hat{A}^\dagger_{ik,\sigma\sigma'} \hat{P}^{R\dagger}_{ik,\sigma\sigma'} \right) + \sum_{ij\in L,\sigma} \hat{B}_{ij,\sigma} \hat{Q}^R_{ij,\sigma} - \sum_{il\in L,\sigma\sigma'} \hat{B}'_{il\sigma\sigma'} \hat{Q}'^R_{il\sigma\sigma'}
\end{aligned}$$

Normal/Complementary Partitioning

With this normal/complementary partitioning, the operators required in left block are

$$\left\{ \hat{H}^L, \hat{1}^L, a^\dagger_{i\sigma}, a_{i\sigma}, \hat{R}'^{L\dagger}_{k\sigma}, \hat{R}'^L_{k\sigma}, \hat{A}_{ij,\sigma\sigma'}, \hat{A}^\dagger_{ij,\sigma\sigma'}, \hat{B}_{ij,\sigma}, \hat{B}'_{ij,\sigma\sigma'} \right\} \quad (i,j \in L,\ k \in R)$$

The operators required in right block are

$$\left\{ \hat{1}^R, \hat{H}^R, \hat{R}'^R_{i\sigma}, \hat{R}'^{R\dagger}_{i\sigma}, a_{k\sigma}, a^\dagger_{k\sigma}, \hat{P}^R_{ij,\sigma\sigma'}, \hat{P}^{R\dagger}_{ij,\sigma\sigma'}, \hat{Q}^R_{ij,\sigma}, \hat{Q}'^R_{ij,\sigma\sigma'} \right\} \quad (i,j \in L,\ k \in R)$$

Assuming that there are K sites in total, and K_L/K_R sites in left/right block (optimally, $K_L \le K_R$), the total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$N_{NC} = 1 + 1 + 4K_L + 4K_R + 8K_L^2 + 2K_L^2 + 4K_L^2 = 14K_L^2 + 4K + 2$$

Complementary/Normal Partitioning

$$\hat{H}^{CN} = \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R + \sum_{i\in L,\sigma} \left(a^\dagger_{i\sigma} \hat{R}'^R_{i\sigma} - a_{i\sigma} \hat{R}'^{R\dagger}_{i\sigma} \right) + \sum_{i\in R,\sigma} \left(\hat{R}'^{L\dagger}_{i\sigma} a_{i\sigma} - \hat{R}'^L_{i\sigma} a^\dagger_{i\sigma} \right)$$
$$+ \frac{1}{2} \sum_{jl\in R,\sigma\sigma'} \left(\hat{P}^L_{jl,\sigma\sigma'} \hat{A}_{jl,\sigma\sigma'} + \hat{P}^{L\dagger}_{jl,\sigma\sigma'} \hat{A}^\dagger_{jl,\sigma\sigma'} \right) + \sum_{kl\in R,\sigma} \hat{Q}^L_{kl,\sigma} \hat{B}_{kl,\sigma} - \sum_{jk\in R,\sigma\sigma'} \hat{Q}'^L_{jk\sigma\sigma'} \hat{B}'_{jk\sigma\sigma'}$$

Now the operators required in left block are

$$\left\{ \hat{H}^L, \hat{1}^L, a^\dagger_{i\sigma}, a_{i\sigma}, \hat{R}'^{L\dagger}_{k\sigma}, \hat{R}'^L_{k\sigma}, \hat{P}^L_{kl,\sigma\sigma'}, \hat{P}^{L\dagger}_{kl,\sigma\sigma'}, \hat{Q}^L_{kl,\sigma}, \hat{Q}'^L_{kl,\sigma\sigma'} \right\} \quad (k,l \in R,\ i \in L)$$

The operators required in right block are

$$\left\{ \hat{1}^R, \hat{H}^R, \hat{R}'^R_{i\sigma}, \hat{R}'^{R\dagger}_{i\sigma}, a_{k\sigma}, a^\dagger_{k\sigma}, \hat{A}_{kl,\sigma\sigma'}, \hat{A}^\dagger_{kl,\sigma\sigma'}, \hat{B}_{kl,\sigma}, \hat{B}'_{kl,\sigma\sigma'} \right\} \quad (k,l \in R,\ i \in L)$$

The total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$N_{CN} = 1 + 1 + 4K_R + 4K_L + 8K_R^2 + 2K_R^2 + 4K_R^2 = 14K_R^2 + 4K + 2$$

Blocking

The enlarged left/right block is denoted as $L*/R*$. Make sure that all L operators are to the left of $*$ operators.

$$\hat{R}'^{L*}_{i\sigma} = \hat{R}'^L_{i\sigma} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}'^*_{i\sigma} + \sum_{j\in L} \left(\sum_{kl\in *,\sigma'} v_{ijkl,\sigma\sigma'} a^\dagger_{k\sigma'} a_{l\sigma'} \right) a_{j\sigma} + \sum_{j\in *} \left(\sum_{kl\in L,\sigma'} v_{ijkl,\sigma\sigma'} a^\dagger_{k\sigma'} a_{l\sigma'} \right) a_{j\sigma}$$
$$+ \sum_{k\in L,\sigma'} a^\dagger_{k\sigma'} \left(\sum_{jl\in *} v_{ijkl,\sigma\sigma'} a_{l\sigma'} a_{j\sigma} \right) + \sum_{k\in *,\sigma'} a^\dagger_{k\sigma'} \left(\sum_{jl\in L} v_{ijkl,\sigma\sigma'} a_{l\sigma'} a_{j\sigma} \right) - \sum_{l\in L,\sigma'} a_{l\sigma'} \left(\sum_{jk\in *} v_{ijkl,\sigma\sigma'} a^\dagger_{k\sigma'} a_{j\sigma} \right) -$$
$$= \hat{R}'^L_{i\sigma} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}'^*_{i\sigma} + \sum_{j\in L} a_{j\sigma} \left(\sum_{kl\in *,\sigma'} v_{ijkl,\sigma\sigma'} a^\dagger_{k\sigma'} a_{l\sigma'} \right) + \sum_{j\in *} \left(\sum_{kl\in L,\sigma'} v_{ijkl,\sigma\sigma'} a^\dagger_{k\sigma'} a_{l\sigma'} \right) a_{j\sigma}$$
$$+ \sum_{k\in L,\sigma'} a^\dagger_{k\sigma'} \left(\sum_{jl\in *} v_{ijkl,\sigma\sigma'} a_{l\sigma'} a_{j\sigma} \right) + \sum_{k\in *,\sigma'} \left(\sum_{jl\in L} v_{ijkl,\sigma\sigma'} a_{l\sigma'} a_{j\sigma} \right) a^\dagger_{k\sigma'} - \sum_{l\in L,\sigma'} a_{l\sigma'} \left(\sum_{jk\in *} v_{ijkl,\sigma\sigma'} a^\dagger_{k\sigma'} a_{j\sigma} \right) -$$

Now there are two possibilities. In NC partition, in L we have $A, A^\dagger, B, B'$ and in $*$ we have $P, P^\dagger, Q, Q'$. In CN partition, the opposite is true. Therefore, we have

$$\hat{R}'^{L*,NC}_{i\sigma} = \hat{R}'^L_{i\sigma} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}'^*_{i\sigma} + \sum_{j\in L} a_{j\sigma} \hat{Q}^*_{ij,\sigma} + \sum_{j\in *,kl\in L,\sigma'} v_{ijkl,\sigma\sigma'} \hat{B}_{kl,\sigma'} a_{j\sigma}$$
$$+ \sum_{k\in L,\sigma'} a^\dagger_{k\sigma'} \hat{P}^*_{ik,\sigma\sigma'} + \sum_{k\in *,jl\in L,\sigma'} v_{ijkl,\sigma\sigma'} \hat{A}^\dagger_{jl,\sigma\sigma'} a^\dagger_{k\sigma'} - \sum_{l\in L,\sigma'} a_{l\sigma'} \hat{Q}'^*_{il,\sigma\sigma'} - \sum_{l\in *,jk\in L,\sigma'} v_{ijkl,\sigma\sigma'} \hat{B}'_{kj,\sigma'\sigma} a_{l\sigma'}$$
$$= \hat{R}'^L_{i\sigma} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}'^*_{i\sigma} + \sum_{k\in L,\sigma'} a^\dagger_{k\sigma'} \hat{P}^*_{ik,\sigma\sigma'} + \sum_{j\in L} a_{j\sigma} \hat{Q}^*_{ij,\sigma} - \sum_{l\in L,\sigma'} a_{l\sigma'} \hat{Q}'^*_{il,\sigma\sigma'}$$
$$+ \sum_{k\in *,jl\in L,\sigma'} v_{ijkl,\sigma\sigma'} \hat{A}^\dagger_{jl,\sigma\sigma'} a^\dagger_{k\sigma'} + \sum_{j\in *,kl\in L,\sigma'} v_{ijkl,\sigma\sigma'} \hat{B}_{kl,\sigma'} a_{j\sigma} - \sum_{l\in *,jk\in L,\sigma'} v_{ijkl,\sigma\sigma'} \hat{B}'_{kj,\sigma'\sigma} a_{l\sigma'}$$

$$\begin{aligned}\hat{R}'^{L*,CN}_{i\sigma} &= \hat{R}'^{L}_{i\sigma} \otimes \hat{1}^{*} + \hat{1}^{L} \otimes \hat{R}'^{*}_{i\sigma} + \sum_{j\in L, kl\in *,\sigma'} v_{ijkl,\sigma\sigma'} a_{j\sigma} \hat{B}_{kl,\sigma'} + \sum_{j\in *} \hat{Q}^{L}_{ij,\sigma} a_{j\sigma} \\ &+ \sum_{k\in L, jl\in *,\sigma'} v_{ijkl,\sigma\sigma'} a^{\dagger}_{k\sigma'} \hat{A}^{\dagger}_{jl,\sigma\sigma'} + \sum_{k\in *,\sigma'} \hat{P}^{L}_{ik,\sigma\sigma'} a^{\dagger}_{k\sigma'} - \sum_{l\in L, jk\in *,\sigma'} v_{ijkl,\sigma\sigma'} a_{l\sigma'} \hat{B}'_{kj,\sigma'\sigma} - \sum_{l\in *,\sigma'} \hat{Q}'^{L}_{il,\sigma\sigma'} a_{l\sigma'} \\ &= \hat{R}'^{L}_{i\sigma} \otimes \hat{1}^{*} + \hat{1}^{L} \otimes \hat{R}'^{*}_{i\sigma} + \sum_{k\in L, jl\in *,\sigma'} v_{ijkl,\sigma\sigma'} a^{\dagger}_{k\sigma'} \hat{A}^{\dagger}_{jl,\sigma\sigma'} + \sum_{j\in L, kl\in *,\sigma'} v_{ijkl,\sigma\sigma'} a_{j\sigma} \hat{B}_{kl,\sigma'} - \sum_{l\in L, jk\in *,\sigma'} v_{ijkl,\sigma\sigma'} a_{l\sigma'} \\ &+ \sum_{k\in *,\sigma'} \hat{P}^{L}_{ik,\sigma\sigma'} a^{\dagger}_{k\sigma'} + \sum_{j\in *} \hat{Q}^{L}_{ij,\sigma} a_{j\sigma} - \sum_{l\in *,\sigma'} \hat{Q}'^{L}_{il,\sigma\sigma'} a_{l\sigma'}\end{aligned}$$

Simplified Form

Define

$$\hat{Q}''^{R}_{ij,\sigma\sigma'} = \delta_{\sigma\sigma'} \hat{Q}^{R}_{ij\sigma} - \hat{Q}'^{R}_{ij\sigma\sigma'}$$

we have N/C form

$$\begin{aligned}\hat{H}^{NC} &= \hat{H}^{L} \otimes \hat{1}^{R} + \hat{1}^{L} \otimes \hat{H}^{R} + \sum_{i\in L,\sigma} \left(a^{\dagger}_{i\sigma} \hat{R}'^{R}_{i\sigma} - a_{i\sigma} \hat{R}'^{R\dagger}_{i\sigma} \right) + \sum_{i\in R,\sigma} \left(\hat{R}'^{L\dagger}_{i\sigma} a_{i\sigma} - \hat{R}'^{L}_{i\sigma} a^{\dagger}_{i\sigma} \right) \\ &+ \frac{1}{2} \sum_{ik\in L,\sigma\sigma'} \left(\hat{A}_{ik,\sigma\sigma'} \hat{P}^{R}_{ik,\sigma\sigma'} + \hat{A}^{\dagger}_{ik,\sigma\sigma'} \hat{P}^{R\dagger}_{ik,\sigma\sigma'} \right) + \sum_{ij\in L,\sigma\sigma'} \hat{B}'_{ij\sigma\sigma'} \hat{Q}''^{R}_{ij\sigma\sigma'}\end{aligned}$$

With this normal/complementary partitioning, the operators required in left block are

$$\left\{ \hat{H}^{L}, \hat{1}^{L}, a^{\dagger}_{i\sigma}, a_{i\sigma}, \hat{R}'^{L\dagger}_{k\sigma}, \hat{R}'^{L}_{k\sigma}, \hat{A}_{ij,\sigma\sigma'}, \hat{A}^{\dagger}_{ij,\sigma\sigma'}, \hat{B}'_{ij,\sigma\sigma'} \right\} \quad (i,j \in L,\ k \in R)$$

The operators required in right block are

$$\left\{ \hat{1}^{R}, \hat{H}^{R}, \hat{R}'^{R}_{i\sigma}, \hat{R}'^{R\dagger}_{i\sigma}, a_{k\sigma}, a^{\dagger}_{k\sigma}, \hat{P}^{R}_{ij,\sigma\sigma'}, \hat{P}^{R\dagger}_{ij,\sigma\sigma'}, \hat{Q}''^{R}_{ij,\sigma\sigma'} \right\} \quad (i,j \in L,\ k \in R)$$

Assuming that there are K sites in total, and K_L/K_R sites in left/right block (optimally, $K_L \le K_R$), the total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$N_{NC} = 1 + 1 + 4K_L + 4K_R + 8K_L^2 + 4K_L^2 = 12K_L^2 + 4K + 2$$

and C/N form

$$\begin{aligned}\hat{H}^{CN} &= \hat{H}^{L} \otimes \hat{1}^{R} + \hat{1}^{L} \otimes \hat{H}^{R} + \sum_{i\in L,\sigma} \left(a^{\dagger}_{i\sigma} \hat{R}'^{R}_{i\sigma} - a_{i\sigma} \hat{R}'^{R\dagger}_{i\sigma} \right) + \sum_{i\in R,\sigma} \left(\hat{R}'^{L\dagger}_{i\sigma} a_{i\sigma} - \hat{R}'^{L}_{i\sigma} a^{\dagger}_{i\sigma} \right) \\ &+ \frac{1}{2} \sum_{jl\in R,\sigma\sigma'} \left(\hat{P}^{L}_{jl,\sigma\sigma'} \hat{A}_{jl,\sigma\sigma'} + \hat{P}^{L\dagger}_{jl,\sigma\sigma'} \hat{A}^{\dagger}_{jl,\sigma\sigma'} \right) + \sum_{kl\in R,\sigma\sigma'} \hat{Q}''^{L}_{kl\sigma\sigma'} \hat{B}'_{kl\sigma\sigma'}\end{aligned}$$

Now the operators required in left block are

$$\left\{ \hat{H}^{L}, \hat{1}^{L}, a^{\dagger}_{i\sigma}, a_{i\sigma}, \hat{R}'^{L\dagger}_{k\sigma}, \hat{R}'^{L}_{k\sigma}, \hat{P}^{L}_{kl,\sigma\sigma'}, \hat{P}^{L\dagger}_{kl,\sigma\sigma'}, \hat{Q}''^{L}_{kl,\sigma\sigma'} \right\} \quad (k,l \in R,\ i \in L)$$

The operators required in right block are

$$\left\{\hat{1}^R, \hat{H}^R, \hat{R}^R_{i\sigma}, \hat{R}'^{R\dagger}_{i\sigma}, a_{k\sigma}, a^\dagger_{k\sigma}, \hat{A}_{kl,\sigma\sigma'}, \hat{A}^\dagger_{kl,\sigma\sigma'}, \hat{B}'_{kl,\sigma\sigma'}\right\} \quad (k,l \in R,\ i \in L)$$

The total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$N_{CN} = 1 + 1 + 4K_R + 4K_L + 8K_R^2 + 4K_R^2 = 12K_R^2 + 4K + 2$$

Then for blocking

$$\begin{aligned}\hat{R}'^{L*,NC}_{i\sigma} &= \hat{R}'^{L}_{i\sigma} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}'^{*}_{i\sigma} + \sum_{k\in L,\sigma'} a^\dagger_{k\sigma'} \hat{P}^*_{ik,\sigma\sigma'} + \sum_{j\in L,\sigma'} a_{j\sigma'} \hat{Q}''^*_{ij,\sigma\sigma'} \\ &+ \sum_{k\in *,jl\in L,\sigma'} v_{ijkl,\sigma\sigma'} \hat{A}^\dagger_{jl,\sigma\sigma'} a^\dagger_{k\sigma'} + \sum_{j\in *,kl\in L,\sigma'} v_{ijkl,\sigma\sigma'} \hat{B}'_{kl,\sigma'\sigma'} a_{j\sigma} - \sum_{l\in *,jk\in L,\sigma'} v_{ijkl,\sigma\sigma'} \hat{B}'_{kj,\sigma'\sigma} a_{l\sigma'}\end{aligned}$$

$$\begin{aligned}\hat{R}'^{L*,CN}_{i\sigma} &= \hat{R}'^{L}_{i\sigma} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}'^{*}_{i\sigma} + \sum_{k\in L,jl\in *,\sigma'} v_{ijkl,\sigma\sigma'} a^\dagger_{k\sigma'} \hat{A}^\dagger_{jl,\sigma\sigma'} + \sum_{j\in L,kl\in *,\sigma'} v_{ijkl,\sigma\sigma'} a_{j\sigma} \hat{B}'_{kl,\sigma'\sigma'} \\ &- \sum_{l\in L,jk\in *,\sigma'} v_{ijkl,\sigma\sigma'} a_{l\sigma'} \hat{B}'_{kj,\sigma'\sigma} + \sum_{k\in *,\sigma'} \hat{P}^L_{ik,\sigma\sigma'} a^\dagger_{k\sigma'} + \sum_{j\in *,\sigma'} \hat{Q}'^L_{ij,\sigma\sigma'} a_{j\sigma'}\end{aligned}$$

7.1.4 Sum MPO Formalism in Unrestricted Spatial Orbitals

Hamiltonian

The quantum chemistry Hamiltonian is written as follows

$$\hat{H} = \sum_{ij,\sigma} t_{ij,\sigma}\ a^\dagger_{i\sigma} a_{j\sigma} + \frac{1}{2} \sum_{ijkl,\sigma\sigma'} v_{ijkl,\sigma\sigma'}\ a^\dagger_{i\sigma} a^\dagger_{k\sigma'} a_{l\sigma'} a_{j\sigma}$$

where

$$\begin{aligned}t_{ij,\sigma} = t_{(ij),\sigma} &= \int \mathrm{d}\mathbf{x}\ \phi^*_{i\sigma}(\mathbf{x}) \left(-\frac{1}{2}\nabla^2 - \sum_a \frac{Z_a}{r_a}\right) \phi_{j\sigma}(\mathbf{x}) \\ v_{ijkl,\sigma\sigma'} = v_{(ij)(kl),\sigma\sigma'} = v_{(kl)(ij),\sigma\sigma'} &= \int \mathrm{d}\mathbf{x}_1 \mathrm{d}\mathbf{x}_2\ \frac{\phi^*_{i\sigma}(\mathbf{x}_1)\phi^*_{k\sigma'}(\mathbf{x}_2)\phi_{l\sigma'}(\mathbf{x}_2)\phi_{j\sigma}(\mathbf{x}_1)}{r_{12}}\end{aligned}$$

Note that here the order of $ijkl$ is the same as that in `FCIDUMP` (chemist's notation $[ij|kl]$).

Derivation

Sum of MPO

$$\hat{H} = \sum_{m\sigma} a^\dagger_{m\sigma} \hat{H}_{m\sigma} = \sum_{m\sigma} a^\dagger_{m\sigma} \left[\sum_j t_{mj,\sigma}\ a_{j\sigma} + \frac{1}{2} \sum_{jkl,\sigma'} v_{mjkl,\sigma\sigma'}\ a^\dagger_{k\sigma'} a_{l\sigma'} a_{j\sigma}\right]$$

Now consider LR partition. There are 8 possibilities: $LLL, LRR, RLR, RRL, LLR, LRL, RLL, RRR$.

$$\begin{aligned}
\hat{H}_{m\sigma} =& \left[\sum_{j\in L} t_{mj,\sigma}\ a_{j\sigma} + \frac{1}{2}\sum_{jkl\in L,\sigma'} v_{mjkl,\sigma\sigma'}\ a^\dagger_{k\sigma'}a_{l\sigma'}a_{j\sigma}\right] + \left[\sum_{j\in R} t_{mj,\sigma}\ a_{j\sigma} + \frac{1}{2}\sum_{jkl\in R,\sigma'} v_{mjkl,\sigma\sigma'}\ a^\dagger_{k\sigma'}a_{l\sigma'}a_{j\sigma}\right] \\
&+ \left[\frac{1}{2}\sum_{j\in L} a_{j\sigma}\sum_{kl\in R,\sigma'} v_{mjkl,\sigma\sigma'}\ a^\dagger_{k\sigma'}a_{l\sigma'} + \frac{1}{2}\sum_{k\in L,\sigma'} a^\dagger_{k\sigma'}\sum_{jl\in R} v_{mjkl,\sigma\sigma'}\ a_{l\sigma'}a_{j\sigma} - \frac{1}{2}\sum_{l\in L,\sigma'} a_{l\sigma'}\sum_{jk\in R} v_{mjkl,\sigma\sigma'}\ a^\dagger_{k\sigma'}a_{j\sigma}\right] \\
&+ \left[\frac{1}{2}\sum_{j\in R}\left(\sum_{kl\in L,\sigma'} v_{mjkl,\sigma\sigma'}\ a^\dagger_{k\sigma'}a_{l\sigma'}\right)a_{j\sigma} + \frac{1}{2}\sum_{k\in R,\sigma'}\left(\sum_{jl\in L} v_{mjkl,\sigma\sigma'}\ a_{l\sigma'}a_{j\sigma}\right)a^\dagger_{k\sigma'} - \frac{1}{2}\sum_{l\in R,\sigma'}\left(\sum_{jk\in L} v_{mjkl,\sigma\sigma'}\ a^\dagger_{k}\right.\right.
\end{aligned}$$

Let

$$\begin{aligned}
\hat{H}^{L/R}_{m\sigma} =& \sum_{j\in L/R} t_{mj,\sigma}\ a_{j\sigma} + \frac{1}{2}\sum_{jkl\in L/R,\sigma'} v_{mjkl,\sigma\sigma'}\ a^\dagger_{k\sigma'}a_{l\sigma'}a_{j\sigma} \\
\hat{P}^{L/R}_{ik,\sigma\sigma'} =& \sum_{jl\in L/R} v_{ijkl,\sigma\sigma'}a_{l\sigma'}a_{j\sigma}, \\
\hat{Q}^{L/R}_{ij,\sigma} =& \sum_{kl\in L/R,\sigma'} v_{ijkl,\sigma\sigma'}a^\dagger_{k\sigma'}a_{l\sigma'}, \\
\hat{Q}'^{L/R}_{il,\sigma\sigma'} =& \sum_{jk\in L/R} v_{ijkl,\sigma\sigma'}a^\dagger_{k\sigma'}a_{j\sigma} \\
\hat{Q}''^{R}_{ij,\sigma\sigma'} =& \delta_{\sigma\sigma'}\hat{Q}^R_{ij\sigma} - \hat{Q}'^R_{ij\sigma\sigma'}
\end{aligned}$$

we have

$$\begin{aligned}
\hat{H}_{m\sigma} &= \hat{H}^L_{m\sigma}\otimes\hat{1}^R + \hat{1}^L\otimes\hat{H}^R_{m\sigma} + \frac{1}{2}\sum_{j\in L} a_{j\sigma}\hat{Q}^R_{mj,\sigma} + \frac{1}{2}\sum_{k\in L,\sigma'} a^\dagger_{k\sigma'}\hat{P}^R_{mk,\sigma\sigma'} - \frac{1}{2}\sum_{l\in L,\sigma'} a_{l\sigma'}\hat{Q}'^R_{ml,\sigma\sigma'} + \frac{1}{2}\sum_{j\in R}\hat{Q}^L_{mj,\sigma}a_{j\sigma} + \frac{1}{2} \\
&= \hat{H}^L_{m\sigma}\otimes\hat{1}^R + \hat{1}^L\otimes\hat{H}^R_{m\sigma} + \frac{1}{2}\sum_{k\in L,\sigma'} a^\dagger_{k\sigma'}\hat{P}^R_{mk,\sigma\sigma'} + \frac{1}{2}\sum_{j\in L,\sigma'} a_{j\sigma'}\left(\delta_{\sigma\sigma'}\hat{Q}^R_{mj,\sigma} - \hat{Q}'^R_{mj,\sigma\sigma'}\right) + \frac{1}{2}\sum_{k\in R,\sigma'}\hat{P}^L_{mk,\sigma\sigma'}a^\dagger_{k\sigma'} + \\
&= \hat{H}^L_{m\sigma}\otimes\hat{1}^R + \hat{1}^L\otimes\hat{H}^R_{m\sigma} + \frac{1}{2}\sum_{k\in L,\sigma'} a^\dagger_{k\sigma'}\hat{P}^R_{mk,\sigma\sigma'} + \frac{1}{2}\sum_{j\in L,\sigma'} a_{j\sigma'}\hat{Q}''^R_{mj,\sigma\sigma'} + \frac{1}{2}\sum_{k\in R,\sigma'}\hat{P}^L_{mk,\sigma\sigma'}a^\dagger_{k\sigma'} + \frac{1}{2}\sum_{j\in R,\sigma'}\hat{Q}''^L_{mj,\sigma\sigma}
\end{aligned}$$

Now consider $m \in L$ or $m \in R$. For $m \in L$:

$$\begin{aligned}
\sum_{m\in L,\sigma} a^\dagger_{m\sigma}\hat{H}_{m\sigma} =& \left(\sum_{m\in L,\sigma} a^\dagger_{m\sigma}\hat{H}^L_{m\sigma}\right)\otimes\hat{1}^R + \sum_{m\in L,\sigma} a^\dagger_{m\sigma}\otimes\hat{H}^R_{m\sigma} \\
&+ \frac{1}{2}\sum_{mk\in L,\sigma\sigma'} a^\dagger_{m\sigma}a^\dagger_{k\sigma'}\hat{P}^R_{mk,\sigma\sigma'} + \frac{1}{2}\sum_{mj\in L,\sigma\sigma'} a^\dagger_{m\sigma}a_{j\sigma'}\hat{Q}''^R_{mj,\sigma\sigma'} + \frac{1}{2}\sum_{k\in R,\sigma'}\left(\sum_{m\in L,\sigma} a^\dagger_{m\sigma}\hat{P}^L_{mk,\sigma\sigma'}\right)a^\dagger_{k\sigma'} + \frac{1}{2} \\
&= \hat{H}^{ML}\otimes\hat{1}^R + \sum_{m\in L,\sigma} a^\dagger_{m\sigma}\otimes\hat{H}^R_{m\sigma} + \frac{1}{2}\sum_{mk\in L,\sigma\sigma'}\hat{A}_{mk,\sigma\sigma'}\hat{P}^R_{mk,\sigma\sigma'} + \frac{1}{2}\sum_{mj\in L,\sigma\sigma'}\hat{B}_{mj,\sigma\sigma'}\hat{Q}''^R_{mj,\sigma\sigma'} + \frac{1}{2}\sum_{k\in R,}
\end{aligned}$$

where

$$
\begin{aligned}
\hat{A}_{ik,\sigma\sigma'} &= a^\dagger_{i\sigma} a^\dagger_{k\sigma'}, \\
\hat{B}_{il,\sigma\sigma'} &= a^\dagger_{i\sigma} a_{l\sigma'}, \\
\hat{H}^{ML/R} &= \sum_{m\in L/R,\sigma} a^\dagger_{m\sigma} \hat{H}^{L/R}_{m\sigma} \\
\hat{P}^{ML/R}_{k\sigma'} &= \sum_{m\in L/R,\sigma} a^\dagger_{m\sigma} \hat{P}^{L/R}_{mk,\sigma\sigma'} \\
\hat{Q}^{ML/R}_{j\sigma'} &= \sum_{m\in L/R,\sigma} a^\dagger_{m\sigma} \hat{Q}''^{L/R}_{mj,\sigma\sigma'}
\end{aligned}
$$

For $m \in R$:

$$
\begin{aligned}
\sum_{m\in R,\sigma} a^\dagger_{m\sigma} \hat{H}_{m\sigma} =& -\sum_{m\in R,\sigma} \hat{H}^L_{m\sigma} \otimes a^\dagger_{m\sigma} + \hat{1}^L \otimes \left(\sum_{m\in R,\sigma} a^\dagger_{m\sigma} \hat{H}^R_{m\sigma} \right) \\
& - \frac{1}{2} \sum_{k\in L,\sigma'} a^\dagger_{k\sigma'} \left(\sum_{m\in R,\sigma} a^\dagger_{m\sigma} \hat{P}^R_{mk,\sigma\sigma'} \right) - \frac{1}{2} \sum_{j\in L,\sigma'} a_{j\sigma'} \left(\sum_{m\in R,\sigma} a^\dagger_{m\sigma} \hat{Q}''^R_{mj,\sigma\sigma'} \right) + \frac{1}{2} \sum_{mk\in R,\sigma\sigma'} \hat{P}^L_{mk,\sigma\sigma'} a^\dagger_{m\sigma} a \\
=& -\sum_{m\in R,\sigma} \hat{H}^L_{m\sigma} \otimes a^\dagger_{m\sigma} + \hat{1}^L \otimes \hat{H}^{MR} - \frac{1}{2} \sum_{k\in L,\sigma'} a^\dagger_{k\sigma'} \hat{P}^{MR}_{k,\sigma'} - \frac{1}{2} \sum_{j\in L,\sigma'} a_{j\sigma'} \hat{Q}^{MR}_{j,\sigma'} + \frac{1}{2} \sum_{mk\in R,\sigma\sigma'} \hat{P}^L_{mk,\sigma\sigma'} \hat{A}
\end{aligned}
$$

In summary

$$
\begin{aligned}
\hat{H} =& \hat{H}^{ML} \otimes \hat{1}^R + \sum_{m\in L,\sigma} a^\dagger_{m\sigma} \otimes \hat{H}^R_{m\sigma} + \frac{1}{2} \sum_{mj\in L,\sigma\sigma'} \hat{A}_{mj,\sigma\sigma'} \hat{P}^R_{mj,\sigma\sigma'} + \frac{1}{2} \sum_{mj\in L,\sigma\sigma'} \hat{B}_{mj,\sigma\sigma'} \hat{Q}''^R_{mj,\sigma\sigma'} + \frac{1}{2} \sum_{k\in R,\sigma'} \hat{P}^{ML}_{k\sigma'} a^\dagger_{k\sigma'} + \\
& - \sum_{n\in R,\sigma} \hat{H}^L_{n\sigma} \otimes a^\dagger_{n\sigma} + \hat{1}^L \otimes \hat{H}^{MR} - \frac{1}{2} \sum_{j\in L,\sigma'} a^\dagger_{j\sigma'} \hat{P}^{MR}_{j,\sigma'} - \frac{1}{2} \sum_{j\in L,\sigma'} a_{j\sigma'} \hat{Q}^{MR}_{j,\sigma'} + \frac{1}{2} \sum_{nk\in R,\sigma\sigma'} \hat{P}^L_{nk,\sigma\sigma'} \hat{A}_{nk,\sigma\sigma'} + \frac{1}{2} \sum_{nk\in R,\sigma\sigma'}
\end{aligned}
$$

The operators required in left block are

$$
\left\{ \hat{H}^{ML}, a^\dagger_{m\sigma}, \hat{A}_{mj,\sigma\sigma'}, \hat{B}_{mj,\sigma\sigma'}, \hat{P}^{ML}_{k\sigma'}, \hat{Q}^{ML}_{k\sigma'}, \hat{H}^L_{n\sigma}, \hat{1}^L, a^\dagger_{j\sigma'}, a_{j\sigma'}, \hat{P}^L_{nk,\sigma\sigma'}, \hat{Q}''^L_{nk,\sigma\sigma'} \right\} \quad (m, j \in L,\ n, k \in R)
$$

The total number of operators is

$$
\begin{aligned}
N &= 1 + 2K_{ML} + 4K_{ML}K_L + 4K_{ML}K_L + 2K_R + 2K_R + 2K_{MR} + 1 + 2K_L + 2K_L + 4K_{MR}K_R + 4K_{MR}K_R \\
&= 2 + 2K_M + 4K + 8K_{ML}K_L + 8K_{MR}K_R
\end{aligned}
$$

Reordered left and right block operators

$$
\begin{aligned}
L &= \left\{ \hat{H}^{ML}, \hat{1}^L, a^\dagger_{m\sigma}, \hat{H}^L_{n\sigma}, a^\dagger_{j\sigma'}, a_{j\sigma'}, \hat{P}^{ML}_{k\sigma'}, \hat{Q}^{ML}_{k\sigma'}, \hat{A}_{mj,\sigma\sigma'}, \hat{B}_{mj,\sigma\sigma'}, \hat{P}^L_{nk,\sigma\sigma'}, \hat{Q}''^L_{nk,\sigma\sigma'} \right\} \quad (m, j \in L,\ n, k \in R) \\
R &= \left\{ \hat{1}^R, \hat{H}^{MR}, \hat{H}^R_{m\sigma}, a^\dagger_{n\sigma}, \hat{P}^{MR}_{j,\sigma'}, \hat{Q}^{MR}_{j,\sigma'}, a^\dagger_{k\sigma'}, a_{k\sigma'}, \hat{P}^R_{mj,\sigma\sigma'}, \hat{Q}''^R_{mj,\sigma\sigma'}, \hat{A}_{nk,\sigma\sigma'}, \hat{B}_{nk,\sigma\sigma'} \right\}
\end{aligned}
$$

Now let

$$
\begin{aligned}
\hat{R}^{ML/R}_{k\sigma} &= -2\delta(k \in M) \hat{H}^{L/R}_{k\sigma} + \hat{P}^{ML/R}_{k\sigma'} \\
\hat{S}^{ML/R}_{k\sigma} &= \hat{Q}^{ML/R}_{k\sigma'}
\end{aligned}
$$

we have

$$L = \left\{\hat{H}^{ML}, \hat{1}^L, a^\dagger_{j\sigma'}, a_{j\sigma'}, \hat{R}^{ML}_{k\sigma'}, \hat{S}^{ML}_{k\sigma'}, \hat{A}_{mj,\sigma\sigma'}, \hat{B}_{mj,\sigma\sigma'}, \hat{P}^L_{nk,\sigma\sigma'}, \hat{Q}''^L_{nk,\sigma\sigma'}\right\} \quad (m, j \in L,\ n, k \in R)$$
$$R = \left\{\hat{1}^R, \hat{H}^{MR}, \hat{R}^{MR}_{j,\sigma'}, \hat{S}^{MR}_{j,\sigma'}, a^\dagger_{k\sigma'}, a_{k\sigma'}, \hat{P}^R_{mj,\sigma\sigma'}, \hat{Q}''^R_{mj,\sigma\sigma'}, \hat{A}_{nk,\sigma\sigma'}, \hat{B}_{nk,\sigma\sigma'}\right\}$$

The total number of operators is

$$N = 2 + 4K + 8K_{ML}K_L + 8K_{MR}K_R$$

Blocking

$$\begin{aligned}
\hat{P}^{ML*}_{k\sigma'} &= \sum_{m\in L*,\sigma} a^\dagger_{m\sigma}\hat{P}^{L*}_{mk,\sigma\sigma'} = \sum_{m\in L*,\sigma} a^\dagger_{m\sigma}\sum_{jl\in L*} v_{mjkl,\sigma\sigma'}a_{l\sigma'}a_{j\sigma} \\
&= \hat{P}^{ML}_{k\sigma'}\otimes\hat{1}^* + \hat{1}^L\otimes\hat{P}^{M*}_{k\sigma'} + \sum_{m\in *,\sigma} a^\dagger_{m\sigma}\hat{P}^L_{mk,\sigma\sigma'} + \sum_{m\in L,\sigma} a^\dagger_{m\sigma}\sum_{j\in *,l\in L} v_{mjkl,\sigma\sigma'}a_{l\sigma'}a_{j\sigma} + \sum_{m\in L,\sigma} a^\dagger_{m\sigma}\sum_{j\in L,l\in *} v_{mjkl,\sigma\sigma} \\
&\quad + \sum_{m\in L,\sigma} a^\dagger_{m\sigma}\hat{P}^*_{mk,\sigma\sigma'} + \sum_{m\in *,\sigma} a^\dagger_{m\sigma}\sum_{j\in *,l\in L} v_{mjkl,\sigma\sigma'}a_{l\sigma'}a_{j\sigma} + \sum_{m\in *,\sigma} a^\dagger_{m\sigma}\sum_{j\in L,l\in *} v_{mjkl,\sigma\sigma'}a_{l\sigma'}a_{j\sigma} \\
&= \hat{P}^{ML}_{k\sigma'}\otimes\hat{1}^* + \hat{1}^L\otimes\hat{P}^{M*}_{k\sigma'} + \sum_{m\in *,\sigma}\hat{P}^L_{mk,\sigma\sigma'}a^\dagger_{m\sigma} + \sum_{m\in L,\sigma} a^\dagger_{m\sigma}\hat{P}^*_{mk,\sigma\sigma'} + \sum_{ml\in L,j\in *,\sigma} v_{mjkl,\sigma\sigma'}a^\dagger_{m\sigma}a_{l\sigma'}a_{j\sigma} - \sum_{mj\in L,l\in *} \\
&\quad - \sum_{mj\in *,l\in L,\sigma} v_{mjkl,\sigma\sigma'}a_{l\sigma'}a^\dagger_{m\sigma}a_{j\sigma} + \sum_{ml\in *,j\in L,\sigma} v_{mjkl,\sigma\sigma'}a_{j\sigma}a^\dagger_{m\sigma}a_{l\sigma'} \\
&= \hat{P}^{ML}_{k\sigma'}\otimes\hat{1}^* + \hat{1}^L\otimes\hat{P}^{M*}_{k\sigma'} + \sum_{m\in *,\sigma}\hat{P}^L_{mk,\sigma\sigma'}a^\dagger_{m\sigma} + \sum_{m\in L,\sigma} a^\dagger_{m\sigma}\hat{P}^*_{mk,\sigma\sigma'} + \sum_{ml\in L,j\in *,\sigma} v_{mjkl,\sigma\sigma'}a^\dagger_{m\sigma}a_{l\sigma'}a_{j\sigma} - \sum_{ml\in L,j\in *} \\
&\quad - \sum_{mj\in *,l\in L,\sigma} v_{mjkl,\sigma\sigma'}a_{l\sigma'}a^\dagger_{m\sigma}a_{j\sigma} + \sum_{mj\in *,l\in L,\sigma} v_{mlkj,\sigma\sigma'}a_{l\sigma}a^\dagger_{m\sigma}a_{j\sigma'} \\
&= \hat{P}^{ML}_{k\sigma'}\otimes\hat{1}^* + \hat{1}^L\otimes\hat{P}^{M*}_{k\sigma'} + \sum_{m\in *,\sigma}\hat{P}^L_{mk,\sigma\sigma'}a^\dagger_{m\sigma} + \sum_{m\in L,\sigma} a^\dagger_{m\sigma}\hat{P}^*_{mk,\sigma\sigma'} \\
&\quad + \sum_{ml\in L,j\in *,\sigma} v_{mjkl,\sigma\sigma'}\hat{B}_{ml,\sigma\sigma'}a_{j\sigma} - \sum_{ml\in L,j\in *,\sigma} v_{mlkj,\sigma\sigma'}\hat{B}_{ml,\sigma\sigma}a_{j\sigma'} + \sum_{mj\in *,l\in L,\sigma} v_{mlkj,\sigma\sigma'}a_{l\sigma}\hat{B}_{mj,\sigma\sigma'} - \sum_{mj\in *,l\in L}
\end{aligned}$$

and

$$\begin{aligned}
\hat{Q}^{ML*}_{j\sigma'} &= \sum_{m\in L*,\sigma} a^\dagger_{m\sigma} \hat{Q}''^{L*}_{mj,\sigma\sigma'} = \sum_{m\in L*,\sigma} a^\dagger_{m\sigma} \sum_{kl\in L*} \left(\delta_{\sigma\sigma'} \sum_{\sigma''} v_{mjkl,\sigma\sigma''} a^\dagger_{k\sigma''} a_{l\sigma''} - v_{mlkj,\sigma\sigma'} a^\dagger_{k\sigma'} a_{l\sigma} \right) \\
&= \hat{Q}^{ML}_{j\sigma'} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{Q}^{M*}_{j\sigma'} + \sum_{m\in *,\sigma} a^\dagger_{m\sigma} \hat{Q}''^{L}_{mj,\sigma\sigma'} + \sum_{m\in L,\sigma} a^\dagger_{m\sigma} \hat{Q}''^{*}_{mj,\sigma\sigma'} \\
&\quad + \sum_{m\in L,\sigma} a^\dagger_{m\sigma} \sum_{k\in *,l\in L} \left(\delta_{\sigma\sigma'} \sum_{\sigma''} v_{mjkl,\sigma\sigma''} a^\dagger_{k\sigma''} a_{l\sigma''} - v_{mlkj,\sigma\sigma'} a^\dagger_{k\sigma'} a_{l\sigma} \right) + \sum_{m\in L,\sigma} a^\dagger_{m\sigma} \sum_{k\in L,l\in *} \left(\delta_{\sigma\sigma'} \sum_{\sigma''} v_{mjkl,\sigma\sigma'} \right. \\
&\quad + \sum_{m\in *,\sigma} a^\dagger_{m\sigma} \sum_{k\in *,l\in L} \left(\delta_{\sigma\sigma'} \sum_{\sigma''} v_{mjkl,\sigma\sigma''} a^\dagger_{k\sigma''} a_{l\sigma''} - v_{mlkj,\sigma\sigma'} a^\dagger_{k\sigma'} a_{l\sigma} \right) + \sum_{m\in *,\sigma} a^\dagger_{m\sigma} \sum_{k\in L,l\in *} \left(\delta_{\sigma\sigma'} \sum_{\sigma''} v_{mjkl,\sigma\sigma''} \right. \\
&= \hat{Q}^{ML}_{j\sigma'} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{Q}^{M*}_{j\sigma'} + \sum_{m\in *,\sigma} a^\dagger_{m\sigma} \hat{Q}''^{L}_{mj,\sigma\sigma'} + \sum_{m\in L,\sigma} a^\dagger_{m\sigma} \hat{Q}''^{*}_{mj,\sigma\sigma'} \\
&\quad + \sum_{ml\in L,k\in *,\sigma} \left(\delta_{\sigma\sigma'} \sum_{\sigma''} v_{mjkl,\sigma\sigma''} a^\dagger_{m\sigma} a^\dagger_{k\sigma''} a_{l\sigma''} - v_{mlkj,\sigma\sigma'} a^\dagger_{m\sigma} a^\dagger_{k\sigma'} a_{l\sigma} \right) + \sum_{mk\in L,l\in *,\sigma} \left(\delta_{\sigma\sigma'} \sum_{\sigma''} v_{mjkl,\sigma\sigma''} a^\dagger_{m\sigma} a^\dagger_{k} \right. \\
&\quad + \sum_{mk\in *,l\in L,\sigma} \left(\delta_{\sigma\sigma'} \sum_{\sigma''} v_{mjkl,\sigma\sigma''} a^\dagger_{m\sigma} a^\dagger_{k\sigma''} a_{l\sigma''} - v_{mlkj,\sigma\sigma'} a^\dagger_{m\sigma} a^\dagger_{k\sigma'} a_{l\sigma} \right) + \sum_{ml\in *,k\in L,\sigma} \left(\delta_{\sigma\sigma'} \sum_{\sigma''} v_{mjkl,\sigma\sigma''} a^\dagger_{m\sigma} a^\dagger_{k} \right. \\
&= \hat{Q}^{ML}_{j\sigma'} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{Q}^{M*}_{j\sigma'} + \sum_{m\in *,\sigma} a^\dagger_{m\sigma} \hat{Q}''^{L}_{mj,\sigma\sigma'} + \sum_{m\in L,\sigma} a^\dagger_{m\sigma} \hat{Q}''^{*}_{mj,\sigma\sigma'} \\
&\quad - \sum_{ml\in L,k\in *,\sigma} \left(\delta_{\sigma\sigma'} \sum_{\sigma''} v_{mjkl,\sigma\sigma''} a^\dagger_{m\sigma} a_{l\sigma''} a^\dagger_{k\sigma''} - v_{mlkj,\sigma\sigma'} a^\dagger_{m\sigma} a_{l\sigma} a^\dagger_{k\sigma'} \right) + \sum_{mk\in L,l\in *,\sigma} \left(\delta_{\sigma\sigma'} \sum_{\sigma''} v_{mjkl,\sigma\sigma''} a^\dagger_{m\sigma} a^\dagger_{k} \right. \\
&\quad + \sum_{mk\in *,l\in L,\sigma} \left(\delta_{\sigma\sigma'} \sum_{\sigma''} v_{mjkl,\sigma\sigma''} a_{l\sigma''} a^\dagger_{m\sigma} a^\dagger_{k\sigma''} - v_{mlkj,\sigma\sigma'} a_{l\sigma} a^\dagger_{m\sigma} a^\dagger_{k\sigma'} \right) - \sum_{ml\in *,k\in L,\sigma} \left(\delta_{\sigma\sigma'} \sum_{\sigma''} v_{mjkl,\sigma\sigma''} a^\dagger_{k\sigma''} a^\dagger_{} \right.
\end{aligned}$$

and

$$\begin{aligned}
&= \hat{Q}^{ML}_{j\sigma'} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{Q}^{M*}_{j\sigma'} + \sum_{m\in *,\sigma} a^\dagger_{m\sigma} \hat{Q}''^{L}_{mj,\sigma\sigma'} + \sum_{m\in L,\sigma} a^\dagger_{m\sigma} \hat{Q}''^{*}_{mj,\sigma\sigma'} \\
&\quad - \sum_{ml\in L,k\in *,\sigma} \left(\delta_{\sigma\sigma'} \sum_{\sigma''} v_{mjkl,\sigma\sigma''} \hat{B}_{ml\sigma\sigma''} a^\dagger_{k\sigma''} - v_{mlkj,\sigma\sigma'} \hat{B}_{ml\sigma\sigma} a^\dagger_{k\sigma'} \right) + \sum_{mk\in L,l\in *,\sigma} \left(\delta_{\sigma\sigma'} \sum_{\sigma''} v_{mjkl,\sigma\sigma''} \hat{A}_{mk\sigma\sigma''} a \right. \\
&\quad + \sum_{mk\in *,l\in L,\sigma} \left(\delta_{\sigma\sigma'} \sum_{\sigma''} v_{mjkl,\sigma\sigma''} a_{l\sigma''} \hat{A}_{mk\sigma\sigma''} - v_{mlkj,\sigma\sigma'} a_{l\sigma} \hat{A}_{mk\sigma\sigma'} \right) - \sum_{ml\in *,k\in L,\sigma} \left(\delta_{\sigma\sigma'} \sum_{\sigma''} v_{mjkl,\sigma\sigma''} a^\dagger_{k\sigma''} \hat{B}_{ml} \right.
\end{aligned}$$

after simplification

$$= \hat{Q}^{ML}_{j\sigma'} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{Q}^{M*}_{j\sigma'} + \sum_{m\in *,\sigma} a^\dagger_{m\sigma} \hat{Q}''^L_{mj,\sigma\sigma'} + \sum_{m\in L,\sigma} a^\dagger_{m\sigma} \hat{Q}''^*_{mj,\sigma\sigma'}$$
$$- \sum_{ml\in L,k\in *,\sigma} \left(v_{mjkl,\sigma'\sigma} \hat{B}_{ml\sigma'\sigma} a^\dagger_{k\sigma} - v_{mlkj,\sigma\sigma'} \hat{B}_{ml\sigma\sigma} a^\dagger_{k\sigma'} \right) + \sum_{ml\in L,k\in *,\sigma} \left(v_{mjlk,\sigma'\sigma} \hat{A}_{ml\sigma'\sigma} a_{k\sigma} - v_{mklj,\sigma\sigma'} \hat{A}_{ml\sigma\sigma'} \right.$$
$$+ \sum_{mk\in *,l\in L,\sigma} \left(v_{mjkl,\sigma'\sigma} a_{l\sigma} \hat{A}_{mk\sigma'\sigma} - v_{mlkj,\sigma\sigma'} a_{l\sigma} \hat{A}_{mk\sigma\sigma'} \right) - \sum_{mk\in *,l\in L,\sigma} \left(v_{mjlk,\sigma'\sigma} a^\dagger_{l\sigma} \hat{B}_{mk\sigma'\sigma} - v_{mklj,\sigma\sigma'} a^\dagger_{l\sigma'} \hat{B}_m \right.$$

For P, Q, we have

$$\hat{P}^{L*}_{ik,\sigma\sigma'} = \sum_{jl\in L*} v_{ijkl,\sigma\sigma'} a_{l\sigma'} a_{j\sigma} = \hat{P}^L_{ik,\sigma\sigma'} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{P}^*_{ik,\sigma\sigma'} + \sum_{j\in L,l\in *} v_{ijkl,\sigma\sigma'} a_{l\sigma'} a_{j\sigma} + \sum_{j\in *,l\in L} v_{ijkl,\sigma\sigma'} a_{l\sigma'} a_{j\sigma}$$
$$= \hat{P}^L_{ik,\sigma\sigma'} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{P}^*_{ik,\sigma\sigma'} - \sum_{j\in L,l\in *} v_{ijkl,\sigma\sigma'} a_{j\sigma} a_{l\sigma'} + \sum_{j\in L,l\in *} v_{ilkj,\sigma\sigma'} a_{j\sigma'} a_{l\sigma}$$
$$\hat{Q}''^{L*}_{ij,\sigma\sigma'} = \delta_{\sigma\sigma'} \hat{Q}^{L*}_{ij\sigma} - \hat{Q}'^{L*}_{ij\sigma\sigma'} = \delta_{\sigma\sigma'} \sum_{kl\in L*,\sigma''} v_{ijkl,\sigma\sigma''} a^\dagger_{k\sigma''} a_{l\sigma''} - \sum_{kl\in L*} v_{ilkj,\sigma\sigma'} a^\dagger_{k\sigma'} a_{l\sigma}$$
$$= \hat{Q}''^L_{ij,\sigma\sigma'} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{Q}''^*_{ij,\sigma\sigma'} + \delta_{\sigma\sigma'} \sum_{k\in L,l\in *,\sigma''} v_{ijkl,\sigma\sigma''} a^\dagger_{k\sigma''} a_{l\sigma''} - \sum_{k\in L,l\in *} v_{ilkj,\sigma\sigma'} a^\dagger_{k\sigma'} a_{l\sigma} + \delta_{\sigma\sigma'} \sum_{k\in *,l\in L,\sigma''} v_{ijkl,\sigma}$$
$$= \hat{Q}''^L_{ij,\sigma\sigma'} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{Q}''^*_{ij,\sigma\sigma'} + \delta_{\sigma\sigma'} \sum_{k\in L,l\in *,\sigma''} v_{ijkl,\sigma\sigma''} a^\dagger_{k\sigma''} a_{l\sigma''} - \sum_{k\in L,l\in *} v_{ilkj,\sigma\sigma'} a^\dagger_{k\sigma'} a_{l\sigma} - \delta_{\sigma\sigma'} \sum_{k\in L,l\in *,\sigma''} v_{ijlk,\sigma}$$

7.1.5 DMRG Quantum Chemistry Hamiltonian in Spin Orbitals

Hamiltonian

The quantum chemistry Hamiltonian is written as follows

$$\hat{H} = \sum_{ij} t_{ij}\ a^\dagger_i a_j + \frac{1}{2} \sum_{ijkl} v_{ijkl}\ a^\dagger_i a^\dagger_k a_l a_j$$

where $ijkl$ are spin orbital indices, and

$$t_{ij} = \int \mathrm{d}\mathbf{x}\ \phi^*_i(\mathbf{x}) \left(-\frac{1}{2}\nabla^2 - \sum_a \frac{Z_a}{r_a} \right) \phi_j(\mathbf{x})$$
$$v_{ijkl} = \int \mathrm{d}\mathbf{x}_1 \mathrm{d}\mathbf{x}_2\ \frac{\phi^*_i(\mathbf{x}_1)\phi^*_k(\mathbf{x}_2)\phi_l(\mathbf{x}_2)\phi_j(\mathbf{x}_1)}{r_{12}}$$

Note that here the order of $ijkl$ is the same as that in `FCIDUMP` (chemist's notation $[ij|kl]$).

When spin index is given, we have

$$t_{i\sigma,j\tau} = t_{ij}\delta_{\sigma\tau}$$
$$v_{i\sigma,j\tau,k\mu,l\nu} = v_{ijkl}\delta_{\sigma\tau}\delta_{\mu\nu}$$

For complex orbitals, we have

$$t_{ij} = t^*_{ji}$$
$$v_{ijkl} = v_{klij} = v^*_{jilk} = v^*_{lkji}$$

For real orbitals, we have

$$
\begin{aligned}
t_{ij} &= t_{ji} \\
v_{ijkl} &= v_{klij} = v_{jilk} = v_{lkji} = v_{jikl} = v_{ijlk} = v_{lkij} = v_{klji}
\end{aligned}
$$

Partitioning in Spin Orbitals

The partitioning of Hamiltonian in left (L) and right (R) blocks is given by

$$
\hat{H} = \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R + \Big(\sum_{i \in L} a_i^\dagger \hat{R}_i^R + h.c. + \sum_{i \in R} a_i^\dagger \hat{R}_i^L + h.c. \Big) + \frac{1}{2} \Big(\sum_{ik \in L} \hat{A}_{ik}^L \hat{P}_{ik}^R + h.c. \Big) + \sum_{ij \in L} \hat{B}_{ij}^L \hat{Q}_{ij}^R
$$

where the normal and complementary operators are defined by

$$
\begin{aligned}
\hat{R}_i^{L/R} &= \frac{1}{2} \sum_{j \in L/R} t_{ij} a_j + \sum_{jkl \in L/R} v_{ijkl} a_k^\dagger a_l a_j, \\
\hat{A}_{ik} &= a_i^\dagger a_k^\dagger, \\
\hat{B}_{ij} &= a_i^\dagger a_j, \\
\hat{P}_{ik}^R &= \sum_{jl \in R} v_{ijkl} a_l a_j, \\
\hat{Q}_{ij}^R &= \sum_{kl \in R} (v_{ijkl} - v_{ilkj}) a_k^\dagger a_l
\end{aligned}
$$

Note that we need to move all on-site interaction into local Hamiltonian, so that when construction interaction terms in Hamiltonian, operators anticommute (without giving extra constant terms).

Derivation

First consider one-electron term. ij indices have only two possibilities: i left, j right, or i right, j left. Index i must be associated with creation operator. So the second case is the Hermitian conjugate of the first case. Namely, consider $\hat{S}_i^{L/R}$ as the one-body part of $\hat{R}_i^{L/R}$, we have

$$
\begin{aligned}
& \Big(\sum_{i \in L} a_i^\dagger \hat{S}_i^R + h.c. + \sum_{i \in R} a_i^\dagger \hat{S}_i^L + h.c. \Big) \\
=& \Big(\sum_{i \in L} a_i^\dagger \hat{S}_i^R + \sum_{i \in L} \hat{S}_i^{R\dagger} a_i + \sum_{i \in R} a_i^\dagger \hat{S}_i^L + \sum_{i \in R} \hat{S}_i^{L\dagger} a_i \Big) \\
=& \frac{1}{2} \Big(\sum_{i \in L, j \in R} t_{ij} a_i^\dagger a_j + \sum_{i \in L, j \in R} t_{ij}^* a_j^\dagger a_i + \sum_{i \in R, j \in L} t_{ij} a_i^\dagger a_j + \sum_{i \in R, j \in L} t_{ij}^* a_j^\dagger a_i \Big)
\end{aligned}
$$

Using $t_{ij}^* = t_{ji}$ and swap the indices ij we have

$$
\begin{aligned}
\cdots &= \frac{1}{2} \Big(\sum_{i \in L, j \in R} t_{ij} a_i^\dagger a_j + \sum_{i \in R, j \in L} t_{ij} a_i^\dagger a_j + \sum_{i \in R, j \in L} t_{ij} a_i^\dagger a_j + \sum_{i \in L, j \in R} t_{ij} a_i^\dagger a_j \Big) \\
&= \sum_{i \in L, j \in R} t_{ij} a_i^\dagger a_j + \sum_{i \in R, j \in L} t_{ij} a_i^\dagger a_j
\end{aligned}
$$

Next consider one of $ijkl$ in left, and three of them in right. These terms are

$$\begin{aligned}\hat{H}_{1L,3R} &= \frac{1}{2}\sum_{i\in L,jkl\in R} v_{ijkl}a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2}\sum_{j\in L,ikl\in R} v_{ijkl}a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2}\sum_{k\in L,ijl\in R} v_{ijkl}a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2}\sum_{l\in L,ijk\in R} v_{ijkl}a_i^\dagger a_k^\dagger a_l a_j \\ &= \left[\frac{1}{2}\sum_{i\in L,jkl\in R} v_{ijkl}a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2}\sum_{k\in L,ijl\in R} v_{ijkl}a_i^\dagger a_k^\dagger a_l a_j\right] + \frac{1}{2}\sum_{j\in L,ikl\in R} v_{ijkl}a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2}\sum_{l\in L,ijk\in R} v_{ijkl}a_i^\dagger a_k^\dagger a_l a\end{aligned}$$

where the terms in bracket equal to first and third terms in left-hand-side. Outside the bracket are second, forth terms.

The conjugate of third term in rhs is second term in rhs

$$\frac{1}{2}\sum_{j\in L,ikl\in R} v_{ijkl}^* a_j^\dagger a_l^\dagger a_k a_i = \frac{1}{2}\sum_{k\in L,ijl\in R} v_{lkji}^* a_k^\dagger a_i^\dagger a_j a_l = \frac{1}{2}\sum_{k\in L,ijl\in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j$$

The conjugate of forth term in rhs is first term in rhs

$$\frac{1}{2}\sum_{l\in L,ijk\in R} v_{ijkl}^* a_j^\dagger a_l^\dagger a_k a_i = \frac{1}{2}\sum_{i\in L,jkl\in R} v_{lkji}^* a_k^\dagger a_i^\dagger a_j a_l = \frac{1}{2}\sum_{i\in L,jkl\in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j$$

Therefore, using $v_{ijkl} = v_{klij}$

$$\begin{aligned}\hat{H}_{1L,3R} &= \left[\frac{1}{2}\sum_{i\in L,jkl\in R} v_{ijkl}a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2}\sum_{k\in L,ijl\in R} v_{ijkl}a_i^\dagger a_k^\dagger a_l a_j\right] + h.c. \\ &= \left[\frac{1}{2}\sum_{i\in L,jkl\in R} v_{ijkl}a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2}\sum_{k\in L,ijl\in R} v_{ijkl}a_k^\dagger a_i^\dagger a_j a_l\right] + h.c. \\ &= \left[\frac{1}{2}\sum_{i\in L,jkl\in R} v_{ijkl}a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2}\sum_{i\in L,jkl\in R} v_{klij}a_i^\dagger a_k^\dagger a_l a_j\right] + h.c. \\ &= \sum_{i\in L,jkl\in R} v_{ijkl}a_i^\dagger a_k^\dagger a_l a_j + h.c. \\ &= \sum_{i\in L} a_i^\dagger \sum_{jkl\in R} v_{ijkl}a_k^\dagger a_l a_j + h.c. = \sum_{i\in L} a_i^\dagger R_i^R + h.c.\end{aligned}$$

Next consider the two creation operators together in left or in together in right. There are two cases. The second case is the conjugate of the first case, namely,

$$\sum_{ik\in R,jl\in L} a_i^\dagger a_k^\dagger v_{ijkl} a_l a_j = \sum_{jl\in R,ik\in L} a_j^\dagger a_l^\dagger v_{jilk} a_k a_i = \sum_{ik\in L,jl\in R} v_{jilk} a_j^\dagger a_l^\dagger a_k a_i = \sum_{ik\in L,jl\in R} v_{ijkl}^* \left(a_i^\dagger a_k^\dagger a_l a_j\right)^\dagger$$

This explains the $\hat{A}\hat{P}$ term. The last situation is, one creation in left and one creation in right. Note that when exchange two elementary operators, one creation and one annihilation, one in left and one in right, they must anticommute.

$$\begin{aligned}\hat{H}_{2L,2R} &= \frac{1}{2}\sum_{il\in L,jk\in R} v_{ijkl}a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2}\sum_{ij\in L,kl\in R} v_{ijkl}a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2}\sum_{kl\in L,ij\in R} v_{ijkl}a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2}\sum_{jk\in L,il\in R} v_{ijkl}a_i^\dagger a_k^\dagger a_l a_j \\ &= -\frac{1}{2}\sum_{il\in L,jk\in R} v_{ijkl}a_i^\dagger a_l a_k^\dagger a_j + \frac{1}{2}\sum_{ij\in L,kl\in R} v_{ijkl}a_i^\dagger a_j a_k^\dagger a_l + \frac{1}{2}\sum_{kl\in L,ij\in R} v_{ijkl}a_i^\dagger a_j a_k^\dagger a_l - \frac{1}{2}\sum_{jk\in L,il\in R} v_{ijkl}a_i^\dagger a_l a_k^\dagger a\end{aligned}$$

First consider the second and third terms

$$\begin{aligned}
&\frac{1}{2}\sum_{ij\in L,kl\in R} v_{ijkl}a_i^\dagger a_j a_k^\dagger a_l + \frac{1}{2}\sum_{kl\in L,ij\in R} v_{ijkl}a_i^\dagger a_j a_k^\dagger a_l \\
=&\frac{1}{2}\sum_{ij\in L,kl\in R} v_{ijkl}a_i^\dagger a_j a_k^\dagger a_l + \frac{1}{2}\sum_{kl\in L,ij\in R} v_{ijkl}a_k^\dagger a_l a_i^\dagger a_j \\
=&\frac{1}{2}\sum_{ij\in L,kl\in R} v_{ijkl}a_i^\dagger a_j a_k^\dagger a_l + \frac{1}{2}\sum_{ij\in L,kl\in R} v_{klij}a_i^\dagger a_j a_k^\dagger a_l \\
=&\sum_{ij\in L,kl\in R} v_{ijkl}a_i^\dagger a_j a_k^\dagger a_l = \sum_{ij\in L} a_i^\dagger a_j \sum_{kl\in R} v_{ijkl}a_k^\dagger a_l = \sum_{ij\in L}\hat{B}_{ij}\hat{Q}^R_{ij\prime}
\end{aligned}$$

For the other two terms,

$$\begin{aligned}
&-\frac{1}{2}\sum_{il\in L,jk\in R} v_{ijkl}a_i^\dagger a_l a_k^\dagger a_j - \frac{1}{2}\sum_{jk\in L,il\in R} v_{ijkl}a_i^\dagger a_l a_k^\dagger a_j \\
=&-\frac{1}{2}\sum_{il\in L,jk\in R} v_{ijkl}a_i^\dagger a_l a_k^\dagger a_j - \frac{1}{2}\sum_{jk\in L,il\in R} v_{ijkl}a_k^\dagger a_j a_i^\dagger a_l \\
=&-\frac{1}{2}\sum_{il\in L,jk\in R} v_{ijkl}a_i^\dagger a_l a_k^\dagger a_j - \frac{1}{2}\sum_{il\in L,jk\in R} v_{klij}a_i^\dagger a_l a_k^\dagger a_j \\
=&-\sum_{il\in L,jk\in R} v_{ijkl}a_i^\dagger a_l a_k^\dagger a_j \\
=&-\sum_{il\in L} a_i^\dagger a_l \sum_{jk\in R} v_{ijkl}a_k^\dagger a_j = \sum_{il\in L}\hat{B}_{il}\hat{Q}^R_{il\prime\prime}
\end{aligned}$$

Then

$$\hat{Q}^R_{ij} = \hat{Q}^R_{ij\prime} + \hat{Q}^R_{ij\prime\prime} = \sum_{kl\in R}\big(v_{ijkl} - v_{ilkj}\big)a_k^\dagger a_l$$

Normal/Complementary Partitioning

The above version is used when left block is short in length. Note that all terms should be written in a way that operators for particles in left block should appear in the left side of operator string, and operators for particles in right block should appear in the right side of operator string. To write the Hermitian conjugate explicitly, we have

$$\begin{aligned}
\hat{H}^{NC} =& \hat{H}^L\otimes\hat{1}^R + \hat{1}^L\otimes\hat{H}^R \\
&+\sum_{i\in L}\left(a_i^\dagger\hat{R}_i^R - a_i\hat{R}_i^{R\dagger}\right) + \sum_{i\in R}\left(\hat{R}_i^{L\dagger}a_i - \hat{R}_i^L a_i^\dagger\right) \\
&+\frac{1}{2}\sum_{ik\in L}\left(\hat{A}_{ik}\hat{P}^R_{ik} + \hat{A}^\dagger_{ik}\hat{P}^{R\dagger}_{ik}\right) + \sum_{ij\in L}\hat{B}_{ij}\hat{Q}^R_{ij}
\end{aligned}$$

Note that no minus sign for Hermitian conjugate terms with A, P because these are not Fermion operators.

With this normal/complementary partitioning, the operators required in left block are

$$\left\{\hat{H}^L, \hat{1}^L, a_i^\dagger, a_i, \hat{R}_k^{L\dagger}, \hat{R}_k^L, \hat{A}_{ij}, \hat{A}_{ij}^\dagger, \hat{B}_{ij}\right\} \quad (i, j \in L,\ k \in R)$$

The operators required in right block are

$$\left\{\hat{1}^R, \hat{H}^R, \hat{R}_i^R, \hat{R}_i^{R\dagger}, a_k, a_k^\dagger, \hat{P}_{ij}^R, \hat{P}_{ij}^{R\dagger}, \hat{Q}_{ij}^R\right\} \quad (i, j \in L,\ k \in R)$$

Assuming that there are K sites in total, and K_L/K_R sites in left/right block (optimally, $K_L \le K_R$), the total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$N_{NC} = 1 + 1 + 2K_L + 2K_R + 2K_L^2 + K_L^2 = 3K_L^2 + 2K + 2$$

Complementary/Normal Partitioning

$$\begin{aligned}\hat{H}^{CN} = \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R + \sum_{i\in L}\left(a_i^\dagger \hat{R}_i^R - a_i \hat{R}_i^{R\dagger}\right) + \sum_{i\in R}\left(\hat{R}_i^{L\dagger} a_i - \hat{R}_i^L a_i^\dagger\right) \\ + \frac{1}{2}\sum_{jl\in R}\left(\hat{P}_{jl}^L \hat{A}_{jl} + \hat{P}_{jl}^{L\dagger}\hat{A}_{jl}^\dagger\right) + \sum_{kl\in R}\hat{Q}_{kl}^L \hat{B}_{kl}\end{aligned}$$

Now the operators required in left block are

$$\left\{\hat{H}^L, \hat{1}^L, a_i^\dagger, a_i, \hat{R}_k^{L\dagger}, \hat{R}_k^L, \hat{P}_{kl}^L, \hat{P}_{kl}^{L\dagger}, \hat{Q}_{kl}^L\right\} \quad (k, l \in R,\ i \in L)$$

The operators required in right block are

$$\left\{\hat{1}^R, \hat{H}^R, \hat{R}_i^R, \hat{R}_i^{R\dagger}, a_k, a_k^\dagger, \hat{A}_{kl}, \hat{A}_{kl}^\dagger, \hat{B}_{kl}\right\} \quad (k, l \in R,\ i \in L)$$

The total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$N_{CN} = 1 + 1 + 2K_R + 2K_L + 2K_R^2 + K_R^2 = 3K_R^2 + 2K + 2$$

Blocking

The enlarged left/right block is denoted as $L*/R*$. Make sure that all L operators are to the left of $*$ operators.

$$\begin{aligned}\hat{R}_i^{L*} &= \hat{R}_i^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_i^* + \sum_{j\in L}\left(\sum_{kl\in *} v_{ijkl} a_k^\dagger a_l\right) a_j + \sum_{j\in *}\left(\sum_{kl\in L} v_{ijkl} a_k^\dagger a_l\right) a_j \\ &\quad + \sum_{k\in L} a_k^\dagger \left(\sum_{jl\in *} v_{ijkl} a_l a_j\right) + \sum_{k\in *} a_k^\dagger \left(\sum_{jl\in L} v_{ijkl} a_l a_j\right) - \sum_{l\in L} a_l \left(\sum_{jk\in *} v_{ijkl} a_k^\dagger a_j\right) - \sum_{l\in *} a_l \left(\sum_{jk\in L} v_{ijkl} a_k^\dagger a_j\right) \\ &= \hat{R}_i^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_i^* + \sum_{j\in L} a_j \left(\sum_{kl\in *} v_{ijkl} a_k^\dagger a_l\right) + \sum_{j\in *}\left(\sum_{kl\in L} v_{ijkl} a_k^\dagger a_l\right) a_j \\ &\quad + \sum_{k\in L} a_k^\dagger \left(\sum_{jl\in *} v_{ijkl} a_l a_j\right) + \sum_{k\in *}\left(\sum_{jl\in L} v_{ijkl} a_l a_j\right) a_k^\dagger - \sum_{l\in L} a_l \left(\sum_{jk\in *} v_{ijkl} a_k^\dagger a_j\right) - \sum_{l\in *}\left(\sum_{jk\in L} v_{ijkl} a_k^\dagger a_j\right) a_l\end{aligned}$$

Now there are two possibilities. In NC partition, in L we have $A, A^\dagger, B, B'$ and in $*$ we have $P, P^\dagger, Q, Q'$. In CN partition, the opposite is true. Therefore, we have

$$\begin{aligned}
\hat{R}_i^{L*,NC} &= \hat{R}_i^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_i^* + \sum_{j\in L} a_j \hat{Q}_{ij}^* + \sum_{j\in *,kl\in L} \big(v_{ijkl} - v_{ilkj}\big) \hat{B}_{kl} a_j + \sum_{k\in L} a_k^\dagger \hat{P}_{ik}^* + \sum_{k\in *,jl\in L,} v_{ijkl} \hat{A}_{jl}^\dagger a_k^\dagger \\
&= \hat{R}_i^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_i^* + \sum_{k\in L} a_k^\dagger \hat{P}_{ik}^* + \sum_{j\in L} a_j \hat{Q}_{ij}^* + \sum_{k\in *,jl\in L,} v_{ijkl} \hat{A}_{jl}^\dagger a_k^\dagger + \sum_{j\in *,kl\in L} \big(v_{ijkl} - v_{ilkj}\big) \hat{B}_{kl} a_j \\
\hat{R}_i^{L*,CN} &= \hat{R}_i^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_i^* + \sum_{j\in L,kl\in *} \big(v_{ijkl} - v_{ilkj}\big) a_j \hat{B}_{kl} + \sum_{j\in *} \hat{Q}_{ij}^L a_j + \sum_{k\in L,jl\in *,} v_{ijkl} a_k^\dagger \hat{A}_{jl}^\dagger + \sum_{k\in *} \hat{P}_{ik}^L a_k^\dagger \\
&= \hat{R}_i^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_i^* + \sum_{k\in L,jl\in *,} v_{ijkl} a_k^\dagger \hat{A}_{jl}^\dagger + \sum_{j\in L,kl\in *} \big(v_{ijkl} - v_{ilkj}\big) a_j \hat{B}_{kl} + \sum_{k\in *} \hat{P}_{ik}^L a_k^\dagger + \sum_{j\in *} \hat{Q}_{ij}^L a_j
\end{aligned}$$

Similarly,

$$\begin{aligned}
\hat{R}_i^{R*,NC} &= \hat{R}_i^* \otimes \hat{1}^R + \hat{1}^* \otimes \hat{R}_i^R + \sum_{k\in *} a_k^\dagger \hat{P}_{ik}^R + \sum_{j\in *} a_j \hat{Q}_{ij}^R + \sum_{k\in R,jl\in *,} v_{ijkl} \hat{A}_{jl}^\dagger a_k^\dagger + \sum_{j\in R,kl\in *} \big(v_{ijkl} - v_{ilkj}\big) \hat{B}_{kl} a_j \\
\hat{R}_i^{R*,CN} &= \hat{R}_i^* \otimes \hat{1}^R + \hat{1}^* \otimes \hat{R}_i^R + \sum_{k\in *,jl\in R,} v_{ijkl} a_k^\dagger \hat{A}_{jl}^\dagger + \sum_{j\in *,kl\in R} \big(v_{ijkl} - v_{ilkj}\big) a_j \hat{B}_{kl} + \sum_{k\in R} \hat{P}_{ik}^* a_k^\dagger + \sum_{j\in R} \hat{Q}_{ij}^* a_j
\end{aligned}$$

Number of terms

$$\begin{aligned}
N_{R,NC} &= (2 + 2K_L + 2K_L^2)K_R + (2 + 2 + 2K_R)K_L = 2K_L^2 K_R + 4K_L K_R + 2K + 2K_L \\
N_{R,CN} &= (2 + 2K_L + 2)K_R + (2 + 2K_R^2 + 2K_R)K_L = 2K_R^2 K_L + 4K_R K_L + 2K + 2K_R
\end{aligned}$$

Blocking of other complementary operators is straightforward

$$\begin{aligned}
\hat{P}_{ik}^{L*,CN} &= \hat{P}_{ik}^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{P}_{ik}^* + \sum_{j\in L,l\in *} v_{ijkl} a_l a_j + \sum_{j\in *,l\in L} v_{ijkl} a_l a_j \\
&= \hat{P}_{ik}^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{P}_{ik}^* - \sum_{j\in L,l\in *} v_{ijkl} a_j a_l + \sum_{j\in *,l\in L} v_{ijkl} a_l a_j \\
\hat{P}_{ik}^{R*,NC} &= \hat{P}_{ik}^* \otimes \hat{1}^R + \hat{1}^* \otimes \hat{P}_{ik}^R + \sum_{j\in *,l\in R} v_{ijkl} a_l a_j + \sum_{j\in R,l\in *} v_{ijkl} a_l a_j \\
&= \hat{P}_{ik}^* \otimes \hat{1}^R + \hat{1}^* \otimes \hat{P}_{ik}^R - \sum_{j\in *,l\in R} v_{ijkl} a_j a_l + \sum_{j\in R,l\in *} v_{ijkl} a_l a_j
\end{aligned}$$

and

$$\begin{aligned}
\hat{Q}_{ij}^{L*,CN} &= \hat{Q}_{ij}^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{Q}_{ij}^* + \sum_{k\in L,l\in *} v_{ijkl} a_k^\dagger a_l + \sum_{k\in *,l\in L} v_{ijkl} a_k^\dagger a_l \\
&= \hat{Q}_{ij}^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{Q}_{ij}^* + \sum_{k\in L,l\in *} v_{ijkl} a_k^\dagger a_l - \sum_{k\in *,l\in L} v_{ijkl} a_l a_k^\dagger \\
\hat{Q}_{ij}^{R*,NC} &= \hat{Q}_{ij}^* \otimes \hat{1}^R + \hat{1}^* \otimes \hat{Q}_{ij}^R + \sum_{k\in *,l\in R} v_{ijkl} a_k^\dagger a_l + \sum_{k\in R,l\in *} v_{ijkl} a_k^\dagger a_l \\
&= \hat{Q}_{ij}^* \otimes \hat{1}^R + \hat{1}^* \otimes \hat{Q}_{ij}^R + \sum_{k\in *,l\in R} v_{ijkl} a_k^\dagger a_l - \sum_{k\in R,l\in *} v_{ijkl} a_l a_k^\dagger
\end{aligned}$$

Middle-Site Transformation

When the sweep is performed from left to right, passing the middle site, we need to switch from NC partition to CN partition. The cost is $O(K^4/16)$. This happens only once in the sweep. The cost of one blocking procedure is $O(K_<^2 K_>)$, but there are K blocking steps in one sweep. So the cost for blocking in one sweep is $O(K K_<^2 K_>)$. Note that the most expensive part in the program should be the Hamiltonian step in Davidson, which scales as $O(K_<^2)$.

$$\hat{P}_{ik}^{L,NC\rightarrow CN} = \sum_{jl\in L} v_{ijkl} a_l a_j = \sum_{jl\in L} v_{ijkl} \hat{A}_{jl}^\dagger$$
$$\hat{Q}_{ij}^{L,NC\rightarrow CN} = \sum_{kl\in L} v_{ijkl} a_k^\dagger a_l = \sum_{kl\in L} v_{ijkl} \hat{B}_{kl}$$

7.1.6 Diagonal Two-Particle Density Matrix

PDM Definition

One-particle density matrix

$$\langle a_{p\sigma}^\dagger a_{q\tau} \rangle$$

Two-particle density matrix

$$\langle a_{p\sigma}^\dagger a_{q\tau}^\dagger a_{r\gamma} a_{s\lambda} \rangle$$

Spatial one-particle density matrix

$$E_{pq} \equiv \sum_\sigma \langle a_{p\sigma}^\dagger a_{q\sigma} \rangle$$

Spatial two-particle density matrix

$$e_{pqrs} \equiv \sum_{\sigma\tau} \langle a_{p\sigma}^\dagger a_{q\tau}^\dagger a_{r\tau} a_{s\sigma} \rangle$$

Spatial two-spin density matrix

$$s_{pqrs} \equiv \sum_{\sigma\tau} (-1)^{1+\delta_{\sigma\tau}} \langle a_{p\sigma}^\dagger a_{q\tau}^\dagger a_{r\tau} a_{s\sigma} \rangle$$

where

$$(-1)^{1+\delta_{\sigma\tau}} = \begin{cases} 1 & \sigma = \tau \\ -1 & \sigma \neq \tau \end{cases}$$

NPC Definition

Number of particle correlation (pure spin)

$$\langle n_{p\sigma} n_{q\tau} \rangle = \langle a^\dagger_{p\sigma} a_{p\sigma} a^\dagger_{q\tau} a_{q\tau} \rangle$$

Number of particle correlation (mixed spin)

$$\langle a^\dagger_{p\sigma} a_{p\tau} a^\dagger_{q\tau} a_{q\sigma} \rangle$$

Spin/Charge Correlation

Spin correlation

$$S_{pq} = \langle (n_{p\alpha} - n_{p\beta})(n_{q\alpha} - n_{q\beta}) \rangle = \langle n_{p\alpha} n_{q\alpha} \rangle - \langle n_{p\alpha} n_{q\beta} \rangle - \langle n_{p\beta} n_{q\alpha} \rangle + \langle n_{p\beta} n_{q\beta} \rangle = \sum_{\sigma\tau} (-1)^{1+\delta_{\sigma\tau}} \langle n_{p\sigma} n_{q\tau} \rangle$$

Charge correlation

$$C_{pq} = \langle (n_{p\alpha} + n_{p\beta})(n_{q\alpha} + n_{q\beta}) \rangle = \langle n_{p\alpha} n_{q\alpha} \rangle + \langle n_{p\alpha} n_{q\beta} \rangle + \langle n_{p\beta} n_{q\alpha} \rangle + \langle n_{p\beta} n_{q\beta} \rangle = \sum_{\sigma\tau} \langle n_{p\sigma} n_{q\tau} \rangle$$

Diagonal Spatial Two-Particle Density Matrix (Pure Spin)

Using anticommutation relation

$$a^\dagger_{q\tau} a_{p\sigma} = -a_{p\sigma} a^\dagger_{q\tau} + \delta_{pq} \delta_{\sigma\tau}$$

We have

$$\langle a^\dagger_{p\sigma} a^\dagger_{q\tau} a_{q\tau} a_{p\sigma} \rangle = -\langle a^\dagger_{p\sigma} a^\dagger_{q\tau} a_{p\sigma} a_{q\tau} \rangle = \langle a^\dagger_{p\sigma} a_{p\sigma} a^\dagger_{q\tau} a_{q\tau} \rangle - \delta_{pq} \delta_{\sigma\tau} \langle a^\dagger_{p\sigma} a_{q\tau} \rangle$$

Then

$$\begin{aligned} e_{pqqp} \equiv \sum_{\sigma\tau} \langle a^\dagger_{p\sigma} a^\dagger_{q\tau} a_{q\tau} a_{p\sigma} \rangle &= -\sum_{\sigma\tau} \langle a^\dagger_{p\sigma} a^\dagger_{q\tau} a_{p\sigma} a_{q\tau} \rangle = \sum_{\sigma\tau} \langle a^\dagger_{p\sigma} a_{p\sigma} a^\dagger_{q\tau} a_{q\tau} \rangle - \delta_{pq} \sum_{\sigma} \langle a^\dagger_{p\sigma} a_{q\sigma} \rangle \\ &= \sum_{\sigma\tau} \langle n_{p\sigma} n_{q\tau} \rangle - \delta_{pq} \sum_{\sigma} \langle a^\dagger_{p\sigma} a_{q\sigma} \rangle \end{aligned}$$

Therefore,

$$\boxed{C_{pq} \equiv \sum_{\sigma\tau} \langle n_{p\sigma} n_{q\tau} \rangle = e_{pqqp} + \delta_{pq} E_{pq}}$$

Similarly,

$$\begin{aligned} s_{pqqp} \equiv \sum_{\sigma\tau} (-1)^{1+\delta_{\sigma\tau}} \langle a^\dagger_{p\sigma} a^\dagger_{q\tau} a_{q\tau} a_{p\sigma} \rangle &= -\sum_{\sigma\tau} (-1)^{1+\delta_{\sigma\tau}} \langle a^\dagger_{p\sigma} a^\dagger_{q\tau} a_{p\sigma} a_{q\tau} \rangle \\ &= \sum_{\sigma\tau} (-1)^{1+\delta_{\sigma\tau}} \langle a^\dagger_{p\sigma} a_{p\sigma} a^\dagger_{q\tau} a_{q\tau} \rangle - \delta_{pq} \sum_{\sigma} \langle a^\dagger_{p\sigma} a_{q\sigma} \rangle \\ &= \sum_{\sigma\tau} (-1)^{1+\delta_{\sigma\tau}} \langle n_{p\sigma} n_{q\tau} \rangle - \delta_{pq} \sum_{\sigma} \langle a^\dagger_{p\sigma} a_{q\sigma} \rangle \end{aligned}$$

Therefore,

$$\boxed{S_{pq} \equiv \sum_{\sigma\tau}(-1)^{1+\delta_{\sigma\tau}}\langle n_{p\sigma}n_{q\tau}\rangle = s_{pqqp} + \delta_{pq}E_{pq}}$$

Diagonal Spatial Two-Particle Density Matrix (Mixed Spin)

Using anticommutation relation

$$a^\dagger_{q\tau}a_{p\tau} = -a_{p\tau}a^\dagger_{q\tau} + \delta_{pq}$$

we have

$$\begin{aligned} e_{pqpq} \equiv& \sum_{\sigma\tau}\langle a^\dagger_{p\sigma}a^\dagger_{q\tau}a_{p\tau}a_{q\sigma}\rangle = -\sum_{\sigma\tau}\langle a^\dagger_{p\sigma}a_{p\tau}a^\dagger_{q\tau}a_{q\sigma}\rangle + \delta_{pq}\sum_{\sigma\tau}\langle a^\dagger_{p\sigma}a_{q\sigma}\rangle \\ =& -\sum_{\sigma\tau}\langle a^\dagger_{p\sigma}a_{p\tau}a^\dagger_{q\tau}a_{q\sigma}\rangle + 2\delta_{pq}\sum_{\sigma}\langle a^\dagger_{p\sigma}a_{q\sigma}\rangle \end{aligned}$$

Therefore,

$$\boxed{\sum_{\sigma\tau}\langle a^\dagger_{p\sigma}a_{p\tau}a^\dagger_{q\tau}a_{q\sigma}\rangle = -e_{pqpq} + 2\delta_{pq}E_{pq}}$$

PYTHON MODULE INDEX

p
pyblock2.driver.core, 375

INDEX

Symbols

__init__() (*pyblock2.driver.core.Block2Wrapper method*), 383
__init__() (*pyblock2.driver.core.DMRGDriver method*), 384
__init__() (*pyblock2.driver.core.ExprBuilder method*), 426
__init__() (*pyblock2.driver.core.SOCDMRGDriver method*), 422

A

add_const() (*pyblock2.driver.core.ExprBuilder method*), 426
add_sum_term() (*pyblock2.driver.core.ExprBuilder method*), 426
add_term() (*pyblock2.driver.core.ExprBuilder method*), 426
add_terms() (*pyblock2.driver.core.ExprBuilder method*), 427
addition() (*pyblock2.driver.core.DMRGDriver method*), 413
adjust_mps() (*pyblock2.driver.core.DMRGDriver method*), 411
adjust_spin_coupling() (*pyblock2.driver.core.WickSpinAdaptation static method*), 429
align_mps_center() (*pyblock2.driver.core.DMRGDriver method*), 411

B

Bipartite (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 378
block2::Allocator (*C++ struct*), 434
block2::Allocator::~Allocator (*C++ function*), 435
block2::Allocator::allocate (*C++ function*), 435
block2::Allocator::Allocator (*C++ function*), 435
block2::Allocator::complex_allocate (*C++ function*), 435
block2::Allocator::complex_deallocate (*C++ function*), 435
block2::Allocator::copy (*C++ function*), 436
block2::Allocator::deallocate (*C++ function*), 435
block2::Allocator::reallocate (*C++ function*), 435
block2::ArchivedSparseMatrix (*C++ struct*), 445
block2::ArchivedSparseMatrix::allocate (*C++ function*), 446
block2::ArchivedSparseMatrix::ArchivedSparseMatrix (*C++ function*), 446
block2::ArchivedSparseMatrix::deallocate (*C++ function*), 446
block2::ArchivedSparseMatrix::filename (*C++ member*), 447
block2::ArchivedSparseMatrix::get_type (*C++ function*), 446
block2::ArchivedSparseMatrix::load_archive (*C++ function*), 446
block2::ArchivedSparseMatrix::offset (*C++ member*), 447
block2::ArchivedSparseMatrix::save_archive (*C++ function*), 446
block2::ArchivedSparseMatrix::sparse_type (*C++ member*), 447
block2::ArchivedTensorFunctions (*C++ struct*), 447
block2::ArchivedTensorFunctions::archive_tensor (*C++ function*), 447

block2::ArchivedTensorFunctions::ArchivedTensorFunctions (C++ function), 447
block2::ArchivedTensorFunctions::filename (C++ member), 454
block2::ArchivedTensorFunctions::get_type (C++ function), 447
block2::ArchivedTensorFunctions::intermediates (C++ function), 452
block2::ArchivedTensorFunctions::left_assign (C++ function), 448
block2::ArchivedTensorFunctions::left_contract (C++ function), 453
block2::ArchivedTensorFunctions::left_rotate (C++ function), 451
block2::ArchivedTensorFunctions::numerical_transform (C++ function), 452
block2::ArchivedTensorFunctions::offset (C++ member), 454
block2::ArchivedTensorFunctions::right_assign (C++ function), 448
block2::ArchivedTensorFunctions::right_contract (C++ function), 453
block2::ArchivedTensorFunctions::right_rotate (C++ function), 451
block2::ArchivedTensorFunctions::tensor_product (C++ function), 451
block2::ArchivedTensorFunctions::tensor_product_diagonal (C++ function), 450
block2::ArchivedTensorFunctions::tensor_product_multiply (C++ function), 449
block2::ArchivedTensorFunctions::tensor_product_partial (C++ function), 450
block2::ArchivedTensorFunctions::tensor_product_partial_multiply (C++ function), 448
block2::BasicFFT (C++ struct), 469
block2::BasicFFT::BasicFFT (C++ function), 469
block2::BasicFFT::fft (C++ function), 469
block2::BasicFFT::init (C++ function), 469
block2::BasicFFT::pad (C++ function), 470
block2::BasicFFT::r (C++ member), 469
block2::BasicFFT::wb (C++ member), 469
block2::BasicFFT::wf (C++ member), 469
block2::BasicFFT::xw (C++ member), 470
block2::BasicFFT<2> (C++ struct), 470
block2::BasicFFT<2>::BasicFFT (C++ function), 470
block2::BasicFFT<2>::fft (C++ function), 470
block2::BasicFFT<2>::init (C++ function), 470
block2::BasicFFT<2>::pad (C++ function), 471
block2::BasicFFT<2>::r (C++ member), 471
block2::BasicFFT<2>::wb (C++ member), 471
block2::BasicFFT<2>::wf (C++ member), 471
block2::binary_repr (C++ function), 455
block2::BitsCodec (C++ struct), 456
block2::BitsCodec::begin_decode (C++ function), 456
block2::BitsCodec::BitsCodec (C++ function), 456
block2::BitsCodec::buf (C++ member), 457
block2::BitsCodec::d_offset (C++ member), 457
block2::BitsCodec::decode (C++ function), 456
block2::BitsCodec::encode (C++ function), 456
block2::BitsCodec::finish_encode (C++ function), 456
block2::BitsCodec::i_length (C++ member), 457
block2::BitsCodec::i_offset (C++ member), 457
block2::BitsCodec::op_data (C++ member), 457
block2::BluesteinFFT (C++ struct), 472
block2::BluesteinFFT::arx (C++ member), 473
block2::BluesteinFFT::b (C++ member), 474
block2::BluesteinFFT::BluesteinFFT (C++ function), 473
block2::BluesteinFFT::cb (C++ member), 473
block2::BluesteinFFT::cf (C++ member), 473
block2::BluesteinFFT::fft (C++ function), 473
block2::BluesteinFFT::init (C++ function), 473
block2::BluesteinFFT::nn (C++ member), 473
block2::BluesteinFFT::wb (C++ member), 473
block2::BluesteinFFT::wf (C++ member),

473
block2::check_signal_ (C++ function), 445
block2::CompressedVector (C++ struct), 460
block2::CompressedVector::~CompressedVector (C++ function), 461
block2::CompressedVector::arr_len (C++ member), 462
block2::CompressedVector::cache_data (C++ member), 462
block2::CompressedVector::cache_dirty (C++ member), 462
block2::CompressedVector::chunk_size (C++ member), 462
block2::CompressedVector::clear (C++ function), 461
block2::CompressedVector::CompressedVector (C++ function), 460
block2::CompressedVector::cp_data (C++ member), 462
block2::CompressedVector::finalize (C++ function), 461
block2::CompressedVector::fpc (C++ member), 462
block2::CompressedVector::icache (C++ member), 462
block2::CompressedVector::ncache (C++ member), 462
block2::CompressedVector::operator[] (C++ function), 461
block2::CompressedVector::shrink_to_fit (C++ function), 461
block2::CompressedVector::size (C++ function), 461
block2::CompressedVectorMT (C++ struct), 462
block2::CompressedVectorMT::cache_datas (C++ member), 463
block2::CompressedVectorMT::CompressedVectorMT (C++ function), 462
block2::CompressedVectorMT::icaches (C++ member), 463
block2::CompressedVectorMT::operator[] (C++ function), 462, 463
block2::CompressedVectorMT::ref_cv (C++ member), 463
block2::CompressedVectorMT::size (C++ function), 463
block2::dalloc_ (C++ function), 439
block2::DataFrame (C++ struct), 439
block2::DataFrame::_t (C++ member), 443
block2::DataFrame::_t2 (C++ member), 443
block2::DataFrame::~DataFrame (C++ function), 440
block2::DataFrame::activate (C++ function), 440
block2::DataFrame::buffer_save_data (C++ function), 444
block2::DataFrame::dallocs (C++ member), 443
block2::DataFrame::DataFrame (C++ function), 439
block2::DataFrame::deallocate (C++ function), 441
block2::DataFrame::dsize (C++ member), 442
block2::DataFrame::fp_codec (C++ member), 444
block2::DataFrame::fpread (C++ member), 443
block2::DataFrame::fpwrite (C++ member), 443
block2::DataFrame::i_frame (C++ member), 442
block2::DataFrame::iallocs (C++ member), 443
block2::DataFrame::isize (C++ member), 442
block2::DataFrame::load_buffering (C++ member), 444
block2::DataFrame::load_buffers (C++ member), 443
block2::DataFrame::load_data (C++ function), 440
block2::DataFrame::load_data_from (C++ function), 440
block2::DataFrame::memory_used (C++ function), 441
block2::DataFrame::minimal_disk_usage (C++ member), 444
block2::DataFrame::minimal_memory_usage (C++ member), 444
block2::DataFrame::mpo_dir (C++ member), 441
block2::DataFrame::mps_dir (C++ member), 441
block2::DataFrame::n_frames (C++ member),

442
block2::DataFrame::operator<< (*C++ function*), 445
block2::DataFrame::partition_can_write (*C++ member*), 442
block2::DataFrame::peak_used_memory (*C++ member*), 443
block2::DataFrame::prefix (*C++ member*), 442
block2::DataFrame::prefix_can_write (*C++ member*), 442
block2::DataFrame::prefix_distri (*C++ member*), 442
block2::DataFrame::present_filenames (*C++ member*), 443
block2::DataFrame::rename_data (*C++ function*), 440
block2::DataFrame::reset (*C++ function*), 440
block2::DataFrame::reset_buffer (*C++ function*), 440
block2::DataFrame::reset_peak_used_memory (*C++ function*), 441
block2::DataFrame::restart_dir (*C++ member*), 441
block2::DataFrame::restart_dir_optimal_mps (*C++ member*), 442
block2::DataFrame::restart_dir_optimal_mps_per_sweep (*C++ member*), 442
block2::DataFrame::restart_dir_per_sweep (*C++ member*), 442
block2::DataFrame::save_buffering (*C++ member*), 444
block2::DataFrame::save_buffers (*C++ member*), 443
block2::DataFrame::save_data (*C++ function*), 441
block2::DataFrame::save_data_to (*C++ function*), 441
block2::DataFrame::save_dir (*C++ member*), 441
block2::DataFrame::save_futures (*C++ member*), 443
block2::DataFrame::tasync (*C++ member*), 443
block2::DataFrame::tread (*C++ member*), 442
block2::DataFrame::twrite (*C++ member*), 443
block2::DataFrame::update_peak_used_memory (*C++ function*), 441
block2::DataFrame::use_main_stack (*C++ member*), 444
block2::DFT (*C++ struct*), 474
block2::DFT::DFT (*C++ function*), 474
block2::DFT::fft (*C++ function*), 474
block2::DFT::init (*C++ function*), 474
block2::FactorizedFFT (*C++ struct*), 474, 475
block2::FactorizedFFT::cooley_tukey (*C++ function*), 476
block2::FactorizedFFT::FactorizedFFT (*C++ function*), 475, 476
block2::FactorizedFFT::fft (*C++ function*), 476
block2::FactorizedFFT::fft_internal (*C++ function*), 475, 476
block2::FactorizedFFT::init (*C++ function*), 476
block2::FactorizedFFT::max_factor (*C++ member*), 477
block2::FactorizedFFT::prime (*C++ member*), 477
block2::FFT (*C++ type*), 477
block2::FFT2 (*C++ type*), 477
block2::FPCodec (*C++ struct*), 457
block2::FPCodec::chunk_size (*C++ member*), 459
block2::FPCodec::decode (*C++ function*), 458
block2::FPCodec::e (*C++ member*), 460
block2::FPCodec::encode (*C++ function*), 458
block2::FPCodec::FPCodec (*C++ function*), 458
block2::FPCodec::m (*C++ member*), 460
block2::FPCodec::n_parallel_chunks (*C++ member*), 459
block2::FPCodec::ncpsd (*C++ member*), 459
block2::FPCodec::ndata (*C++ member*), 459
block2::FPCodec::prec (*C++ member*), 459
block2::FPCodec::prec_u (*C++ member*), 459
block2::FPCodec::read_array (*C++ function*), 459
block2::FPCodec::read_chunks (*C++ function*), 459
block2::FPCodec::s (*C++ member*), 460
block2::FPCodec::write_array (*C++ function*), 458

block2::FPCodec::x (C++ *member*), 460
block2::FPtraits (C++ *struct*), 454
block2::FPtraits::ebits (C++ *member*), 454
block2::FPtraits::mbits (C++ *member*), 454
block2::FPtraits::U (C++ *type*), 454
block2::FPtraits<double> (C++ *struct*), 455
block2::FPtraits<double>::ebits (C++ *member*), 455
block2::FPtraits<double>::mbits (C++ *member*), 455
block2::FPtraits<double>::U (C++ *type*), 455
block2::FPtraits<float> (C++ *struct*), 454
block2::FPtraits<float>::ebits (C++ *member*), 455
block2::FPtraits<float>::mbits (C++ *member*), 455
block2::FPtraits<float>::U (C++ *type*), 455
block2::frame_ (C++ *function*), 445
block2::ialloc_ (C++ *function*), 439
block2::KuhnMunkres (C++ *struct*), 463
block2::KuhnMunkres::cost (C++ *member*), 464
block2::KuhnMunkres::eps (C++ *member*), 464
block2::KuhnMunkres::inf (C++ *member*), 464
block2::KuhnMunkres::KuhnMunkres (C++ *function*), 464
block2::KuhnMunkres::lx (C++ *member*), 464
block2::KuhnMunkres::ly (C++ *member*), 464
block2::KuhnMunkres::match (C++ *function*), 464
block2::KuhnMunkres::n (C++ *member*), 464
block2::KuhnMunkres::slack (C++ *member*), 465
block2::KuhnMunkres::solve (C++ *function*), 464
block2::KuhnMunkres::st (C++ *member*), 465
block2::Prime (C++ *struct*), 465
block2::Prime::euler (C++ *function*), 465
block2::Prime::exgcd (C++ *function*), 466
block2::Prime::factors (C++ *function*), 465
block2::Prime::gcd (C++ *function*), 467
block2::Prime::init_primes (C++ *function*), 465
block2::Prime::inv (C++ *function*), 467
block2::Prime::is_prime (C++ *function*), 465
block2::Prime::miller_rabin (C++ *function*), 468
block2::Prime::np (C++ *member*), 466
block2::Prime::pmod (C++ *function*), 466
block2::Prime::pollard_rho (C++ *function*), 468
block2::Prime::power (C++ *function*), 467
block2::Prime::Prime (C++ *function*), 465
block2::Prime::primes (C++ *member*), 466
block2::Prime::primitive_root (C++ *function*), 466
block2::Prime::primitive_roots (C++ *function*), 466
block2::Prime::quick_multiply (C++ *function*), 467
block2::Prime::quick_power (C++ *function*), 468
block2::Prime::sqrt (C++ *function*), 467
block2::print_trace (C++ *function*), 445
block2::RaderFFT (C++ *struct*), 471
block2::RaderFFT::arx (C++ *member*), 472
block2::RaderFFT::b (C++ *member*), 472
block2::RaderFFT::cb (C++ *member*), 472
block2::RaderFFT::cf (C++ *member*), 472
block2::RaderFFT::fft (C++ *function*), 472
block2::RaderFFT::init (C++ *function*), 471
block2::RaderFFT::nn (C++ *member*), 472
block2::RaderFFT::prime (C++ *member*), 472
block2::RaderFFT::RaderFFT (C++ *function*), 471
block2::RaderFFT::wb (C++ *member*), 472
block2::RaderFFT::wf (C++ *member*), 472
block2::SeqTypes (C++ *enum*), 431
block2::SeqTypes::Auto (C++ *enumerator*), 431
block2::SeqTypes::None (C++ *enumerator*), 431
block2::SeqTypes::Simple (C++ *enumerator*), 431
block2::SeqTypes::SimpleTasked (C++ *enumerator*), 431
block2::SeqTypes::Tasked (C++ *enumerator*), 431
block2::StackAllocator (C++ *struct*), 436
block2::StackAllocator::allocate (C++ *function*), 436
block2::StackAllocator::data (C++ *member*), 437

block2::StackAllocator::deallocate (C++ function), 436
block2::StackAllocator::operator<< (C++ function), 437
block2::StackAllocator::reallocate (C++ function), 436
block2::StackAllocator::shift (C++ member), 437
block2::StackAllocator::size (C++ member), 437
block2::StackAllocator::StackAllocator (C++ function), 436
block2::StackAllocator::used (C++ member), 437
block2::Threading (C++ struct), 431
block2::Threading::activate_global (C++ function), 432
block2::Threading::activate_global_mkl (C++ function), 432
block2::Threading::activate_normal (C++ function), 433
block2::Threading::activate_operator (C++ function), 433
block2::Threading::activate_quanta (C++ function), 433
block2::Threading::blis_available (C++ function), 432
block2::Threading::complex_available (C++ function), 432
block2::Threading::get_mkl_threading_type (C++ function), 432
block2::Threading::get_mkl_version (C++ function), 432
block2::Threading::get_seq_type (C++ function), 432
block2::Threading::get_thread_id (C++ function), 432
block2::Threading::ksymm_available (C++ function), 432
block2::Threading::mkl_available (C++ function), 432
block2::Threading::n_levels (C++ member), 434
block2::Threading::n_threads_global (C++ member), 434
block2::Threading::n_threads_mkl (C++ member), 434
block2::Threading::n_threads_op (C++ member), 433
block2::Threading::n_threads_quanta (C++ member), 434
block2::Threading::openmp_available (C++ function), 432
block2::Threading::operator<< (C++ function), 434
block2::Threading::seq_type (C++ member), 433
block2::Threading::single_precision_available (C++ function), 432
block2::Threading::tbb_available (C++ function), 432
block2::Threading::Threading (C++ function), 433
block2::Threading::type (C++ member), 433
block2::threading_ (C++ function), 434
block2::ThreadingTypes (C++ enum), 430
block2::ThreadingTypes::BatchedGEMM (C++ enumerator), 430
block2::ThreadingTypes::Global (C++ enumerator), 430
block2::ThreadingTypes::Operator (C++ enumerator), 430
block2::ThreadingTypes::OperatorBatchedGEMM (C++ enumerator), 430
block2::ThreadingTypes::OperatorQuanta (C++ enumerator), 430
block2::ThreadingTypes::OperatorQuantaBatchedGEMM (C++ enumerator), 430
block2::ThreadingTypes::Quanta (C++ enumerator), 430
block2::ThreadingTypes::QuantaBatchedGEMM (C++ enumerator), 430
block2::ThreadingTypes::SequentialGEMM (C++ enumerator), 430
block2::VectorAllocator (C++ struct), 437
block2::VectorAllocator::allocate (C++ function), 438
block2::VectorAllocator::copy (C++ function), 438
block2::VectorAllocator::data (C++ member), 439
block2::VectorAllocator::deallocate (C++ function), 438
block2::VectorAllocator::operator<< (C++ function), 439
block2::VectorAllocator::reallocate (C++

function), 438
block2::VectorAllocator::VectorAllocator (*C++ function*), 438
Block2Wrapper (*class in pyblock2.driver.core*), 382
Blocked (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 378
BlockedBipartite (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
BlockedConstrainedSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
BlockedDisjointSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
BlockedLengthSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
BlockedRescaledConstrainedSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
BlockedRescaledDisjointSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
BlockedRescaledSumConstrainedSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
BlockedRescaledSumDisjointSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 378
BlockedRescaledSumSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
BlockedRescaledSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
BlockedSumBipartite (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
BlockedSumConstrainedSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
BlockedSumDisjointSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 378
BlockedSumSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
BlockedSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379

C

CN (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
compress_mps() (*pyblock2.driver.core.DMRGDriver method*), 410
Compressed (*pyblock2.driver.core.NPDMAlgorithmTypes attribute*), 381
Constrained (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 378
ConstrainedSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
Conventional (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
Conventional (*pyblock2.driver.core.NPDMAlgorithmTypes attribute*), 381
ConventionalCN (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
ConventionalNC (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
copy_mps() (*pyblock2.driver.core.DMRGDriver method*), 417
CPX (*pyblock2.driver.core.SymmetryTypes attribute*), 375

D

def_gctr() (*pyblock2.driver.core.NormalOrder static method*), 425
def_gctr_sz() (*pyblock2.driver.core.NormalOrder static method*), 425
def_ix() (*pyblock2.driver.core.NormalOrder static method*), 425
Default (*pyblock2.driver.core.NPDMAlgorithmTypes attribute*), 381
Disjoint (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 378

DisjointSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 378
divide_nprocs() (*pyblock2.driver.core.DMRGDriver method*), 387
dmrg() (*pyblock2.driver.core.DMRGDriver method*), 400
DMRGDriver (*class in pyblock2.driver.core*), 384

E

expectation() (*pyblock2.driver.core.DMRGDriver method*), 415
expr_builder() (*pyblock2.driver.core.DMRGDriver method*), 422
ExprBuilder (*class in pyblock2.driver.core*), 425

F

Fast (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 378
Fast (*pyblock2.driver.core.NPDMAlgorithmTypes attribute*), 381
FastBipartite (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
FastBlockedBipartite (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
FastBlockedConstrainedSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
FastBlockedDisjointSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
FastBlockedLengthSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
FastBlockedRescaledConstrainedSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
FastBlockedRescaledDisjointSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
FastBlockedRescaledSumConstrainedSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
FastBlockedRescaledSumDisjointSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
FastBlockedRescaledSumSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
FastBlockedRescaledSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
FastBlockedSumBipartite (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
FastBlockedSumConstrainedSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
FastBlockedSumDisjointSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 378
FastBlockedSumSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
FastBlockedSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
FastConstrainedSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
FastDisjointSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
FastRescaledConstrainedSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
FastRescaledDisjointSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
FastRescaledSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
FastSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
FermionTransform (*class in pyblock2.driver.core*), 428
finalize() (*pyblock2.driver.core.DMRGDriver method*), 422
finalize() (*pyblock2.driver.core.ExprBuilder method*), 428
fix_restarting_mps() (*py-*

block2.driver.core.DMRGDriver method), 417

G

get_1pdm() (*pyblock2.driver.core.DMRGDriver method*), 409
get_2pdm() (*pyblock2.driver.core.DMRGDriver method*), 409
get_3pdm() (*pyblock2.driver.core.DMRGDriver method*), 409
get_4pdm() (*pyblock2.driver.core.DMRGDriver method*), 409
get_ancilla_mps() (*pyblock2.driver.core.DMRGDriver method*), 421
get_bipartite_entanglement() (*pyblock2.driver.core.DMRGDriver method*), 404
get_conventional_1pdm() (*pyblock2.driver.core.DMRGDriver method*), 407
get_conventional_2pdm() (*pyblock2.driver.core.DMRGDriver method*), 407
get_conventional_npdm() (*pyblock2.driver.core.DMRGDriver method*), 406
get_conventional_qc_mpo() (*pyblock2.driver.core.DMRGDriver method*), 391
get_conventional_trans_1pdm() (*pyblock2.driver.core.DMRGDriver method*), 407
get_conventional_trans_2pdm() (*pyblock2.driver.core.DMRGDriver method*), 407
get_csf_coefficients() (*pyblock2.driver.core.DMRGDriver method*), 410
get_custom_hamiltonian() (*pyblock2.driver.core.DMRGDriver method*), 388
get_dmrg_results() (*pyblock2.driver.core.DMRGDriver method*), 403
get_eq_exprs() (*pyblock2.driver.core.WickSpinAdaptation static method*), 429
get_identity_mpo() (*pyblock2.driver.core.DMRGDriver method*), 391
get_lz_hamiltonian() (*pyblock2.driver.core.DMRGDriver method*), 388
get_mpo() (*pyblock2.driver.core.DMRGDriver method*), 396
get_mpo_any_fermionic() (*pyblock2.driver.core.DMRGDriver method*), 398
get_mpo_any_pauli() (*pyblock2.driver.core.DMRGDriver method*), 398
get_mps_from_csf_coefficients() (*pyblock2.driver.core.DMRGDriver method*), 421
get_n_orb_rdm_mpos() (*pyblock2.driver.core.DMRGDriver method*), 404
get_npdm() (*pyblock2.driver.core.DMRGDriver method*), 407
get_one_orb_rdm_exprs() (*pyblock2.driver.core.OrbitalEntropy static method*), 429
get_one_orb_rdm_h_terms() (*pyblock2.driver.core.OrbitalEntropy static method*), 428
get_orbital_entropies() (*pyblock2.driver.core.DMRGDriver method*), 404
get_orbital_entropies_use_npdm() (*pyblock2.driver.core.DMRGDriver method*), 405
get_orbital_interaction_matrix() (*pyblock2.driver.core.DMRGDriver method*), 405
get_pauli_hamiltonian() (*pyblock2.driver.core.DMRGDriver method*), 388
get_phsu2_hamiltonian() (*pyblock2.driver.core.DMRGDriver method*), 388
get_qc_mpo() (*pyblock2.driver.core.DMRGDriver method*), 392
get_random_mps() (*pyblock2.driver.core.DMRGDriver method*), 420

get_site_mpo() (*pyblock2.driver.core.DMRGDriver method*), 397
get_so3_hamiltonian() (*pyblock2.driver.core.DMRGDriver method*), 388
get_so4_hamiltonian() (*pyblock2.driver.core.DMRGDriver method*), 388
get_spin_square_mpo() (*pyblock2.driver.core.DMRGDriver method*), 398
get_trans_1pdm() (*pyblock2.driver.core.DMRGDriver method*), 409
get_trans_2pdm() (*pyblock2.driver.core.DMRGDriver method*), 409
get_trans_3pdm() (*pyblock2.driver.core.DMRGDriver method*), 409
get_trans_4pdm() (*pyblock2.driver.core.DMRGDriver method*), 409
get_two_orb_rdm_eigvals() (*pyblock2.driver.core.OrbitalEntropy static method*), 429
get_two_orb_rdm_exprs() (*pyblock2.driver.core.OrbitalEntropy static method*), 429
get_two_orb_rdm_h_terms() (*pyblock2.driver.core.OrbitalEntropy static method*), 429
greens_function() (*pyblock2.driver.core.DMRGDriver method*), 415

H

H (*pyblock2.driver.core.STTypes attribute*), 381
H_HT (*pyblock2.driver.core.STTypes attribute*), 382
H_HT_HT2T2 (*pyblock2.driver.core.STTypes attribute*), 382
H_HT_HT2T2_HT1T3 (*pyblock2.driver.core.STTypes attribute*), 382
H_HT_HT2T2_HT2T3 (*pyblock2.driver.core.STTypes attribute*), 382
H_HT_HT2T2_HT3 (*pyblock2.driver.core.STTypes attribute*), 382
H_HT_HTT (*pyblock2.driver.core.STTypes attribute*), 382
H_HT_HTT_HT1T3 (*pyblock2.driver.core.STTypes attribute*), 382
H_HT_HTT_HT2T3 (*pyblock2.driver.core.STTypes attribute*), 382
H_HT_HTT_HT3 (*pyblock2.driver.core.STTypes attribute*), 382
HT (*pyblock2.driver.core.STTypes attribute*), 381
HT1T2 (*pyblock2.driver.core.STTypes attribute*), 382
HT1T3 (*pyblock2.driver.core.STTypes attribute*), 382
HT2T2 (*pyblock2.driver.core.STTypes attribute*), 382
HT2T3 (*pyblock2.driver.core.STTypes attribute*), 382
hybrid_mpo_dmrg() (*pyblock2.driver.core.SOCDMRGDriver method*), 423

I

I (*pyblock2.driver.core.ParallelTypes attribute*), 377
ialloc (*C macro*), 439
initialize_system() (*pyblock2.driver.core.DMRGDriver method*), 385
integral_symmetrize() (*pyblock2.driver.core.DMRGDriver method*), 390
iscale() (*pyblock2.driver.core.ExprBuilder method*), 427

J

J (*pyblock2.driver.core.ParallelTypes attribute*), 377
jordan_wigner() (*pyblock2.driver.core.FermionTransform static method*), 428

L

Length (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 378
load_mps() (*pyblock2.driver.core.DMRGDriver method*), 417

LowMem (*pyblock2.driver.core.NPDMAlgorithmTypes attribute*), 381
LZ (*pyblock2.driver.core.SymmetryTypes attribute*), 376

M

make_sgf() (*pyblock2.driver.core.NormalOrder static method*), 425
make_sgf() (*pyblock2.driver.core.SimilarityTransform static method*), 429
make_sgf() (*pyblock2.driver.core.WickNormalOrder static method*), 425
make_su2() (*pyblock2.driver.core.NormalOrder static method*), 425
make_su2() (*pyblock2.driver.core.SimilarityTransform static method*), 429
make_su2() (*pyblock2.driver.core.WickNormalOrder static method*), 425
make_su2_open_shell() (*pyblock2.driver.core.WickNormalOrder static method*), 425
make_sz() (*pyblock2.driver.core.NormalOrder static method*), 425
make_sz() (*pyblock2.driver.core.SimilarityTransform static method*), 429
make_sz() (*pyblock2.driver.core.WickNormalOrder static method*), 425
MixUIJSI (*pyblock2.driver.core.ParallelTypes attribute*), 377
MixUIJUKL (*pyblock2.driver.core.ParallelTypes attribute*), 377
MixUKLUIJ (*pyblock2.driver.core.ParallelTypes attribute*), 377
module
    pyblock2.driver.core, 375
MPOAlgorithmTypes (*class in pyblock2.driver.core*), 378
mps_change_complex() (*pyblock2.driver.core.DMRGDriver method*), 419
mps_change_from_singlet_embedding() (*pyblock2.driver.core.DMRGDriver method*), 418
mps_change_precision() (*pyblock2.driver.core.DMRGDriver method*), 419
mps_change_singlet_embedding() (*pyblock2.driver.core.DMRGDriver method*), 417
mps_change_to_singlet_embedding() (*pyblock2.driver.core.DMRGDriver method*), 418
mps_change_to_sz() (*pyblock2.driver.core.DMRGDriver method*), 419
multiply() (*pyblock2.driver.core.DMRGDriver method*), 412

N

NC (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
NoRIntermed (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
NoRIntermedConventional (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 381
Normal (*pyblock2.driver.core.NPDMAlgorithmTypes attribute*), 381
NormalOrder (*class in pyblock2.driver.core*), 425
Nothing (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 378
Nothing (*pyblock2.driver.core.NPDMAlgorithmTypes attribute*), 381
Nothing (*pyblock2.driver.core.ParallelTypes attribute*), 376
Nothing (*pyblock2.driver.core.SymmetryTypes attribute*), 375
NoTransConventional (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
NoTransConventionalCN (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 381
NoTransConventionalNC (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 381
NoTransNoRIntermedConventional (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 381
NoTranspose (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
NPDMAlgorithmTypes (*class in pyblock2.driver.core*), 381

O

ops (*pyblock2.driver.core.OrbitalEntropy attribute*), 428
ops_ghf (*pyblock2.driver.core.OrbitalEntropy attribute*), 428
orbital_reordering() (*pyblock2.driver.core.DMRGDriver method*), 399
orbital_reordering_interaction_matrix() (*pyblock2.driver.core.DMRGDriver method*), 399
OrbitalEntropy (*class in pyblock2.driver.core*), 428

P

parallelize_integrals() (*pyblock2.driver.core.DMRGDriver method*), 387
ParallelTypes (*class in pyblock2.driver.core*), 376
parse_expr() (*pyblock2.driver.core.OrbitalEntropy static method*), 428
PHSU2 (*pyblock2.driver.core.SymmetryTypes attribute*), 376
pyblock2.driver.core
    module, 375

R

read_fcidump() (*pyblock2.driver.core.DMRGDriver method*), 389
Rescaled (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 378
RescaledConstrainedSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
RescaledDisjointSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 379
RescaledSVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 380
restart_dir (*pyblock2.driver.core.DMRGDriver property*), 385

S

SAny (*pyblock2.driver.core.SymmetryTypes attribute*), 376
SAnyPHSU2 (*pyblock2.driver.core.SymmetryTypes attribute*), 376
SAnySGF (*pyblock2.driver.core.SymmetryTypes attribute*), 376
SAnySGFCPX (*pyblock2.driver.core.SymmetryTypes attribute*), 376
SAnySGFLZ (*pyblock2.driver.core.SymmetryTypes attribute*), 376
SAnySO3 (*pyblock2.driver.core.SymmetryTypes attribute*), 376
SAnySO4 (*pyblock2.driver.core.SymmetryTypes attribute*), 376
SAnySU2 (*pyblock2.driver.core.SymmetryTypes attribute*), 376
SAnySU2LZ (*pyblock2.driver.core.SymmetryTypes attribute*), 376
SAnySZ (*pyblock2.driver.core.SymmetryTypes attribute*), 376
SAnySZLZ (*pyblock2.driver.core.SymmetryTypes attribute*), 376
scratch (*pyblock2.driver.core.DMRGDriver property*), 385
set_symm_type() (*pyblock2.driver.core.DMRGDriver method*), 385
set_symmetry_groups() (*pyblock2.driver.core.Block2Wrapper method*), 383
set_symmetry_groups() (*pyblock2.driver.core.DMRGDriver method*), 385
SGB (*pyblock2.driver.core.SymmetryTypes attribute*), 375
SGF (*pyblock2.driver.core.SymmetryTypes attribute*), 375
SGFCPX (*pyblock2.driver.core.SymmetryTypes attribute*), 375
SI (*pyblock2.driver.core.ParallelTypes attribute*), 377
SIJ (*pyblock2.driver.core.ParallelTypes attribute*), 377
SimilarityTransform (*class in pyblock2.driver.core*), 429
SJ (*pyblock2.driver.core.ParallelTypes attribute*), 377
SKL (*pyblock2.driver.core.ParallelTypes attribute*), 377

SO3 (*pyblock2.driver.core.SymmetryTypes attribute*), 376
SO4 (*pyblock2.driver.core.SymmetryTypes attribute*), 376
soc_two_step() (*pyblock2.driver.core.SOCDMRGDriver method*), 424
SOCDMRGDriver (*class in pyblock2.driver.core*), 422
SP (*pyblock2.driver.core.SymmetryTypes attribute*), 375
SPCPX (*pyblock2.driver.core.SymmetryTypes attribute*), 375
spin_tag_to_pattern() (*pyblock2.driver.core.WickSpinAdaptation static method*), 429
split_mps() (*pyblock2.driver.core.DMRGDriver method*), 412
STTypes (*class in pyblock2.driver.core*), 381
SU2 (*pyblock2.driver.core.SymmetryTypes attribute*), 375
su2_to_sgf() (*pyblock2.driver.core.DMRGDriver method*), 390
Sum (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 378
SVD (*pyblock2.driver.core.MPOAlgorithmTypes attribute*), 378
SymbolFree (*pyblock2.driver.core.NPDMAlgorithmTypes attribute*), 381
symm_type (*pyblock2.driver.core.DMRGDriver property*), 385
SymmetryTypes (*class in pyblock2.driver.core*), 375
SZ (*pyblock2.driver.core.SymmetryTypes attribute*), 375

T

td_dmrg() (*pyblock2.driver.core.DMRGDriver method*), 402
threading (*C macro*), 434

U

UIJ (*pyblock2.driver.core.ParallelTypes attribute*), 377
UIJM2 (*pyblock2.driver.core.ParallelTypes attribute*), 377
UIJM4 (*pyblock2.driver.core.ParallelTypes attribute*), 377
UKL (*pyblock2.driver.core.ParallelTypes attribute*), 377
UKLM2 (*pyblock2.driver.core.ParallelTypes attribute*), 377
UKLM4 (*pyblock2.driver.core.ParallelTypes attribute*), 377
unpack_g2e() (*pyblock2.driver.core.DMRGDriver method*), 391

W

WickNormalOrder (*class in pyblock2.driver.core*), 425
WickSpinAdaptation (*class in pyblock2.driver.core*), 429
write_fcidump() (*pyblock2.driver.core.DMRGDriver method*), 389