
block2

Huanchen Zhai

Oct 16, 2023

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	170
4.3	Custom Hamiltonian	178
4.4	Green's Function and TD-DMRG	183
4.5	Hubbard Model	208
4.6	Heisenberg Model	228
5	Developer Guide	233
5.1	DMRG Options	233
5.2	MPS Orbital Rotation	235
5.3	Point Group Mapping	246
5.4	MPO Reloading	252
5.5	Debugging Hints	260
5.6	Notes	269
6	API Reference	273
6.1	Global Settings	273
6.2	Sparse Matrix	288
6.3	Tensor Functions	290

6.4	Tools	297
7	Theory	321
7.1	DMRG Hamiltonian	321
	Index	359

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>

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

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

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 re-installing 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 point 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 small 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 instructions 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 environment (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 directory. 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 appear, 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 `nospinadapted` 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 combinations 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 roots 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-averaged 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 dmrq.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 excited 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 calculation 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 `onedpm`, `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)
```

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 indices $[:, a, b]$ are for $\langle a_{a\alpha}^\dagger a_{b\alpha} \rangle$, and $\langle a_{a\beta}^\dagger a_{b\beta} \rangle$, 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 indices $[:, a, b, c, d]$ are for $\langle a_{a\alpha}^\dagger a_{b\alpha}^\dagger a_{c\alpha} a_{d\alpha} \rangle$, $\langle a_{a\alpha}^\dagger a_{b\beta}^\dagger a_{c\beta} a_{d\alpha} \rangle$, and $\langle a_{a\beta}^\dagger a_{b\beta}^\dagger a_{c\beta} a_{d\beta} \rangle$, 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

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

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 indices $[:, a, b, c, d, e, f]$ are for $\langle a_{a\alpha}^\dagger a_{b\alpha}^\dagger a_{c\alpha}^\dagger a_{d\alpha} a_{e\alpha} a_{f\alpha} \rangle$, $\langle a_{a\alpha}^\dagger a_{b\alpha}^\dagger a_{c\beta}^\dagger a_{d\beta} a_{e\alpha} a_{f\alpha} \rangle$, $\langle a_{a\alpha}^\dagger a_{b\beta}^\dagger a_{c\beta}^\dagger a_{d\beta} a_{e\beta} a_{f\alpha} \rangle$, and $\langle a_{a\beta}^\dagger a_{b\beta}^\dagger a_{c\beta}^\dagger a_{d\beta} a_{e\beta} a_{f\beta} \rangle$, 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

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

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 indices $[:, a, b, c, d, e, f, g, h]$ are for $\langle a_{a\alpha}^\dagger a_{b\alpha}^\dagger a_{c\alpha}^\dagger a_{d\alpha}^\dagger a_{e\alpha} a_{f\alpha} a_{g\alpha} a_{h\alpha} \rangle$, $\langle a_{a\alpha}^\dagger a_{b\alpha}^\dagger a_{c\alpha}^\dagger a_{d\beta}^\dagger a_{e\beta} a_{f\alpha} a_{g\alpha} a_{h\alpha} \rangle$, $\langle a_{a\alpha}^\dagger a_{b\alpha}^\dagger a_{c\beta}^\dagger a_{d\beta}^\dagger a_{e\beta} a_{f\beta} a_{g\alpha} a_{h\alpha} \rangle$, $\langle a_{a\alpha}^\dagger a_{b\beta}^\dagger a_{c\beta}^\dagger a_{d\beta}^\dagger a_{e\beta} a_{f\beta} a_{g\beta} a_{h\alpha} \rangle$, and $\langle a_{a\beta}^\dagger a_{b\beta}^\dagger a_{c\beta}^\dagger a_{d\beta}^\dagger a_{e\beta} a_{f\beta} a_{g\beta} a_{h\beta} \rangle$, 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 $[1, n, n]$, $[1, n, n, n, n]$, $[1, n, n, n, n, n, n, n]$, and $[1, n, n, n, n, n, n, n, n, n]$ respectively, where n is the number of spin orbitals. The content is the expectation value for $\langle a_a^\dagger a_b \rangle$, $\langle a_a^\dagger a_b^\dagger a_c a_d \rangle$, $\langle a_a^\dagger a_b^\dagger a_c^\dagger a_d a_e a_f \rangle$, and $\langle a_a^\dagger a_b^\dagger a_c^\dagger a_d^\dagger a_e a_f a_g a_h \rangle$, 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` and `conventional_npdms` 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 calculation 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 scratch 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 directory).

You may need to change the (custom) schedule 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-averaged MPSs or state-specific MPSs (stored in scratch 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
conventional_npd
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:

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

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	DE =	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 guarantee 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:

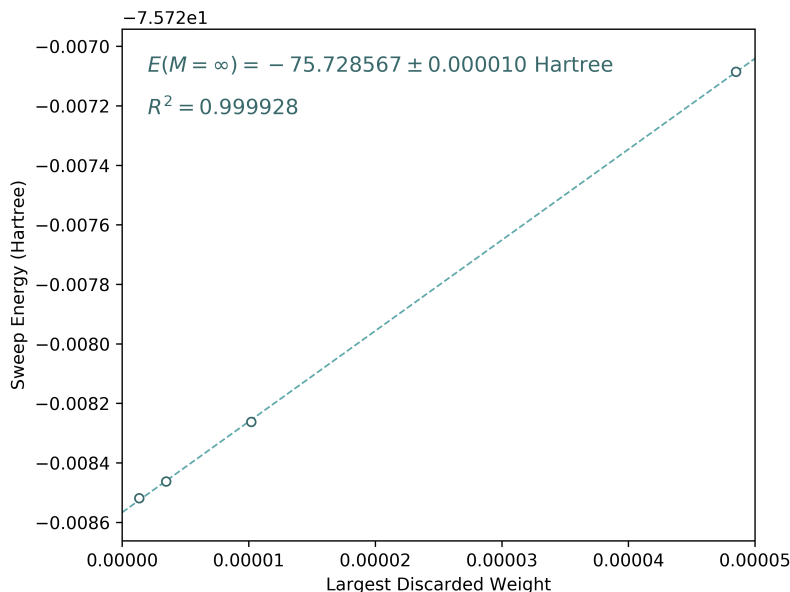
In the above script, we have used the largest discarded weights and associated sweep energies in the last sweep iteration of each bond dimension ($M = 800, 600, 400, 200$) to make linear regression. The extrapolated DMRG sweep energy is -75.728567 Hartree.

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 $1.07\text{e-}09$, and the norm of MPS is not far away from 1. So the transformation 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 scratch 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 $9.7\text{E-}6$. 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 transform 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 = KRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR 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 = KRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR 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 = CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC 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)

```
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 = KRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR 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_twsz` indicates that the transformed SZ MPS will have projected spin 2

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:

Some reference outputs for this example:

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+000000000+0200000-000-00 = -0.140063445607095
CSF 3 20+000000000+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`.

npv 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 obtained 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 $H|MPS\rangle$ 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' dmrq-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 Z_4 or Z_2 . 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 keyword `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.
↪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 = LLLLLLLLLLLLLLLLLLLLLLLLLLLLLLJ 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 = LLLLLLLLLLLLLLLLLLLLLLLLLLT
----- root =  1 /  2 -----
      final tag = KET-CPX-1
      final canonical form = LLLLLLLLLLLLLLLLLLLLLLLLLLT
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.

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 $\exp(-0.2i \cdot H)$ to the input.

The output :

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.

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

59

```
$ grep 'final' dmrg-4.out
mps final tag = BRAEX
mps final canonical form = MRRRRRRRRRRRRRRRRRRRRRRRRR
$ 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      1.0000000000000002 + IM      0.0000000000000000
OH Energy      1 -      0 = RE     -0.845792004408687 + IM     -0.533433527528264
OH Energy      1 -      1 = RE      0.999986418355938 + IM      0.0000000000000000
OH Energy      2 -      0 = RE     -0.0000000000000000 + IM      0.0000000000000000
OH Energy      2 -      1 = RE     -0.0000000827506956 + IM     -0.000000742303613
OH Energy      2 -      2 = RE      1.0000000000000004 + IM      0.0000000000000000
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 $\langle 1|1 \rangle$ 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.
↪ 0000000000000002)
= -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 $E \cdot t$ divided by 2π :

```
-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 $\langle 2|0 \rangle$ is exactly zero. The corresponding overlap between the time evolved states $\langle 3|1 \rangle$ is slightly different from zero, mainly due to the time step error and truncation error.

We can also get the energy expectation, 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   -0.0000000000000007
OH Energy    2 -    0 = RE     0.000000000000000 + IM     0.000000000000000
OH Energy    2 -    1 = RE     0.0000061050951670 + 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 $\langle A|H|B \rangle$, but the energy is $\langle A|H|B \rangle / \langle A|B \rangle$. 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 libdmnet). 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 ~ 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.

nospinadapted

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 interpreted 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 reinterpreted 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 storing 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_npdms.

fourpdm / restart_fourpdm

Four-particle density matrix calculation on the DMRG optimized MPS or reloaded MPS. Cannot be used together with conventional_npdms.

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_npdms.

tran_fourpdm / restart_tran_fourpdm

Four-particle transition density matrix among a set of MPSs. Cannot be used together with conventional_npdms.

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_dmrp

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 dimension 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_npdn

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 contraction 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_npd

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 and transition density matrix between different irreps, 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 $\log(\text{condense_mpo})$ times, until the total number of sites is $n_{\text{sites}} / \text{condense_mpo}$. 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 Hubbard 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 ending 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 ending 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 construction 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.

mrcisdtr

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 Dyll'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 default 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 *i*th 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.

n_sub_sweeps

Optional. Number of sweeps for each time step. Default 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 calculation 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 unphysical eigenvalues as $E_i + w_i$, where E_i 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-separated 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 dmrghscf, which can be obtained from <https://github.com/pyscf/dmrghscf>. If it is installed using pip, one also needs to create a file named settings.py under the dmrghscf folder, as follows:

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

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

(continues on next page)

(continued from previous page)

```
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
natorb, natelec, coeff = avas.avas(mf, ["C 2p", "C 3p", "C 2s", "C 3s"])
print('CAS = ', natorb, natelec)

mc = dmrgscf.DMRGSCF(mf, natorb, natelec, 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
natorb, natelec, coeff = avas.avas(mf, ["C 2p", "C 3p", "C 2s", "C 3s"])
print('CAS = ', natorb, natelec)

mc = dmrgscf.DMRGSCF(mf, natorb, natelec, 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
natorb, natelec, coeff = avas.avas(mf, ["C 2p", "C 3p", "C 2s", "C 3s"])
print('CAS = ', natorb, natelec)

mc = mcscf.CASSCF(mf, natorb, natelec)
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
natorb, natelec, coeff = avas.avas(mf, ["C 2p", "C 3p", "C 2s", "C 3s"])
print('CAS = ', natorb, natelec)

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, natorb, natelec)
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 = 0.00000000
State 1 weight 0.5  E = -75.298522666384   S^2 = 2.00000000

```

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.nospinAdapted = 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.dmrghi.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 orbitals 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
natorb, natelec, coeff = avas.avas(mf, ["C 2p", "C 3p", "C 2s", "C 3s"])
print('CAS = ', natorb, natelec)
```

(continues on next page)

(continued from previous page)

```

mc = mcscf.CASSCF(mf, natorb, 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/dmrghscf/nevpt_mpi.py` has the execute permission. You can do `chmod +x ${PYSCFHOME}/pyscf/dmrghscf/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, dmrghscf, lib
import os

dmrghscf.settings.BLOCKEXE = os.popen("which block2main").read().strip()
dmrghscf.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 O O O; O O O 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 WickICNEVP2
wic = WickICNEVP2(mc).run()
```

This will generate the following output:

```
$ grep 'E(WickSCNEVP2)' dmrg-nevpt2.out
E(WickSCNEVP2) = -149.9578400627551 E_corr_pt = -0.2491822915198339
$ grep 'E(WickICNEVP2)' dmrg-nevpt2.out
E(WickICNEVP2) = -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 $\geq 0.5.1rc17$), 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 block2 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

  0 0 0 -0.6035
  0 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
  CIRCUT = 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 orbital 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.
↳] # 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

  0 0 0 -0.6035
  0 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
CIR00T = 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
  CIR00T = 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

  0 0 0 -0.6035
  0 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
  CIRROOT = 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
  CIRROOT = 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

  0 0 0 -0.6035
  0 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
  CIRROOT = 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 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.

Then we can build `psi4` as follows:

```
git clone https://github.com/psi4/psi4
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.

Third, we need to build `ambit` as follows:

```
git clone https://github.com/jturney/ambit
cd ambit
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=../install
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_dmrp` branch of the forked repo <https://github.com/hczhai/forte>. To build it, we can:

```
git clone https://github.com/hczhai/forte
cd forte
git checkout block2_dmrp
$(psi4 --plugin-compile) -Dambit_DIR=${AMBIT_DIR}/share/cmake/ambit \
  -DENABLE_block2=ON \
  -Dblock2_DIR=${AMBIT_DIR}/share/cmake/block2 \
  -DCMAKE_PREFIX_PATH=${CONDA_PREFIX}
make -j 10
export FORTE_DIR=$PWD
cd ..
```

Then we can add the following environment variables:

```
export PYTHONPATH=${FORTE_DIR}:$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 DM-RGSCF state as the reference state:

```
import psi4
import forte

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

psi4.set_options(
    {
        'basis': 'cc-pvdz',
        'scf_type': 'direct',
        'e_convergence': 20,
        'reference': 'rohfc',
        '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
0 0.0 0.0 -0.6035
0 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 xzf 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 version 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 directory.

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 following 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 evaluated 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' dmrq2.out | tail -1
Time elapsed = 3.883 | E = -75.7284436933 | DE = -3.85e-07 | DW = 3.76e-16
```

The following commands can be used to translate the MPS. Please make sure that the environment variables `STACKBLOCK`, `PYBLOCKHOME`, and `BLOCK2HOME` are correctly set.

[illegible]

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 evaluate 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 dmr2.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|" dmr2.conf
$ READWFN=${BLOCK2HOME}/pyblock2/driver/readwfn.py
$ python3 $READWFN dmr2.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. arXiv preprint [arXiv:2310.03920](https://arxiv.org/abs/2310.03920).

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.

For the large site code, 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.

You can find a bibtex file in [CITATIONS.bib](#).

The other algorithms implemented in **block2** are based on the following papers.

3.10.1 Quantum Chemistry 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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/10.1016/0009-2614(84)85617-1)
- Laidig, W. D., Saxe, P., Bartlett, R. J. The Description of N₂ and F₂ Potential Energy Surfaces Using Multireference Coupled Cluster Theory. *The Journal of Chemical Physics* 1987, **86**, 887-907. doi: [10.1063/1.452291](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/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](https://doi.org/10.1063/1.4905329)

PYTHON INTERFACE TUTORIAL

4.1 Quantum Chemistry Hamiltonians

```
[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
```

4.1.1 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 symmetry that can be used in the DMRG calculation is thus has 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
```

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)

bond_dims = [250] * 4 + [500] * 4
noises = [1e-4] * 4 + [1e-5] * 4 + [0]
thrds = [1e-10] * 8

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.dmrp(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 = 5.977257773040786e-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 = 0.010
Site = 1 / 10 .. Mmpo = 34 DW = 0.00e+00 NNZ = 63 SPT = 0.8575 Tmvc_
↪= 0.000 T = 0.009
Site = 2 / 10 .. Mmpo = 56 DW = 0.00e+00 NNZ = 121 SPT = 0.9364 Tmvc_
↪= 0.000 T = 0.009
Site = 3 / 10 .. Mmpo = 74 DW = 0.00e+00 NNZ = 373 SPT = 0.9100 Tmvc_
↪= 0.000 T = 0.010
Site = 4 / 10 .. Mmpo = 80 DW = 0.00e+00 NNZ = 269 SPT = 0.9546 Tmvc_
↪= 0.000 T = 0.007
Site = 5 / 10 .. Mmpo = 94 DW = 0.00e+00 NNZ = 169 SPT = 0.9775 Tmvc_
↪= 0.000 T = 0.009
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 = 0.007
Site = 8 / 10 .. Mmpo = 14 DW = 0.00e+00 NNZ = 41 SPT = 0.9024 Tmvc_
↪= 0.000 T = 0.006
Site = 9 / 10 .. Mmpo = 1 DW = 0.00e+00 NNZ = 14 SPT = 0.0000 Tmvc_
↪= 0.000 T = 0.010
Ttotal = 0.082 Tmvc-total = 0.003 MPO bond dimension = 94 MaxDW = 0.00e+00
NNZ = 1317 SIZE = 27073 SPT = 0.9514

Rank = 0 Ttotal = 0.197 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 = 1.239 | 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 = 1.842 | E = -107.6541224475 | DE = -5.40e-12 | DW = 5.66e-20

Sweep = 2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |_
↪Dav threshold = 1.00e-10
Time elapsed = 2.663 | 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

```

(continues on next page)

(continued from previous page)

```

Time elapsed =      3.203 | E =      -107.6541224475 | DE = -7.96e-13 | DW = 5.72e-20

Sweep =      4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =      3.918 | E =      -107.6541224475 | DE = -2.84e-14 | DW = 1.92e-20

Sweep =      5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =      4.457 | E =      -107.6541224475 | DE = 0.00e+00 | DW = 5.42e-20

Sweep =      6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =      5.188 | E =      -107.6541224475 | DE = 0.00e+00 | DW = 3.01e-20

Sweep =      7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =      5.683 | E =      -107.6541224475 | DE = -2.84e-14 | DW = 5.57e-20

Sweep =      8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 | _
↳ Dav threshold = 1.00e-09
Time elapsed =      6.061 | E =      -107.6541224475 | DE = 2.84e-14 | DW = 2.64e-20

DMRG energy = -107.654122447524443
Energy from pdms = -107.654122447524387
Energy from expectation = -107.654122447524344

```

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.dmrp(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

```

```

integral symmetrize error = 7.512924107430347e-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 = 0.005
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 = 0.006
Site = 2 / 10 .. Mmpo = 110 DW = 0.00e+00 NNZ = 283 SPT = 0.9610 Tmvc_
↪ = 0.001 T = 0.008
Site = 3 / 10 .. Mmpo = 138 DW = 0.00e+00 NNZ = 1023 SPT = 0.9326 Tmvc_
↪ = 0.004 T = 0.013
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 = 0.010
Site = 6 / 10 .. Mmpo = 106 DW = 0.00e+00 NNZ = 415 SPT = 0.9790 Tmvc_
↪ = 0.000 T = 0.010
Site = 7 / 10 .. Mmpo = 58 DW = 0.00e+00 NNZ = 163 SPT = 0.9735 Tmvc_
↪ = 0.000 T = 0.025
Site = 8 / 10 .. Mmpo = 26 DW = 0.00e+00 NNZ = 87 SPT = 0.9423 Tmvc_
↪ = 0.000 T = 0.010
Site = 9 / 10 .. Mmpo = 1 DW = 0.00e+00 NNZ = 26 SPT = 0.0000 Tmvc_
↪ = 0.000 T = 0.005
Ttotal = 0.103 Tmvc-total = 0.009 MPO bond dimension = 186 MaxDW = 0.00e+00
NNZ = 3164 SIZE = 102772 SPT = 0.9692

Rank = 0 Ttotal = 0.178 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 = 1.281 | 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 = 2.089 | E = -107.6541224475 | DE = -1.31e-11 | DW = 5.10e-09

Sweep = 2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |_
↪ Dav threshold = 1.00e-10
Time elapsed = 2.909 | 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 = 4.136 | E = -107.6541224475 | DE = 1.42e-12 | DW = 5.19e-09

Sweep = 4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 |_
↪ Dav threshold = 1.00e-10
Time elapsed = 5.572 | E = -107.6541224475 | DE = -1.17e-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 =      6.850 | E =    -107.6541224475 | DE = 9.95e-13 | DW = 1.60e-19

Sweep =      6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 |
↳ Dav threshold = 1.00e-10
Time elapsed =      7.709 | 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 =      8.581 | E =    -107.6541224475 | DE = -1.25e-12 | DW = 1.60e-19

Sweep =      8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 |
↳ Dav threshold = 1.00e-09
Time elapsed =      9.169 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 5.07e-20

DMRG energy = -107.654122447524500

```

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.dmrp(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

integral symmetrize error = 7.512924107430346e-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 = 0.006
Site = 2 / 20 .. Mmpo = 45 DW = 0.00e+00 NNZ = 45 SPT = 0.9500 Tmvc
↳ = 0.001 T = 0.005

```

(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 = 0.005
Site =    4 /    20 .. Mmpo =    81 DW = 0.00e+00 NNZ =    159 SPT = 0.9683 Tmvc_
↪= 0.001 T = 0.005
Site =    5 /    20 .. Mmpo =   104 DW = 0.00e+00 NNZ =    203 SPT = 0.9759 Tmvc_
↪= 0.001 T = 0.006
Site =    6 /    20 .. Mmpo =   125 DW = 0.00e+00 NNZ =    265 SPT = 0.9796 Tmvc_
↪= 0.001 T = 0.006
Site =    7 /    20 .. Mmpo =   126 DW = 0.00e+00 NNZ =    974 SPT = 0.9382 Tmvc_
↪= 0.001 T = 0.008
Site =    8 /    20 .. Mmpo =   151 DW = 0.00e+00 NNZ =    188 SPT = 0.9901 Tmvc_
↪= 0.001 T = 0.010
Site =    9 /    20 .. Mmpo =   148 DW = 0.00e+00 NNZ =    344 SPT = 0.9846 Tmvc_
↪= 0.001 T = 0.037
Site =   10 /    20 .. Mmpo =   177 DW = 0.00e+00 NNZ =    246 SPT = 0.9906 Tmvc_
↪= 0.001 T = 0.007
Site =   11 /    20 .. Mmpo =   178 DW = 0.00e+00 NNZ =    402 SPT = 0.9872 Tmvc_
↪= 0.001 T = 0.006
Site =   12 /    20 .. Mmpo =   147 DW = 0.00e+00 NNZ =    306 SPT = 0.9883 Tmvc_
↪= 0.001 T = 0.005
Site =   13 /    20 .. Mmpo =   100 DW = 0.00e+00 NNZ =    242 SPT = 0.9835 Tmvc_
↪= 0.000 T = 0.004
Site =   14 /    20 .. Mmpo =    77 DW = 0.00e+00 NNZ =    150 SPT = 0.9805 Tmvc_
↪= 0.000 T = 0.004
Site =   15 /    20 .. Mmpo =    54 DW = 0.00e+00 NNZ =     94 SPT = 0.9774 Tmvc_
↪= 0.000 T = 0.003
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 = 0.004
Site =   18 /    20 .. Mmpo =     7 DW = 0.00e+00 NNZ =     20 SPT = 0.8571 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 =    0.134 Tmvc-total = 0.012 MPO bond dimension =    178 MaxDW = 0.00e+00
NNZ =    3934 SIZE =    200866 SPT = 0.9804

Rank =     0 Ttotal =    0.186 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 =    0.861 | 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 =      1.471 | 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 =      2.030 | 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 =      2.597 | E =    -107.6541224347 | DE = 8.81e-13 | DW = 6.15e-08

Sweep =      4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↪Dav threshold = 1.00e-10
Time elapsed =      3.707 | 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 =      5.003 | E =    -107.6541224379 | DE = -3.37e-11 | DW = 7.34e-20

Sweep =      6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↪Dav threshold = 1.00e-10
Time elapsed =      6.306 | 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 =      7.088 | E =    -107.6541224379 | DE = 7.11e-13 | DW = 7.10e-20

Sweep =      8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 | _
↪Dav threshold = 1.00e-09
Time elapsed =      7.514 | E =    -107.6541224379 | DE = 2.84e-14 | DW = 8.86e-20

DMRG energy = -107.654122437940984

```

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 information 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)

bond_dims = [250] * 4 + [500] * 4
noises = [1e-4] * 4 + [1e-5] * 4 + [0]
thrds = [1e-10] * 8

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.dmrp(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

symmetrize error = 2.33000000000000018e-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_
= 0.000 T = 0.011
Site = 1 / 10 .. Mmpo = 34 DW = 0.00e+00 NNZ = 63 SPT = 0.8575 Tmvc_
= 0.000 T = 0.018
Site = 2 / 10 .. Mmpo = 56 DW = 0.00e+00 NNZ = 121 SPT = 0.9364 Tmvc_
= 0.000 T = 0.032
Site = 3 / 10 .. Mmpo = 74 DW = 0.00e+00 NNZ = 373 SPT = 0.9100 Tmvc_
= 0.001 T = 0.007
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 = 0.012

```

(continues on next page)

(continued from previous page)

```

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 = 0.007
Site =      8 /      10 .. Mmpo =      14 DW = 0.00e+00 NNZ =      41 SPT = 0.9024 Tmvc_
↪= 0.000 T = 0.004
Site =      9 /      10 .. Mmpo =       1 DW = 0.00e+00 NNZ =      14 SPT = 0.0000 Tmvc_
↪= 0.000 T = 0.004
Ttotal =      0.103 Tmvc-total = 0.003 MPO bond dimension =      94 MaxDW = 0.00e+00
NNZ =      1317 SIZE =      27073 SPT = 0.9514

Rank =      0 Ttotal =      0.171 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 =      0.556 | 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 =      0.903 | E = -107.6541224475 | DE = -3.52e-12 | DW = 4.66e-20

Sweep =      2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |_
↪Dav threshold = 1.00e-10
Time elapsed =      1.295 | 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 =      1.686 | E = -107.6541224475 | DE = -5.97e-13 | DW = 5.90e-20

Sweep =      4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 |_
↪Dav threshold = 1.00e-10
Time elapsed =      2.136 | E = -107.6541224475 | DE = -5.68e-14 | DW = 2.02e-20

Sweep =      5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 |_
↪Dav threshold = 1.00e-10
Time elapsed =      2.523 | E = -107.6541224475 | DE = 0.00e+00 | DW = 4.86e-20

Sweep =      6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 |_
↪Dav threshold = 1.00e-10
Time elapsed =      2.946 | E = -107.6541224475 | DE = 2.84e-14 | DW = 3.08e-20

Sweep =      7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 |_
↪Dav threshold = 1.00e-10
Time elapsed =      3.342 | E = -107.6541224475 | DE = -2.84e-14 | DW = 5.58e-20

```

(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 =      3.607 | E = -107.6541224475 | DE = 0.00e+00 | DW = 2.33e-20

DMRG energy = -107.654122447524614

```

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.dmrp(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

integral symmetrize error = 1.5366482610151817e-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 = 0.018
Site =      1 /      10 .. Mmpo =      66 DW = 0.00e+00 NNZ =     143 SPT = 0.9167 Tmvc
↳ = 0.001 T = 0.014
Site =      2 /      10 .. Mmpo =     110 DW = 0.00e+00 NNZ =     283 SPT = 0.9610 Tmvc
↳ = 0.001 T = 0.012
Site =      3 /      10 .. Mmpo =     138 DW = 0.00e+00 NNZ =    1023 SPT = 0.9326 Tmvc
↳ = 0.002 T = 0.015
Site =      4 /      10 .. Mmpo =     158 DW = 0.00e+00 NNZ =     535 SPT = 0.9755 Tmvc
↳ = 0.001 T = 0.012
Site =      5 /      10 .. Mmpo =     186 DW = 0.00e+00 NNZ =     463 SPT = 0.9842 Tmvc
↳ = 0.001 T = 0.007

```

(continues on next page)

(continued from previous page)

```

Site =      6 /      10 .. Mmpo =    106 DW = 0.00e+00 NNZ =      415 SPT = 0.9790 Tmvc_
↪= 0.000 T = 0.006
Site =      7 /      10 .. Mmpo =     58 DW = 0.00e+00 NNZ =      163 SPT = 0.9735 Tmvc_
↪= 0.000 T = 0.004
Site =      8 /      10 .. Mmpo =     26 DW = 0.00e+00 NNZ =       87 SPT = 0.9423 Tmvc_
↪= 0.000 T = 0.003
Site =      9 /      10 .. Mmpo =      1 DW = 0.00e+00 NNZ =       26 SPT = 0.0000 Tmvc_
↪= 0.000 T = 0.003
Ttotal =      0.093 Tmvc-total = 0.007 MPO bond dimension =    186 MaxDW = 0.00e+00
NNZ =      3164 SIZE =      102772 SPT = 0.9692

Rank =      0 Ttotal =      0.141 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 =      0.783 | 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 =      1.233 | E =    -107.6541224475 | DE = -1.78e-11 | DW = 5.23e-09

Sweep =    2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |_
↪Dav threshold = 1.00e-10
Time elapsed =      1.662 | E =    -107.6541224475 | DE = -1.19e-12 | DW = 4.14e-08

Sweep =    3 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 |_
↪Dav threshold = 1.00e-10
Time elapsed =      2.104 | E =    -107.6541224475 | DE = 1.28e-12 | DW = 5.02e-09

Sweep =    4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 |_
↪Dav threshold = 1.00e-10
Time elapsed =      2.629 | E =    -107.6541224475 | DE = -6.82e-13 | DW = 3.61e-11

Sweep =    5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 |_
↪Dav threshold = 1.00e-10
Time elapsed =      3.139 | E =    -107.6541224475 | DE = 8.53e-13 | DW = 1.92e-19

Sweep =    6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 |_
↪Dav threshold = 1.00e-10
Time elapsed =      3.672 | 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 =      4.173 | E =    -107.6541224475 | DE = -1.68e-12 | DW = 2.25e-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 =      4.529 | E = -107.6541224475 | DE = 2.84e-14 | DW = 7.36e-20

DMRG energy = -107.654122447524529

```

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.dmrp(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

integral symmetrize error = 2.1082787907764326e-13
integral cutoff error = 0.0
mpo terms = 5914

Build MPO | Nsites = 20 | Nterms = 5914 | Algorithm = FastBIP | Cutoff = 1.
↳ 00e-20
Site = 0 / 20 .. Mmpo = 7 DW = 0.00e+00 NNZ = 7 SPT = 0.0000 Tmvc
↳ = 0.001 T = 0.007
Site = 1 / 20 .. Mmpo = 20 DW = 0.00e+00 NNZ = 19 SPT = 0.8643 Tmvc
↳ = 0.001 T = 0.009
Site = 2 / 20 .. Mmpo = 47 DW = 0.00e+00 NNZ = 49 SPT = 0.9479 Tmvc
↳ = 0.002 T = 0.010
Site = 3 / 20 .. Mmpo = 62 DW = 0.00e+00 NNZ = 251 SPT = 0.9139 Tmvc
↳ = 0.002 T = 0.013
Site = 4 / 20 .. Mmpo = 81 DW = 0.00e+00 NNZ = 273 SPT = 0.9456 Tmvc
↳ = 0.004 T = 0.019
Site = 5 / 20 .. Mmpo = 104 DW = 0.00e+00 NNZ = 357 SPT = 0.9576 Tmvc
↳ = 0.003 T = 0.013

```

(continues on next page)

(continued from previous page)

```

Site =    6 /    20 .. Mmpo =   129 DW = 0.00e+00 NNZ =    563 SPT = 0.9580 Tmvc_
↪= 0.001 T = 0.012
Site =    7 /    20 .. Mmpo =   126 DW = 0.00e+00 NNZ =   2318 SPT = 0.8574 Tmvc_
↪= 0.001 T = 0.014
Site =    8 /    20 .. Mmpo =   155 DW = 0.00e+00 NNZ =    208 SPT = 0.9893 Tmvc_
↪= 0.002 T = 0.012
Site =    9 /    20 .. Mmpo =   148 DW = 0.00e+00 NNZ =    686 SPT = 0.9701 Tmvc_
↪= 0.001 T = 0.014
Site =   10 /    20 .. Mmpo =   181 DW = 0.00e+00 NNZ =    388 SPT = 0.9855 Tmvc_
↪= 0.001 T = 0.010
Site =   11 /    20 .. Mmpo =   178 DW = 0.00e+00 NNZ =   720 SPT = 0.9777 Tmvc_
↪= 0.001 T = 0.011
Site =   12 /    20 .. Mmpo =   147 DW = 0.00e+00 NNZ =    496 SPT = 0.9810 Tmvc_
↪= 0.001 T = 0.006
Site =   13 /    20 .. Mmpo =   100 DW = 0.00e+00 NNZ =    412 SPT = 0.9720 Tmvc_
↪= 0.000 T = 0.004
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 = 0.003
Site =   16 /    20 .. Mmpo =    39 DW = 0.00e+00 NNZ =    124 SPT = 0.9411 Tmvc_
↪= 0.000 T = 0.003
Site =   17 /    20 .. Mmpo =    20 DW = 0.00e+00 NNZ =     76 SPT = 0.9026 Tmvc_
↪= 0.000 T = 0.004
Site =   18 /    20 .. Mmpo =     7 DW = 0.00e+00 NNZ =     22 SPT = 0.8429 Tmvc_
↪= 0.000 T = 0.003
Site =   19 /    20 .. Mmpo =     1 DW = 0.00e+00 NNZ =      7 SPT = 0.0000 Tmvc_
↪= 0.000 T = 0.003
Ttotal =    0.176 Tmvc-total = 0.023 MPO bond dimension =   181 MaxDW = 0.00e+00
NNZ =      7328 SIZE =      204350 SPT = 0.9641

Rank =      0 Ttotal =    0.269 MPO method = FastBipartite bond dimension =   181_
↪NNZ =      7328 SIZE =      204350 SPT = 0.9641

Sweep =    0 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |_
↪Dav threshold = 1.00e-10
Time elapsed =    1.012 | E =   -107.6541220013 | DW = 8.06e-08

Sweep =    1 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 |_
↪Dav threshold = 1.00e-10
Time elapsed =    1.754 | E =   -107.6541223288 | DE = -3.27e-07 | DW = 7.52e-08

Sweep =    2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |_
↪Dav threshold = 1.00e-10
Time elapsed =    2.474 | E =   -107.6541224307 | DE = -1.02e-07 | DW = 8.06e-08

```

(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 =    3.210 | E = -107.6541224307 | DE = 1.14e-12 | DW = 6.96e-08

Sweep =    4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    4.149 | E = -107.6541224313 | DE = -5.22e-10 | DW = 8.65e-11

Sweep =    5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    5.121 | E = -107.6541224313 | DE = 1.65e-12 | DW = 7.88e-20

Sweep =    6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    6.140 | E = -107.6541224313 | DE = -5.68e-14 | DW = 8.65e-11

Sweep =    7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    7.097 | E = -107.6541224313 | DE = -2.47e-12 | DW = 7.25e-20

Sweep =    8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 | _
↳ Dav threshold = 1.00e-09
Time elapsed =    7.676 | E = -107.6541224313 | DE = 0.00e+00 | DW = 8.25e-20

DMRG energy = -107.654122431266543

```

4.1.6 The SGB Mode

In this section, we try to solve the problem by first transforming 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, _

```

(continues on next page)

(continued from previous page)

```

↪ 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)
energy = driver.dmrp(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

integral symmetrize error = 2.140864794887064e-13
integral cutoff error = 0.0
mpo terms = 5984

Build MPO | Nsites = 20 | Nterms = 5984 | Algorithm = FastBIP | Cutoff = 1.
↪ 00e-20
Site = 0 / 20 .. Mmpo = 7 DW = 0.00e+00 NNZ = 7 SPT = 0.0000 Tmvc_
↪ = 0.001 T = 0.006
Site = 1 / 20 .. Mmpo = 20 DW = 0.00e+00 NNZ = 19 SPT = 0.8643 Tmvc_
↪ = 0.001 T = 0.009
Site = 2 / 20 .. Mmpo = 47 DW = 0.00e+00 NNZ = 49 SPT = 0.9479 Tmvc_
↪ = 0.002 T = 0.011
Site = 3 / 20 .. Mmpo = 62 DW = 0.00e+00 NNZ = 251 SPT = 0.9139 Tmvc_
↪ = 0.002 T = 0.014
Site = 4 / 20 .. Mmpo = 81 DW = 0.00e+00 NNZ = 273 SPT = 0.9456 Tmvc_
↪ = 0.002 T = 0.014
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_
↪ = 0.002 T = 0.016
Site = 7 / 20 .. Mmpo = 126 DW = 0.00e+00 NNZ = 2316 SPT = 0.8575 Tmvc_
↪ = 0.002 T = 0.015
Site = 8 / 20 .. Mmpo = 155 DW = 0.00e+00 NNZ = 214 SPT = 0.9890 Tmvc_
↪ = 0.001 T = 0.008
Site = 9 / 20 .. Mmpo = 148 DW = 0.00e+00 NNZ = 710 SPT = 0.9690 Tmvc_
↪ = 0.001 T = 0.012
Site = 10 / 20 .. Mmpo = 181 DW = 0.00e+00 NNZ = 398 SPT = 0.9851 Tmvc_
↪ = 0.001 T = 0.013
Site = 11 / 20 .. Mmpo = 178 DW = 0.00e+00 NNZ = 720 SPT = 0.9777 Tmvc_
↪ = 0.001 T = 0.013
Site = 12 / 20 .. Mmpo = 147 DW = 0.00e+00 NNZ = 496 SPT = 0.9810 Tmvc_
↪ = 0.001 T = 0.007
Site = 13 / 20 .. Mmpo = 100 DW = 0.00e+00 NNZ = 412 SPT = 0.9720 Tmvc_
↪ = 0.000 T = 0.005
Site = 14 / 20 .. Mmpo = 77 DW = 0.00e+00 NNZ = 254 SPT = 0.9670 Tmvc_
↪ = 0.000 T = 0.006
Site = 15 / 20 .. Mmpo = 54 DW = 0.00e+00 NNZ = 120 SPT = 0.9711 Tmvc_
↪ = 0.000 T = 0.004

```

(continues on next page)

(continued from previous page)

```

Site = 16 / 20 .. Mmpo = 39 DW = 0.00e+00 NNZ = 124 SPT = 0.9411 Tmvc_
↪ = 0.000 T = 0.003
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 = 0.177 Tmvc-total = 0.018 MPO bond dimension = 181 MaxDW = 0.00e+00
NNZ = 7388 SIZE = 204350 SPT = 0.9638

Rank = 0 Ttotal = 0.249 MPO method = FastBipartite bond dimension = 181_
↪ NNZ = 7388 SIZE = 204350 SPT = 0.9638

Sweep = 0 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |_
↪ Dav threshold = 1.00e-10
Time elapsed = 1.079 | E = -107.6541211530 | DW = 6.36e-08

Sweep = 1 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 |_
↪ Dav threshold = 1.00e-10
Time elapsed = 1.838 | E = -107.6541223677 | DE = -1.21e-06 | DW = 5.89e-08

Sweep = 2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |_
↪ Dav threshold = 1.00e-10
Time elapsed = 2.571 | E = -107.6541224370 | DE = -6.93e-08 | DW = 6.35e-08

Sweep = 3 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 |_
↪ Dav threshold = 1.00e-10
Time elapsed = 3.284 | E = -107.6541224370 | DE = 4.26e-13 | DW = 5.71e-08

Sweep = 4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 |_
↪ Dav threshold = 1.00e-10
Time elapsed = 4.581 | E = -107.6541224379 | DE = -9.81e-10 | DW = 9.19e-11

Sweep = 5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 |_
↪ Dav threshold = 1.00e-10
Time elapsed = 6.311 | E = -107.6541224379 | DE = 1.96e-12 | DW = 6.93e-20

Sweep = 6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 |_
↪ Dav threshold = 1.00e-10
Time elapsed = 7.757 | E = -107.6541224379 | DE = -5.68e-14 | DW = 9.19e-11

Sweep = 7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 |_
↪ Dav threshold = 1.00e-10
Time elapsed = 8.724 | E = -107.6541224379 | DE = -1.88e-12 | DW = 7.39e-20

```

(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 =      9.323 | E = -107.6541224379 | DE = -2.84e-14 | DW = 8.26e-20

DMRG energy = -107.654122437940899

```

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.dmrp(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 = 1.5501285354427146e-20
mpo terms = 44346

```

```

Build MPO | Nsites = 20 | Nterms = 44346 | 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 = 0.025
Site = 2 / 20 .. Mmpo = 63 DW = 0.00e+00 NNZ = 404 SPT = 0.7710 Tmvc
↳ = 0.006 T = 0.023
Site = 3 / 20 .. Mmpo = 78 DW = 0.00e+00 NNZ = 592 SPT = 0.8795 Tmvc
↳ = 0.006 T = 0.028

```

(continues on next page)

(continued from previous page)

```

Site =    4 /    20 .. Mmpo =    97 DW = 0.00e+00 NNZ =    932 SPT = 0.8768 Tmvc_
↪ = 0.010 T = 0.035
Site =    5 /    20 .. Mmpo =   120 DW = 0.00e+00 NNZ =   1315 SPT = 0.8870 Tmvc_
↪ = 0.008 T = 0.030
Site =    6 /    20 .. Mmpo =   147 DW = 0.00e+00 NNZ =   1722 SPT = 0.9024 Tmvc_
↪ = 0.008 T = 0.033
Site =    7 /    20 .. Mmpo =   178 DW = 0.00e+00 NNZ =   2142 SPT = 0.9181 Tmvc_
↪ = 0.012 T = 0.038
Site =    8 /    20 .. Mmpo =   213 DW = 0.00e+00 NNZ =   2597 SPT = 0.9315 Tmvc_
↪ = 0.009 T = 0.038
Site =    9 /    20 .. Mmpo =   252 DW = 0.00e+00 NNZ =   2985 SPT = 0.9444 Tmvc_
↪ = 0.011 T = 0.040
Site =   10 /    20 .. Mmpo =   213 DW = 0.00e+00 NNZ =  17958 SPT = 0.6654 Tmvc_
↪ = 0.014 T = 0.068
Site =   11 /    20 .. Mmpo =   178 DW = 0.00e+00 NNZ =   2590 SPT = 0.9317 Tmvc_
↪ = 0.003 T = 0.020
Site =   12 /    20 .. Mmpo =   147 DW = 0.00e+00 NNZ =   2184 SPT = 0.9165 Tmvc_
↪ = 0.003 T = 0.016
Site =   13 /    20 .. Mmpo =   120 DW = 0.00e+00 NNZ =   1783 SPT = 0.8989 Tmvc_
↪ = 0.002 T = 0.019
Site =   14 /    20 .. Mmpo =    97 DW = 0.00e+00 NNZ =   1397 SPT = 0.8800 Tmvc_
↪ = 0.002 T = 0.013
Site =   15 /    20 .. Mmpo =    78 DW = 0.00e+00 NNZ =   1008 SPT = 0.8668 Tmvc_
↪ = 0.001 T = 0.010
Site =   16 /    20 .. Mmpo =    63 DW = 0.00e+00 NNZ =    653 SPT = 0.8671 Tmvc_
↪ = 0.001 T = 0.010
Site =   17 /    20 .. Mmpo =    28 DW = 0.00e+00 NNZ =    494 SPT = 0.7200 Tmvc_
↪ = 0.001 T = 0.007
Site =   18 /    20 .. Mmpo =     9 DW = 0.00e+00 NNZ =     26 SPT = 0.8968 Tmvc_
↪ = 0.000 T = 0.003
Site =   19 /    20 .. Mmpo =     1 DW = 0.00e+00 NNZ =      9 SPT = 0.0000 Tmvc_
↪ = 0.000 T = 0.002
Ttotal =    0.475 Tmvc-total = 0.105 MPO bond dimension =   252 MaxDW = 0.00e+00
NNZ =    40823 SIZE =    323082 SPT = 0.8736

Rank =     0 Ttotal =    0.567 MPO method = FastBipartite bond dimension =   252_
↪ NNZ =    40823 SIZE =    323082 SPT = 0.8736

Sweep =    0 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |_
↪ Dav threshold = 1.00e-10
Time elapsed =   23.177 | E =  -107.6929206929 | DW = 4.40e-10

Sweep =    1 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 |_
↪ Dav threshold = 1.00e-10
Time elapsed =   35.078 | E =  -107.6929209450 | DE = -2.52e-07 | DW = 8.35e-10

```

(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 =    45.366 | E =    -107.6929209492 | DE = -4.14e-09 | DW = 2.09e-10

Sweep =    3 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    53.524 | E =    -107.6929209492 | DE = -3.41e-13 | DW = 8.36e-10

Sweep =    4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    71.358 | E =    -107.6929209492 | DE = -4.26e-13 | DW = 6.57e-20

Sweep =    5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    86.911 | E =    -107.6929209492 | DE = 2.84e-14 | DW = 1.18e-19

Sweep =    6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =   103.723 | E =    -107.6929209492 | DE = 2.84e-14 | DW = 7.77e-20

Sweep =    7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =   119.169 | E =    -107.6929209492 | DE = -2.84e-14 | DW = 1.19e-19

Sweep =    8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 | _
↳ Dav threshold = 1.00e-09
Time elapsed =   128.099 | E =    -107.6929209492 | DE = 2.84e-14 | DW = 8.33e-20

DMRG energy = -107.692920949170755

```

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)
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)

bond_dims = [250] * 4 + [500] * 4
noises = [1e-4] * 4 + [1e-5] * 4 + [0]
thrds = [1e-10] * 8

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.dmrp(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 = 0.0
mpo terms = 1881

Build MPO | Nsites = 10 | Nterms = 1881 | Algorithm = FastBIP | Cutoff = 1.
00e-20
Site = 0 / 10 .. Mmpo = 14 DW = 0.00e+00 NNZ = 14 SPT = 0.0000 Tmvc
= 0.001 T = 0.031
Site = 1 / 10 .. Mmpo = 34 DW = 0.00e+00 NNZ = 107 SPT = 0.7752 Tmvc
= 0.001 T = 0.054
Site = 2 / 10 .. Mmpo = 56 DW = 0.00e+00 NNZ = 189 SPT = 0.9007 Tmvc
= 0.001 T = 0.063
Site = 3 / 10 .. Mmpo = 66 DW = 0.00e+00 NNZ = 822 SPT = 0.7776 Tmvc
= 0.001 T = 0.063
Site = 4 / 10 .. Mmpo = 88 DW = 0.00e+00 NNZ = 154 SPT = 0.9735 Tmvc
= 0.000 T = 0.033
Site = 5 / 10 .. Mmpo = 94 DW = 0.00e+00 NNZ = 318 SPT = 0.9616 Tmvc
= 0.000 T = 0.031
Site = 6 / 10 .. Mmpo = 64 DW = 0.00e+00 NNZ = 236 SPT = 0.9608 Tmvc
= 0.000 T = 0.023
Site = 7 / 10 .. Mmpo = 40 DW = 0.00e+00 NNZ = 145 SPT = 0.9434 Tmvc
= 0.000 T = 0.016
Site = 8 / 10 .. Mmpo = 14 DW = 0.00e+00 NNZ = 49 SPT = 0.9125 Tmvc
= 0.000 T = 0.009
Site = 9 / 10 .. Mmpo = 1 DW = 0.00e+00 NNZ = 14 SPT = 0.0000 Tmvc
```

(continues on next page)

(continued from previous page)

```

↪ = 0.000 T = 0.007
Ttotal =      0.332 Tmvc-total = 0.005 MPO bond dimension =      94 MaxDW = 0.00e+00
NNZ =      2048 SIZE =      29320 SPT = 0.9302

Rank =      0 Ttotal =      0.402 MPO method = FastBipartite bond dimension =      94
↪ NNZ =      2048 SIZE =      29320 SPT = 0.9302

Sweep =      0 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |
↪ Dav threshold = 1.00e-10
Time elapsed =      0.679 | 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 =      1.030 | E = -107.6541224475 | DE = 1.19e-12 | DW = 2.77e-20

Sweep =      2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |
↪ Dav threshold = 1.00e-10
Time elapsed =      1.383 | 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 =      1.721 | E = -107.6541224475 | DE = -8.67e-12 | DW = 3.29e-20

Sweep =      4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 |
↪ Dav threshold = 1.00e-10
Time elapsed =      2.225 | E = -107.6541224475 | DE = -2.84e-14 | DW = 4.20e-20

Sweep =      5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 |
↪ Dav threshold = 1.00e-10
Time elapsed =      2.894 | E = -107.6541224475 | DE = 5.68e-14 | DW = 2.00e-20

Sweep =      6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 |
↪ Dav threshold = 1.00e-10
Time elapsed =      3.682 | E = -107.6541224475 | DE = -2.84e-14 | DW = 4.90e-20

Sweep =      7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 |
↪ Dav threshold = 1.00e-10
Time elapsed =      4.379 | E = -107.6541224475 | DE = 0.00e+00 | DW = 1.69e-20

Sweep =      8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 |
↪ Dav threshold = 1.00e-09
Time elapsed =      4.809 | E = -107.6541224475 | DE = 2.84e-14 | DW = 3.21e-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)
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)

bond_dims = [250] * 4 + [500] * 4
noises = [1e-4] * 4 + [1e-5] * 4 + [0]
thrds = [1e-10] * 8

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.dmrp(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 = 0.0
mpo terms = 5231

Build MPO | Nsites = 10 | Nterms = 5231 | Algorithm = FastBIP | Cutoff = 1.
00e-20
Site = 0 / 10 .. Mmpo = 30 DW = 0.00e+00 NNZ = 30 SPT = 0.0000 Tmvc
= 0.002 T = 0.080
Site = 1 / 10 .. Mmpo = 66 DW = 0.00e+00 NNZ = 271 SPT = 0.8631 Tmvc
= 0.001 T = 0.135
Site = 2 / 10 .. Mmpo = 110 DW = 0.00e+00 NNZ = 480 SPT = 0.9339 Tmvc
= 0.001 T = 0.136
Site = 3 / 10 .. Mmpo = 124 DW = 0.00e+00 NNZ = 2273 SPT = 0.8334 Tmvc
= 0.001 T = 0.135
Site = 4 / 10 .. Mmpo = 171 DW = 0.00e+00 NNZ = 363 SPT = 0.9829 Tmvc
= 0.001 T = 0.066
Site = 5 / 10 .. Mmpo = 186 DW = 0.00e+00 NNZ = 811 SPT = 0.9745 Tmvc
= 0.001 T = 0.072
Site = 6 / 10 .. Mmpo = 126 DW = 0.00e+00 NNZ = 583 SPT = 0.9751 Tmvc
= 0.001 T = 0.045
Site = 7 / 10 .. Mmpo = 82 DW = 0.00e+00 NNZ = 263 SPT = 0.9745 Tmvc
= 0.000 T = 0.024
Site = 8 / 10 .. Mmpo = 30 DW = 0.00e+00 NNZ = 182 SPT = 0.9260 Tmvc
= 0.000 T = 0.020
Site = 9 / 10 .. Mmpo = 1 DW = 0.00e+00 NNZ = 30 SPT = 0.0000 Tmvc
```

(continues on next page)

(continued from previous page)

```

↪ = 0.000 T = 0.009
Ttotal =      0.722 Tmvc-total = 0.009 MPO bond dimension =      186 MaxDW = 0.00e+00
NNZ =      5286 SIZE =      112178 SPT = 0.9529

Rank =      0 Ttotal =      0.773 MPO method = FastBipartite bond dimension =      186
↪ NNZ =      5286 SIZE =      112178 SPT = 0.9529

Sweep =      0 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |
↪ Dav threshold = 1.00e-10
Time elapsed =      1.157 | E =      -107.6541224475 | DW = 3.14e-08

Sweep =      1 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 |
↪ Dav threshold = 1.00e-10
Time elapsed =      1.836 | E =      -107.6541224475 | DE = -1.13e-11 | DW = 3.32e-08

Sweep =      2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |
↪ Dav threshold = 1.00e-10
Time elapsed =      2.477 | E =      -107.6541224455 | DE = 2.05e-09 | DW = 3.14e-08

Sweep =      3 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 |
↪ Dav threshold = 1.00e-10
Time elapsed =      3.168 | E =      -107.6541224455 | DE = -2.27e-13 | DW = 3.27e-08

Sweep =      4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 |
↪ Dav threshold = 1.00e-10
Time elapsed =      3.974 | E =      -107.6541224455 | DE = -1.42e-13 | DW = 3.08e-11

Sweep =      5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 |
↪ Dav threshold = 1.00e-10
Time elapsed =      4.811 | E =      -107.6541224455 | DE = 2.84e-14 | DW = 1.35e-19

Sweep =      6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 |
↪ Dav threshold = 1.00e-10
Time elapsed =      6.270 | E =      -107.6541224455 | DE = -2.84e-14 | DW = 3.08e-11

Sweep =      7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 |
↪ Dav threshold = 1.00e-10
Time elapsed =      7.758 | E =      -107.6541224455 | DE = 2.56e-13 | DW = 1.55e-19

Sweep =      8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 |
↪ Dav threshold = 1.00e-09
Time elapsed =      8.655 | E =      -107.6541224455 | DE = -2.84e-14 | DW = 9.59e-20

DMRG energy = -107.654122445468332

```

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 orbitals
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)

bond_dims = [250] * 4 + [500] * 4
noises = [1e-4] * 4 + [1e-5] * 4 + [0]
thrds = [1e-10] * 8
```

(continues on next page)

(continued from previous page)

```

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.dmrp(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  0 -1 -1  1  1  0  0  1  1 -1 -1  0  0]
integral symmetrize error = 9.305335353847714e-15
integral cutoff error = 5.457899320407648e-20
mpo terms = 12773

Build MPO | Nsites = 20 | Nterms = 12773 | Algorithm = FastBIP | Cutoff = 1.
↪ 00e-20
Site = 0 / 20 .. Mmpo = 9 DW = 0.00e+00 NNZ = 9 SPT = 0.0000 Tmvc_
↪ = 0.002 T = 0.098
Site = 1 / 20 .. Mmpo = 28 DW = 0.00e+00 NNZ = 25 SPT = 0.9008 Tmvc_
↪ = 0.002 T = 0.170
Site = 2 / 20 .. Mmpo = 47 DW = 0.00e+00 NNZ = 309 SPT = 0.7652 Tmvc_
↪ = 0.004 T = 0.245
Site = 3 / 20 .. Mmpo = 62 DW = 0.00e+00 NNZ = 357 SPT = 0.8775 Tmvc_
↪ = 0.003 T = 0.264
Site = 4 / 20 .. Mmpo = 81 DW = 0.00e+00 NNZ = 514 SPT = 0.8977 Tmvc_
↪ = 0.003 T = 0.280
Site = 5 / 20 .. Mmpo = 104 DW = 0.00e+00 NNZ = 667 SPT = 0.9208 Tmvc_
↪ = 0.003 T = 0.308
Site = 6 / 20 .. Mmpo = 129 DW = 0.00e+00 NNZ = 2106 SPT = 0.8430 Tmvc_
↪ = 0.003 T = 0.299
Site = 7 / 20 .. Mmpo = 118 DW = 0.00e+00 NNZ = 3960 SPT = 0.7399 Tmvc_
↪ = 0.003 T = 0.249
Site = 8 / 20 .. Mmpo = 153 DW = 0.00e+00 NNZ = 250 SPT = 0.9862 Tmvc_
↪ = 0.001 T = 0.127
Site = 9 / 20 .. Mmpo = 164 DW = 0.00e+00 NNZ = 690 SPT = 0.9725 Tmvc_
↪ = 0.001 T = 0.134
Site = 10 / 20 .. Mmpo = 185 DW = 0.00e+00 NNZ = 1223 SPT = 0.9597 Tmvc_
↪ = 0.002 T = 0.131
Site = 11 / 20 .. Mmpo = 178 DW = 0.00e+00 NNZ = 910 SPT = 0.9724 Tmvc_
↪ = 0.001 T = 0.101
Site = 12 / 20 .. Mmpo = 147 DW = 0.00e+00 NNZ = 814 SPT = 0.9689 Tmvc_
↪ = 0.001 T = 0.077
Site = 13 / 20 .. Mmpo = 120 DW = 0.00e+00 NNZ = 632 SPT = 0.9642 Tmvc_
↪ = 0.001 T = 0.052
Site = 14 / 20 .. Mmpo = 97 DW = 0.00e+00 NNZ = 383 SPT = 0.9671 Tmvc_
↪ = 0.001 T = 0.040
Site = 15 / 20 .. Mmpo = 78 DW = 0.00e+00 NNZ = 249 SPT = 0.9671 Tmvc_
↪ = 0.000 T = 0.038
Site = 16 / 20 .. Mmpo = 57 DW = 0.00e+00 NNZ = 382 SPT = 0.9141 Tmvc_

```

(continues on next page)

(continued from previous page)

```

↪= 0.000 T = 0.022
Site = 17 / 20 .. Mmpo = 28 DW = 0.00e+00 NNZ = 91 SPT = 0.9430 Tmvc↪
↪= 0.000 T = 0.009
Site = 18 / 20 .. Mmpo = 9 DW = 0.00e+00 NNZ = 28 SPT = 0.8889 Tmvc↪
↪= 0.000 T = 0.011
Site = 19 / 20 .. Mmpo = 1 DW = 0.00e+00 NNZ = 9 SPT = 0.0000 Tmvc↪
↪= 0.000 T = 0.004
Ttotal = 2.660 Tmvc-total = 0.031 MPO bond dimension = 185 MaxDW = 0.00e+00
NNZ = 13608 SIZE = 222306 SPT = 0.9388

Rank = 0 Ttotal = 2.764 MPO method = FastBipartite bond dimension = 185↪
↪NNZ = 13608 SIZE = 222306 SPT = 0.9388

Sweep = 0 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |↪
↪Dav threshold = 1.00e-10
Time elapsed = 1.655 | E = -107.6541219864 | DW = 1.34e-08

Sweep = 1 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 |↪
↪Dav threshold = 1.00e-10
Time elapsed = 3.673 | E = -107.6541222928 | DE = -3.06e-07 | DW = 4.92e-08

Sweep = 2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |↪
↪Dav threshold = 1.00e-10
Time elapsed = 5.224 | E = -107.6541224418 | DE = -1.49e-07 | DW = 1.33e-08

Sweep = 3 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 |↪
↪Dav threshold = 1.00e-10
Time elapsed = 6.259 | E = -107.6541224418 | DE = -1.02e-12 | DW = 4.94e-08

Sweep = 4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 |↪
↪Dav threshold = 1.00e-10
Time elapsed = 7.849 | E = -107.6541224449 | DE = -3.13e-09 | DW = 6.76e-12

Sweep = 5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 |↪
↪Dav threshold = 1.00e-10
Time elapsed = 9.282 | E = -107.6541224455 | DE = -5.68e-10 | DW = 5.49e-20

Sweep = 6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 |↪
↪Dav threshold = 1.00e-10
Time elapsed = 10.873 | E = -107.6541224455 | DE = 5.68e-14 | DW = 6.76e-12

Sweep = 7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 |↪
↪Dav threshold = 1.00e-10
Time elapsed = 12.297 | E = -107.6541224455 | DE = 2.84e-14 | DW = 6.04e-20

```

(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 =      13.165 | E =      -107.6541224455 | DE = 2.84e-14 | DW = 7.22e-20

DMRG energy = -107.654122445469270

```

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.dmrp(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 =    0.557 | 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 =    0.833 | E = -106.9391328597 | DE = -3.24e-12 | DW = 9.75e-19

Sweep =    2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 | _
↪ Dav threshold = 1.00e-10
Time elapsed =    1.130 | E = -106.9391328597 | DE = 2.84e-14 | DW = 3.38e-10

Sweep =    3 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 | _
↪ Dav threshold = 1.00e-10
Time elapsed =    1.421 | E = -106.9391328597 | DE = 1.71e-13 | DW = 1.52e-18

Sweep =    4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↪ Dav threshold = 1.00e-10
Time elapsed =    1.755 | E = -106.9391328597 | DE = 0.00e+00 | DW = 1.12e-16

Sweep =    5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↪ Dav threshold = 1.00e-10
Time elapsed =    2.063 | E = -106.9391328597 | DE = 5.68e-14 | DW = 1.40e-19

Sweep =    6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↪ Dav threshold = 1.00e-10
Time elapsed =    2.402 | E = -106.9391328597 | DE = 0.00e+00 | DW = 1.12e-16

Sweep =    7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↪ Dav threshold = 1.00e-10
Time elapsed =    2.709 | E = -106.9391328597 | DE = 0.00e+00 | DW = 1.38e-19

Sweep =    8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 | _
↪ Dav threshold = 1.00e-09
Time elapsed =    2.933 | E = -106.9391328597 | DE = 0.00e+00 | DW = 3.39e-20

DMRG energy = -106.939132859666600
Norm = 1.0000000000000000
Energy expectation = -106.93913285966614
<S^2> expectation = 2.0000000000000000

```

We can also evaluate expectation of arbitray operator such as the occupancy in the first orbital

$$\hat{N}_0 = a_{0\alpha}^\dagger a_{0\alpha} + a_{0\beta}^\dagger a_{0\beta} = \sqrt{2}(a_0^\dagger)^{[1/2]} \otimes_{[0]} (a_0)^{[1/2]}$$

```
[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 =      1.999995824361892
```

We can compute the 3PDM and compare the result with the FCI 3PDM. Note that in pyscf the 3PDM is defined as

$$DM_{ijklmn} := \langle E_{ij} E_{kl} E_{mn} \rangle$$

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 =  4.622993128790701e-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)
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.dmrp(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 =    0.639 | 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 =    1.016 | E = -107.6541224475 | DE = -3.38e-12 | DW = 3.81e-20

Sweep =    2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    1.407 | 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 =    1.777 | E = -107.6541224475 | DE = -5.68e-13 | DW = 4.69e-20

Sweep =    4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    2.209 | E = -107.6541224475 | DE = 0.00e+00 | DW = 1.73e-20

Sweep =    5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    2.604 | E = -107.6541224475 | DE = 2.84e-14 | DW = 5.35e-20

Sweep =    6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    3.011 | E = -107.6541224475 | DE = 0.00e+00 | DW = 1.66e-20
```

(continues on next page)

(continued from previous page)

```

Sweep =    7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 |
↳ Dav threshold = 1.00e-10
Time elapsed =    3.376 | E = -107.6541224475 | DE = 0.00e+00 | DW = 5.98e-20

Sweep =    8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 |
↳ Dav threshold = 1.00e-09
Time elapsed =    3.647 | E = -107.6541224475 | DE = 2.84e-14 | DW = 5.56e-20

DMRG energy = -107.654122447524614
dtrie finished 0.01697694599999977
Number of CSF =          9 (cutoff =    0.05)
Sum of weights of included CSF =    0.984368811359554

CSF      0 2222222000 =    0.957506528669401
CSF      1 2222202200 =   -0.131287867807394
CSF      2 2222022020 =   -0.131287794595557
CSF      3 2222+-2+-0 =    0.118594301088608
CSF      4 2222++2--0 =   -0.084968224982022
CSF      5 2220222020 =   -0.054710635444627
CSF      6 2220222200 =   -0.054710465272207
CSF      7 2222+2+0-- =    0.053881241407894
CSF      8 22222+-0- =    0.053881215166769

```

Extracting determinant coefficients from 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.dmrp(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)

```

(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 =    1.373 | 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 =    2.089 | E =    -107.6541224475 | DE = -2.07e-12 | DW = 5.05e-09

Sweep =    2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    2.609 | E =    -107.6541224475 | DE = -1.34e-12 | DW = 4.14e-08

Sweep =    3 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    3.051 | E =    -107.6541224475 | DE = 1.28e-12 | DW = 5.21e-09

Sweep =    4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    3.572 | E =    -107.6541224475 | DE = -9.95e-13 | DW = 3.60e-11

Sweep =    5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    4.085 | E =    -107.6541224475 | DE = 8.81e-13 | DW = 1.37e-19

Sweep =    6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    4.625 | 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 =    5.142 | E =    -107.6541224475 | DE = -1.28e-12 | DW = 1.52e-19

Sweep =    8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 | _
↳ Dav threshold = 1.00e-09
Time elapsed =    5.505 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 8.34e-20

DMRG energy = -107.654122447524443
dtrie finished 0.02674672699998837
Number of DET =          7 (cutoff =          0.05)
Sum of weights of included DET =    0.971331638354918

DET          0 2222222000 =    0.957506568620143
DET          1 2222022020 =   -0.131287835040785
DET          2 2222202200 =   -0.131287814797523
DET          3 2222ab2ab0 =    0.083825279692261

```

(continues on next page)

(continued from previous page)

DET	4	2222ba2ba0	=	0.083825279691480
DET	5	2220222200	=	-0.054710476471258
DET	6	2220222020	=	-0.054710439530678

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 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.dmrp(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.dmrp(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 = 0.496 | E = -107.6541224475 | DW = 1.87e-10
```

(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 =    0.733 | E =    -107.6541224475 | DE = -1.24e-11 | DW = 4.89e-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 =    1.208 | E =    -107.6541224475 | DE = -1.36e-12 | DW = 3.97e-20

Sweep =    4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    1.488 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 1.48e-20

Sweep =    5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    1.713 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 6.11e-20

Sweep =    6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    1.974 | E =    -107.6541224475 | DE = 5.68e-14 | DW = 2.41e-20

Sweep =    7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    2.209 | E =    -107.6541224475 | DE = -2.84e-14 | DW = 6.14e-20

Sweep =    8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 | _
↳ Dav threshold = 1.00e-09
Time elapsed =    2.482 | E =    -107.6541224475 | DE = 5.68e-14 | DW = 2.91e-20

DMRG energy = -107.654122447524358
dtrie finished 0.016507582999963688
Number of CSF =          9 (cutoff =      0.05)
Sum of weights of included CSF =    0.984368809657036

CSF      0 2222222000 =    0.957506521875127
CSF      1 2222202200 =   -0.131287813613834
CSF      2 2222022020 =   -0.131287676854091
CSF      3 2222+-2+-0 =    0.118594392287595
CSF      4 2222++2--0 =   -0.084968279487231
CSF      5 2220222020 =   -0.054710673444202
CSF      6 2220222200 =   -0.054710596031681
CSF      7 2222+2+0-- =    0.053881289057939

```

(continues on next page)

(continued from previous page)

```

CSF          8 22222+-0- =    0.053881233355398
0.9843688096570361
-107.6155366353171

Sweep =    0 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↪ Dav threshold = 1.00e-10
Time elapsed =    0.054 | E =   -107.6285084535 | DW = 3.52e-21

Sweep =    1 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↪ Dav threshold = 1.00e-10
Time elapsed =    0.109 | E =   -107.6288109091 | DE = -3.02e-04 | DW = 9.04e-21

Sweep =    2 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↪ Dav threshold = 1.00e-10
Time elapsed =    0.164 | E =   -107.6289019710 | DE = -9.11e-05 | DW = 4.57e-21

Sweep =    3 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↪ Dav threshold = 1.00e-10
Time elapsed =    0.217 | E =   -107.6289019713 | DE = -3.35e-10 | DW = 2.07e-20

Ground state energy = -107.628901971339317

```

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.dmrp(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))

```

(continues on next page)

(continued from previous page)

```
print(driver.expectation(mps, mpo, mps) / driver.expectation(mps, impo, mps))
```

```
energy = driver.dmrp(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 = 0.809 | 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 = 1.294 | E = -107.6541224475 | DE = -5.80e-12 | DW = 5.17e-09
```

```
Sweep = 2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 | _
```

```
↳ Dav threshold = 1.00e-10
```

```
Time elapsed = 1.713 | E = -107.6541224475 | DE = -9.38e-13 | DW = 4.15e-08
```

```
Sweep = 3 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 | _
```

```
↳ Dav threshold = 1.00e-10
```

```
Time elapsed = 2.170 | E = -107.6541224475 | DE = 1.36e-12 | DW = 5.37e-09
```

```
Sweep = 4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
```

```
↳ Dav threshold = 1.00e-10
```

```
Time elapsed = 2.687 | E = -107.6541224475 | DE = -1.36e-12 | DW = 3.60e-11
```

```
Sweep = 5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
```

```
↳ Dav threshold = 1.00e-10
```

```
Time elapsed = 3.212 | E = -107.6541224475 | DE = 1.05e-12 | DW = 1.85e-19
```

```
Sweep = 6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
```

```
↳ Dav threshold = 1.00e-10
```

```
Time elapsed = 3.737 | 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 = 4.252 | E = -107.6541224475 | DE = -1.14e-12 | DW = 1.77e-19
```

```
Sweep = 8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 | _
```

```
↳ Dav threshold = 1.00e-09
```

```
Time elapsed = 4.598 | E = -107.6541224475 | DE = 0.00e+00 | DW = 8.68e-20
```

```
DMRG energy = -107.654122447524529
```

```
dtrie finished 0.03569589099998893
```

```
Number of DET = 7 (cutoff = 0.05)
```

(continues on next page)

(continued from previous page)

```

Sum of weights of included DET =    0.971331637808218

DET      0 2222222000 =  -0.957506566009713
DET      1 2222022020 =   0.131287836810967
DET      2 2222202200 =   0.131287818787713
DET      3 2222ab2ab0 =  -0.083825284176507
DET      4 2222ba2ba0 =  -0.083825284073794
DET      5 2220222200 =   0.054710477807739
DET      6 2220222020 =   0.054710451475846
0.971331637808218
-107.59447325662364

Sweep =    0 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    0.047 | E =    -107.6101211360 | DW = 6.22e-21

Sweep =    1 | Direction =  forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    0.104 | E =    -107.6102787809 | DE = -1.58e-04 | DW = 8.84e-21

Sweep =    2 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    0.163 | E =    -107.6104759466 | DE = -1.97e-04 | DW = 1.22e-20

Sweep =    3 | Direction =  forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    0.218 | E =    -107.6104759470 | DE = -4.45e-10 | DW = 1.18e-20

Ground state energy = -107.610475947024796

```

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)
```

(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)

bias = 0.1 # make it more random
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.dmrp(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 =    0.044 | E = -107.5033999317 | DW = 2.59e-21

Sweep =    1 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    0.135 | E = -107.5830591524 | DE = -7.97e-02 | DW = 3.51e-21

Sweep =    2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    0.217 | E = -107.5836402922 | DE = -5.81e-04 | DW = 9.17e-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 = 1.12e-20

Sweep =    4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    0.371 | 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 =    0.443 | E = -107.5868152464 | DE = -9.07e-06 | DW = 1.23e-20

Sweep =    6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    0.517 | E = -107.5868152464 | DE = -5.68e-14 | DW = 1.45e-20

```

(continues on next page)

(continued from previous page)

```

Sweep =    7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    0.589 | E = -107.5868152464 | DE = 2.84e-14 | DW = 8.04e-21

Sweep =    8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 | _
↳ Dav threshold = 1.00e-09
Time elapsed =    0.655 | E = -107.5868152464 | DE = 2.84e-14 | DW = 1.69e-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.dmrp(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 =    0.097 | E = -107.5033999316 | DW = 3.37e-21

Sweep =    1 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    0.267 | E = -107.5830606948 | DE = -7.97e-02 | DW = 7.61e-21

Sweep =    2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 | _
↳ Dav threshold = 1.00e-10
Time elapsed =    0.376 | E = -107.5836402307 | DE = -5.80e-04 | DW = 1.27e-20

Sweep =    3 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 | _

```

(continues on next page)

(continued from previous page)

```

↳Dav threshold = 1.00e-10
Time elapsed = 0.505 | E = -107.5867932582 | DE = -3.15e-03 | DW = 1.21e-20

Sweep = 4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳Dav threshold = 1.00e-10
Time elapsed = 0.624 | E = -107.5868061673 | DE = -1.29e-05 | DW = 1.84e-20

Sweep = 5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳Dav threshold = 1.00e-10
Time elapsed = 0.847 | E = -107.5868152464 | DE = -9.08e-06 | DW = 1.03e-20

Sweep = 6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳Dav threshold = 1.00e-10
Time elapsed = 1.008 | E = -107.5868152464 | DE = 0.00e+00 | DW = 1.88e-20

Sweep = 7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳Dav threshold = 1.00e-10
Time elapsed = 1.132 | E = -107.5868152464 | DE = 0.00e+00 | DW = 1.76e-20

Sweep = 8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 | _
↳Dav threshold = 1.00e-09
Time elapsed = 1.218 | E = -107.5868152464 | DE = 0.00e+00 | DW = 1.33e-20

DMRG energy = -107.586815246376730

```

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.dmrp(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 = 0.853 | E = -107.6538893531 | DW = 2.24e-08

Sweep = 1 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 1.548 | E = -107.6539226742 | DE = -3.33e-05 | DW = 8.93e-09

Sweep = 2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 2.168 | E = -107.6539462697 | DE = -2.36e-05 | DW = 5.04e-08

Sweep = 3 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 2.808 | E = -107.6539506829 | DE = -4.41e-06 | DW = 1.95e-08

Sweep = 4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 3.596 | E = -107.6539506829 | DE = 3.24e-12 | DW = 1.84e-11

Sweep = 5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 4.386 | E = -107.6539506829 | DE = -4.12e-12 | DW = 5.67e-20

Sweep = 6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 5.236 | E = -107.6539506829 | DE = -2.84e-14 | DW = 1.84e-11

Sweep = 7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 6.014 | E = -107.6539506829 | DE = 2.05e-12 | DW = 4.05e-20

Sweep = 8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 | _
↳ Dav threshold = 1.00e-09
Time elapsed = 6.497 | E = -107.6539506829 | DE = 0.00e+00 | DW = 7.66e-20

DMRG energy = -107.653950682884570
```

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.dmrp(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 = 0.902 | 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 = 1.281 | E = -107.6541224475 | DE = -2.87e-12 | DW = 4.49e-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 =      1.682 | 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 =      2.068 | E =      -107.6541224475 | DE = -5.12e-13 | DW = 5.07e-20

Sweep =      4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =      2.610 | E =      -107.6541224475 | DE = -5.68e-14 | DW = 2.62e-20

Sweep =      5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =      3.019 | E =      -107.6541224475 | DE = 2.84e-14 | DW = 4.59e-20

Sweep =      6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =      3.435 | E =      -107.6541224475 | DE = 2.84e-14 | DW = 2.01e-20

Sweep =      7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed =      3.808 | E =      -107.6541224475 | DE = -5.68e-14 | DW = 3.69e-20

Sweep =      8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 | _
↳ Dav threshold = 1.00e-09
Time elapsed =      4.095 | E =      -107.6541224475 | DE = 2.84e-14 | DW = 3.68e-20

DMRG energy = -107.654122447524671
dtrie finished 0.026360811000017748
Number of CSF =          9 (cutoff =      0.05)
Sum of weights of included CSF =    0.984368806039259

CSF      0 2222222000 = -0.957506495595307
CSF      1 2222022020 =  0.131287987564377
CSF      2 2222202200 =  0.131287965813458
CSF      3 2222+-2+-0 = -0.118594461577159
CSF      4 2222++2--0 =  0.084967942537817
CSF      5 2220222020 =  0.054710523921704
CSF      6 2220222200 =  0.054710517406346
CSF      7 2222+2+0-- = -0.053881222618422
CSF      8 22222+-0- = -0.053881215803966
-107.65412244752456
dtrie finished 0.0680388100000755
Number of DET =          7 (cutoff =      0.05)
Sum of weights of included DET =    0.971331619872548

```

(continues on next page)

(continued from previous page)

DET	0	2222222000	=	-0.957506495595307
DET	1	2222022020	=	0.131287987564377
DET	2	2222022000	=	0.131287965813458
DET	3	2222ab2ab0	=	-0.083825363036928
DET	4	2222ba2ba0	=	-0.083825363036928
DET	5	2220222020	=	0.054710523921704
DET	6	2220222000	=	0.054710517406346

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.dmrp(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))
```

(continues on next page)

(continued from previous page)

```

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 = 0.804 | 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 = 1.144 | E = -107.6541224475 | DE = -3.95e-12 | DW = 4.49e-20

Sweep = 2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 1.379 | 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 = 1.601 | E = -107.6541224475 | DE = -7.67e-13 | DW = 5.89e-20

Sweep = 4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 1.868 | E = -107.6541224475 | DE = -2.84e-14 | DW = 2.31e-20

Sweep = 5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 2.110 | E = -107.6541224475 | DE = -2.84e-14 | DW = 6.59e-20

Sweep = 6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 2.366 | E = -107.6541224475 | DE = 0.00e+00 | DW = 8.80e-21

Sweep = 7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 2.606 | E = -107.6541224475 | DE = -5.68e-14 | DW = 6.06e-20

Sweep = 8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 | _
↳ Dav threshold = 1.00e-09
Time elapsed = 2.780 | E = -107.6541224475 | DE = 2.84e-14 | DW = 4.42e-20

DMRG energy = -107.654122447524671
1.0
-107.65412244752456
integral symmetrize error = 1.76819336753967e-14
integral cutoff error = 0.0
mpo terms = 1030

```

(continues on next page)

(continued from previous page)

```

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 = 0.007
Site =    1 /    10 .. Mmpo =    34 DW = 0.00e+00 NNZ =    63 SPT = 0.8575 Tmvc_
↪ = 0.001 T = 0.005
Site =    2 /    10 .. Mmpo =    56 DW = 0.00e+00 NNZ =   121 SPT = 0.9364 Tmvc_
↪ = 0.000 T = 0.006
Site =    3 /    10 .. Mmpo =    74 DW = 0.00e+00 NNZ =   373 SPT = 0.9100 Tmvc_
↪ = 0.000 T = 0.006
Site =    4 /    10 .. Mmpo =    80 DW = 0.00e+00 NNZ =   269 SPT = 0.9546 Tmvc_
↪ = 0.000 T = 0.007
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 = 0.008
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 = 0.004
Site =    9 /    10 .. Mmpo =     1 DW = 0.00e+00 NNZ =    14 SPT = 0.0000 Tmvc_
↪ = 0.000 T = 0.002
Ttotal =    0.054 Tmvc-total = 0.003 MPO bond dimension =    94 MaxDW = 0.00e+00
NNZ =    1317 SIZE =    27073 SPT = 0.9514

Rank =     0 Ttotal =    0.108 MPO method = FastBipartite bond dimension =    94_
↪ NNZ =    1317 SIZE =    27073 SPT = 0.9514
(0.9999999999999999+0j)
(-107.65412244752457+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)

```

(continues on next page)

(continued from previous page)

```

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
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 =    1.189 | 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 =    2.240 | E = -107.6541224475 | DE = -1.41e-11 | DW = 2.95e-19

Sweep =    2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 | _
↪Dav threshold = 1.00e-10
Time elapsed =    3.105 | 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 =    3.655 | E = -107.6541224475 | DE = -2.05e-12 | DW = 1.13e-19

Sweep =    4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↪Dav threshold = 1.00e-10
Time elapsed =    4.299 | E = -107.6541224475 | DE = 0.00e+00 | DW = 1.31e-19

Sweep =    5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↪Dav threshold = 1.00e-10
Time elapsed =    4.784 | E = -107.6541224475 | DE = -2.84e-14 | DW = 1.06e-19

Sweep =    6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↪Dav threshold = 1.00e-10
Time elapsed =    5.305 | E = -107.6541224475 | DE = 5.68e-14 | DW = 1.24e-19

```

(continues on next page)

(continued from previous page)

```

Sweep =    7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 |
↳ Dav threshold = 1.00e-10
Time elapsed =    5.770 | E =    -107.6541224475 | DE = -5.68e-14 | DW = 1.04e-19

Sweep =    8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 |
↳ Dav threshold = 1.00e-09
Time elapsed =    6.086 | E =    -107.6541224475 | DE = 0.00e+00 | DW = 4.25e-20

DMRG energy = -107.654122447523960
(1.0000000000000002-7.905175135209388e-18j)
(-107.6541224475238-8.610748230937272e-16j)
integral symmetrize error = 1.6922854083945986e-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 = 0.004
Site =    1 /    10 .. Mmpo =    34 DW = 0.00e+00 NNZ =    63 SPT = 0.8575 Tmvc
↳ = 0.001 T = 0.004
Site =    2 /    10 .. Mmpo =    56 DW = 0.00e+00 NNZ =   121 SPT = 0.9364 Tmvc
↳ = 0.000 T = 0.004
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 = 0.004
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 = 0.003
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 = 0.002
Ttotal =    0.036 Tmvc-total = 0.003 MPO bond dimension =    94 MaxDW = 0.00e+00
NNZ =    1317 SIZE =    27073 SPT = 0.9514

Rank =    0 Ttotal =    0.064 MPO method = FastBipartite bond dimension =    94
↳ NNZ =    1317 SIZE =    27073 SPT = 0.9514
0.9626534962131837
-107.65412244752437

```

4.1.15 MPS Bipartite Entanglement

We can get 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 mode, where Λ_k are singular values in the bond at site k .

```
[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)
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.dmrp(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
    thrds=thrds, iprint=1)
print('DMRG energy = %20.15f' % energy)

bip_ent = driver.get_bipartite_entanglement()

import matplotlib.pyplot as plt
plt.plot(np.arange(len(bip_ent)), bip_ent, linestyle='-', marker='o',
    mfc='white', mec="#7FB685", color="#7FB685")
plt.xlabel("site index $k$")
plt.ylabel("bipartite entanglement $S_k$")
plt.show()

Sweep = 0 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |
↳ Dav threshold = 1.00e-10
Time elapsed = 0.769 | 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 = 1.254 | E = -107.6541224475 | DE = -1.43e-11 | DW = 5.08e-09

Sweep = 2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |
↳ Dav threshold = 1.00e-10
Time elapsed = 1.670 | E = -107.6541224475 | DE = -5.12e-13 | DW = 4.14e-08

Sweep = 3 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 |
↳ Dav threshold = 1.00e-10
Time elapsed = 2.130 | E = -107.6541224475 | DE = 1.31e-12 | DW = 5.21e-09
```

(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 = 2.655 | E = -107.6541224475 | DE = -1.14e-12 | DW = 3.60e-11

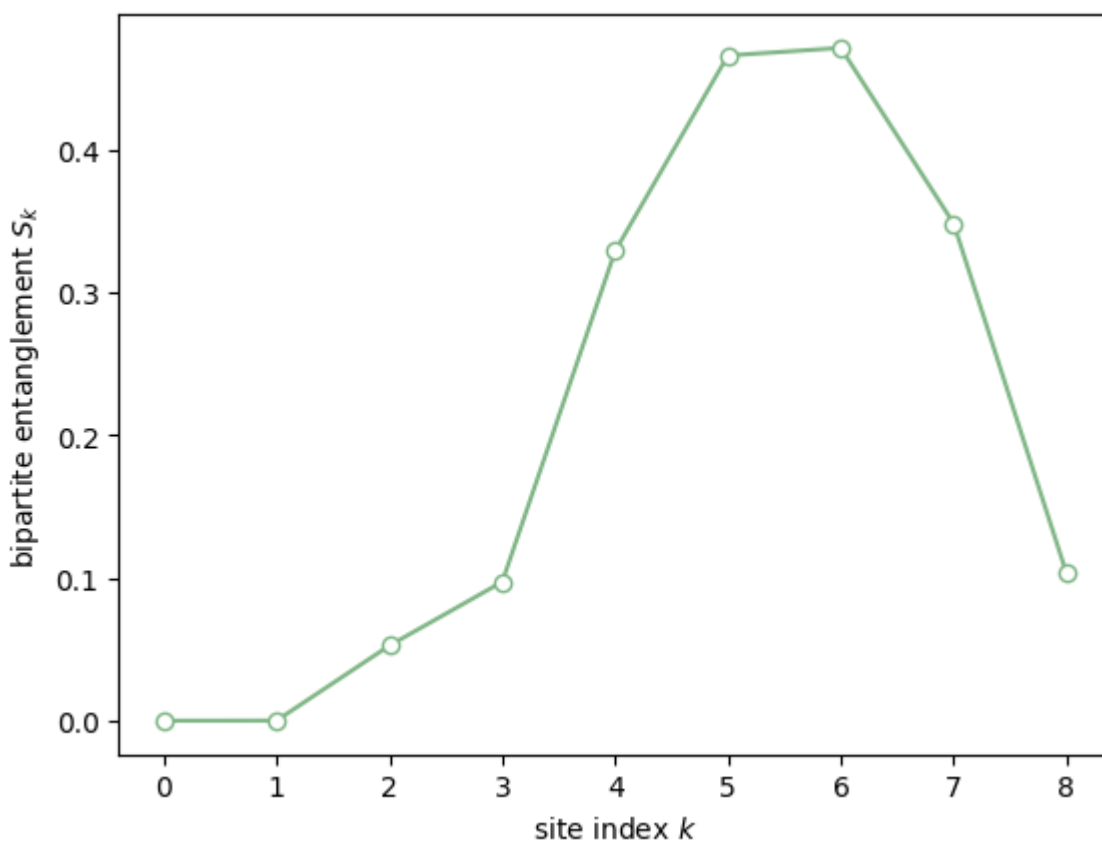
Sweep = 5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | └
↳ Dav threshold = 1.00e-10
Time elapsed = 3.189 | E = -107.6541224475 | DE = 9.38e-13 | DW = 1.75e-19

Sweep = 6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | └
↳ Dav threshold = 1.00e-10
Time elapsed = 3.722 | 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 = 4.245 | E = -107.6541224475 | DE = -1.22e-12 | DW = 1.54e-19

Sweep = 8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 | └
↳ Dav threshold = 1.00e-09
Time elapsed = 4.595 | E = -107.6541224475 | DE = 2.84e-14 | DW = 7.71e-20

DMRG energy = -107.654122447524443



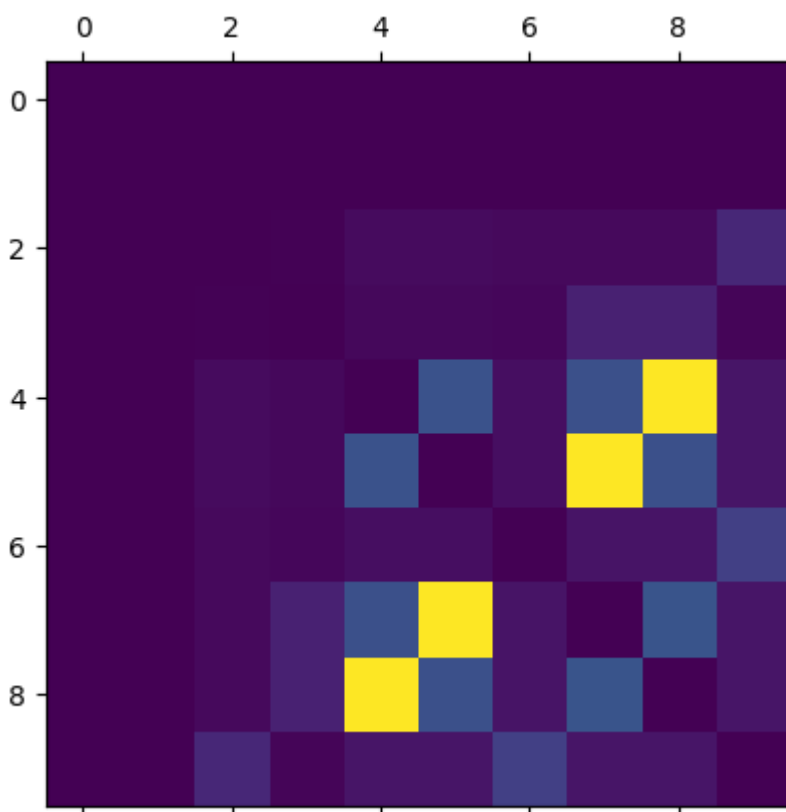
4.1.16 Orbital Entropy and Mutual Information

For the optimized MPS in the SZ mode, we can compute the 1- and 2- orbital density matrices and mutual information for pairs of orbitals.

```
[29]: odm1 = driver.get_orbital_entropies(ket, orb_type=1)
      odm2 = driver.get_orbital_entropies(ket, orb_type=2)
      minfo = 0.5 * (odm1[:, None] + odm1[None, :] - odm2) * (1 - np.identity(len(odm1)))

      import matplotlib.pyplot as plt
      plt.matshow(minfo)
```

```
[29]: <matplotlib.image.AxesImage at 0x7d0b6521bd30>
```



4.1.17 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.

```
[30]: 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)
```

(continues on next page)

(continued from previous page)

```

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.dmrp(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):
    energy = driver.dmrp(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 = 2.037 | E[ 3] = -107.6541193407 -107.0314407783 -106.
    ↪ 9595996333 | DW = 8.51e-05

Sweep = 1 | Direction = backward | Bond dimension = 100 | Noise = 1.00e-05 |
    ↪ Dav threshold = 1.00e-10
Time elapsed = 3.220 | E[ 3] = -107.6541193407 -107.0314407783 -106.
    ↪ 9595996332 | DE = 7.11e-12 | DW = 1.62e-04

Sweep = 2 | Direction = forward | Bond dimension = 100 | Noise = 1.00e-05 |
    ↪ Dav threshold = 1.00e-10
Time elapsed = 4.616 | E[ 3] = -107.6541138874 -107.0314486179 -106.
    ↪ 9594564026 | DE = 1.43e-04 | DW = 7.20e-05

Sweep = 3 | Direction = backward | Bond dimension = 100 | Noise = 1.00e-05 |
    ↪ Dav threshold = 1.00e-10
Time elapsed = 5.853 | E[ 3] = -107.6541138874 -107.0314486178 -106.
    ↪ 9594564026 | DE = -3.97e-11 | DW = 1.47e-04

Sweep = 4 | Direction = forward | Bond dimension = 100 | Noise = 0.00e+00 |
    ↪ Dav threshold = 1.00e-10
Time elapsed = 6.621 | E[ 3] = -107.6540972369 -107.0314487271 -106.

```

(continues on next page)

(continued from previous page)

```

↪9594391421 | 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 =      7.410 | E[ 3] =      -107.6540972369      -107.0314487271      -106.
↪9594391422 | DE = -1.72e-11 | DW = 1.43e-04

State-averaged MPS energies = [-107.654097236905088 -107.031448727142006 -106.
↪959439142163390]
State-specific MPS E[0] = -107.654122447496988
State-specific MPS E[1] = -107.031449471612873
State-specific MPS E[2] = -106.959626154678844

```

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,

```

(continues on next page)

(continued from previous page)

```

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 increases). 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.dmrp(mpo, ket, n_sweeps=28, bond_dims=bond_dims, noises=noises,
                    thrds=thrds, iprint=1, twosite_to_onsite=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 | _

```

(continues on next page)

(continued from previous page)

```

↪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

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

```

(continues on next page)

(continued from previous page)

```

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

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

```

(continues on next page)

(continued from previous page)

```

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 folder, 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 obtained 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.dmrp(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

```

(continues on next page)

(continued from previous page)

```

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
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

```

(continues on next page)

(continued from previous page)

```

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))

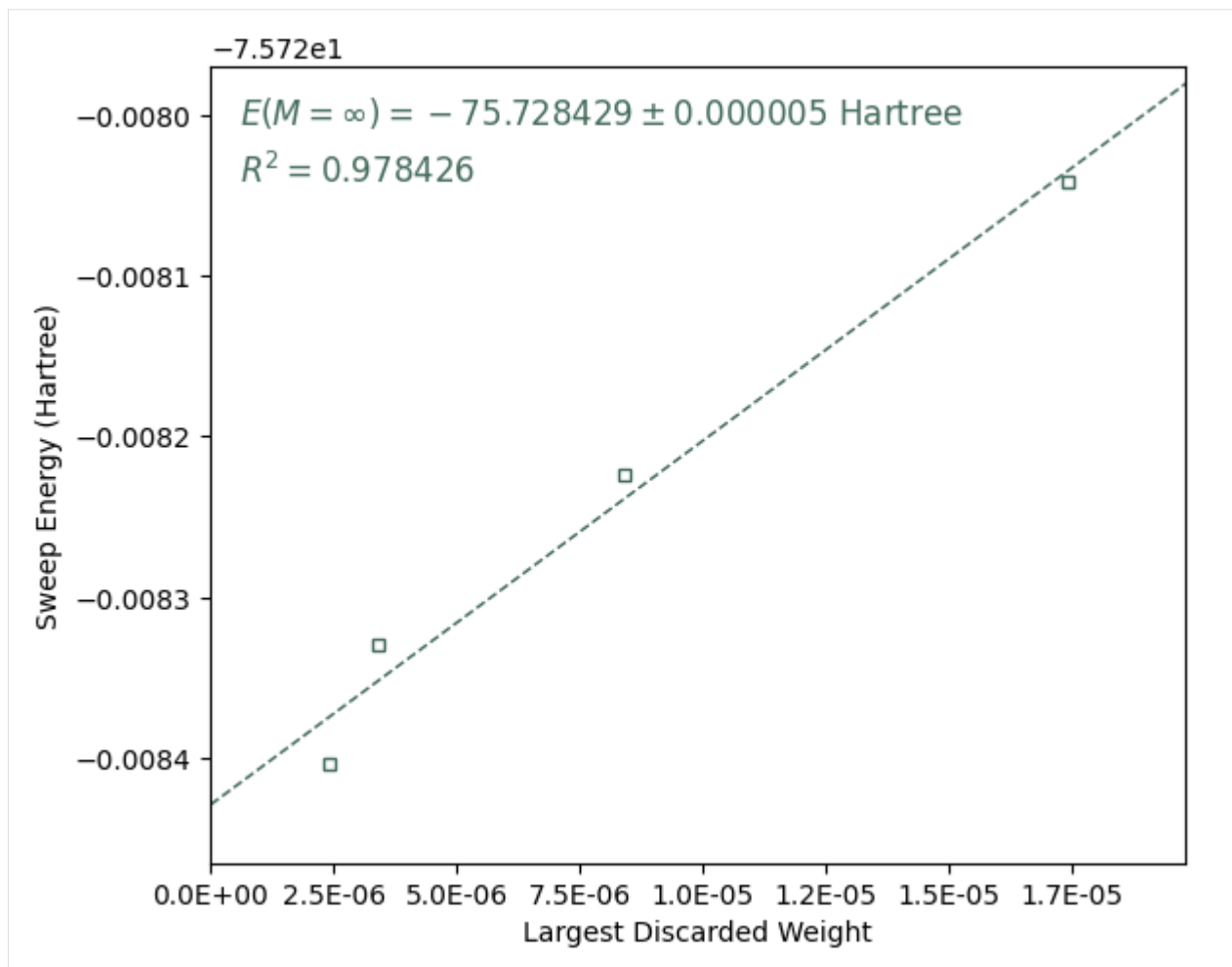
BOND DIMS          = [800 700 600 500]
Discarded Weights = [2.39910258e-06 3.39687452e-06 8.41354132e-06 1.74228067e-05]
Energies           = [-75.72840404 -75.7283302  -75.72822444 -75.72804214]
DMRG energy (extrapolated) = -75.728429118584003 +/- 0.0000050161

```

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, "$E(M=\infty) = %.6f \pm %.6f \mathrm{\infty}$",
↪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 Custom Hamiltonian

```
[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 this tutorial, we provide an example python script 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.

Note the following examples is only supposed to work in the SZ mode.

4.3.1 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]]), #_
        ↪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()
```

(continues on next page)

(continued from previous page)

```

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), algo_type=MPOAlgorithmTypes.
    ↪FastBipartite)

```

```

mps = driver.get_random_mps(tag="KET", bond_dim=250, nroots=1)

```

```

energy = driver.dmrp(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 =    0.342 | 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 =    0.456 | E =    -6.2256341447 | DE = -1.07e-14 | DW = 4.93e-16

```

```

Sweep =    2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 |
    ↪Dav threshold = 1.00e-10

```

```

Time elapsed =    0.585 | E =    -6.2256341447 | DE = -1.78e-15 | DW = 9.81e-17

```

```

Sweep =    3 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 |
    ↪Dav threshold = 1.00e-10

```

```

Time elapsed =    0.694 | E =    -6.2256341447 | DE = -3.55e-15 | DW = 1.20e-16

```

```

Sweep =    4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 |
    ↪Dav threshold = 1.00e-10

```

```

Time elapsed =    0.873 | E =    -6.2256341447 | DE = -8.88e-16 | DW = 4.50e-20

```

```

Sweep =    5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 |
    ↪Dav threshold = 1.00e-10

```

```

Time elapsed =    1.028 | E =    -6.2256341447 | DE = -2.66e-15 | DW = 4.87e-20

```

```

Sweep =    6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 |
    ↪Dav threshold = 1.00e-10

```

```

Time elapsed =    1.178 | E =    -6.2256341447 | DE = 5.33e-15 | DW = 3.86e-20

```

```

Sweep =    7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 |
    ↪Dav threshold = 1.00e-10

```

```

Time elapsed =    1.320 | E =    -6.2256341447 | DE = 1.78e-15 | DW = 4.94e-20

```

(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 =      1.472 | E =      -6.2256341447 | DE = 0.00e+00 | DW = 2.86e-20

DMRG energy =      -6.225634144657919

```

4.3.2 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 = {
            "'": 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 = {

```

(continues on next page)

(continued from previous page)

```

        """ : 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)]).
    ↪ravel(), G)

# [Part C] Perform DMRG
mpo = driver.get_mpo(b.finalize(adjust_order=True), algo_type=MPOAlgorithmTypes.
    ↪FastBipartite)
mps = driver.get_random_mps(tag="KET", bond_dim=250, nroots=1)
energy = driver.dmrp(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 = 108.945 | E = -6.9568929229 | DW = 3.62e-09

```

(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 = 142.030 | E = -6.9568932112 | DE = -2.88e-07 | DW = 3.07e-19

Sweep =    2 | Direction = forward | Bond dimension = 250 | Noise = 1.00e-04 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 154.346 | E = -6.9568932112 | DE = -1.78e-15 | DW = 1.38e-19

Sweep =    3 | Direction = backward | Bond dimension = 250 | Noise = 1.00e-04 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 165.954 | E = -6.9568932112 | DE = -1.78e-15 | DW = 6.71e-20

Sweep =    4 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 176.280 | E = -6.9568932112 | DE = -8.88e-16 | DW = 7.26e-20

Sweep =    5 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 186.605 | E = -6.9568932112 | DE = 2.66e-15 | DW = 6.17e-20

Sweep =    6 | Direction = forward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 196.209 | E = -6.9568932112 | DE = -1.78e-15 | DW = 9.34e-20

Sweep =    7 | Direction = backward | Bond dimension = 500 | Noise = 1.00e-05 | _
↳ Dav threshold = 1.00e-10
Time elapsed = 205.734 | E = -6.9568932112 | DE = 2.66e-15 | DW = 6.36e-20

Sweep =    8 | Direction = forward | Bond dimension = 500 | Noise = 0.00e+00 | _
↳ Dav threshold = 1.00e-09
Time elapsed = 215.378 | E = -6.9568932112 | DE = -9.77e-15 | DW = 7.87e-20

DMRG energy = -6.956893211180049

```

4.4 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 func-

tion 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).

4.4.1 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)

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,
```

(continues on next page)

(continued from previous page)

```

↪iprint=1)
ket = driver.get_random_mps(tag="KET", bond_dim=150, nroots=1)
energy = driver.dmrp(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)
gformat = np.zeros((len(freqs), ), dtype=complex)
for iw, freq in enumerate(freqs):
    bra = driver.copy_mps(dket, tag="BRA") # initial guess
    gformat[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,
↪gformat[iw].real, gformat[iw].imag))

ldos = -1 / np.pi * gformat.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
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

```

(continues on next page)

(continued from previous page)

```

↪= 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

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

```

(continues on next page)

(continued from previous page)

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
 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

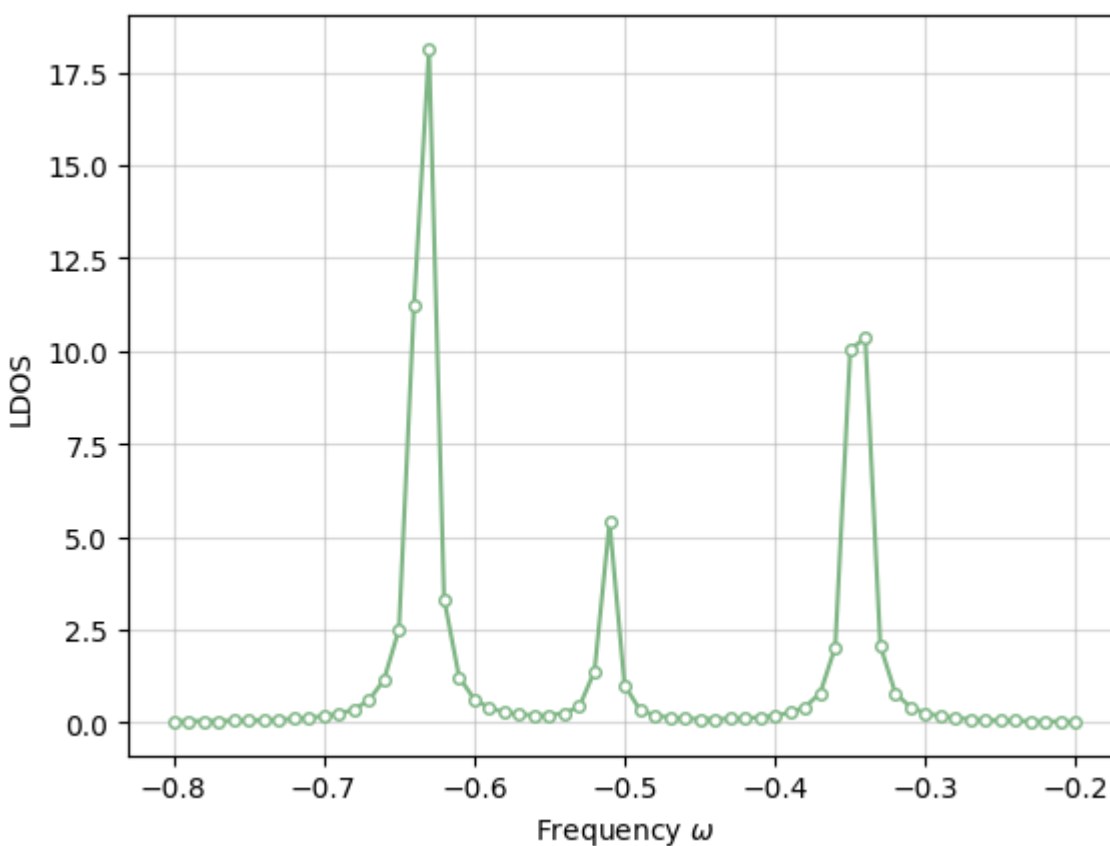
(continues on next page)

(continued from previous page)

```

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)
gformat = np.zeros((len(freqs), ), dtype=complex)
for iw, freq in enumerate(freqs):
    bra = driver.copy_mps(dket, tag="BRA") # initial guess
    gformat[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,
        gformat[iw].real, gformat[iw].imag))

ldos = -2 / np.pi * gformat.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

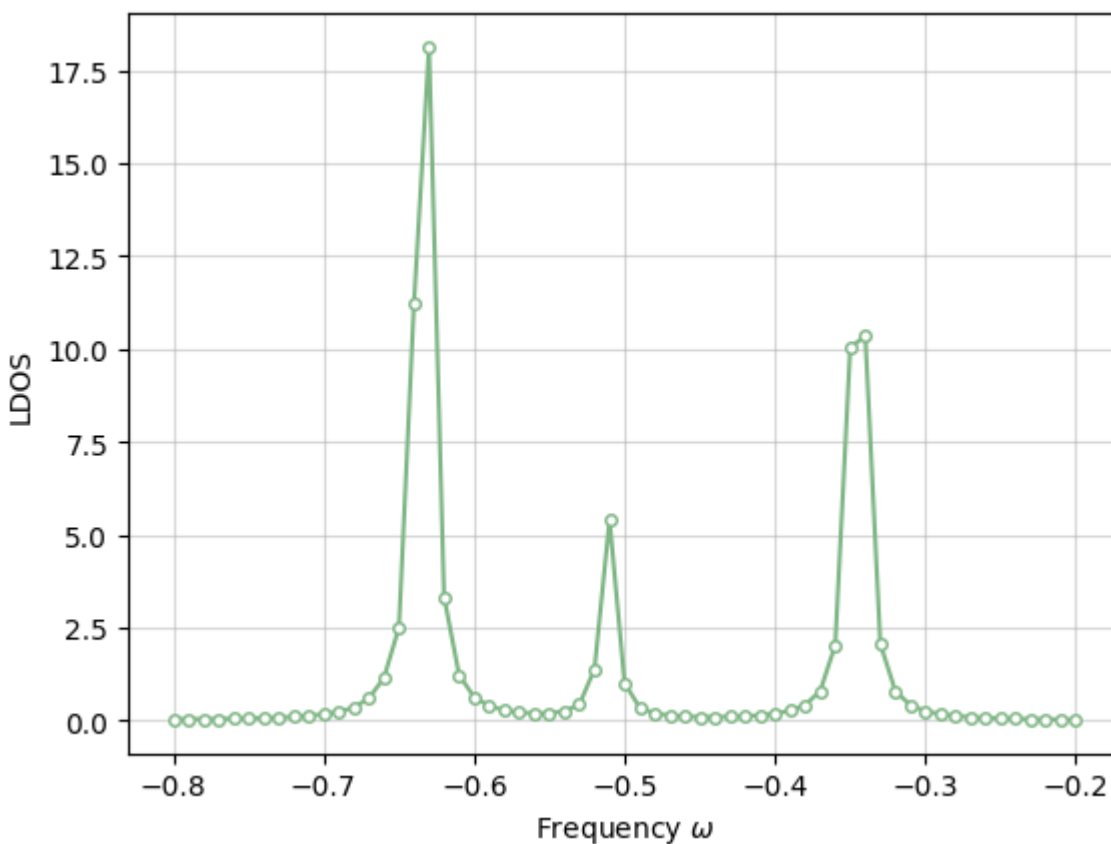
FREQ = -0.64 GF[2,2] = -22.488537 + -17.623137 i

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



4.4.2 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:

$$G_{ij}(t) = -i\langle\Psi_0|a_j^\dagger e^{-i(\hat{H}_0 - E_0)t} a_i|\Psi_0\rangle$$

$$G_{ij}(\omega) = \int_{-\infty}^{\infty} dt e^{-i\omega t} G_{ij}(t) e^{-\eta t}$$

where $e^{-\eta t}$ is a broadening 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.drmg(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)

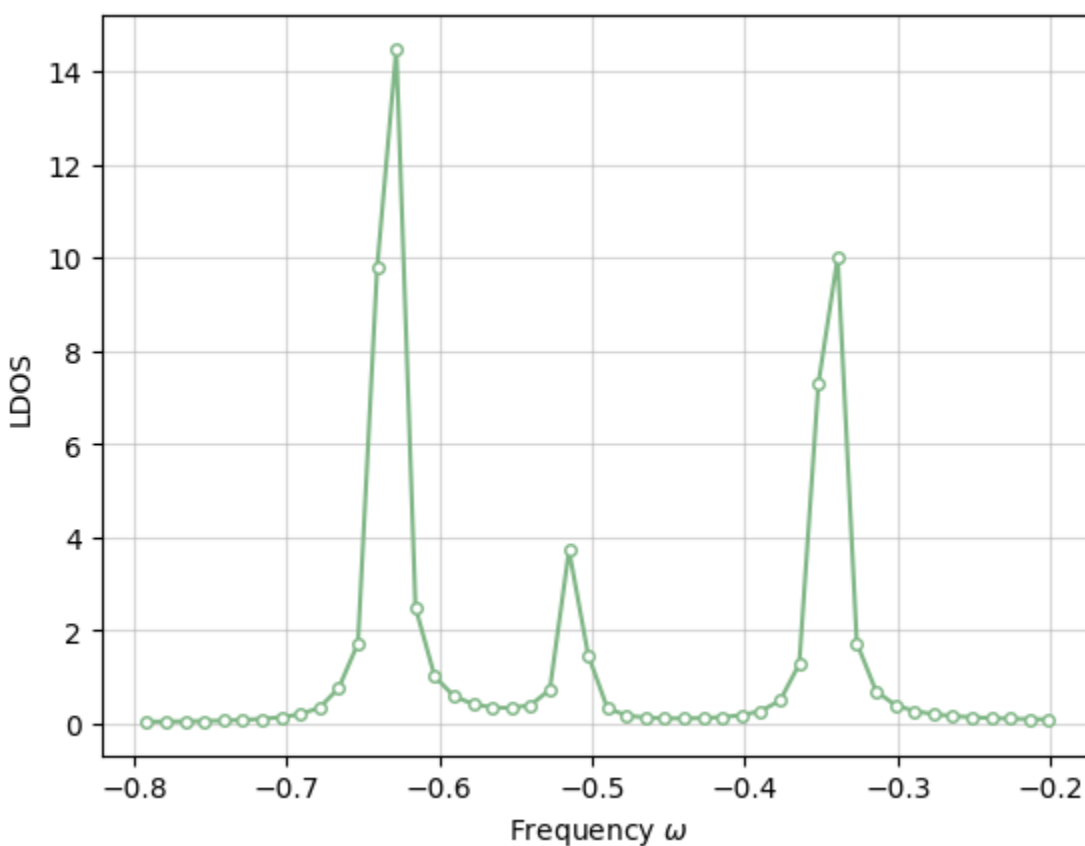
(continued from previous page)

```
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.
↪ 0.0e-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)

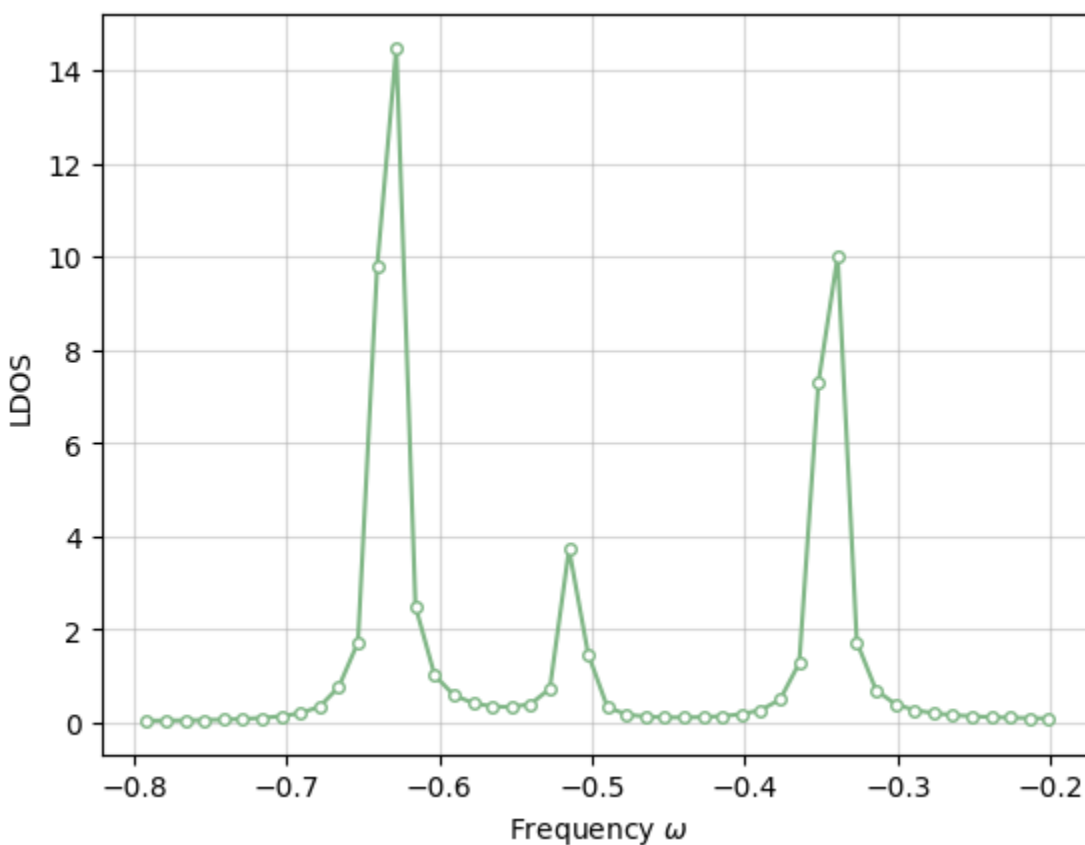
(continued from previous page)

```
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 %)
```



4.5 Hubbard Model

```
[1]: !pip install block2==0.5.2rc13 -qq --progress-bar off --extra-index-url=https://
    ↪block-hczhai.github.io/block2-preview/pypi/
```

4.5.1 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 $U(1) \times U(1)$, 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 $U(1) \times SU(2)$. $SU(2)$ 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 $U(1)$. 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) $SU(2)$ symmetry. If we label states using (N, S_z) , the symmetry group is $SU(2) \times U(1)$. 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 $SU(2) \times SU(2) / Z_2 = SO(4)$. 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
```

4.5.2 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 temporary 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 character (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)

# 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
```

(continues on next page)

(continued from previous page)

```

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 solve 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.dmrp` 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_dmrp(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.dmrp(mpo, ket, n_sweeps=20, bond_dims=bond_dims, noises=noises,
                       thrds=thrds, cutoff=0, iprint=1)
```

```
energies = run_dmrp(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
```

(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 =    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 documentation introduces some other alternative symmetry to solve the same problem.

4.5.3 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 $a_\alpha^\dagger$ 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

$$n_{i\uparrow}n_{i\downarrow} = a_{i\alpha}^\dagger a_{i\alpha} a_{i\beta}^\dagger a_{i\beta} = a_{i\alpha}^\dagger a_{i\beta}^\dagger a_{i\beta} a_{i\alpha} = a_{i\beta}^\dagger a_{i\alpha}^\dagger a_{i\alpha} a_{i\beta} = \frac{1}{2} \sum_{\sigma} \sum_{\sigma' \neq \sigma} a_{i\sigma}^\dagger a_{i\sigma'}^\dagger a_{i\sigma'} a_{i\sigma} = \frac{1}{2} \sum_{\sigma\sigma'} a_{i\sigma}^\dagger a_{i\sigma'}^\dagger a_{i\sigma'} a_{i\sigma}$$

We can rewrite the Hubbard Hamiltonian as

$$\hat{H} = -t \sum_{\langle ij \rangle, \sigma} a_{i\sigma}^\dagger a_{j\sigma} + \frac{U}{2} \sum_{i, \sigma\sigma'} a_{i\sigma}^\dagger a_{i\sigma'}^\dagger a_{i\sigma'} a_{i\sigma}$$

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

$$(a_p^\dagger)^{[1/2]} := \begin{pmatrix} a_{p\alpha}^\dagger \\ a_{p\beta}^\dagger \end{pmatrix}^{[1/2]} \quad (a_p)^{[1/2]} := \begin{pmatrix} -a_{p\beta} \\ a_{p\alpha} \end{pmatrix}^{[1/2]}$$

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 $(a_p^\dagger)^{[1/2]}$ and $(a_q)^{[1/2]}$, we have to indicate which resulting total spin is desired. So we need the following notation to differentiate the two cases:

$$\begin{aligned} (a_p^\dagger)^{[1/2]} \otimes_{[0]} (a_q)^{[1/2]} & \quad (\text{singlet product}) \\ (a_p^\dagger)^{[1/2]} \otimes_{[1]} (a_q)^{[1/2]} & \quad (\text{triplet product}) \end{aligned}$$

- 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>)

$$\hat{H}^{[0]} = -\sqrt{2} t \sum_{\langle ij \rangle} (a_i^\dagger)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} + U \sum_i \left((a_i^\dagger)^{[1/2]} \otimes_{[1/2]} \left((a_i^\dagger)^{[1/2]} \otimes_{[0]} (a_i)^{[1/2]} \right) \right) \otimes_{[0]} (a_i)^{[1/2]}$$

Next, we have to introduce a string notation to represent the above formula. We can use C and D for $a^\dagger$ and a , respectively. 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
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

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
```

4.5.4 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
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
```

(continues on next page)

(continued from previous page)

```

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

```

(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 =    1.272 | 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

```

4.5.5 The SGB Mode

Initialization

In this section, we try to solve the same Hubbard problem by first transforming 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 characters 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
```

(continues on next page)

(continued from previous page)

```

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_dmrp(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

```

(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 =    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

```

4.5.6 The PHSU2 Mode

Let

$$(E_p)^{[1/2]} := \begin{pmatrix} a_{p,\uparrow}^\dagger \\ (-1)^p a_{p,\downarrow} \end{pmatrix}^{[1/2]} \quad (F_p)^{[1/2]} := \begin{pmatrix} -(-1)^p a_{p,\downarrow}^\dagger \\ a_{p,\uparrow} \end{pmatrix}^{[1/2]}$$

We have (assuming p and q differ by 1)

$$\begin{aligned} (E_p)^{[1/2]} \otimes_{[0]} (F_q)^{[1/2]} &= \begin{pmatrix} a_{p,\uparrow}^\dagger \\ (-1)^p a_{p,\downarrow} \end{pmatrix}^{[1/2]} \otimes_{[0]} \begin{pmatrix} -(-1)^q a_{q,\downarrow}^\dagger \\ a_{q,\uparrow} \end{pmatrix}^{[1/2]} \\ &= \frac{1}{\sqrt{2}} \begin{pmatrix} a_{p,\uparrow}^\dagger a_{q,\uparrow} - a_{p,\downarrow} a_{q,\downarrow}^\dagger \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} a_{p,\uparrow}^\dagger a_{q,\uparrow} + a_{q,\downarrow}^\dagger a_{p,\downarrow} \end{pmatrix} \end{aligned}$$

So

$$\begin{aligned}
& (E_p)^{[1/2]} \otimes_{[0]} (F_{p+1})^{[1/2]} + (E_{p+1})^{[1/2]} \otimes_{[0]} (F_p)^{[1/2]} \\
&= \frac{1}{\sqrt{2}} \left(a_{p,\uparrow}^\dagger a_{p+1,\uparrow} + a_{p+1,\downarrow}^\dagger a_{p,\downarrow} \right) + \frac{1}{\sqrt{2}} \left(a_{p+1,\uparrow}^\dagger a_{p,\uparrow} + a_{p,\downarrow}^\dagger a_{p+1,\downarrow} \right) \\
&= \frac{1}{\sqrt{2}} \left(a_{p,\uparrow}^\dagger a_{p+1,\uparrow} + a_{p,\downarrow}^\dagger a_{p+1,\downarrow} + a_{p+1,\uparrow}^\dagger a_{p,\uparrow} + a_{p+1,\downarrow}^\dagger a_{p,\downarrow} \right) \\
&= \frac{1}{\sqrt{2}} \sum_{\sigma} \left(a_{p,\sigma}^\dagger a_{p+1,\sigma} + \text{H.c.} \right) \\
&-t \sum_{\langle i,j \rangle, \sigma} \left(c_{i,\sigma}^\dagger c_{j,\sigma} + \text{H.c.} \right) = -\sqrt{2}t \sum_{\langle i,j \rangle} \left[(E_i)^{[1/2]} \otimes_{[0]} (F_j)^{[1/2]} + (E_j)^{[1/2]} \otimes_{[0]} (F_i)^{[1/2]} \right]
\end{aligned}$$

Now consider the onsite term

$$\begin{aligned}
(E_p)^{[1/2]} \otimes_{[0]} (F_p)^{[1/2]} &= \begin{pmatrix} a_{p,\uparrow}^\dagger \\ (-1)^p a_{p,\downarrow} \end{pmatrix}^{[1/2]} \otimes_{[0]} \begin{pmatrix} -(-1)^p a_{p,\downarrow}^\dagger \\ a_{p,\uparrow} \end{pmatrix}^{[1/2]} \\
&= \frac{1}{\sqrt{2}} \left(a_{p,\uparrow}^\dagger a_{p,\uparrow} + a_{p,\downarrow}^\dagger a_{p,\downarrow} \right) = \frac{1}{\sqrt{2}} \left(a_{p,\uparrow}^\dagger a_{p,\uparrow} - a_{p,\downarrow}^\dagger a_{p,\downarrow} + 1 \right) \\
(E_p)^{[1/2]} \otimes_{[0]} (E_p)^{[1/2]} &= \begin{pmatrix} a_{p,\uparrow}^\dagger \\ (-1)^p a_{p,\downarrow} \end{pmatrix}^{[1/2]} \otimes_{[0]} \begin{pmatrix} a_{p,\uparrow}^\dagger \\ (-1)^p a_{p,\downarrow} \end{pmatrix}^{[1/2]} \\
&= \frac{1}{\sqrt{2}} \left((-1)^p a_{p,\uparrow}^\dagger a_{p,\downarrow} - (-1)^p a_{p,\downarrow}^\dagger a_{p,\uparrow} \right) = (-1)^p \sqrt{2} a_{p,\uparrow}^\dagger a_{p,\downarrow} \\
(F_p)^{[1/2]} \otimes_{[0]} (F_p)^{[1/2]} &= \begin{pmatrix} -(-1)^p a_{p,\downarrow}^\dagger \\ a_{p,\uparrow} \end{pmatrix}^{[1/2]} \otimes_{[0]} \begin{pmatrix} -(-1)^p a_{p,\downarrow}^\dagger \\ a_{p,\uparrow} \end{pmatrix}^{[1/2]} \\
&= \frac{1}{\sqrt{2}} \left(-(-1)^p a_{p,\downarrow}^\dagger a_{p,\uparrow} + (-1)^p a_{p,\uparrow}^\dagger a_{p,\downarrow} \right) = -(-1)^p \sqrt{2} a_{p,\downarrow}^\dagger a_{p,\uparrow}
\end{aligned}$$

So the product of above terms

$$\begin{aligned}
& \left((E_p)^{[1/2]} \otimes_{[0]} (E_p)^{[1/2]} \right) \otimes_{[0]} \left((F_p)^{[1/2]} \otimes_{[0]} (F_p)^{[1/2]} \right) \\
&= -2a_{p,\uparrow}^\dagger a_{p,\downarrow} a_{p,\downarrow}^\dagger a_{p,\uparrow} = -2a_{p,\uparrow}^\dagger a_{p,\uparrow} a_{p,\downarrow} a_{p,\downarrow}^\dagger = 2a_{p,\uparrow}^\dagger a_{p,\uparrow} a_{p,\downarrow}^\dagger a_{p,\downarrow} - 2a_{p,\uparrow}^\dagger a_{p,\uparrow} = 2n_{p\uparrow} n_{p\downarrow} - 2n_{p\uparrow}
\end{aligned}$$

```
[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.
```

(continues on next page)

(continued from previous page)

```

↪ 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.dmrp(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

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 |

```

(continues on next page)

(continued from previous page)

```

↪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

```

4.5.7 The S04 Mode

Similarly, we can consider the $SU(2) \times SU(2)$ operator G , where the row represents the pseudo spin symmetry and the column represents the spin symmetry.

$$(G_p)^{[1/2,1/2]} := \begin{pmatrix} a_{p,\uparrow}^\dagger & a_{p,\downarrow}^\dagger \\ (-1)^p a_{p,\downarrow} & -(-1)^p a_{p,\uparrow} \end{pmatrix}^{[1/2,1/2]}$$

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

(continues on next page)

(continued from previous page)

```

n_elec = L

driver = DMRGDriver(scratch='./tmp', symm_type=SymmetryTypes.SAnyS04, 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.dmrp(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
    ↪ = 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

```

(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 =    0.066 | 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
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

```

4.6 Heisenberg Model

```
[1]: !pip install block2==0.5.2rc13 -qq --progress-bar off --extra-index-url=https://
    ↪block-hczhai.github.io/block2-preview/pypi/
```

4.6.1 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.

4.6.2 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)
    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):
```

(continues on next page)

(continued from previous page)

```

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.dmrp(
    mpo,
    ket,
    n_sweeps=8,
    bond_dims=bond_dims,
    noises=noises,
    thrds=thrds,
    cutoff=1E-24,
    iprint=1,
)

energies = run_dmrp(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 | _
↳ 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

```

(continues on next page)

(continued from previous page)

```
DMRG energy = -44.127739882610513
```

4.6.3 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

$$\begin{aligned} (T_p)^{[1]} \otimes_{[0]} (T_q)^{[1]} &= \begin{pmatrix} -S_p^+ \\ \sqrt{2}S_p^z \\ S_p^- \end{pmatrix}^{[1]} \otimes_{[0]} \begin{pmatrix} -S_q^+ \\ \sqrt{2}S_q^z \\ S_q^- \end{pmatrix}^{[1]} \\ &= \frac{1}{\sqrt{3}} (-S_p^+ S_q^- - S_p^- S_q^+ - 2S_p^z S_q^z)^{[0]} = -\frac{2}{\sqrt{3}} \left(\frac{1}{2} S_p^+ S_q^- + \frac{1}{2} S_p^- S_q^+ + S_p^z S_q^z \right)^{[0]} \end{aligned}$$

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)
```

(continues on next page)

(continued from previous page)

```

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

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 $N_{\text{dav}} = 0$ $E = 0.0$.
- **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 = $\langle n - 1 \rangle$ 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 transform 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 permutation 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:

```

# Hamiltonian 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:

```
# Hamiltonian 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 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> =   -0.0000000000000000 <Norm^2> =    0.999999979398520
T =    0.10000 <E> =   -0.0000000000000000 <Norm^2> =    0.999999926838107
...
T =    0.95000 <E> =    0.0000000000000000 <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> = 0.0000000000000000 <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)

# Hamiltonian 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.normsq[-1]))

# Hamiltonian 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> =    0.0000000000000000 <Norm^2> =    0.999999980443349
T =    0.10000 <E> =   -0.0000000000000000 <Norm^2> =    0.999999930992521
...
T =    0.95000 <E> =    0.0000000000000000 <Norm^2> =    0.999996944337650
Time elapsed =    6.035 | E =   -0.000000000000 | Norm^2 =    0.9999966399 | ↪
↪ DW = 3.84e-08
T =    1.00000 <E> =   -0.0000000000000000 <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 D_{2h} symmetry in the molecule, and another integral with C_{2v} 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 =    1.0000000000000001
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 = 0.99999999999821
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 processors (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 processors (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 = CSR0peratorFunctions(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 inconsistency 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 different 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 stuck. 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.

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'
```

(continues on next page)

(continued from previous page)

```
↪ 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_
```

(continues on next page)

(continued from previous page)

```
↪ 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)'
```


API REFERENCE

6.1 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.

6.1.1 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 possible 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 guaranteed 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.

6.1.2 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 element 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 element 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.

6.1.3 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** – Scratch 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 **~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 &if_) const

Load one data frame from input stream.

Parameters

- **i** – The index of the data frame.
- **if_** – 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.

6.1.4 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.

6.2 Sparse Matrix

In **block2**, we support a few different representations of the block-sparse matrix.

6.2.1 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.

6.3 Tensor Functions

Tensor functions are drivers for performing operations for operator tensors. For operator tensors with different internal data representations or distributed parallelization schemes, a few different implementations of the tensor function drivers are defined. They all have the same interface.

6.3.1 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 vec-  
                                                    tor<vector<shared_ptr<typename  
                                                    SparseMatrix-  
                                                    Info<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: $\text{vmat} = \text{expr}[\text{L part} \mid \text{R part}] \times \text{cmat}$. 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 SparseMatrix-
                                                    Info<S>::ConnectionInfo>> &cinfos,
                                                    S opdq, FL factor, bool all_reduce)
                                                    const override
```

Tensor product multiplication operation (multi-root case): $\text{vmats} = \text{expr} \times \text{cmats}$. 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).
- **vinfos** – 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** – Scaling 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): $\text{vmat} = \text{expr} \times \text{cmat}$.

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: $c = \text{mpst_bra} \times a \times \text{mpst_ket}$. 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. 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: $c = a \times b$.

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 constructed 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: $c = b \times a$.

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 constructed 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).

6.4 Tools

6.4.1 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 element 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 decompress 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 – Element 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 element.

inline virtual *T* **operator**[(size_t i)] const

Read one element from the array.

Parameters

i – Array index.

Returns

The array element.

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 – Element 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 element.

```
inline virtual T operator[](size_t i) const
```

Read one element from the array.

Parameters

i – Array index.

Returns

The array element.

```
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.

6.4.2 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, then 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).

6.4.3 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

$a^{-1} \pmod n$ 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 ($i \geq 0$).

Returns

n^i .

static inline LL **sqrt**(LL x)

Find the largest number r such that $r * r \leq x$. Using binary algorithm.

Parameters

x – Integer x.

Returns

floor(sqrt(x)).

static inline LL **quick_multiply**(LL x, LL y, LL p)

Find $(x * y) \pmod p$. Note that the expression $x * y \% p$ may overflow if the intermediate $x * y > 2^{63} - 1$. This function will not overflow (if $p \leq 2^{62} - 1$)

Parameters

- **x** – Integer x.
- **y** – Integer y.

- **p** – Modulus p.

Returns

$(x * y) \bmod p$.

static inline LL **quick_power**(LL n, LL i, LL p)

Find $(n^i) \bmod p$ using binary algorithm.

Parameters

- **n** – Integer n.
- **i** – Integer i ($i \geq 0$).
- **p** – Modulus p.

Returns

$(n^i) \bmod p$.

static inline bool **milller_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 ($n \geq 3$).

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.

6.4.4 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 $5929741 = 181 \times 3$ and $5929742 = 2 \times 7 \times 13 \times 31 \times 1051$.

template<int **base**>

struct **BasicFFT**

Fast Fourier Transform (FFT) with array length of base^k ($k \geq 0$). The complexity is $O(n \log_{\text{base}} n \times \text{base}^2)$.

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 $\exp(i2\pi k/n)$ for backward FFT.

```
vector<complex<double>> wf
```

The precomputed primitive n th root of 1 $\exp(-i2\pi k/n)$ for forward FFT.

```
array<array<complex<double>, base>, base> xw[2]
```

The precomputed primitive base -th root of 1 $\exp(-/+ i2\pi jk/\text{base})$ for forward/backward FFT.

Public Static Functions

```
static inline size_t pad(size_t n)
```

Find smallest number x such that $x = \text{base}^k \geq n$.

Parameters

n – The array length.

Returns

The padded array length.

```
template<>
```

```
struct BasicFFT<2>
```

Radix-2 Fast Fourier Transform (FFT) with complexity $O(n \log n)$.

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 $\exp(i2\pi k/n)$ for backward FFT.

vector<complex<double>> **wf**

The precomputed primitive nth root of 1 $\exp(-i2\pi k/n)$ for forward FFT.

Public Static Functions

static inline size_t **pad**(size_t n)

Find smallest number x such that $x = 2^k \geq n$.

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 $g^{-k} \bmod n$ for $k = 0, 1, \dots, n - 1$.

```
vector<LL> wf
```

Precomputed power of primitive root $g^k \bmod n$ for $k = 0, 1, \dots, n - 1$.

```
vector<complex<double>> cb
```

FFT transformed $\exp(i2\pi g^{-k}/n)$.

```
vector<complex<double>> cf
```

FFT transformed $\exp(-i2\pi g^{-k}/n)$.

```
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 $\exp(i2\pi k/n)$ for backward FFT.

vector<complex<double>> **wf**

The precomputed primitive nth root of 1 $\exp(-i2\pi k/n)$ for forward FFT.

vector<complex<double>> **cb**

FFT transformed $\exp(i2\pi [((k - n) - (k - n)(k - n)) / 2] / n)$.

vector<complex<double>> **cf**

FFT transformed $\exp(-i2\pi [((k - n) - (k - n)(k - n)) / 2] / n)$.

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: $X[k] = \sum_{j=0}^{n-1} x[j] \exp(-2\pi i j k / n)$

Backward DFT: $X[k] = \frac{1}{n} \sum_{j=0}^{n-1} x[j] \exp(2\pi i j k / n)$

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: $X[k] = \sum_{j=0}^{n-1} x[j] \exp(-2\pi i j k / n)$

Backward FFT: $X[k] = \frac{1}{n} \sum_{j=0}^{n-1} x[j] \exp(2\pi i j k / n)$

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 implemented using Rader's algorithm.

typedef *FactorizedFFT*<*BluesteinFFT*<>, 2, 3, 5, 7, 11> block2::**FFT**

FFT with small prime factorization implemented using Bluestein's algorithm.

7.1 DMRG Hamiltonian

7.1.1 DMRG Quantum Chemistry Hamiltonian in Spatial Orbitals

Hamiltonian

The quantum chemistry Hamiltonian is written as follows

$$\hat{H} = \sum_{ij,\sigma} t_{ij} a_{i\sigma}^\dagger a_{j\sigma} + \frac{1}{2} \sum_{ijkl,\sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma}$$

where

$$t_{ij} = t_{(ij)} = \int d\mathbf{x} \phi_i^*(\mathbf{x}) \left(-\frac{1}{2} \nabla^2 - \sum_a \frac{Z_a}{r_a} \right) \phi_j(\mathbf{x})$$

$$v_{ijkl} = v_{(ij)(kl)} = v_{(kl)(ij)} = \int d\mathbf{x}_1 d\mathbf{x}_2 \frac{\phi_i^*(\mathbf{x}_1) \phi_k^*(\mathbf{x}_2) \phi_l(\mathbf{x}_2) \phi_j(\mathbf{x}_1)}{r_{12}}$$

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

$$\begin{aligned} \hat{H} &= \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R \\ &+ \left(\sum_{i \in L, \sigma} a_{i\sigma}^\dagger \hat{S}_{i\sigma}^R + h.c. \right) + \left(\sum_{i \in L, \sigma} a_{i\sigma}^\dagger \hat{R}_{i\sigma}^R + h.c. + \sum_{i \in R, \sigma} a_{i\sigma}^\dagger \hat{R}_{i\sigma}^L + h.c. \right) \\ &+ \frac{1}{2} \left(\sum_{ik \in L, \sigma\sigma'} \hat{A}_{ik, \sigma\sigma'}^L \hat{P}_{ik, \sigma\sigma'}^R + h.c. \right) + \sum_{ij \in L} \hat{B}_{ij} \hat{Q}_{ij}^R - \sum_{il \in L, \sigma\sigma'} \hat{B}'_{il\sigma\sigma'} \hat{Q}'_{il\sigma\sigma'}^R \end{aligned}$$

where the normal and complementary operators are defined by

$$\begin{aligned}
\hat{S}_{i\sigma}^{L/R} &= \sum_{j \in L/R} t_{ij} a_{j\sigma}, \\
\hat{R}_{i\sigma}^{L/R} &= \sum_{jkl \in L/R, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma}, \\
\hat{A}_{ik, \sigma\sigma'} &= a_{i\sigma}^\dagger a_{k\sigma'}^\dagger, \\
\hat{B}_{ij} &= \sum_{\sigma} a_{i\sigma}^\dagger a_{j\sigma}, \\
\hat{B}'_{il, \sigma\sigma'} &= a_{i\sigma}^\dagger a_{l\sigma'}, \\
\hat{P}_{ik, \sigma\sigma'}^R &= \sum_{jl \in R} v_{ijkl} a_{l\sigma'} a_{j\sigma}, \\
\hat{Q}_{ij}^R &= \sum_{kl \in R, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'}, \\
\hat{Q}'_{il, \sigma\sigma'}^R &= \sum_{jk \in R} v_{ijkl} a_{k\sigma'}^\dagger a_{j\sigma}
\end{aligned}$$

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,

$$\sum_{i \in L, \sigma} a_{i\sigma}^\dagger \hat{S}_{i\sigma}^R + h.c. = \sum_{i \in L, \sigma} a_{i\sigma}^\dagger \hat{S}_{i\sigma}^R + \sum_{j \in L, \sigma} \hat{S}_{j\sigma}^{R\dagger} a_{j\sigma} = \sum_{i \in L/R, j \in R/L, \sigma} t_{ij} a_{i\sigma}^\dagger a_{j\sigma}$$

Next consider one of $ijkl$ in left, and three of them in right. These terms are

$$\begin{aligned}
\hat{H}_{1L,3R} &= \frac{1}{2} \sum_{i \in L, jkl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} + \frac{1}{2} \sum_{j \in L, ikl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} + \frac{1}{2} \sum_{k \in L, ij \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} + \\
&= \left[\frac{1}{2} \sum_{i \in L, jkl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} + \frac{1}{2} \sum_{k \in L, ij \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} \right] + \frac{1}{2} \sum_{j \in L, ikl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma}
\end{aligned}$$

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

$$\frac{1}{2} \sum_{j \in L, ikl \in R, \sigma\sigma'} v_{ijkl} a_{j\sigma}^\dagger a_{l\sigma'}^\dagger a_{k\sigma'} a_{i\sigma} = \frac{1}{2} \sum_{k \in L, ij \in R, \sigma\sigma'} v_{lkji} a_{k\sigma}^\dagger a_{i\sigma'}^\dagger a_{j\sigma'} a_{l\sigma} = \frac{1}{2} \sum_{k \in L, ij \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma} a_{j\sigma'}$$

The conjugate of forth term in rhs is first term in rhs

$$\frac{1}{2} \sum_{l \in L, ijk \in R, \sigma\sigma'} v_{ijkl} a_{j\sigma}^\dagger a_{l\sigma'}^\dagger a_{k\sigma'} a_{i\sigma} = \frac{1}{2} \sum_{i \in L, jkl \in R, \sigma\sigma'} v_{lkji} a_{k\sigma}^\dagger a_{i\sigma'}^\dagger a_{j\sigma'} a_{l\sigma} = \frac{1}{2} \sum_{i \in L, jkl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma} a_{j\sigma'}$$

Therefore, using $v_{ijkl} = v_{klij}$

$$\begin{aligned}
\hat{H}_{1L,3R} &= \left[\frac{1}{2} \sum_{i \in L, jkl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} + \frac{1}{2} \sum_{k \in L, ijl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} \right] + h.c. \\
&= \left[\frac{1}{2} \sum_{i \in L, jkl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} + \frac{1}{2} \sum_{k \in L, ijl \in R, \sigma\sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{i\sigma}^\dagger a_{j\sigma} a_{l\sigma'} \right] + h.c. \\
&= \left[\frac{1}{2} \sum_{i \in L, jkl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} + \frac{1}{2} \sum_{i \in L, jkl \in R, \sigma\sigma'} v_{klij} a_{i\sigma}^\dagger a_{k\sigma}^\dagger a_{l\sigma} a_{j\sigma'} \right] + h.c. \\
&= \sum_{i \in L, jkl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} + h.c. \\
&= \sum_{i \in L, \sigma} a_{i\sigma}^\dagger \sum_{jkl \in R, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} + h.c. = \sum_{i \in L, \sigma} a_{i\sigma}^\dagger R_{i\sigma}^R + h.c.
\end{aligned}$$

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,

$$\sum_{ik \in R, jl \in L, \sigma\sigma'} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger v_{ijkl} a_{l\sigma'} a_{j\sigma} = \sum_{jl \in R, ik \in L, \sigma\sigma'} a_{j\sigma}^\dagger a_{l\sigma'}^\dagger v_{jikl} a_{k\sigma'} a_{i\sigma} = \sum_{ik \in L, jl \in R, \sigma\sigma'} v_{jikl} a_{j\sigma}^\dagger a_{l\sigma'}^\dagger a_{k\sigma'} a_{i\sigma} = \sum_{ik \in L, jl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma}$$

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.

$$\begin{aligned}
\hat{H}_{2L,2R} &= \frac{1}{2} \sum_{il \in L, jk \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} + \frac{1}{2} \sum_{ij \in L, kl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} + \frac{1}{2} \sum_{kl \in L, ij \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} + \\
&= -\frac{1}{2} \sum_{il \in L, jk \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{l\sigma'} a_{k\sigma'}^\dagger a_{j\sigma} + \frac{1}{2} \sum_{ij \in L, kl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{j\sigma} a_{k\sigma'}^\dagger a_{l\sigma'} + \frac{1}{2} \sum_{kl \in L, ij \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{j\sigma} a_{k\sigma'}^\dagger a_{l\sigma'}
\end{aligned}$$

where the first, forth terms are combing different spins. The second, third terms are for the same spin. First consider the same-spin case

$$\begin{aligned}
&\frac{1}{2} \sum_{ij \in L, kl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{j\sigma} a_{k\sigma'}^\dagger a_{l\sigma'} + \frac{1}{2} \sum_{kl \in L, ij \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{j\sigma} a_{k\sigma'}^\dagger a_{l\sigma'} \\
&= \frac{1}{2} \sum_{ij \in L, kl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{j\sigma} a_{k\sigma'}^\dagger a_{l\sigma'} + \frac{1}{2} \sum_{kl \in L, ij \in R, \sigma\sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'} a_{i\sigma}^\dagger a_{j\sigma} \\
&= \frac{1}{2} \sum_{ij \in L, kl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{j\sigma} a_{k\sigma'}^\dagger a_{l\sigma'} + \frac{1}{2} \sum_{ij \in L, kl \in R, \sigma\sigma'} v_{klij} a_{i\sigma}^\dagger a_{j\sigma} a_{k\sigma}^\dagger a_{l\sigma} \\
&= \sum_{ij \in L, kl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{j\sigma} a_{k\sigma'}^\dagger a_{l\sigma'} = \sum_{ij \in L} \sum_{\sigma} a_{i\sigma}^\dagger a_{j\sigma} \sum_{kl \in R_k} \sum_{\sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'} = \sum_{ij \in L} \hat{B}_{ij} \hat{Q}_{ij}^R
\end{aligned}$$

For the different-spin case,

$$\begin{aligned}
&-\frac{1}{2} \sum_{il \in L, jk \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{l\sigma'} a_{k\sigma'}^\dagger a_{j\sigma} - \frac{1}{2} \sum_{jk \in L, il \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{l\sigma'} a_{k\sigma'}^\dagger a_{j\sigma} = - \sum_{il \in L, jk \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{l\sigma'} a_{k\sigma'}^\dagger a_{j\sigma} \\
&= - \sum_{il \in L, \sigma\sigma'} a_{i\sigma}^\dagger a_{l\sigma'} \sum_{jk \in R} v_{ijkl} a_{k\sigma'}^\dagger a_{j\sigma} = - \sum_{il \in L, \sigma\sigma'} \hat{B}'_{il\sigma\sigma'} \hat{Q}'_{il\sigma\sigma'}^R
\end{aligned}$$

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

$$\begin{aligned}\hat{H}^{NC} &= \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R \\ &+ \sum_{i \in L, \sigma} \left(a_{i\sigma}^\dagger \hat{S}_{i\sigma}^R - a_{i\sigma} \hat{S}_{i\sigma}^{R\dagger} \right) + \sum_{i \in L, \sigma} \left(a_{i\sigma}^\dagger \hat{R}_{i\sigma}^R - a_{i\sigma} \hat{R}_{i\sigma}^{R\dagger} \right) + \sum_{i \in R, \sigma} \left(\hat{R}_{i\sigma}^{L\dagger} a_{i\sigma} - \hat{R}_{i\sigma}^L a_{i\sigma}^\dagger \right) \\ &+ \frac{1}{2} \sum_{ik \in L, \sigma\sigma'} \left(\hat{A}_{ik, \sigma\sigma'} \hat{P}_{ik, \sigma\sigma'}^R + \hat{A}_{ik, \sigma\sigma'}^\dagger \hat{P}_{ik, \sigma\sigma'}^{R\dagger} \right) + \sum_{ij \in L} \hat{B}_{ij} \hat{Q}_{ij}^R - \sum_{il \in L, \sigma\sigma'} \hat{B}'_{il\sigma\sigma'} \hat{Q}_{il\sigma\sigma'}^R\end{aligned}$$

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

$$\hat{R}_{i\sigma}^{L/R} = \frac{1}{2} \hat{S}_{i\sigma}^{L/R} + \hat{R}_{i\sigma}^{L/R} = \frac{1}{2} \sum_{j \in L/R} t_{ij} a_{j\sigma} + \sum_{jkl \in L/R, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma}$$

we have

$$\begin{aligned}\hat{H}^{NC} &= \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R + \sum_{i \in L, \sigma} \left(a_{i\sigma}^\dagger \hat{R}_{i\sigma}^{R'} - a_{i\sigma} \hat{R}_{i\sigma}^{R'\dagger} \right) + \sum_{i \in R, \sigma} \left(\hat{R}_{i\sigma}^{L\dagger} a_{i\sigma} - \hat{R}_{i\sigma}^L a_{i\sigma}^\dagger \right) \\ &+ \frac{1}{2} \sum_{ik \in L, \sigma\sigma'} \left(\hat{A}_{ik, \sigma\sigma'} \hat{P}_{ik, \sigma\sigma'}^R + \hat{A}_{ik, \sigma\sigma'}^\dagger \hat{P}_{ik, \sigma\sigma'}^{R\dagger} \right) + \sum_{ij \in L} \hat{B}_{ij} \hat{Q}_{ij}^R - \sum_{il \in L, \sigma\sigma'} \hat{B}'_{il\sigma\sigma'} \hat{Q}_{il\sigma\sigma'}^R\end{aligned}$$

With this normal/complementary partitioning, the operators required in left block are

$$\{ \hat{H}^L, \hat{1}^L, a_{i\sigma}^\dagger, a_{i\sigma}, \hat{R}_{k\sigma}^{L\dagger}, \hat{R}_{k\sigma}^L, \hat{A}_{ij, \sigma\sigma'}, \hat{A}_{ij, \sigma\sigma'}^\dagger, \hat{B}_{ij}, \hat{B}'_{ij, \sigma\sigma'} \} \quad (i, j \in L, k \in R)$$

The operators required in right block are

$$\{ \hat{1}^R, \hat{H}^R, \hat{R}_{i\sigma}^{R'}, \hat{R}_{i\sigma}^{R'\dagger}, a_{k\sigma}, a_{k\sigma}^\dagger, \hat{P}_{ij, \sigma\sigma'}^R, \hat{P}_{ij, \sigma\sigma'}^{R\dagger}, \hat{Q}_{ij}^R, \hat{Q}_{ij, \sigma\sigma'}^R \} \quad (i, j \in L, k \in R)$$

Assuming that there are K sites in total, and K_L/K_R sites in left/right block (optimally, $K_L \leq K_R$), 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

$$\begin{aligned}\hat{H}^{CN} = & \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R + \sum_{i \in L, \sigma} \left(a_{i\sigma}^\dagger \hat{R}_{i\sigma}^R - a_{i\sigma} \hat{R}_{i\sigma}^{R\dagger} \right) + \sum_{i \in R, \sigma} \left(\hat{R}_{i\sigma}^{L\dagger} a_{i\sigma} - \hat{R}_{i\sigma}^L a_{i\sigma}^\dagger \right) \\ & + \frac{1}{2} \sum_{jl \in R, \sigma\sigma'} \left(\hat{P}_{jl, \sigma\sigma'}^L \hat{A}_{jl, \sigma\sigma'} + \hat{P}_{jl, \sigma\sigma'}^{L\dagger} \hat{A}_{jl, \sigma\sigma'}^\dagger \right) + \sum_{kl \in R} \hat{Q}_{kl}^L \hat{B}_{kl} - \sum_{jk \in R, \sigma\sigma'} \hat{Q}_{jk, \sigma\sigma'}^{L\dagger} \hat{B}_{jk, \sigma\sigma'}'\end{aligned}$$

Now the operators required in left block are

$$\{ \hat{H}^L, \hat{1}^L, a_{i\sigma}^\dagger, a_{i\sigma}, \hat{R}_{k\sigma}^{L\dagger}, \hat{R}_{k\sigma}^L, \hat{P}_{kl, \sigma\sigma'}^L, \hat{P}_{kl, \sigma\sigma'}^{L\dagger}, \hat{Q}_{kl}^L, \hat{Q}_{kl, \sigma\sigma'}^{L\dagger} \} \quad (k, l \in R, i \in L)$$

The operators required in right block are

$$\{ \hat{1}^R, \hat{H}^R, \hat{R}_{i\sigma}^R, \hat{R}_{i\sigma}^{R\dagger}, a_{k\sigma}, a_{k\sigma}^\dagger, \hat{A}_{kl, \sigma\sigma'}, \hat{A}_{kl, \sigma\sigma'}^\dagger, \hat{B}_{kl}, \hat{B}_{kl, \sigma\sigma'}' \} \quad (k, l \in R, i \in L)$$

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.

$$\begin{aligned}\hat{R}_{i\sigma}^{L*} = & \hat{R}_{i\sigma}^{L\dagger} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_{i\sigma}^{*} + \sum_{j \in L} \left(\sum_{kl \in *, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'} \right) a_{j\sigma} + \sum_{j \in *} \left(\sum_{kl \in L, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'} \right) a_{j\sigma} \\ & + \sum_{k \in L, \sigma'} a_{k\sigma'}^\dagger \left(\sum_{jl \in *} v_{ijkl} a_{l\sigma'} a_{j\sigma} \right) + \sum_{k \in *, \sigma'} a_{k\sigma'}^\dagger \left(\sum_{jl \in L} v_{ijkl} a_{l\sigma'} a_{j\sigma} \right) - \sum_{l \in L, \sigma'} a_{l\sigma'} \left(\sum_{jk \in *} v_{ijkl} a_{k\sigma'}^\dagger a_{j\sigma} \right) - \sum_{l \in *, \sigma'} a_{l\sigma'} \left(\sum_{jk \in L} v_{ijkl} a_{k\sigma'}^\dagger a_{j\sigma} \right) \\ = & \hat{R}_{i\sigma}^{L\dagger} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_{i\sigma}^{*} + \sum_{j \in L} a_{j\sigma} \left(\sum_{kl \in *, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'} \right) + \sum_{j \in *} \left(\sum_{kl \in L, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'} \right) a_{j\sigma} \\ & + \sum_{k \in L, \sigma'} a_{k\sigma'}^\dagger \left(\sum_{jl \in *} v_{ijkl} a_{l\sigma'} a_{j\sigma} \right) + \sum_{k \in *, \sigma'} \left(\sum_{jl \in L} v_{ijkl} a_{l\sigma'} a_{j\sigma} \right) a_{k\sigma'}^\dagger - \sum_{l \in L, \sigma'} a_{l\sigma'} \left(\sum_{jk \in *} v_{ijkl} a_{k\sigma'}^\dagger a_{j\sigma} \right) - \sum_{l \in *, \sigma'} \left(\sum_{jk \in L} v_{ijkl} a_{k\sigma'}^\dagger a_{j\sigma} \right) a_{l\sigma'}\end{aligned}$$

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

$$\begin{aligned}\hat{R}_{i\sigma}^{L*, NC} = & \hat{R}_{i\sigma}^{L\dagger} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_{i\sigma}^{*} + \sum_{j \in L} a_{j\sigma} \hat{Q}_{ij}^* + \sum_{j \in *, kl \in L} v_{ijkl} \hat{B}_{kl} a_{j\sigma} \\ & + \sum_{k \in L, \sigma'} a_{k\sigma'}^\dagger \hat{P}_{ik, \sigma\sigma'}^* + \sum_{k \in *, jl \in L, \sigma'} v_{ijkl} \hat{A}_{jl, \sigma\sigma'}^\dagger a_{k\sigma'}^\dagger - \sum_{l \in L, \sigma'} a_{l\sigma'} \hat{Q}_{il, \sigma\sigma'}^* - \sum_{l \in *, jk \in L, \sigma'} v_{ijkl} \hat{B}_{kj, \sigma'\sigma}^* a_{l\sigma'} \\ = & \hat{R}_{i\sigma}^{L\dagger} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_{i\sigma}^{*} + \sum_{k \in L, \sigma'} a_{k\sigma'}^\dagger \hat{P}_{ik, \sigma\sigma'}^* + \sum_{j \in L} a_{j\sigma} \hat{Q}_{ij}^* - \sum_{l \in L, \sigma'} a_{l\sigma'} \hat{Q}_{il, \sigma\sigma'}^* \\ & + \sum_{k \in *, jl \in L, \sigma'} v_{ijkl} \hat{A}_{jl, \sigma\sigma'}^\dagger a_{k\sigma'}^\dagger + \sum_{j \in *, kl \in L} v_{ijkl} \hat{B}_{kl} a_{j\sigma} - \sum_{l \in *, jk \in L, \sigma'} v_{ijkl} \hat{B}_{kj, \sigma'\sigma}^* a_{l\sigma'}\end{aligned}$$

$$\begin{aligned}
\hat{R}_{i\sigma}^{L*,CN} &= \hat{R}_{i\sigma}^{L*} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_{i\sigma}^{L*} + \sum_{j \in L, kl \in *} v_{ijkl} a_{j\sigma} \hat{B}_{kl} + \sum_{j \in *} \hat{Q}_{ij}^L a_{j\sigma} \\
&\quad + \sum_{k \in L, jl \in *, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger \hat{A}_{jl, \sigma\sigma'}^\dagger + \sum_{k \in *, \sigma'} \hat{P}_{ik, \sigma\sigma'}^L a_{k\sigma'}^\dagger - \sum_{l \in L, jk \in *, \sigma'} v_{ijkl} a_{l\sigma'} \hat{B}_{kj, \sigma'\sigma} - \sum_{l \in *, \sigma'} \hat{Q}_{il, \sigma\sigma'}^L a_{l\sigma'} \\
&= \hat{R}_{i\sigma}^{L*} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_{i\sigma}^{L*} + \sum_{k \in L, jl \in *, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger \hat{A}_{jl, \sigma\sigma'}^\dagger + \sum_{j \in L, kl \in *} v_{ijkl} a_{j\sigma} \hat{B}_{kl} - \sum_{l \in L, jk \in *, \sigma'} v_{ijkl} a_{l\sigma'} \hat{B}_{kj, \sigma'\sigma} \\
&\quad + \sum_{k \in *, \sigma'} \hat{P}_{ik, \sigma\sigma'}^L a_{k\sigma'}^\dagger + \sum_{j \in *} \hat{Q}_{ij}^L a_{j\sigma} - \sum_{l \in *, \sigma'} \hat{Q}_{il, \sigma\sigma'}^L a_{l\sigma'}
\end{aligned}$$

Similarly,

$$\begin{aligned}
\hat{R}_{i\sigma}^{R*,NC} &= \hat{R}_{i\sigma}^{R*} \otimes \hat{1}^R + \hat{1}^* \otimes \hat{R}_{i\sigma}^{R*} + \sum_{k \in *, \sigma'} a_{k\sigma'}^\dagger \hat{P}_{ik, \sigma\sigma'}^R + \sum_{j \in *} a_{j\sigma} \hat{Q}_{ij}^R - \sum_{l \in *, \sigma'} a_{l\sigma'} \hat{Q}_{il, \sigma\sigma'}^R \\
&\quad + \sum_{k \in R, jl \in *, \sigma'} v_{ijkl} \hat{A}_{jl, \sigma\sigma'}^\dagger a_{k\sigma'}^\dagger + \sum_{j \in R, kl \in *} v_{ijkl} \hat{B}_{kl} a_{j\sigma} - \sum_{l \in R, jk \in *, \sigma'} v_{ijkl} \hat{B}_{kj, \sigma'\sigma} a_{l\sigma'} \\
\hat{R}_{i\sigma}^{R*,CN} &= \hat{R}_{i\sigma}^{R*} \otimes \hat{1}^R + \hat{1}^* \otimes \hat{R}_{i\sigma}^{R*} + \sum_{k \in *, jl \in R, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger \hat{A}_{jl, \sigma\sigma'}^\dagger + \sum_{j \in *, kl \in R} v_{ijkl} a_{j\sigma} \hat{B}_{kl} - \sum_{l \in *, jk \in R, \sigma'} v_{ijkl} a_{l\sigma'} \hat{B}_{kj, \sigma'\sigma} \\
&\quad + \sum_{k \in R, \sigma'} \hat{P}_{ik, \sigma\sigma'}^* a_{k\sigma'}^\dagger + \sum_{j \in R} \hat{Q}_{ij}^* a_{j\sigma} - \sum_{l \in R, \sigma'} \hat{Q}_{il, \sigma\sigma'}^* a_{l\sigma'}
\end{aligned}$$

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

$$\begin{aligned}
\hat{P}_{ik, \sigma\sigma'}^{L*,CN} &= \hat{P}_{ik, \sigma\sigma'}^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{P}_{ik, \sigma\sigma'}^* + \sum_{j \in L, l \in *} v_{ijkl} a_{l\sigma'} a_{j\sigma} + \sum_{j \in *, l \in L} v_{ijkl} a_{l\sigma'} a_{j\sigma} \\
&= \hat{P}_{ik, \sigma\sigma'}^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{P}_{ik, \sigma\sigma'}^* - \sum_{j \in L, l \in *} v_{ijkl} a_{j\sigma} a_{l\sigma'} + \sum_{j \in *, l \in L} v_{ijkl} a_{l\sigma'} a_{j\sigma} \\
\hat{P}_{ik, \sigma\sigma'}^{R*,NC} &= \hat{P}_{ik, \sigma\sigma'}^* \otimes \hat{1}^R + \hat{1}^* \otimes \hat{P}_{ik, \sigma\sigma'}^R + \sum_{j \in *, l \in R} v_{ijkl} a_{l\sigma'} a_{j\sigma} + \sum_{j \in R, l \in *} v_{ijkl} a_{l\sigma'} a_{j\sigma} \\
&= \hat{P}_{ik, \sigma\sigma'}^* \otimes \hat{1}^R + \hat{1}^* \otimes \hat{P}_{ik, \sigma\sigma'}^R - \sum_{j \in *, l \in R} v_{ijkl} a_{j\sigma} a_{l\sigma'} + \sum_{j \in R, l \in *} v_{ijkl} a_{l\sigma'} a_{j\sigma}
\end{aligned}$$

and

$$\begin{aligned}
\hat{Q}_{ij}^{L*,CN} &= \hat{Q}_{ij}^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{Q}_{ij}^* + \sum_{k \in L, l \in *, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'} + \sum_{k \in *, l \in L, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'} \\
&= \hat{Q}_{ij}^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{Q}_{ij}^* + \sum_{k \in L, l \in *, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'} - \sum_{k \in *, l \in L, \sigma'} v_{ijkl} a_{l\sigma'} a_{k\sigma'}^\dagger \\
\hat{Q}_{ij}^{R*,NC} &= \hat{Q}_{ij}^* \otimes \hat{1}^R + \hat{1}^* \otimes \hat{Q}_{ij}^R + \sum_{k \in *, l \in R, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'} + \sum_{k \in R, l \in *, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'} \\
&= \hat{Q}_{ij}^* \otimes \hat{1}^R + \hat{1}^* \otimes \hat{Q}_{ij}^R + \sum_{k \in *, l \in R, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'} - \sum_{k \in R, l \in *, \sigma'} v_{ijkl} a_{l\sigma'} a_{k\sigma'}^\dagger
\end{aligned}$$

and

$$\begin{aligned}
\hat{Q}_{il,\sigma\sigma'}^{L*,CN} &= \hat{Q}_{il,\sigma\sigma'}^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{Q}_{il,\sigma\sigma'}^{*'} + \sum_{j \in L, k \in *} v_{ijkl} a_{k\sigma'}^\dagger a_{j\sigma} + \sum_{j \in *, k \in L} v_{ijkl} a_{k\sigma'}^\dagger a_{j\sigma} \\
&= \hat{Q}_{il,\sigma\sigma'}^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{Q}_{il,\sigma\sigma'}^{*'} - \sum_{j \in L, k \in *} v_{ijkl} a_{j\sigma} a_{k\sigma'}^\dagger + \sum_{j \in *, k \in L} v_{ijkl} a_{k\sigma'}^\dagger a_{j\sigma} \\
\hat{Q}_{il,\sigma\sigma'}^{R*,NC} &= \hat{Q}_{il,\sigma\sigma'}^{*'} \otimes \hat{1}^R + \hat{1}^* \otimes \hat{Q}_{il,\sigma\sigma'}^R + \sum_{j \in *, k \in R} v_{ijkl} a_{k\sigma'}^\dagger a_{j\sigma} + \sum_{j \in R, k \in *} v_{ijkl} a_{k\sigma'}^\dagger a_{j\sigma} \\
&= \hat{Q}_{il,\sigma\sigma'}^{*'} \otimes \hat{1}^R + \hat{1}^* \otimes \hat{Q}_{il,\sigma\sigma'}^R - \sum_{j \in *, k \in R} v_{ijkl} a_{j\sigma} a_{k\sigma'}^\dagger + \sum_{j \in R, k \in *} v_{ijkl} a_{k\sigma'}^\dagger a_{j\sigma}
\end{aligned}$$

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 $O(K_{<}^2)$.

$$\begin{aligned}
\hat{P}_{ik,\sigma\sigma'}^{L,NC \rightarrow CN} &= \sum_{jl \in L} v_{ijkl} a_{l\sigma'} a_{j\sigma} = \sum_{jl \in L} v_{ijkl} \hat{A}_{jl,\sigma\sigma'}^\dagger \\
\hat{Q}_{ij}^{L,NC \rightarrow CN} &= \sum_{kl \in L, \sigma'} v_{ijkl} a_{k\sigma'}^\dagger a_{l\sigma'} = \sum_{kl \in L} v_{ijkl} \hat{B}_{kl} \\
\hat{Q}_{il,\sigma\sigma'}^{L,NC \rightarrow CN} &= \sum_{jk \in L} v_{ijkl} a_{k\sigma'}^\dagger a_{j\sigma} = \sum_{jk \in L} v_{ijkl} \hat{B}'_{kj,\sigma'\sigma}
\end{aligned}$$

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

$$\begin{aligned}
(\hat{H})^{[0]} &= (\hat{H}^L)^{[0]} \otimes_{[0]} (\hat{1}^R)^{[0]} + (\hat{1}^L)^{[0]} \otimes_{[0]} (\hat{H}^R)^{[0]} \\
&\quad + \sqrt{2} \sum_{i \in L} \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (\hat{S}_i^R)^{[\frac{1}{2}]} + h.c. \right] \\
&\quad + 2 \sum_{i \in L} \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (\hat{R}_i^R)^{[\frac{1}{2}]} + h.c. \right] + 2 \sum_{i \in R} \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (\hat{R}_i^L)^{[\frac{1}{2}]} + h.c. \right] \\
&\quad - \frac{1}{2} \sum_{ik \in L} \left[\sqrt{3} (\hat{A}_{ik})^{[1]} \otimes_{[0]} (\hat{P}_{ik}^R)^{[1]} + (\hat{A}_{ik})^{[0]} \otimes_{[0]} (\hat{P}_{ik}^R)^{[0]} + h.c. \right] \\
&\quad + \sum_{ij \in L} \left[(\hat{B}_{ij})^{[0]} \otimes_{[0]} \left(2(\hat{Q}_{ij}^R)^{[0]} - (\hat{Q}_{ij}^{*R})^{[0]} \right) + \sqrt{3} (\hat{B}_{ij}')^{[1]} \otimes_{[0]} (\hat{Q}_{ij}^{*R})^{[1]} \right]
\end{aligned}$$

where the normal and complementary operators are defined by

$$\begin{aligned}
(\hat{S}_i^{L/R})^{[\frac{1}{2}]} &= \sum_{j \in L/R} t_{ij} (a_j)^{[\frac{1}{2}]} \\
(\hat{R}_i^{L/R})^{[\frac{1}{2}]} &= \sum_{jkl \in L/R} v_{ijkl} \left[(a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]} \right] \otimes_{[\frac{1}{2}]} (a_j)^{[\frac{1}{2}]} \\
(\hat{A}_{ik})^{[0/1]} &= (a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0/1]} (a_k^\dagger)^{[\frac{1}{2}]} \\
(\hat{P}_{ik}^R)^{[0/1]} &= \sum_{jl \in R} v_{ijkl} (a_j)^{[\frac{1}{2}]} \otimes_{[0/1]} (a_l)^{[\frac{1}{2}]} \\
(\hat{B}_{ij})^{[0]} &= (a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_j)^{[\frac{1}{2}]} \\
(\hat{B}'_{ij})^{[1]} &= (a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[1]} (a_j)^{[\frac{1}{2}]} \\
(\hat{Q}_{ij}^R)^{[0]} &= \sum_{kl \in R} v_{ijkl} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]} \\
(\hat{Q}'_{ij})^{[0/1]} &= \sum_{kl \in R} v_{ilkj} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0/1]} (a_l)^{[\frac{1}{2}]} \\
(\hat{Q}''_{ij})^{[0]} &:= 2(\hat{Q}_{ij}^R)^{[0]} - (\hat{Q}'_{ij})^{[0]} = \sum_{kl \in R} (2v_{ijkl} - v_{ilkj}) (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]}
\end{aligned}$$

Derivation

CG Factors

From $j_2 = 1/2$ CG factors

$$\begin{aligned}
\left\langle j_1 \left(M - \frac{1}{2} \right) \frac{1}{2} \frac{1}{2} \left| \left(j_1 \pm \frac{1}{2} \right) M \right\rangle &= \pm \sqrt{\frac{1}{2} \left(1 \pm \frac{M}{j_1 + \frac{1}{2}} \right)} \\
\left\langle j_1 \left(M + \frac{1}{2} \right) \frac{1}{2} \left(-\frac{1}{2} \right) \left| \left(j_1 \pm \frac{1}{2} \right) M \right\rangle &= \sqrt{\frac{1}{2} \left(1 \mp \frac{M}{j_1 + \frac{1}{2}} \right)}
\end{aligned}$$

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

$$\begin{aligned}
\left\langle \frac{1}{2} \frac{1}{2} j_1 \left(M - \frac{1}{2} \right) \left| \left(j_1 \pm \frac{1}{2} \right) M \right\rangle &= \sqrt{\frac{1}{2} \left(1 \pm \frac{M}{j_1 + \frac{1}{2}} \right)} \\
\left\langle \frac{1}{2} \left(-\frac{1}{2} \right) j_1 \left(M + \frac{1}{2} \right) \left| \left(j_1 \pm \frac{1}{2} \right) M \right\rangle &= \pm \sqrt{\frac{1}{2} \left(1 \mp \frac{M}{j_1 + \frac{1}{2}} \right)}
\end{aligned}$$

let $j_1 = 1$, we have

$$\begin{aligned}\langle \frac{1}{2} \frac{1}{2} 1 (M - \frac{1}{2}) | \frac{1}{2} M \rangle &= \sqrt{\frac{1}{2}(1 - \frac{M}{\frac{3}{2}})} \\ \langle \frac{1}{2} (-\frac{1}{2}) 1 (M + \frac{1}{2}) | \frac{1}{2} M \rangle &= -\sqrt{\frac{1}{2}(1 + \frac{M}{\frac{3}{2}})}\end{aligned}$$

So the coefficients for $[\frac{1}{2}] \otimes_{[\frac{1}{2}]} [1]$ are

$$\begin{aligned}[\frac{1}{2} + 0 = \frac{1}{2}] &= \sqrt{\frac{1}{3}}, \quad [-\frac{1}{2} + 1 = \frac{1}{2}] = -\sqrt{\frac{2}{3}} \\ [\frac{1}{2} + (-1) = -\frac{1}{2}] &= \sqrt{\frac{2}{3}}, \quad [-\frac{1}{2} + 0 = -\frac{1}{2}] = -\sqrt{\frac{1}{3}}\end{aligned}$$

The coefficients for $[1] \otimes_{[\frac{1}{2}]} [\frac{1}{2}]$ are

$$\begin{aligned}[0 + \frac{1}{2} = \frac{1}{2}] &= -\sqrt{\frac{1}{3}}, \quad [1 - \frac{1}{2} = \frac{1}{2}] = \sqrt{\frac{2}{3}} \\ [(-1) + \frac{1}{2} = -\frac{1}{2}] &= -\sqrt{\frac{2}{3}}, \quad [0 - \frac{1}{2} = -\frac{1}{2}] = \sqrt{\frac{1}{3}}\end{aligned}$$

This means that the SU(2) operator exchange factor for $[\frac{1}{2}] \otimes_{[\frac{1}{2}]} [1] \rightarrow [1] \otimes_{[\frac{1}{2}]} [\frac{1}{2}]$ is -1 . The fermion factor is $+1$. So the overall exchange factor for this case is -1 .

Tensor Product Formulas

Singlet

$$\begin{aligned}(a_p^\dagger)^{[1/2]} \otimes_{[0]} (a_q^\dagger)^{[1/2]} &= \begin{pmatrix} a_{p\alpha}^\dagger \\ a_{p\beta}^\dagger \end{pmatrix}^{[1/2]} \otimes_{[0]} \begin{pmatrix} a_{q\alpha}^\dagger \\ a_{q\beta}^\dagger \end{pmatrix}^{[1/2]} = \frac{1}{\sqrt{2}} (a_{p\alpha}^\dagger a_{q\beta}^\dagger - a_{p\beta}^\dagger a_{q\alpha}^\dagger)^{[0]} \\ (a_p^\dagger)^{[1/2]} \otimes_{[0]} (a_q)^{[1/2]} &= \begin{pmatrix} a_{p\alpha}^\dagger \\ a_{p\beta}^\dagger \end{pmatrix}^{[1/2]} \otimes_{[0]} \begin{pmatrix} -a_{q\beta} \\ a_{q\alpha} \end{pmatrix}^{[1/2]} = \frac{1}{\sqrt{2}} (a_{p\alpha}^\dagger a_{q\alpha} + a_{p\beta}^\dagger a_{q\beta})^{[0]} \\ (a_p)^{[1/2]} \otimes_{[0]} (a_q)^{[1/2]} &= \begin{pmatrix} -a_{p\beta} \\ a_{p\alpha} \end{pmatrix}^{[1/2]} \otimes_{[0]} \begin{pmatrix} -a_{q\beta} \\ a_{q\alpha} \end{pmatrix}^{[1/2]} = \frac{1}{\sqrt{2}} (-a_{p\beta} a_{q\alpha} + a_{p\alpha} a_{q\beta})^{[0]}\end{aligned}$$

Triplet

$$\begin{aligned}(a_p^\dagger)^{[1/2]} \otimes_{[1]} (a_q^\dagger)^{[1/2]} &= \begin{pmatrix} a_{p\alpha}^\dagger \\ a_{p\beta}^\dagger \end{pmatrix}^{[1/2]} \otimes_{[1]} \begin{pmatrix} a_{q\alpha}^\dagger \\ a_{q\beta}^\dagger \end{pmatrix}^{[1/2]} = \begin{pmatrix} a_{p\alpha}^\dagger a_{q\alpha}^\dagger \\ \frac{1}{\sqrt{2}} (a_{p\alpha}^\dagger a_{q\beta}^\dagger + a_{p\beta}^\dagger a_{q\alpha}^\dagger) \\ a_{p\beta}^\dagger a_{q\beta}^\dagger \end{pmatrix}^{[1]} \\ (a_p^\dagger)^{[1/2]} \otimes_{[1]} (a_q)^{[1/2]} &= \begin{pmatrix} a_{p\alpha}^\dagger \\ a_{p\beta}^\dagger \end{pmatrix}^{[1/2]} \otimes_{[1]} \begin{pmatrix} -a_{q\beta} \\ a_{q\alpha} \end{pmatrix}^{[1/2]} = \begin{pmatrix} -a_{p\alpha}^\dagger a_{q\beta} \\ \frac{1}{\sqrt{2}} (a_{p\alpha}^\dagger a_{q\alpha} - a_{p\beta}^\dagger a_{q\beta}) \\ a_{p\beta}^\dagger a_{q\alpha} \end{pmatrix}^{[1]} \\ (a_p)^{[1/2]} \otimes_{[1]} (a_q)^{[1/2]} &= \begin{pmatrix} -a_{p\beta} \\ a_{p\alpha} \end{pmatrix}^{[1/2]} \otimes_{[1]} \begin{pmatrix} -a_{q\beta} \\ a_{q\alpha} \end{pmatrix}^{[1/2]} = \begin{pmatrix} a_{p\beta} a_{q\beta} \\ -\frac{1}{\sqrt{2}} (a_{p\beta} a_{q\alpha} + a_{p\alpha} a_{q\beta}) \\ a_{p\alpha} a_{q\alpha} \end{pmatrix}^{[1]}\end{aligned}$$

Doublet times singlet/triplet

$$\begin{aligned}
U^{[1/2]} &= (a_p^\dagger)^{[1/2]} \otimes_{[1/2]} \left[(a_r)^{[1/2]} \otimes_{[1]} (a_s)^{[1/2]} \right] = \begin{pmatrix} a_{p\alpha}^\dagger \\ a_{p\beta}^\dagger \end{pmatrix}^{[1/2]} \otimes_{[1/2]} \begin{pmatrix} a_{r\beta} a_{s\beta} \\ -\frac{1}{\sqrt{2}} (a_{r\beta} a_{s\alpha} + a_{r\alpha} a_{s\beta}) \\ a_{r\alpha} a_{s\alpha} \end{pmatrix}^{[1]} \\
&= \begin{pmatrix} -\frac{1}{\sqrt{2}} \frac{1}{\sqrt{3}} a_{p\alpha}^\dagger (a_{r\beta} a_{s\alpha} + a_{r\alpha} a_{s\beta}) - \frac{\sqrt{2}}{\sqrt{3}} a_{p\beta}^\dagger a_{r\beta} a_{s\beta} \\ \frac{\sqrt{2}}{\sqrt{3}} a_{p\alpha}^\dagger a_{r\alpha} a_{s\alpha} + (-\frac{1}{\sqrt{3}}) (-\frac{1}{\sqrt{2}}) a_{p\beta}^\dagger (a_{r\beta} a_{s\alpha} + a_{r\alpha} a_{s\beta}) \end{pmatrix}^{[1/2]} = \frac{1}{\sqrt{6}} \begin{pmatrix} -a_{p\alpha}^\dagger a_{r\beta} a_{s\alpha} - a_{p\alpha}^\dagger a_{r\alpha} a_{s\beta} - 2a_{p\beta}^\dagger a_{r\beta} a_{s\beta} \\ 2a_{p\alpha}^\dagger a_{r\alpha} a_{s\alpha} + a_{p\beta}^\dagger a_{r\beta} a_{s\alpha} + a_{p\beta}^\dagger a_{r\alpha} a_{s\beta} \end{pmatrix} \\
V^{[1/2]} &= (a_p^\dagger)^{[1/2]} \otimes_{[1/2]} \left[(a_r)^{[1/2]} \otimes_{[0]} (a_s)^{[1/2]} \right] = \frac{1}{\sqrt{2}} \begin{pmatrix} a_{p\alpha}^\dagger \\ a_{p\beta}^\dagger \end{pmatrix}^{[1/2]} \otimes_{[1/2]} (-a_{r\beta} a_{s\alpha} + a_{r\alpha} a_{s\beta})^{[0]} \\
&= \frac{1}{\sqrt{2}} \begin{pmatrix} -a_{p\alpha}^\dagger a_{r\beta} a_{s\alpha} + a_{p\alpha}^\dagger a_{r\alpha} a_{s\beta} \\ -a_{p\beta}^\dagger a_{r\beta} a_{s\alpha} + a_{p\beta}^\dagger a_{r\alpha} a_{s\beta} \end{pmatrix}^{[1/2]}
\end{aligned}$$

Therefore,

$$\begin{aligned}
\sqrt{3}U^{[1/2]} - V^{[1/2]} &= \frac{1}{\sqrt{2}} \begin{pmatrix} -a_{p\alpha}^\dagger a_{r\beta} a_{s\alpha} - a_{p\alpha}^\dagger a_{r\alpha} a_{s\beta} - 2a_{p\beta}^\dagger a_{r\beta} a_{s\beta} \\ 2a_{p\alpha}^\dagger a_{r\alpha} a_{s\alpha} + a_{p\beta}^\dagger a_{r\beta} a_{s\alpha} + a_{p\beta}^\dagger a_{r\alpha} a_{s\beta} \end{pmatrix}^{[1/2]} - \frac{1}{\sqrt{2}} \begin{pmatrix} -a_{p\alpha}^\dagger a_{r\beta} a_{s\alpha} + a_{p\alpha}^\dagger a_{r\alpha} a_{s\beta} \\ -a_{p\beta}^\dagger a_{r\beta} a_{s\alpha} + a_{p\beta}^\dagger a_{r\alpha} a_{s\beta} \end{pmatrix}^{[1/2]} \\
&= \frac{1}{\sqrt{2}} \begin{pmatrix} -a_{p\alpha}^\dagger a_{r\beta} a_{s\alpha} - a_{p\alpha}^\dagger a_{r\alpha} a_{s\beta} - 2a_{p\beta}^\dagger a_{r\beta} a_{s\beta} + a_{p\alpha}^\dagger a_{r\beta} a_{s\alpha} - a_{p\alpha}^\dagger a_{r\alpha} a_{s\beta} \\ 2a_{p\alpha}^\dagger a_{r\alpha} a_{s\alpha} + a_{p\beta}^\dagger a_{r\beta} a_{s\alpha} + a_{p\beta}^\dagger a_{r\alpha} a_{s\beta} + a_{p\beta}^\dagger a_{r\beta} a_{s\alpha} - a_{p\beta}^\dagger a_{r\alpha} a_{s\beta} \end{pmatrix}^{[1/2]} \\
&= \sqrt{2} \begin{pmatrix} -a_{p\alpha}^\dagger a_{r\alpha} a_{s\beta} - a_{p\beta}^\dagger a_{r\beta} a_{s\beta} \\ a_{p\alpha}^\dagger a_{r\alpha} a_{s\alpha} + a_{p\beta}^\dagger a_{r\beta} a_{s\alpha} \end{pmatrix}^{[1/2]}
\end{aligned}$$

Another case

$$\begin{aligned}
S^{[1/2]} &= (a_r)^{[1/2]} \otimes_{[1/2]} \left[(a_p^\dagger)^{[1/2]} \otimes_{[1]} (a_q)^{[1/2]} \right] = \begin{pmatrix} -a_{r\beta} \\ a_{r\alpha} \end{pmatrix}^{[1/2]} \otimes_{[1/2]} \begin{pmatrix} -a_{p\alpha}^\dagger a_{q\beta} \\ \frac{1}{\sqrt{2}} (a_{p\alpha}^\dagger a_{q\alpha} - a_{p\beta}^\dagger a_{q\beta}) \\ a_{p\beta}^\dagger a_{q\alpha} \end{pmatrix}^{[1]} \\
&= \begin{pmatrix} \frac{1}{\sqrt{2}} \frac{1}{\sqrt{3}} (-a_{r\beta}) (a_{p\alpha}^\dagger a_{q\alpha} - a_{p\beta}^\dagger a_{q\beta}) + \frac{\sqrt{2}}{\sqrt{3}} a_{r\alpha} a_{p\alpha}^\dagger a_{q\beta} \\ -\frac{\sqrt{2}}{\sqrt{3}} a_{r\beta} a_{p\beta}^\dagger a_{q\alpha} - \frac{1}{\sqrt{2}} \frac{1}{\sqrt{3}} a_{r\alpha} (a_{p\alpha}^\dagger a_{q\alpha} - a_{p\beta}^\dagger a_{q\beta}) \end{pmatrix}^{[1/2]} = \frac{1}{\sqrt{6}} \begin{pmatrix} -a_{r\beta} a_{p\alpha}^\dagger a_{q\alpha} + a_{r\beta} a_{p\beta}^\dagger a_{q\beta} + 2a_{r\alpha} a_{p\alpha}^\dagger a_{q\beta} \\ -2a_{r\beta} a_{p\beta}^\dagger a_{q\alpha} - a_{r\alpha} a_{p\alpha}^\dagger a_{q\alpha} + a_{r\alpha} a_{p\beta}^\dagger a_{q\beta} \end{pmatrix}^{[1/2]} \\
T^{[1/2]} &= (a_r)^{[1/2]} \otimes_{[1/2]} \left[(a_p^\dagger)^{[1/2]} \otimes_{[0]} (a_q)^{[1/2]} \right] = \frac{1}{\sqrt{2}} \begin{pmatrix} -a_{r\beta} \\ a_{r\alpha} \end{pmatrix}^{[1/2]} \otimes_{[1/2]} (a_{p\alpha}^\dagger a_{q\alpha} + a_{p\beta}^\dagger a_{q\beta})^{[0]} \\
&= \frac{1}{\sqrt{2}} \begin{pmatrix} -a_{r\beta} a_{p\alpha}^\dagger a_{q\alpha} - a_{r\beta} a_{p\beta}^\dagger a_{q\beta} \\ a_{r\alpha} a_{p\alpha}^\dagger a_{q\alpha} + a_{r\alpha} a_{p\beta}^\dagger a_{q\beta} \end{pmatrix}^{[1/2]}
\end{aligned}$$

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

$$\begin{aligned}
X^{[0]} &= \left[(a_p^\dagger)^{[1/2]} \otimes_{[1]} (a_q^\dagger)^{[1/2]} \right] \otimes_{[0]} \left[(a_r)^{[1/2]} \otimes_{[1]} (a_s)^{[1/2]} \right] \\
&= \begin{pmatrix} a_{p\alpha}^\dagger a_{q\alpha}^\dagger \\ \frac{1}{\sqrt{2}} (a_{p\alpha}^\dagger a_{q\beta}^\dagger + a_{p\beta}^\dagger a_{q\alpha}^\dagger) \\ a_{p\beta}^\dagger a_{q\beta}^\dagger \end{pmatrix}^{[1]} \otimes_{[0]} \begin{pmatrix} a_{r\beta} a_{s\beta} \\ -\frac{1}{\sqrt{2}} (a_{r\beta} a_{s\alpha} + a_{r\alpha} a_{s\beta}) \\ a_{r\alpha} a_{s\alpha} \end{pmatrix}^{[1]} \\
&= \frac{1}{\sqrt{3}} \left(a_{p\alpha}^\dagger a_{q\alpha}^\dagger a_{r\alpha} a_{s\alpha} + \frac{1}{2} (a_{p\alpha}^\dagger a_{q\beta}^\dagger + a_{p\beta}^\dagger a_{q\alpha}^\dagger) (a_{r\beta} a_{s\alpha} + a_{r\alpha} a_{s\beta}) + a_{p\beta}^\dagger a_{q\beta}^\dagger a_{r\beta} a_{s\beta} \right) \\
Y^{[0]} &= \left[(a_p^\dagger)^{[1/2]} \otimes_{[0]} (a_q^\dagger)^{[1/2]} \right] \otimes_{[0]} \left[(a_r)^{[1/2]} \otimes_{[0]} (a_s)^{[1/2]} \right] \\
&= \frac{1}{\sqrt{2}} \left(a_{p\alpha}^\dagger a_{q\beta}^\dagger - a_{p\beta}^\dagger a_{q\alpha}^\dagger \right)^{[0]} \otimes_{[0]} \frac{1}{\sqrt{2}} \left(-a_{r\beta} a_{s\alpha} + a_{r\alpha} a_{s\beta} \right)^{[0]} \\
&= \frac{1}{2} \left(a_{p\alpha}^\dagger a_{q\beta}^\dagger - a_{p\beta}^\dagger a_{q\alpha}^\dagger \right) \left(-a_{r\beta} a_{s\alpha} + a_{r\alpha} a_{s\beta} \right)
\end{aligned}$$

Using

$$(a+b)(c+d) + (a-b)(-c+d) = (a+b)(2d) - 2b(-c+d) = 2(ad+bc)$$

we have

$$\begin{aligned}
\sqrt{3}X^{[0]} + Y^{[0]} &= a_{p\alpha}^\dagger a_{q\alpha}^\dagger a_{r\alpha} a_{s\alpha} + a_{p\beta}^\dagger a_{q\beta}^\dagger a_{r\beta} a_{s\beta} + a_{p\alpha}^\dagger a_{q\beta}^\dagger a_{r\alpha} a_{s\beta} + a_{p\beta}^\dagger a_{q\alpha}^\dagger a_{r\beta} a_{s\alpha} \\
&= \sum_{\sigma\sigma'} a_{p\sigma}^\dagger a_{q\sigma'}^\dagger a_{r\sigma} a_{s\sigma'}
\end{aligned}$$

Another case

$$\begin{aligned}
Z^{[0]} &= \left[(a_p^\dagger)^{[1/2]} \otimes_{[1]} (a_q)^{[1/2]} \right] \otimes_{[0]} \left[(a_r^\dagger)^{[1/2]} \otimes_{[1]} (a_s)^{[1/2]} \right] \\
&= \begin{pmatrix} -a_{p\alpha}^\dagger a_{q\beta} \\ \frac{1}{\sqrt{2}} (a_{p\alpha}^\dagger a_{q\alpha} - a_{p\beta}^\dagger a_{q\beta}) \\ a_{p\beta}^\dagger a_{q\alpha} \end{pmatrix}^{[1]} \otimes_{[0]} \begin{pmatrix} -a_{r\alpha}^\dagger a_{s\beta} \\ \frac{1}{\sqrt{2}} (a_{r\alpha}^\dagger a_{s\alpha} - a_{r\beta}^\dagger a_{s\beta}) \\ a_{r\beta}^\dagger a_{s\alpha} \end{pmatrix}^{[1]} \\
&= \frac{1}{\sqrt{3}} \left(-a_{p\alpha}^\dagger a_{q\beta} a_{r\beta}^\dagger a_{s\alpha} - \frac{1}{2} (a_{p\alpha}^\dagger a_{q\alpha} - a_{p\beta}^\dagger a_{q\beta}) (a_{r\alpha}^\dagger a_{s\alpha} - a_{r\beta}^\dagger a_{s\beta}) - a_{p\beta}^\dagger a_{q\alpha} a_{r\alpha}^\dagger a_{s\beta} \right) \\
W^{[0]} &= \left[(a_p^\dagger)^{[1/2]} \otimes_{[0]} (a_q)^{[1/2]} \right] \otimes_{[0]} \left[(a_r^\dagger)^{[1/2]} \otimes_{[0]} (a_s)^{[1/2]} \right] \\
&= \frac{1}{\sqrt{2}} \left(a_{p\alpha}^\dagger a_{q\alpha} + a_{p\beta}^\dagger a_{q\beta} \right)^{[0]} \otimes_{[0]} \frac{1}{\sqrt{2}} \left(a_{r\alpha}^\dagger a_{s\alpha} + a_{r\beta}^\dagger a_{s\beta} \right)^{[0]} \\
&= \frac{1}{2} \left(a_{p\alpha}^\dagger a_{q\alpha} + a_{p\beta}^\dagger a_{q\beta} \right) \left(a_{r\alpha}^\dagger a_{s\alpha} + a_{r\beta}^\dagger a_{s\beta} \right)
\end{aligned}$$

Using

$$(a - b)(c - d) + (a + b)(c + d) = (a + b)(2c) - (2b)(c - d) = 2(ac + bd)$$

we have

$$\begin{aligned} -\sqrt{3}Z^{[0]} + W^{[0]} &= a_{p\alpha}^\dagger a_{q\beta} a_{r\beta}^\dagger a_{s\alpha} + a_{p\beta}^\dagger a_{q\alpha} a_{r\alpha}^\dagger a_{s\beta} + a_{p\alpha}^\dagger a_{q\alpha} a_{r\alpha}^\dagger a_{s\alpha} + a_{p\beta}^\dagger a_{q\beta} a_{r\beta}^\dagger a_{s\beta} \\ &= \sum_{\sigma\sigma'} a_{p\sigma}^\dagger a_{q\sigma'} a_{r\sigma'}^\dagger a_{s\sigma} \end{aligned}$$

S Term

From second singlet formula we have

$$\sqrt{2} \sum_{i \in L} (a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (\hat{S}_i^R)^{[\frac{1}{2}]} = \sum_{i \in L} (t_{ij} a_{i\alpha}^\dagger a_{j\alpha} + t_{ij} a_{i\beta}^\dagger a_{j\beta})$$

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

$$\begin{aligned} (\hat{A}_{ik})^{[0/1]} &= (a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0/1]} (a_k^\dagger)^{[\frac{1}{2}]} \\ (\hat{P}_{ik}^R)^{[0/1]} &= - \sum_{jl \in R} v_{ijkl} (a_j)^{[\frac{1}{2}]} \otimes_{[0/1]} (a_l)^{[\frac{1}{2}]} \end{aligned}$$

We have

$$\begin{aligned} &\sum_{ik \in L} \left[\sqrt{3} (\hat{A}_{ik})^{[1]} \otimes_{[0]} (\hat{P}_{ik}^R)^{[1]} + (\hat{A}_{ik})^{[0]} \otimes_{[0]} (\hat{P}_{ik}^R)^{[0]} \right] \\ &= \sum_{ik \in L, jl \in R} v_{ijkl} \left[\sqrt{3} \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[1]} (a_k^\dagger)^{[\frac{1}{2}]} \right] \otimes_{[0]} \left[(a_j)^{[\frac{1}{2}]} \otimes_{[1]} (a_l)^{[\frac{1}{2}]} \right] + \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_k^\dagger)^{[\frac{1}{2}]} \right] \otimes_{[0]} \left[(a_j)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]} \right] \right] \\ &= \sum_{ik \in L, jl \in R} v_{ijkl} \left[\sum_{\sigma\sigma'} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{j\sigma} a_{l\sigma'} \right] = - \sum_{ik \in L, jl \in R, \sigma\sigma'} v_{ijkl} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} \end{aligned}$$

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,

$$2W^{[0]} - (-\sqrt{3}Z^{[0]} + W^{[0]}) = \sqrt{3}Z^{[0]} + W^{[0]}$$

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 $(\hat{B}_{ij})^{[0]} = (\hat{B}'_{ij})^{[0]}$)

$$\begin{aligned} & \sum_{ij \in L} \left[2(\hat{B}_{ij})^{[0]} \otimes_{[0]} (\hat{Q}_{ij}^R)^{[0]} - (\hat{B}'_{ij})^{[0]} \otimes_{[0]} (\hat{Q}_{ij}^R)^{[0]} + \sqrt{3}(\hat{B}'_{ij})^{[1]} \otimes_{[0]} (\hat{Q}_{ij}^R)^{[1]} \right] \\ &= \sum_{ij \in L} \left[(\hat{B}_{ij})^{[0]} \otimes_{[0]} \left((2\hat{Q}_{ij}^R)^{[0]} - (\hat{Q}_{ij}^R)^{[0]} \right) + \sqrt{3}(\hat{B}'_{ij})^{[1]} \otimes_{[0]} (\hat{Q}_{ij}^R)^{[1]} \right] \end{aligned}$$

Note that B, Q do not have $[1]$ form.

Normal/Complementary Partitioning

Note that

$$\sqrt{2} \sum_{i \in L} \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (\hat{S}_i^R)^{[\frac{1}{2}]} + h.c. \right] = \sqrt{2} \sum_{i \in R} \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (\hat{S}_i^L)^{[\frac{1}{2}]} + h.c. \right]$$

Therefore,

$$\begin{aligned} & \sqrt{2} \sum_{i \in L} \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (\hat{S}_i^R)^{[\frac{1}{2}]} + h.c. \right] + 2 \sum_{i \in L} \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (\hat{R}_i^R)^{[\frac{1}{2}]} + h.c. \right] + 2 \sum_{i \in R} \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (\hat{R}_i^L)^{[\frac{1}{2}]} + h.c. \right] \\ &= \frac{\sqrt{2}}{2} \sum_{i \in L} \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (\hat{S}_i^R)^{[\frac{1}{2}]} + h.c. \right] + \frac{\sqrt{2}}{2} \sum_{i \in R} \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (\hat{S}_i^L)^{[\frac{1}{2}]} + h.c. \right] \\ & \quad + 2 \sum_{i \in L} \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (\hat{R}_i^R)^{[\frac{1}{2}]} + h.c. \right] + 2 \sum_{i \in R} \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (\hat{R}_i^L)^{[\frac{1}{2}]} + h.c. \right] \\ &= 2 \sum_{i \in L} \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} \left[(\hat{R}_i^R)^{[\frac{1}{2}]} + \frac{\sqrt{2}}{4} (\hat{S}_i^R)^{[\frac{1}{2}]} \right] + h.c. \right] + 2 \sum_{i \in R} \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} \left[(\hat{R}_i^L)^{[\frac{1}{2}]} + \frac{\sqrt{2}}{4} (\hat{S}_i^L)^{[\frac{1}{2}]} \right] + h.c. \right] \end{aligned}$$

So define

$$(\hat{R}_i^{L/R})^{[\frac{1}{2}]} := \frac{\sqrt{2}}{4} (\hat{S}_i^L)^{[\frac{1}{2}]} + (\hat{R}_i^L)^{[\frac{1}{2}]} = \frac{\sqrt{2}}{4} \sum_{j \in L/R} t_{ij} (a_j)^{[\frac{1}{2}]} + \sum_{jkl \in L/R} v_{ijkl} \left[(a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]} \right] \otimes_{[\frac{1}{2}]} (a_j)^{[\frac{1}{2}]}$$

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

$$\begin{aligned} \left\langle j_1 \left(M - \frac{1}{2} \right) \frac{1}{2} \frac{1}{2} \left| \left(j_1 \pm \frac{1}{2} \right) M \right\rangle &= \pm \sqrt{\frac{1}{2} \left(1 \pm \frac{M}{j_1 + \frac{1}{2}} \right)} \\ \left\langle j_1 \left(M + \frac{1}{2} \right) \frac{1}{2} \left(-\frac{1}{2} \right) \left| \left(j_1 \pm \frac{1}{2} \right) M \right\rangle &= \sqrt{\frac{1}{2} \left(1 \mp \frac{M}{j_1 + \frac{1}{2}} \right)} \end{aligned}$$

Let $j_1 = \frac{1}{2}$ we have

$$\begin{aligned} \left\langle \frac{1}{2} \left(-\frac{1}{2} \right) \frac{1}{2} \frac{1}{2} \left| \left(\frac{1}{2} \pm \frac{1}{2} \right) 0 \right\rangle &= \pm \sqrt{\frac{1}{2}} \\ \left\langle \frac{1}{2} \frac{1}{2} \frac{1}{2} \left(-\frac{1}{2} \right) \left| \left(\frac{1}{2} \pm \frac{1}{2} \right) 0 \right\rangle &= \sqrt{\frac{1}{2}} \end{aligned}$$

The exchange factor formula is

$$\begin{aligned} \left(\hat{X}_1^{[S_1]} \otimes_{[S]} \hat{X}_2^{[S_2]} \right)^{[S_z]} &= \sum_{S_{1z}, S_{2z}} \hat{X}_1^{[S_1][S_{1z}]} \hat{X}_2^{[S_2][S_{2z}]} \langle S S_z | S_1 S_{1z}, S_2 S_{2z} \rangle \\ &= P_{\text{fermi}}^{\text{exchange}}(N_1, N_2) \sum_{S_{1z}, S_{2z}} \hat{X}_2^{[S_2][S_{2z}]} \hat{X}_1^{[S_1][S_{1z}]} \langle S S_z | S_1 S_{1z}, S_2 S_{2z} \rangle \\ &= P_{\text{fermi}}^{\text{exchange}}(N_1, N_2) \frac{\langle S S_z | S_1 S_{1z}, S_2 S_{2z} \rangle}{\langle S S_z | S_2 S_{2z}, S_1 S_{1z} \rangle} \left(\hat{X}_2^{[S_2]} \otimes_{[S]} \hat{X}_1^{[S_1]} \right)^{[S_z]} \\ \hat{X}_1^{[S_1]} \otimes_{[S]} \hat{X}_2^{[S_2]} &= P_{\text{fermi}}^{\text{exchange}}(N_1, N_2) P_{\text{SU}(2)}^{\text{exchange}}(S_1, S_2, S) \hat{X}_2^{[S_2]} \otimes_{[S]} \hat{X}_1^{[S_1]} \end{aligned}$$

For $[1/2] \otimes_{[0]} [1/2]$, this is

$$P_{\text{exchange}}\left(\frac{1}{2}, \frac{1}{2}, 0\right) = (-1) \frac{\langle \frac{1}{2} \frac{1}{2} \frac{1}{2} \left(-\frac{1}{2} \right) | 0 0 \rangle}{\langle \frac{1}{2} \left(-\frac{1}{2} \right) \frac{1}{2} \frac{1}{2} | 0 0 \rangle} = (-1) \frac{\sqrt{\frac{1}{2}}}{-\sqrt{\frac{1}{2}}} = 1$$

For $[1/2] \otimes_{[1]} [1/2]$, this is

$$P_{\text{exchange}}\left(\frac{1}{2}, \frac{1}{2}, 1\right) = (-1) \frac{\langle \frac{1}{2} \frac{1}{2} \frac{1}{2} \left(-\frac{1}{2} \right) | 1 0 \rangle}{\langle \frac{1}{2} \left(-\frac{1}{2} \right) \frac{1}{2} \frac{1}{2} | 1 0 \rangle} = (-1) \frac{\sqrt{\frac{1}{2}}}{\sqrt{\frac{1}{2}}} = -1$$

From CG factors

$$\langle 1 m_1 1 (-m_1) | 0 0 \rangle = \frac{(-1)^{1-m_1}}{\sqrt{3}}$$

we have

$$P_{\text{exchange}}(1, 1, 0) = (+1) \frac{\langle 1 1 1 -1 | 0 0 \rangle}{\langle 1 -1 1 1 | 0 0 \rangle} = (+1) \frac{\frac{(-1)^0}{\sqrt{3}}}{\frac{(-1)^2}{\sqrt{3}}} = 1$$

we have

$$\begin{aligned}
(\hat{H})^{[0],NC} &= (\hat{H}^L)^{[0]} \otimes_{[0]} (\hat{1}^R)^{[0]} + (\hat{1}^L)^{[0]} \otimes_{[0]} (\hat{H}^R)^{[0]} \\
&\quad + 2 \sum_{i \in L} \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (\hat{R}_i'^R)^{[\frac{1}{2}]} + (a_i)^{[\frac{1}{2}]} \otimes_{[0]} (\hat{R}_i'^{R\dagger})^{[\frac{1}{2}]} \right] + 2 \sum_{i \in R} \left[(\hat{R}_i'^{L\dagger})^{[\frac{1}{2}]} \otimes_{[0]} (a_i)^{[\frac{1}{2}]} + (\hat{R}_i'^L)^{[\frac{1}{2}]} \otimes_{[0]} (a_i^\dagger)^{[\frac{1}{2}]} \right] \\
&\quad - \frac{1}{2} \sum_{ik \in L} \left[(\hat{A}_{ik})^{[0]} \otimes_{[0]} (\hat{P}_{ik}^R)^{[0]} + \sqrt{3}(\hat{A}_{ik})^{[1]} \otimes_{[0]} (\hat{P}_{ik}^R)^{[1]} + (\hat{A}_{ik}^\dagger)^{[0]} \otimes_{[0]} (\hat{P}_{ik}^{R\dagger})^{[0]} + \sqrt{3}(\hat{A}_{ik}^\dagger)^{[1]} \otimes_{[0]} (\hat{P}_{ik}^{R\dagger})^{[1]} \right] \\
&\quad + \sum_{ij \in L} \left[(\hat{B}_{ij})^{[0]} \otimes_{[0]} (\hat{Q}_{ij}''^R)^{[0]} + \sqrt{3}(\hat{B}_{ij}')^{[1]} \otimes_{[0]} (\hat{Q}_{ij}^R)^{[1]} \right]
\end{aligned}$$

With this normal/complementary partitioning, the operators required in left block are

$$\{(\hat{H}^L)^{[0]}, (\hat{1}^L)^{[0]}, (a_i^\dagger)^{[\frac{1}{2}]}, (a_i)^{[\frac{1}{2}]}, (\hat{R}_k'^{L\dagger})^{[\frac{1}{2}]}, (\hat{R}_k'^L)^{[\frac{1}{2}]}, (\hat{A}_{ij})^{[0]}, (\hat{A}_{ij})^{[1]}, (\hat{A}_{ij}^\dagger)^{[0]}, (\hat{A}_{ij}^\dagger)^{[1]}, (\hat{B}_{ij})^{[0]}, (\hat{B}_{ij}')^{[1]}\} \quad (i, j \in L,$$

The operators required in right block are

$$\{(\hat{1}^R)^{[0]}, (\hat{H}^R)^{[0]}, (\hat{R}_i'^R)^{[\frac{1}{2}]}, (\hat{R}_i'^{R\dagger})^{[\frac{1}{2}]}, (a_k)^{[\frac{1}{2}]}, (a_k^\dagger)^{[\frac{1}{2}]}, (\hat{P}_{ij}^R)^{[0]}, (\hat{P}_{ij}^R)^{[1]}, (\hat{P}_{ij}^{R\dagger})^{[0]}, (\hat{P}_{ij}^{R\dagger})^{[1]}, (\hat{Q}_{ij}''^R)^{[0]}, (\hat{Q}_{ij}^R)^{[1]}\} \quad (i, j \in R,$$

Assuming that there are K sites in total, and K_L/K_R sites in left/right block (optimally, $K_L \leq K_R$), 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.

$$\begin{aligned}
(\hat{H})^{[0],CN} &= (\hat{H}^L)^{[0]} \otimes_{[0]} (\hat{1}^R)^{[0]} + (\hat{1}^L)^{[0]} \otimes_{[0]} (\hat{H}^R)^{[0]} \\
&\quad + 2 \sum_{i \in L} \left[(a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (\hat{R}_i'^R)^{[\frac{1}{2}]} + (a_i)^{[\frac{1}{2}]} \otimes_{[0]} (\hat{R}_i'^{R\dagger})^{[\frac{1}{2}]} \right] + 2 \sum_{i \in R} \left[(\hat{R}_i'^{L\dagger})^{[\frac{1}{2}]} \otimes_{[0]} (a_i)^{[\frac{1}{2}]} + (\hat{R}_i'^L)^{[\frac{1}{2}]} \otimes_{[0]} (a_i^\dagger)^{[\frac{1}{2}]} \right] \\
&\quad - \frac{1}{2} \sum_{jl \in R} \left[(\hat{P}_{jl}^L)^{[0]} \otimes_{[0]} (\hat{A}_{jl})^{[0]} + \sqrt{3}(\hat{P}_{jl}^L)^{[1]} \otimes_{[0]} (\hat{A}_{jl})^{[1]} + (\hat{P}_{jl}^{L\dagger})^{[0]} \otimes_{[0]} (\hat{A}_{jl}^\dagger)^{[0]} + \sqrt{3}(\hat{P}_{jl}^{L\dagger})^{[1]} \otimes_{[0]} (\hat{A}_{jl}^\dagger)^{[1]} \right] \\
&\quad + \sum_{kl \in R} \left[(\hat{Q}_{kl}''^L)^{[0]} \otimes_{[0]} (\hat{B}_{kl})^{[0]} + \sqrt{3}(\hat{Q}_{kl}^L)^{[1]} \otimes_{[0]} (\hat{B}_{kl}')^{[1]} \right]
\end{aligned}$$

Now the operators required in left block are

$$\{(\hat{H}^L)^{[0]}, (\hat{1}^L)^{[0]}, (a_i^\dagger)^{[\frac{1}{2}]}, (a_i)^{[\frac{1}{2}]}, (\hat{R}_k'^{L\dagger})^{[\frac{1}{2}]}, (\hat{R}_k'^L)^{[\frac{1}{2}]}, (\hat{P}_{kl}^L)^{[0]}, (\hat{P}_{kl}^L)^{[1]}, (\hat{P}_{kl}^{L\dagger})^{[0]}, (\hat{P}_{kl}^{L\dagger})^{[1]}, (\hat{Q}_{kl}''^L)^{[0]}, (\hat{Q}_{kl}^L)^{[1]}\} \quad (k, l \in L,$$

The operators required in right block are

$$\{(\hat{1}^R)^{[0]}, (\hat{H}^R)^{[0]}, (\hat{R}_i'^R)^{[\frac{1}{2}]}, (\hat{R}_i'^{R\dagger})^{[\frac{1}{2}]}, (a_k)^{[\frac{1}{2}]}, (a_k^\dagger)^{[\frac{1}{2}]}, (\hat{A}_{kl})^{[0]}, (\hat{A}_{kl})^{[1]}, (\hat{A}_{kl}^\dagger)^{[0]}, (\hat{A}_{kl}^\dagger)^{[1]}, (\hat{B}_{kl})^{[0]}, (\hat{B}_{kl}')^{[1]}\} \quad (k, l \in R,$$

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

$$\begin{aligned} (\hat{R}_i^{L/R})^{[1/2]} &= \sum_{jkl \in L/R} v_{ijkl} \left[(a_k^\dagger)^{[1/2]} \otimes_{[0]} (a_l)^{[1/2]} \right] \otimes_{[1/2]} (a_j)^{[1/2]} \\ &= \frac{1}{\sqrt{2}} \sum_{jkl \in L/R} v_{ijkl} \left(a_{k\alpha}^\dagger a_{l\alpha} + a_{k\beta}^\dagger a_{l\beta} \right)^{[0]} \otimes_{[1/2]} (a_j)^{[1/2]} \\ &= \frac{1}{\sqrt{2}} \sum_{jkl \in L/R} v_{ijkl} \left(\begin{array}{c} -a_{k\alpha}^\dagger a_{l\alpha} a_{j\beta} - a_{k\beta}^\dagger a_{l\beta} a_{j\beta} \\ a_{k\alpha}^\dagger a_{l\alpha} a_{j\alpha} + a_{k\beta}^\dagger a_{l\beta} a_{j\alpha} \end{array} \right)^{[1/2]} \end{aligned}$$

From the formula $\sqrt{3}U^{[1/2]} - V^{[1/2]}$ we have

$$(\hat{R}_i^{L/R})^{[1/2]} = \frac{\sqrt{3}}{2} \sum_{jkl \in L/R} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} \left[(a_l)^{[1/2]} \otimes_{[1]} (a_j)^{[1/2]} \right] - \frac{1}{2} \sum_{jkl \in L/R} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} \left[(a_l)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right]$$

From the formula $\sqrt{3}S^{[1/2]} - T^{[1/2]}$ we have (for $k \neq l$)

$$(\hat{R}_i^{L/R})^{[1/2]} = \frac{\sqrt{3}}{2} \sum_{jkl \in L/R} v_{ijkl} (a_l)^{[1/2]} \otimes_{[1/2]} \left[(a_k^\dagger)^{[1/2]} \otimes_{[1]} (a_j)^{[1/2]} \right] - \frac{1}{2} \sum_{jkl \in L/R} v_{ijkl} (a_l)^{[1/2]} \otimes_{[1/2]} \left[(a_k^\dagger)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right]$$

We have

$$\begin{aligned}
(\hat{R}_i'^{L*})^{[1/2]} &= (\hat{R}_i^L)^{[1/2]} \otimes_{[1/2]} (\hat{1}^*)^{[0]} + (\hat{1}^L)^{[0]} \otimes_{[1/2]} (\hat{R}_i'^*)^{[1/2]} \\
&+ \sum_{j \in L} \left[\sum_{kl \in *} v_{ijkl} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]} \right] \otimes_{[\frac{1}{2}]} (a_j)^{[\frac{1}{2}]} + \sum_{j \in *} \left[\sum_{kl \in L} v_{ijkl} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]} \right] \otimes_{[\frac{1}{2}]} (a_j)^{[\frac{1}{2}]} \\
&- \frac{1}{2} \sum_{k \in L} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl \in *} v_{ijkl} (a_l)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] + \frac{\sqrt{3}}{2} \sum_{k \in L} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl \in *} v_{ijkl} (a_l)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] \\
&- \frac{1}{2} \sum_{k \in *} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl \in L} v_{ijkl} (a_l)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] + \frac{\sqrt{3}}{2} \sum_{k \in *} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl \in L} v_{ijkl} (a_l)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] \\
&- \frac{1}{2} \sum_{l \in L} (a_l)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk \in *} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] + \frac{\sqrt{3}}{2} \sum_{l \in L} (a_l)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk \in *} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] \\
&- \frac{1}{2} \sum_{l \in *} (a_l)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk \in L} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] + \frac{\sqrt{3}}{2} \sum_{l \in *} (a_l)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk \in L} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] \\
&= (\hat{R}_i^L)^{[1/2]} \otimes_{[1/2]} (\hat{1}^*)^{[0]} + (\hat{1}^L)^{[0]} \otimes_{[1/2]} (\hat{R}_i'^*)^{[1/2]} \\
&+ \sum_{j \in L} (a_j)^{[\frac{1}{2}]} \otimes_{[\frac{1}{2}]} \left[\sum_{kl \in *} v_{ijkl} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]} \right] + \sum_{j \in *} \left[\sum_{kl \in L} v_{ijkl} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]} \right] \otimes_{[\frac{1}{2}]} (a_j)^{[\frac{1}{2}]} \\
&- \frac{1}{2} \sum_{k \in L} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl \in *} v_{ijkl} (a_l)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] + \frac{\sqrt{3}}{2} \sum_{k \in L} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl \in *} v_{ijkl} (a_l)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] \\
&- \frac{1}{2} \sum_{k \in *} \left[\sum_{jl \in L} v_{ijkl} (a_l)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] \otimes_{[1/2]} (a_k^\dagger)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{k \in *} \left[\sum_{jl \in L} v_{ijkl} (a_l)^{[1/2]} \otimes_{[1]} (a_j)^{[1/2]} \right] \otimes_{[1/2]} (a_k^\dagger)^{[1/2]} \\
&- \frac{1}{2} \sum_{l \in L} (a_l)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk \in *} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] + \frac{\sqrt{3}}{2} \sum_{l \in L} (a_l)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk \in *} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] \\
&- \frac{1}{2} \sum_{l \in *} \left[\sum_{jk \in L} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] \otimes_{[1/2]} (a_l)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{l \in *} \left[\sum_{jk \in L} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[1]} (a_j)^{[1/2]} \right] \otimes_{[1/2]} (a_l)^{[1/2]}
\end{aligned}$$

After reordering of terms

$$\begin{aligned}
(\hat{R}_i'^{L*})^{[1/2]} &= (\hat{R}_i'^L)^{[1/2]} \otimes_{[1/2]} (\hat{1}^*)^{[0]} + (\hat{1}^L)^{[0]} \otimes_{[1/2]} (\hat{R}_i'^*)^{[1/2]} \\
&\quad - \frac{1}{2} \sum_{k \in L} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl \in *} v_{ijkl} (a_l)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] + \frac{\sqrt{3}}{2} \sum_{k \in L} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl \in *} v_{ijkl} (a_l)^{[1/2]} \otimes_{[1]} (a_j)^{[1/2]} \right] \\
&\quad + \sum_{j \in L} (a_j)^{[\frac{1}{2}]} \otimes_{[\frac{1}{2}]} \left[\sum_{kl \in *} v_{ijkl} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]} \right] \\
&\quad - \frac{1}{2} \sum_{l \in L} (a_l)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk \in *} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] + \frac{\sqrt{3}}{2} \sum_{l \in L} (a_l)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk \in *} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[1]} (a_j)^{[1/2]} \right] \\
&\quad - \frac{1}{2} \sum_{k \in *} \left[\sum_{jl \in L} v_{ijkl} (a_l)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] \otimes_{[1/2]} (a_k^\dagger)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{k \in *} \left[\sum_{jl \in L} v_{ijkl} (a_l)^{[1/2]} \otimes_{[1]} (a_j)^{[1/2]} \right] \otimes_{[1/2]} (a_k^\dagger)^{[1/2]} \\
&\quad + \sum_{j \in *} \left[\sum_{kl \in L} v_{ijkl} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]} \right] \otimes_{[\frac{1}{2}]} (a_j)^{[\frac{1}{2}]} \\
&\quad - \frac{1}{2} \sum_{l \in *} \left[\sum_{jk \in L} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] \otimes_{[1/2]} (a_l)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{l \in *} \left[\sum_{jk \in L} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[1]} (a_j)^{[1/2]} \right] \otimes_{[1/2]} (a_l)^{[1/2]} \\
&= (\hat{R}_i'^L)^{[1/2]} \otimes_{[1/2]} (\hat{1}^*)^{[0]} + (\hat{1}^L)^{[0]} \otimes_{[1/2]} (\hat{R}_i'^*)^{[1/2]} \\
&\quad - \frac{1}{2} \sum_{k \in L} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl \in *} v_{ijkl} (a_l)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] + \frac{\sqrt{3}}{2} \sum_{k \in L} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jl \in *} v_{ijkl} (a_l)^{[1/2]} \otimes_{[1]} (a_j)^{[1/2]} \right] \\
&\quad + \frac{1}{2} \sum_{j \in L} (a_j)^{[1/2]} \otimes_{[1/2]} \left[\sum_{kl \in *} (2v_{ijkl} - v_{ilkj}) (a_k^\dagger)^{[1/2]} \otimes_{[0]} (a_l)^{[1/2]} \right] + \frac{\sqrt{3}}{2} \sum_{l \in L} (a_l)^{[1/2]} \otimes_{[1/2]} \left[\sum_{jk \in *} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[1]} (a_j)^{[1/2]} \right] \\
&\quad - \frac{1}{2} \sum_{k \in *} \left[\sum_{jl \in L} v_{ijkl} (a_l)^{[1/2]} \otimes_{[0]} (a_j)^{[1/2]} \right] \otimes_{[1/2]} (a_k^\dagger)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{k \in *} \left[\sum_{jl \in L} v_{ijkl} (a_l)^{[1/2]} \otimes_{[1]} (a_j)^{[1/2]} \right] \otimes_{[1/2]} (a_k^\dagger)^{[1/2]} \\
&\quad + \frac{1}{2} \sum_{j \in *} \left[\sum_{kl \in L} (2v_{ijkl} - v_{ilkj}) (a_k^\dagger)^{[1/2]} \otimes_{[0]} (a_l)^{[1/2]} \right] \otimes_{[1/2]} (a_j)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{l \in *} \left[\sum_{jk \in L} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[1]} (a_j)^{[1/2]} \right] \otimes_{[1/2]} (a_l)^{[1/2]}
\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)

$$\begin{aligned}
(\hat{A}_{ik})^{[0/1]} &= (a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0/1]} (a_k^\dagger)^{[\frac{1}{2}]} \\
(\hat{A}_{ik}^\dagger)^{[0]} &= (a_i)^{[\frac{1}{2}]} \otimes_{[0]} (a_k)^{[\frac{1}{2}]} = (a_k)^{[\frac{1}{2}]} \otimes_{[0]} (a_i)^{[\frac{1}{2}]} \\
(\hat{A}_{ik}^\dagger)^{[1]} &= - (a_i)^{[\frac{1}{2}]} \otimes_{[1]} (a_k)^{[\frac{1}{2}]} = (a_k)^{[\frac{1}{2}]} \otimes_{[1]} (a_i)^{[\frac{1}{2}]} \\
(\hat{P}_{ik}^R)^{[0/1]} &= \sum_{jl \in R} v_{ijkl} (a_l)^{[\frac{1}{2}]} \otimes_{[0/1]} (a_j)^{[\frac{1}{2}]} \\
(\hat{B}_{ij})^{[0]} &= (a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_j)^{[\frac{1}{2}]} \\
(\hat{B}_{ij}')^{[1]} &= (a_i^\dagger)^{[\frac{1}{2}]} \otimes_{[1]} (a_j)^{[\frac{1}{2}]} \\
(\hat{Q}_{ij}^R)^{[1]} &= \sum_{kl \in R} v_{ilkj} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[1]} (a_l)^{[\frac{1}{2}]} \\
(\hat{Q}_{ij}^R)^{[0]} &= \sum_{kl \in R} (2v_{ijkl} - v_{ilkj}) (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]}
\end{aligned}$$

we have

$$\begin{aligned}
(\hat{R}_i^{L*,NC})^{[1/2]} &= (\hat{R}_i^L)^{[1/2]} \otimes_{[1/2]} (\hat{1}^*)^{[0]} + (\hat{1}^L)^{[0]} \otimes_{[1/2]} (\hat{R}_i^*)^{[1/2]} \\
&\quad - \frac{1}{2} \sum_{k \in L} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} (\hat{P}_{ik}^*)^{[0]} + \frac{\sqrt{3}}{2} \sum_{k \in L} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} (\hat{P}_{ik}^*)^{[1]} \\
&\quad + \frac{1}{2} \sum_{j \in L} (a_j)^{[1/2]} \otimes_{[1/2]} (\hat{Q}_{ij}^{\prime*})^{[0]} + \frac{\sqrt{3}}{2} \sum_{l \in L} (a_l)^{[1/2]} \otimes_{[1/2]} (\hat{Q}_{il}^{\prime*})^{[1]} \\
&\quad - \frac{1}{2} \sum_{k \in *, j \in L} v_{ijkl} (\hat{A}_{jl}^\dagger)^{[0]} \otimes_{[1/2]} (a_k^\dagger)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{k \in *, j \in L} v_{ijkl} (\hat{A}_{jl}^\dagger)^{[1]} \otimes_{[1/2]} (a_k^\dagger)^{[1/2]} \\
&\quad + \frac{1}{2} \sum_{j \in *, kl \in L} (2v_{ijkl} - v_{ilkj}) (\hat{B}_{kl})^{[0]} \otimes_{[1/2]} (a_j)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{l \in *, jk \in L} v_{ijkl} (\hat{B}_{kj}')^{[1]} \otimes_{[1/2]} (a_l)^{[1/2]} \\
(\hat{R}_i^{L*,CN})^{[1/2]} &= (\hat{R}_i^L)^{[1/2]} \otimes_{[1/2]} (\hat{1}^*)^{[0]} + (\hat{1}^L)^{[0]} \otimes_{[1/2]} (\hat{R}_i^*)^{[1/2]} \\
&\quad - \frac{1}{2} \sum_{k \in L, jl \in *} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} (\hat{A}_{jl}^\dagger)^{[0]} + \frac{\sqrt{3}}{2} \sum_{k \in L, jl \in *} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} (\hat{A}_{jl}^\dagger)^{[1]} \\
&\quad + \frac{1}{2} \sum_{j \in L, kl \in *} (2v_{ijkl} - v_{ilkj}) (a_j)^{[1/2]} \otimes_{[1/2]} (\hat{B}_{kl})^{[0]} + \frac{\sqrt{3}}{2} \sum_{l \in L, jk \in *} v_{ijkl} (a_l)^{[1/2]} \otimes_{[1/2]} (\hat{B}_{kj}')^{[1]} \\
&\quad - \frac{1}{2} \sum_{k \in *} (\hat{P}_{ik}^L)^{[0]} \otimes_{[1/2]} (a_k^\dagger)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{k \in *} (\hat{P}_{ik}^L)^{[1]} \otimes_{[1/2]} (a_k^\dagger)^{[1/2]} \\
&\quad + \frac{1}{2} \sum_{j \in *} (\hat{Q}_{ij}^{\prime L})^{[0]} \otimes_{[1/2]} (a_j)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{l \in *} (\hat{Q}_{il}^{\prime L})^{[1]} \otimes_{[1/2]} (a_l)^{[1/2]}
\end{aligned}$$

To generate symmetrized P , we need to change the A line to the following

$$-\frac{1}{4} \sum_{k \in *, jl \in L} (v_{ijkl} + v_{ilkj}) (\hat{A}_{jl}^\dagger)^{[0]} \otimes_{[1/2]} (a_k^\dagger)^{[1/2]} - \frac{\sqrt{3}}{4} \sum_{k \in *, jl \in L} (v_{ijkl} - v_{ilkj}) (\hat{A}_{jl}^\dagger)^{[1]} \otimes_{[1/2]} (a_k^\dagger)^{[1/2]}$$

Similarly,

$$\begin{aligned}
(\hat{R}_i^{R*,NC})^{[1/2]} &= (\hat{R}_i^{t*})^{[1/2]} \otimes_{[1/2]} (\hat{1}^R)^{[0]} + (\hat{1}^*)^{[0]} \otimes_{[1/2]} (\hat{R}_i^R)^{[1/2]} \\
&\quad - \frac{1}{2} \sum_{k \in *} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} (\hat{P}_{ik}^R)^{[0]} + \frac{\sqrt{3}}{2} \sum_{k \in *} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} (\hat{P}_{ik}^R)^{[1]} \\
&\quad + \frac{1}{2} \sum_{j \in *} (a_j)^{[1/2]} \otimes_{[1/2]} (\hat{Q}_{ij}^{R*})^{[0]} + \frac{\sqrt{3}}{2} \sum_{l \in *} (a_l)^{[1/2]} \otimes_{[1/2]} (\hat{Q}_{il}^R)^{[1]} \\
&\quad - \frac{1}{2} \sum_{k \in R, j \in *} v_{ijkl} (\hat{A}_{jl}^\dagger)^{[0]} \otimes_{[1/2]} (a_k^\dagger)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{k \in R, j \in *} v_{ijkl} (\hat{A}_{jl}^\dagger)^{[1]} \otimes_{[1/2]} (a_k^\dagger)^{[1/2]} \\
&\quad + \frac{1}{2} \sum_{j \in R, k \in *} (2v_{ijkl} - v_{ilkj}) (\hat{B}_{kl})^{[0]} \otimes_{[1/2]} (a_j)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{l \in R, j \in *} v_{ijkl} (\hat{B}_{kj}')^{[1]} \otimes_{[1/2]} (a_l)^{[1/2]} \\
(\hat{R}_i^{R*,CN})^{[1/2]} &= (\hat{R}_i^{t*})^{[1/2]} \otimes_{[1/2]} (\hat{1}^R)^{[0]} + (\hat{1}^*)^{[0]} \otimes_{[1/2]} (\hat{R}_i^R)^{[1/2]} \\
&\quad - \frac{1}{2} \sum_{k \in *, j \in R} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} (\hat{A}_{jl}^\dagger)^{[0]} + \frac{\sqrt{3}}{2} \sum_{k \in *, j \in R} v_{ijkl} (a_k^\dagger)^{[1/2]} \otimes_{[1/2]} (\hat{A}_{jl}^\dagger)^{[1]} \\
&\quad + \frac{1}{2} \sum_{j \in *, k \in R} (2v_{ijkl} - v_{ilkj}) (a_j)^{[1/2]} \otimes_{[1/2]} (\hat{B}_{kl})^{[0]} + \frac{\sqrt{3}}{2} \sum_{l \in *, j \in R} v_{ijkl} (a_l)^{[1/2]} \otimes_{[1/2]} (\hat{B}_{kj}')^{[1]} \\
&\quad - \frac{1}{2} \sum_{k \in R} (\hat{P}_{ik}^*)^{[0]} \otimes_{[1/2]} (a_k^\dagger)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{k \in R} (\hat{P}_{ik}^*)^{[1]} \otimes_{[1/2]} (a_k^\dagger)^{[1/2]} \\
&\quad + \frac{1}{2} \sum_{j \in R} (\hat{Q}_{ij}^{R*})^{[0]} \otimes_{[1/2]} (a_j)^{[1/2]} - \frac{\sqrt{3}}{2} \sum_{l \in R} (\hat{Q}_{il}^R)^{[1]} \otimes_{[1/2]} (a_l)^{[1/2]}
\end{aligned}$$

Number of terms

$$N_{R',NC} = (2 + 4K_L + 4K_L^2)K_R + (2 + 4 + 4K_R)K_L = 4K_L^2K_R + 8K_LK_R + 2K + 4K_L$$

$$N_{R',CN} = (2 + 4K_L + 4)K_R + (2 + 4K_R^2 + 4K_R)K_L = 4K_R^2K_L + 8K_RK_L + 2K + 4K_R$$

Blocking of other complementary operators is straightforward

$$\begin{aligned}
(\hat{P}_{ik}^{L*,CN})^{[0/1]} &= (\hat{P}_{ik}^L)^{[0/1]} \otimes_{[0/1]} (\hat{1}^*)^{[0]} + (\hat{1}^L)^{[0]} \otimes_{[0/1]} (\hat{P}_{ik}^*)^{[0/1]} + \sum_{j \in L, l \in *} v_{ijkl} (a_l)^{[\frac{1}{2}]} \otimes_{[0/1]} (a_j)^{[\frac{1}{2}]} + \sum_{j \in *, l \in L} v_{ijkl} (a_l)^{[\frac{1}{2}]} \otimes_{[0/1]} (a_j)^{[\frac{1}{2}]} \\
&= (\hat{P}_{ik}^L)^{[0/1]} \otimes_{[0/1]} (\hat{1}^*)^{[0]} + (\hat{1}^L)^{[0]} \otimes_{[0/1]} (\hat{P}_{ik}^*)^{[0/1]} \pm \sum_{j \in L, l \in *} v_{ijkl} (a_j)^{[\frac{1}{2}]} \otimes_{[0/1]} (a_l)^{[\frac{1}{2}]} + \sum_{j \in *, l \in L} v_{ijkl} (a_l)^{[\frac{1}{2}]} \otimes_{[0/1]} (a_j)^{[\frac{1}{2}]} \\
(\hat{P}_{ik}^{R*,NC})^{[0/1]} &= (\hat{P}_{ik}^*)^{[0/1]} \otimes_{[0/1]} (\hat{1}^R)^{[0]} + (\hat{1}^*)^{[0]} \otimes_{[0/1]} (\hat{P}_{ik}^R)^{[0/1]} \pm \sum_{j \in *, l \in R} v_{ijkl} (a_j)^{[\frac{1}{2}]} \otimes_{[0/1]} (a_l)^{[\frac{1}{2}]} + \sum_{j \in R, l \in *} v_{ijkl} (a_l)^{[\frac{1}{2}]} \otimes_{[0/1]} (a_j)^{[\frac{1}{2}]}
\end{aligned}$$

and

$$\begin{aligned}
(\hat{Q}_{ij}^{R*,CN})^{[0]} &= (\hat{Q}_{ij}^{R*})^{[0]} \otimes_{[0]} (\hat{1}^*)^{[0]} + (\hat{1}^R)^{[0]} \otimes_{[0]} (\hat{Q}_{ij}^{R*})^{[0]} + \sum_{k \in L, l \in *} (2v_{ijkl} - v_{ilkj}) (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]} + \sum_{k \in *, l \in L} (2v_{ijkl} - v_{ilkj}) (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]} \\
&= (\hat{Q}_{ij}^{R*})^{[0]} \otimes_{[0]} (\hat{1}^*)^{[0]} + (\hat{1}^R)^{[0]} \otimes_{[0]} (\hat{Q}_{ij}^{R*})^{[0]} + \sum_{k \in L, l \in *} (2v_{ijkl} - v_{ilkj}) (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]} + \sum_{k \in *, l \in L} (2v_{ijkl} - v_{ilkj}) (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]} \\
(\hat{Q}_{ij}^{R*,NC})^{[0]} &= (\hat{Q}_{ij}^{R*})^{[0]} \otimes_{[0]} (\hat{1}^R)^{[0]} + (\hat{1}^*)^{[0]} \otimes_{[0]} (\hat{Q}_{ij}^{R*})^{[0]} + \sum_{k \in *, l \in R} (2v_{ijkl} - v_{ilkj}) (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]} + \sum_{k \in R, l \in *} (2v_{ijkl} - v_{ilkj}) (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]}
\end{aligned}$$

and

$$\begin{aligned}
(\hat{Q}_{ij}^{\prime L*,CN})^{[1]} &= (\hat{Q}_{ij}^{\prime L})^{[1]} \otimes_{[1]} (\hat{1}^*)^{[0]} + (\hat{1}^L)^{[0]} \otimes_{[1]} (\hat{Q}_{ij}^{\prime*})^{[1]} + \sum_{k \in L, l \in *} v_{ilkj} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[1]} (a_l)^{[\frac{1}{2}]} + \sum_{k \in *, l \in L} v_{ilkj} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[1]} (a_l)^{[\frac{1}{2}]} \\
&= (\hat{Q}_{ij}^{\prime L})^{[1]} \otimes_{[1]} (\hat{1}^*)^{[0]} + (\hat{1}^L)^{[0]} \otimes_{[1]} (\hat{Q}_{ij}^{\prime*})^{[1]} + \sum_{k \in L, l \in *} v_{ilkj} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[1]} (a_l)^{[\frac{1}{2}]} - \sum_{k \in *, l \in L} v_{ilkj} (a_l)^{[\frac{1}{2}]} \otimes_{[1]} (a_k^\dagger)^{[\frac{1}{2}]} \\
(\hat{Q}_{ij}^{\prime R*,CN})^{[1]} &= (\hat{Q}_{ij}^{\prime*})^{[1]} \otimes_{[1]} (\hat{1}^R)^{[0]} + (\hat{1}^*)^{[0]} \otimes_{[1]} (\hat{Q}_{ij}^{\prime R})^{[1]} + \sum_{k \in *, l \in R} v_{ilkj} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[1]} (a_l)^{[\frac{1}{2}]} - \sum_{k \in R, l \in *} v_{ilkj} (a_l)^{[\frac{1}{2}]} \otimes_{[1]} (a_k^\dagger)^{[\frac{1}{2}]}
\end{aligned}$$

Middle-Site Transformation

$$\begin{aligned}
(\hat{P}_{ik}^{L,NC \rightarrow CN})^{[0/1]} &= \sum_{jl \in L} v_{ijkl} (a_l)^{[\frac{1}{2}]} \otimes_{[0/1]} (a_j)^{[\frac{1}{2}]} = \sum_{jl \in L} v_{ijkl} (\hat{A}_{jl}^\dagger)^{[0/1]} \\
(\hat{Q}_{ij}^{\prime L,NC \rightarrow CN})^{[0]} &= \sum_{kl \in R} (2v_{ijkl} - v_{ilkj}) (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[0]} (a_l)^{[\frac{1}{2}]} = \sum_{kl \in R} (2v_{ijkl} - v_{ilkj}) (\hat{B}_{kl})^{[0]} \\
(\hat{Q}_{ij}^{\prime L,NC \rightarrow CN})^{[1]} &= \sum_{kl \in R} v_{ilkj} (a_k^\dagger)^{[\frac{1}{2}]} \otimes_{[1]} (a_l)^{[\frac{1}{2}]} = \sum_{kl \in R} v_{ilkj} (\hat{B}_{kl}')^{[1]}
\end{aligned}$$

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

$$\begin{aligned}
t_{ij,\sigma} &= t_{(ij),\sigma} = \int d\mathbf{x} \phi_{i\sigma}^*(\mathbf{x}) \left(-\frac{1}{2} \nabla^2 - \sum_a \frac{Z_a}{r_a} \right) \phi_{j\sigma}(\mathbf{x}) \\
v_{ijkl,\sigma\sigma'} &= v_{(ij)(kl),\sigma\sigma'} = v_{(kl)(ij),\sigma\sigma'} = \int d\mathbf{x}_1 d\mathbf{x}_2 \frac{\phi_{i\sigma}^*(\mathbf{x}_1) \phi_{k\sigma'}^*(\mathbf{x}_2) \phi_{l\sigma'}(\mathbf{x}_2) \phi_{j\sigma}(\mathbf{x}_1)}{r_{12}}
\end{aligned}$$

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

$$\begin{aligned}
\hat{H} &= \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R \\
&+ \left(\sum_{i \in L, \sigma} a_{i\sigma}^\dagger \hat{S}_{i\sigma}^R + h.c. \right) + \left(\sum_{i \in L, \sigma} a_{i\sigma}^\dagger \hat{R}_{i\sigma}^R + h.c. + \sum_{i \in R, \sigma} a_{i\sigma}^\dagger \hat{R}_{i\sigma}^L + h.c. \right) \\
&+ \frac{1}{2} \left(\sum_{ik \in L, \sigma\sigma'} \hat{A}_{ik,\sigma\sigma'}^L \hat{P}_{ik,\sigma\sigma'}^R + h.c. \right) + \sum_{ij \in L, \sigma} \hat{B}_{ij\sigma} \hat{Q}_{ij\sigma}^R - \sum_{il \in L, \sigma\sigma'} \hat{B}'_{il\sigma\sigma'} \hat{Q}'_{il\sigma\sigma'}^R
\end{aligned}$$

where the normal and complementary operators are defined by

$$\begin{aligned}
\hat{S}_{i\sigma}^{L/R} &= \sum_{j \in L/R} t_{ij,\sigma} a_{j\sigma}, \\
\hat{R}_{i\sigma}^{L/R} &= \sum_{jkl \in L/R, \sigma'} v_{ijkl, \sigma \sigma'} a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma}, \\
\hat{A}_{ik, \sigma \sigma'} &= a_{i\sigma}^\dagger a_{k\sigma'}^\dagger, \\
\hat{B}_{ij, \sigma} &= a_{i\sigma}^\dagger a_{j\sigma}, \\
\hat{B}'_{il, \sigma \sigma'} &= a_{i\sigma}^\dagger a_{l\sigma'}, \\
\hat{P}_{ik, \sigma \sigma'}^R &= \sum_{jl \in R} v_{ijkl, \sigma \sigma'} a_{l\sigma'} a_{j\sigma}, \\
\hat{Q}_{ij, \sigma}^R &= \sum_{kl \in R, \sigma'} v_{ijkl, \sigma \sigma'} a_{k\sigma'}^\dagger a_{l\sigma'}, \\
\hat{Q}'_{il, \sigma \sigma'}^R &= \sum_{jk \in R} v_{ijkl, \sigma \sigma'} a_{k\sigma'}^\dagger a_{j\sigma}
\end{aligned}$$

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

$$\hat{R}_{i\sigma}^{L/R} = \frac{1}{2} \hat{S}_{i\sigma}^{L/R} + \hat{R}_{i\sigma}^{L/R} = \frac{1}{2} \sum_{j \in L/R} t_{ij,\sigma} a_{j\sigma} + \sum_{jkl \in L/R, \sigma'} v_{ijkl, \sigma \sigma'} a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma}$$

Then we have

$$\begin{aligned}
\hat{H}^{NC} &= \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R + \sum_{i \in L, \sigma} \left(a_{i\sigma}^\dagger \hat{R}_{i\sigma}^{R\dagger} - a_{i\sigma} \hat{R}_{i\sigma}^{R\dagger} \right) + \sum_{i \in R, \sigma} \left(\hat{R}_{i\sigma}^{L\dagger} a_{i\sigma} - \hat{R}_{i\sigma}^{L\dagger} a_{i\sigma}^\dagger \right) \\
&+ \frac{1}{2} \sum_{ik \in L, \sigma \sigma'} \left(\hat{A}_{ik, \sigma \sigma'} \hat{P}_{ik, \sigma \sigma'}^R + \hat{A}_{ik, \sigma \sigma'}^\dagger \hat{P}_{ik, \sigma \sigma'}^{R\dagger} \right) + \sum_{ij \in L, \sigma} \hat{B}_{ij, \sigma} \hat{Q}_{ij, \sigma}^R - \sum_{il \in L, \sigma \sigma'} \hat{B}'_{il, \sigma \sigma'} \hat{Q}'_{il, \sigma \sigma'}^R
\end{aligned}$$

Normal/Complementary Partitioning

With this normal/complementary partitioning, the operators required in left block are

$$\{ \hat{H}^L, \hat{1}^L, a_{i\sigma}^\dagger, a_{i\sigma}, \hat{R}_{k\sigma}^{L\dagger}, \hat{R}_{k\sigma}^{L\dagger}, \hat{A}_{ij, \sigma \sigma'}, \hat{A}_{ij, \sigma \sigma'}^\dagger, \hat{B}_{ij, \sigma}, \hat{B}'_{ij, \sigma \sigma'} \} \quad (i, j \in L, k \in R)$$

The operators required in right block are

$$\{ \hat{1}^R, \hat{H}^R, \hat{R}_{i\sigma}^{R\dagger}, \hat{R}_{i\sigma}^{R\dagger}, a_{k\sigma}, a_{k\sigma}^\dagger, \hat{P}_{ij, \sigma \sigma'}^R, \hat{P}_{ij, \sigma \sigma'}^{R\dagger}, \hat{Q}_{ij, \sigma}^R, \hat{Q}'_{ij, \sigma \sigma'}^R \} \quad (i, j \in L, k \in R)$$

Assuming that there are K sites in total, and K_L/K_R sites in left/right block (optimally, $K_L \leq K_R$), 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

$$\begin{aligned}\hat{H}^{CN} = & \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R + \sum_{i \in L, \sigma} \left(a_{i\sigma}^\dagger \hat{R}_{i\sigma}^{R'} - a_{i\sigma} \hat{R}_{i\sigma}^{R\dagger} \right) + \sum_{i \in R, \sigma} \left(\hat{R}_{i\sigma}^{L\dagger} a_{i\sigma} - \hat{R}_{i\sigma}^L a_{i\sigma}^\dagger \right) \\ & + \frac{1}{2} \sum_{jl \in R, \sigma\sigma'} \left(\hat{P}_{jl, \sigma\sigma'}^L \hat{A}_{jl, \sigma\sigma'} + \hat{P}_{jl, \sigma\sigma'}^{L\dagger} \hat{A}_{jl, \sigma\sigma'}^\dagger \right) + \sum_{kl \in R, \sigma} \hat{Q}_{kl, \sigma}^L \hat{B}_{kl, \sigma} - \sum_{jk \in R, \sigma\sigma'} \hat{Q}_{jk, \sigma\sigma'}^{L'} \hat{B}'_{jk, \sigma\sigma'}\end{aligned}$$

Now the operators required in left block are

$$\{ \hat{H}^L, \hat{1}^L, a_{i\sigma}^\dagger, a_{i\sigma}, \hat{R}_{k\sigma}^{L\dagger}, \hat{R}_{k\sigma}^L, \hat{P}_{kl, \sigma\sigma'}^L, \hat{P}_{kl, \sigma\sigma'}^{L\dagger}, \hat{Q}_{kl, \sigma}^L, \hat{Q}_{kl, \sigma}^{L'} \} \quad (k, l \in R, i \in L)$$

The operators required in right block are

$$\{ \hat{1}^R, \hat{H}^R, \hat{R}_{i\sigma}^{R'}, \hat{R}_{i\sigma}^{R\dagger}, a_{k\sigma}, a_{k\sigma}^\dagger, \hat{A}_{kl, \sigma\sigma'}, \hat{A}_{kl, \sigma\sigma'}^\dagger, \hat{B}_{kl, \sigma}, \hat{B}'_{kl, \sigma\sigma'} \} \quad (k, l \in R, i \in L)$$

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.

$$\begin{aligned}\hat{R}_{i\sigma}^{L*} = & \hat{R}_{i\sigma}^{L'} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_{i\sigma}^{*} + \sum_{j \in L} \left(\sum_{kl \in *, \sigma'} v_{ijkl, \sigma\sigma'} a_{k\sigma'}^\dagger a_{l\sigma'} \right) a_{j\sigma} + \sum_{j \in *} \left(\sum_{kl \in L, \sigma'} v_{ijkl, \sigma\sigma'} a_{k\sigma'}^\dagger a_{l\sigma'} \right) a_{j\sigma} \\ & + \sum_{k \in L, \sigma'} a_{k\sigma'}^\dagger \left(\sum_{jl \in *} v_{ijkl, \sigma\sigma'} a_{l\sigma'} a_{j\sigma} \right) + \sum_{k \in *, \sigma'} a_{k\sigma'}^\dagger \left(\sum_{jl \in L} v_{ijkl, \sigma\sigma'} a_{l\sigma'} a_{j\sigma} \right) - \sum_{l \in L, \sigma'} a_{l\sigma'} \left(\sum_{jk \in *} v_{ijkl, \sigma\sigma'} a_{k\sigma'}^\dagger a_{j\sigma} \right) \\ = & \hat{R}_{i\sigma}^{L'} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_{i\sigma}^{*} + \sum_{j \in L} a_{j\sigma} \left(\sum_{kl \in *, \sigma'} v_{ijkl, \sigma\sigma'} a_{k\sigma'}^\dagger a_{l\sigma'} \right) + \sum_{j \in *} \left(\sum_{kl \in L, \sigma'} v_{ijkl, \sigma\sigma'} a_{k\sigma'}^\dagger a_{l\sigma'} \right) a_{j\sigma} \\ & + \sum_{k \in L, \sigma'} a_{k\sigma'}^\dagger \left(\sum_{jl \in *} v_{ijkl, \sigma\sigma'} a_{l\sigma'} a_{j\sigma} \right) + \sum_{k \in *, \sigma'} \left(\sum_{jl \in L} v_{ijkl, \sigma\sigma'} a_{l\sigma'} a_{j\sigma} \right) a_{k\sigma'}^\dagger - \sum_{l \in L, \sigma'} a_{l\sigma'} \left(\sum_{jk \in *} v_{ijkl, \sigma\sigma'} a_{k\sigma'}^\dagger a_{j\sigma} \right)\end{aligned}$$

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

$$\begin{aligned}\hat{R}_{i\sigma}^{L*, NC} = & \hat{R}_{i\sigma}^{L'} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_{i\sigma}^{*} + \sum_{j \in L} a_{j\sigma} \hat{Q}_{ij, \sigma}^* + \sum_{j \in *, kl \in L, \sigma'} v_{ijkl, \sigma\sigma'} \hat{B}_{kl, \sigma'} a_{j\sigma} \\ & + \sum_{k \in L, \sigma'} a_{k\sigma'}^\dagger \hat{P}_{ik, \sigma\sigma'}^* + \sum_{k \in *, jl \in L, \sigma'} v_{ijkl, \sigma\sigma'} \hat{A}_{jl, \sigma\sigma'}^\dagger a_{k\sigma'}^\dagger - \sum_{l \in L, \sigma'} a_{l\sigma'} \hat{Q}_{il, \sigma\sigma'}^* - \sum_{l \in *, jk \in L, \sigma'} v_{ijkl, \sigma\sigma'} \hat{B}'_{kj, \sigma'} a_{l\sigma'} \\ = & \hat{R}_{i\sigma}^{L'} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_{i\sigma}^{*} + \sum_{k \in L, \sigma'} a_{k\sigma'}^\dagger \hat{P}_{ik, \sigma\sigma'}^* + \sum_{j \in L} a_{j\sigma} \hat{Q}_{ij, \sigma}^* - \sum_{l \in L, \sigma'} a_{l\sigma'} \hat{Q}_{il, \sigma\sigma'}^* \\ & + \sum_{k \in *, jl \in L, \sigma'} v_{ijkl, \sigma\sigma'} \hat{A}_{jl, \sigma\sigma'}^\dagger a_{k\sigma'}^\dagger + \sum_{j \in *, kl \in L, \sigma'} v_{ijkl, \sigma\sigma'} \hat{B}_{kl, \sigma'} a_{j\sigma} - \sum_{l \in *, jk \in L, \sigma'} v_{ijkl, \sigma\sigma'} \hat{B}'_{kj, \sigma'} a_{l\sigma'}\end{aligned}$$

$$\begin{aligned}
\hat{R}_{i\sigma}^{'L*,CN} &= \hat{R}_{i\sigma}^{'L} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_{i\sigma}^{'*} + \sum_{j \in L, kl \in *, \sigma'} v_{ijkl, \sigma \sigma'} a_{j\sigma} \hat{B}_{kl, \sigma'} + \sum_{j \in *} \hat{Q}_{ij, \sigma}^L a_{j\sigma} \\
&+ \sum_{k \in L, jl \in *, \sigma'} v_{ijkl, \sigma \sigma'} a_{k\sigma'}^\dagger \hat{A}_{jl, \sigma \sigma'}^\dagger + \sum_{k \in *, \sigma'} \hat{P}_{ik, \sigma \sigma'}^L a_{k\sigma'}^\dagger - \sum_{l \in L, jk \in *, \sigma'} v_{ijkl, \sigma \sigma'} a_{l\sigma'} \hat{B}_{kj, \sigma'}^\dagger - \sum_{l \in *, \sigma'} \hat{Q}_{il, \sigma \sigma'}^L a_{l\sigma'} \\
&= \hat{R}_{i\sigma}^{'L} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_{i\sigma}^{'*} + \sum_{k \in L, jl \in *, \sigma'} v_{ijkl, \sigma \sigma'} a_{k\sigma'}^\dagger \hat{A}_{jl, \sigma \sigma'}^\dagger + \sum_{j \in L, kl \in *, \sigma'} v_{ijkl, \sigma \sigma'} a_{j\sigma} \hat{B}_{kl, \sigma'} - \sum_{l \in L, jk \in *, \sigma'} v_{ijkl, \sigma \sigma'} a_{l\sigma'} \\
&+ \sum_{k \in *, \sigma'} \hat{P}_{ik, \sigma \sigma'}^L a_{k\sigma'}^\dagger + \sum_{j \in *} \hat{Q}_{ij, \sigma}^L a_{j\sigma} - \sum_{l \in *, \sigma'} \hat{Q}_{il, \sigma \sigma'}^L a_{l\sigma'}
\end{aligned}$$

Simplified Form

Define

$$\hat{Q}_{ij, \sigma \sigma'}^{''R} = \delta_{\sigma \sigma'} \hat{Q}_{ij\sigma}^R - \hat{Q}_{ij\sigma \sigma'}^R$$

we have N/C form

$$\begin{aligned}
\hat{H}^{NC} &= \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R + \sum_{i \in L, \sigma} \left(a_{i\sigma}^\dagger \hat{R}_{i\sigma}^{'R} - a_{i\sigma} \hat{R}_{i\sigma}^{'R\dagger} \right) + \sum_{i \in R, \sigma} \left(\hat{R}_{i\sigma}^{'L\dagger} a_{i\sigma} - \hat{R}_{i\sigma}^{'L} a_{i\sigma}^\dagger \right) \\
&+ \frac{1}{2} \sum_{ik \in L, \sigma \sigma'} \left(\hat{A}_{ik, \sigma \sigma'} \hat{P}_{ik, \sigma \sigma'}^R + \hat{A}_{ik, \sigma \sigma'}^\dagger \hat{P}_{ik, \sigma \sigma'}^{R\dagger} \right) + \sum_{ij \in L, \sigma \sigma'} \hat{B}_{ij\sigma \sigma'}' \hat{Q}_{ij\sigma \sigma'}^{''R}
\end{aligned}$$

With this normal/complementary partitioning, the operators required in left block are

$$\{ \hat{H}^L, \hat{1}^L, a_{i\sigma}^\dagger, a_{i\sigma}, \hat{R}_{k\sigma}^{'L\dagger}, \hat{R}_{k\sigma}^{'L}, \hat{A}_{ij, \sigma \sigma'}^\dagger, \hat{A}_{ij, \sigma \sigma'}, \hat{B}_{ij, \sigma \sigma'}' \} \quad (i, j \in L, k \in R)$$

The operators required in right block are

$$\{ \hat{1}^R, \hat{H}^R, \hat{R}_{i\sigma}^{'R}, \hat{R}_{i\sigma}^{'R\dagger}, a_{k\sigma}, a_{k\sigma}^\dagger, \hat{P}_{ij, \sigma \sigma'}^R, \hat{P}_{ij, \sigma \sigma'}^{R\dagger}, \hat{Q}_{ij, \sigma \sigma'}^{''R} \} \quad (i, j \in L, k \in R)$$

Assuming that there are K sites in total, and K_L/K_R sites in left/right block (optimally, $K_L \leq K_R$), 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

$$\begin{aligned}
\hat{H}^{CN} &= \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R + \sum_{i \in L, \sigma} \left(a_{i\sigma}^\dagger \hat{R}_{i\sigma}^{'R} - a_{i\sigma} \hat{R}_{i\sigma}^{'R\dagger} \right) + \sum_{i \in R, \sigma} \left(\hat{R}_{i\sigma}^{'L\dagger} a_{i\sigma} - \hat{R}_{i\sigma}^{'L} a_{i\sigma}^\dagger \right) \\
&+ \frac{1}{2} \sum_{jl \in R, \sigma \sigma'} \left(\hat{P}_{jl, \sigma \sigma'}^L \hat{A}_{jl, \sigma \sigma'}^\dagger + \hat{P}_{jl, \sigma \sigma'}^{L\dagger} \hat{A}_{jl, \sigma \sigma'} \right) + \sum_{kl \in R, \sigma \sigma'} \hat{Q}_{kl\sigma \sigma'}^{''L} \hat{B}_{kl\sigma \sigma'}'
\end{aligned}$$

Now the operators required in left block are

$$\{ \hat{H}^L, \hat{1}^L, a_{i\sigma}^\dagger, a_{i\sigma}, \hat{R}_{k\sigma}^{'L\dagger}, \hat{R}_{k\sigma}^{'L}, \hat{P}_{kl, \sigma \sigma'}^L, \hat{P}_{kl, \sigma \sigma'}^{L\dagger}, \hat{Q}_{kl, \sigma \sigma'}^{''L} \} \quad (k, l \in R, i \in L)$$

The operators required in right block are

$$\{\hat{1}^R, \hat{H}^R, \hat{R}_{i\sigma}^R, \hat{R}_{i\sigma}^{R\dagger}, a_{k\sigma}, a_{k\sigma}^\dagger, \hat{A}_{kl,\sigma\sigma'}, \hat{A}_{kl,\sigma\sigma'}^\dagger, \hat{B}'_{kl,\sigma\sigma'}\} \quad (k, l \in R, i \in L)$$

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

$$\begin{aligned} \hat{R}_{i\sigma}^{\prime L*,NC} &= \hat{R}_{i\sigma}^{\prime L} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_{i\sigma}^{\prime*} + \sum_{k \in L, \sigma'} a_{k\sigma'}^\dagger \hat{P}_{ik,\sigma\sigma'}^* + \sum_{j \in L, \sigma'} a_{j\sigma'} \hat{Q}_{ij,\sigma\sigma'}^{\prime\prime*} \\ &+ \sum_{k \in *, j \in L, \sigma'} v_{ijkl,\sigma\sigma'} \hat{A}_{jl,\sigma\sigma'}^\dagger a_{k\sigma'}^\dagger + \sum_{j \in *, k \in L, \sigma'} v_{ijkl,\sigma\sigma'} \hat{B}'_{kl,\sigma'\sigma'} a_{j\sigma} - \sum_{l \in *, j \in L, \sigma'} v_{ijkl,\sigma\sigma'} \hat{B}'_{kj,\sigma'\sigma} a_{l\sigma'} \\ \hat{R}_{i\sigma}^{\prime L*,CN} &= \hat{R}_{i\sigma}^{\prime L} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_{i\sigma}^{\prime*} + \sum_{k \in L, j \in *, \sigma'} v_{ijkl,\sigma\sigma'} a_{k\sigma'}^\dagger \hat{A}_{jl,\sigma\sigma'}^\dagger + \sum_{j \in L, k \in *, \sigma'} v_{ijkl,\sigma\sigma'} a_{j\sigma} \hat{B}'_{kl,\sigma'\sigma'} \\ &- \sum_{l \in L, j \in *, \sigma'} v_{ijkl,\sigma\sigma'} a_{l\sigma'} \hat{B}'_{kj,\sigma'\sigma} + \sum_{k \in *, \sigma'} \hat{P}_{ik,\sigma\sigma'}^L a_{k\sigma'}^\dagger + \sum_{j \in *, \sigma'} \hat{Q}_{ij,\sigma\sigma'}^{\prime L} a_{j\sigma'} \end{aligned}$$

7.1.4 Sum MPO Formalism in Unrestricted Spatial Orbitals

Hamiltonian

The quantum chemistry Hamiltonian is written as follows

$$\hat{H} = \sum_{ij,\sigma} t_{ij,\sigma} a_{i\sigma}^\dagger a_{j\sigma} + \frac{1}{2} \sum_{ijkl,\sigma\sigma'} v_{ijkl,\sigma\sigma'} a_{i\sigma}^\dagger a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma}$$

where

$$\begin{aligned} t_{ij,\sigma} &= t_{(ij),\sigma} = \int d\mathbf{x} \phi_{i\sigma}^*(\mathbf{x}) \left(-\frac{1}{2} \nabla^2 - \sum_a \frac{Z_a}{r_a} \right) \phi_{j\sigma}(\mathbf{x}) \\ v_{ijkl,\sigma\sigma'} &= v_{(ij)(kl),\sigma\sigma'} = v_{(kl)(ij),\sigma\sigma'} = \int d\mathbf{x}_1 d\mathbf{x}_2 \frac{\phi_{i\sigma}^*(\mathbf{x}_1) \phi_{k\sigma'}^*(\mathbf{x}_2) \phi_{l\sigma'}(\mathbf{x}_2) \phi_{j\sigma}(\mathbf{x}_1)}{r_{12}} \end{aligned}$$

Note that here the order of $ijkl$ is the same as that in FCIDUMP (chemist's notation $[ij|kl]$).

Derivation

Sum of MPO

$$\hat{H} = \sum_{m\sigma} a_{m\sigma}^\dagger \hat{H}_{m\sigma} = \sum_{m\sigma} a_{m\sigma}^\dagger \left[\sum_j t_{mj,\sigma} a_{j\sigma} + \frac{1}{2} \sum_{jkl,\sigma'} v_{mjkl,\sigma\sigma'} a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} \right]$$

Now consider LR partition. There are 8 possibilities: $LLL, LRR, RLR, RRL, LLR, LRL, RLL, RRR$.

$$\begin{aligned}\hat{H}_{m\sigma} &= \left[\sum_{j \in L} t_{mj,\sigma} a_{j\sigma} + \frac{1}{2} \sum_{jkl \in L, \sigma'} v_{mjkl, \sigma\sigma'} a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} \right] + \left[\sum_{j \in R} t_{mj,\sigma} a_{j\sigma} + \frac{1}{2} \sum_{jkl \in R, \sigma'} v_{mjkl, \sigma\sigma'} a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} \right] \\ &+ \left[\frac{1}{2} \sum_{j \in L} a_{j\sigma} \sum_{kl \in R, \sigma'} v_{mjkl, \sigma\sigma'} a_{k\sigma'}^\dagger a_{l\sigma'} + \frac{1}{2} \sum_{k \in L, \sigma'} a_{k\sigma'}^\dagger \sum_{jl \in R} v_{mjkl, \sigma\sigma'} a_{l\sigma'} a_{j\sigma} - \frac{1}{2} \sum_{l \in L, \sigma'} a_{l\sigma'} \sum_{jk \in R} v_{mjkl, \sigma\sigma'} a_{k\sigma'}^\dagger a_{j\sigma} \right] \\ &+ \left[\frac{1}{2} \sum_{j \in R} \left(\sum_{kl \in L, \sigma'} v_{mjkl, \sigma\sigma'} a_{k\sigma'}^\dagger a_{l\sigma'} \right) a_{j\sigma} + \frac{1}{2} \sum_{k \in R, \sigma'} \left(\sum_{jl \in L} v_{mjkl, \sigma\sigma'} a_{l\sigma'} a_{j\sigma} \right) a_{k\sigma'}^\dagger - \frac{1}{2} \sum_{l \in R, \sigma'} \left(\sum_{jk \in L} v_{mjkl, \sigma\sigma'} a_{k\sigma'}^\dagger a_{j\sigma} \right) \right]\end{aligned}$$

Let

$$\begin{aligned}\hat{H}_{m\sigma}^{L/R} &= \sum_{j \in L/R} t_{mj,\sigma} a_{j\sigma} + \frac{1}{2} \sum_{jkl \in L/R, \sigma'} v_{mjkl, \sigma\sigma'} a_{k\sigma'}^\dagger a_{l\sigma'} a_{j\sigma} \\ \hat{P}_{ik, \sigma\sigma'}^{L/R} &= \sum_{jl \in L/R} v_{ijkl, \sigma\sigma'} a_{l\sigma'} a_{j\sigma}, \\ \hat{Q}_{ij, \sigma}^{L/R} &= \sum_{kl \in L/R, \sigma'} v_{ijkl, \sigma\sigma'} a_{k\sigma'}^\dagger a_{l\sigma'}, \\ \hat{Q}_{il, \sigma\sigma'}^{L/R} &= \sum_{jk \in L/R} v_{ijkl, \sigma\sigma'} a_{k\sigma'}^\dagger a_{j\sigma} \\ \hat{Q}_{ij, \sigma\sigma'}^{''R} &= \delta_{\sigma\sigma'} \hat{Q}_{ij\sigma}^R - \hat{Q}_{ij\sigma\sigma'}^R\end{aligned}$$

we have

$$\begin{aligned}\hat{H}_{m\sigma} &= \hat{H}_{m\sigma}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}_{m\sigma}^R + \frac{1}{2} \sum_{j \in L} a_{j\sigma} \hat{Q}_{mj, \sigma}^R + \frac{1}{2} \sum_{k \in L, \sigma'} a_{k\sigma'}^\dagger \hat{P}_{mk, \sigma\sigma'}^R - \frac{1}{2} \sum_{l \in L, \sigma'} a_{l\sigma'} \hat{Q}_{ml, \sigma\sigma'}^R + \frac{1}{2} \sum_{j \in R} \hat{Q}_{mj, \sigma}^L a_{j\sigma} + \frac{1}{2} \\ &= \hat{H}_{m\sigma}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}_{m\sigma}^R + \frac{1}{2} \sum_{k \in L, \sigma'} a_{k\sigma'}^\dagger \hat{P}_{mk, \sigma\sigma'}^R + \frac{1}{2} \sum_{j \in L, \sigma'} a_{j\sigma'} \left(\delta_{\sigma\sigma'} \hat{Q}_{mj, \sigma}^R - \hat{Q}_{mj, \sigma\sigma'}^R \right) + \frac{1}{2} \sum_{k \in R, \sigma'} \hat{P}_{mk, \sigma\sigma'}^L a_{k\sigma'}^\dagger + \\ &= \hat{H}_{m\sigma}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}_{m\sigma}^R + \frac{1}{2} \sum_{k \in L, \sigma'} a_{k\sigma'}^\dagger \hat{P}_{mk, \sigma\sigma'}^R + \frac{1}{2} \sum_{j \in L, \sigma'} a_{j\sigma'} \hat{Q}_{mj, \sigma\sigma'}^{''R} + \frac{1}{2} \sum_{k \in R, \sigma'} \hat{P}_{mk, \sigma\sigma'}^L a_{k\sigma'}^\dagger + \frac{1}{2} \sum_{j \in R, \sigma'} \hat{Q}_{mj, \sigma\sigma'}^{''L}\end{aligned}$$

Now consider $m \in L$ or $m \in R$. For $m \in L$:

$$\begin{aligned}\sum_{m \in L, \sigma} a_{m\sigma}^\dagger \hat{H}_{m\sigma} &= \left(\sum_{m \in L, \sigma} a_{m\sigma}^\dagger \hat{H}_{m\sigma}^L \right) \otimes \hat{1}^R + \sum_{m \in L, \sigma} a_{m\sigma}^\dagger \otimes \hat{H}_{m\sigma}^R \\ &+ \frac{1}{2} \sum_{mk \in L, \sigma\sigma'} a_{m\sigma}^\dagger a_{k\sigma'}^\dagger \hat{P}_{mk, \sigma\sigma'}^R + \frac{1}{2} \sum_{mj \in L, \sigma\sigma'} a_{m\sigma}^\dagger a_{j\sigma'} \hat{Q}_{mj, \sigma\sigma'}^{''R} + \frac{1}{2} \sum_{k \in R, \sigma'} \left(\sum_{m \in L, \sigma} a_{m\sigma}^\dagger \hat{P}_{mk, \sigma\sigma'}^L \right) a_{k\sigma'}^\dagger + \frac{1}{2} \sum_{j \in R, \sigma'} \left(\sum_{m \in L, \sigma} a_{m\sigma}^\dagger \hat{Q}_{mj, \sigma\sigma'}^{''L} \right) a_{j\sigma} \\ &= \hat{H}^{ML} \otimes \hat{1}^R + \sum_{m \in L, \sigma} a_{m\sigma}^\dagger \otimes \hat{H}_{m\sigma}^R + \frac{1}{2} \sum_{mk \in L, \sigma\sigma'} \hat{A}_{mk, \sigma\sigma'} \hat{P}_{mk, \sigma\sigma'}^R + \frac{1}{2} \sum_{mj \in L, \sigma\sigma'} \hat{B}_{mj, \sigma\sigma'} \hat{Q}_{mj, \sigma\sigma'}^{''R} + \frac{1}{2} \sum_{k \in R, \sigma'} \hat{C}_{k\sigma'}\end{aligned}$$

where

$$\begin{aligned}
\hat{A}_{ik,\sigma\sigma'} &= a_{i\sigma}^\dagger a_{k\sigma'}^\dagger, \\
\hat{B}_{il,\sigma\sigma'} &= a_{i\sigma}^\dagger a_{l\sigma'}, \\
\hat{H}^{ML/R} &= \sum_{m \in L/R, \sigma} a_{m\sigma}^\dagger \hat{H}_{m\sigma}^{L/R} \\
\hat{P}_{k\sigma'}^{ML/R} &= \sum_{m \in L/R, \sigma} a_{m\sigma}^\dagger \hat{P}_{mk, \sigma\sigma'}^{L/R} \\
\hat{Q}_{j\sigma'}^{ML/R} &= \sum_{m \in L/R, \sigma} a_{m\sigma}^\dagger \hat{Q}_{mj, \sigma\sigma'}^{L/R}
\end{aligned}$$

For $m \in R$:

$$\begin{aligned}
\sum_{m \in R, \sigma} a_{m\sigma}^\dagger \hat{H}_{m\sigma} &= - \sum_{m \in R, \sigma} \hat{H}_{m\sigma}^L \otimes a_{m\sigma}^\dagger + \hat{1}^L \otimes \left(\sum_{m \in R, \sigma} a_{m\sigma}^\dagger \hat{H}_{m\sigma}^R \right) \\
&\quad - \frac{1}{2} \sum_{k \in L, \sigma'} a_{k\sigma'}^\dagger \left(\sum_{m \in R, \sigma} a_{m\sigma}^\dagger \hat{P}_{mk, \sigma\sigma'}^R \right) - \frac{1}{2} \sum_{j \in L, \sigma'} a_{j\sigma'} \left(\sum_{m \in R, \sigma} a_{m\sigma}^\dagger \hat{Q}_{mj, \sigma\sigma'}^{R} \right) + \frac{1}{2} \sum_{mk \in R, \sigma\sigma'} \hat{P}_{mk, \sigma\sigma'}^L a_{m\sigma}^\dagger a_{k\sigma'} \\
&= - \sum_{m \in R, \sigma} \hat{H}_{m\sigma}^L \otimes a_{m\sigma}^\dagger + \hat{1}^L \otimes \hat{H}^{MR} - \frac{1}{2} \sum_{k \in L, \sigma'} a_{k\sigma'}^\dagger \hat{P}_{k, \sigma'}^{MR} - \frac{1}{2} \sum_{j \in L, \sigma'} a_{j\sigma'} \hat{Q}_{j, \sigma'}^{MR} + \frac{1}{2} \sum_{mk \in R, \sigma\sigma'} \hat{P}_{mk, \sigma\sigma'}^L \hat{A}_{mk, \sigma\sigma'}
\end{aligned}$$

In summary

$$\begin{aligned}
\hat{H} &= \hat{H}^{ML} \otimes \hat{1}^R + \sum_{m \in L, \sigma} a_{m\sigma}^\dagger \otimes \hat{H}_{m\sigma}^R + \frac{1}{2} \sum_{mj \in L, \sigma\sigma'} \hat{A}_{mj, \sigma\sigma'} \hat{P}_{mj, \sigma\sigma'}^R + \frac{1}{2} \sum_{mj \in L, \sigma\sigma'} \hat{B}_{mj, \sigma\sigma'} \hat{Q}_{mj, \sigma\sigma'}^{R} + \frac{1}{2} \sum_{k \in R, \sigma'} \hat{P}_{k\sigma'}^{ML} a_{k\sigma'}^\dagger + \\
&\quad - \sum_{n \in R, \sigma} \hat{H}_{n\sigma}^L \otimes a_{n\sigma}^\dagger + \hat{1}^L \otimes \hat{H}^{MR} - \frac{1}{2} \sum_{j \in L, \sigma'} a_{j\sigma'}^\dagger \hat{P}_{j, \sigma'}^{MR} - \frac{1}{2} \sum_{j \in L, \sigma'} a_{j\sigma'} \hat{Q}_{j, \sigma'}^{MR} + \frac{1}{2} \sum_{nk \in R, \sigma\sigma'} \hat{P}_{nk, \sigma\sigma'}^L \hat{A}_{nk, \sigma\sigma'} + \frac{1}{2} \sum_{nk \in R, \sigma\sigma'} \hat{B}_{nk, \sigma\sigma'} \hat{Q}_{nk, \sigma\sigma'}^{R}
\end{aligned}$$

The operators required in left block are

$$\{ \hat{H}^{ML}, a_{m\sigma}^\dagger, \hat{A}_{mj, \sigma\sigma'}, \hat{B}_{mj, \sigma\sigma'}, \hat{P}_{k\sigma'}^{ML}, \hat{Q}_{k\sigma'}^{ML}, \hat{H}_{n\sigma}^L, \hat{1}^L, a_{j\sigma'}, a_{j\sigma'}, \hat{P}_{nk, \sigma\sigma'}^L, \hat{Q}_{nk, \sigma\sigma'}^{R} \} \quad (m, j \in L, n, k \in R)$$

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

$$\begin{aligned}
L &= \{ \hat{H}^{ML}, \hat{1}^L, a_{m\sigma}^\dagger, \hat{H}_{n\sigma}^L, a_{j\sigma'}^\dagger, a_{j\sigma'}, \hat{P}_{k\sigma'}^{ML}, \hat{Q}_{k\sigma'}^{ML}, \hat{A}_{mj, \sigma\sigma'}, \hat{B}_{mj, \sigma\sigma'}, \hat{P}_{nk, \sigma\sigma'}^L, \hat{Q}_{nk, \sigma\sigma'}^{R} \} \quad (m, j \in L, n, k \in R) \\
R &= \{ \hat{1}^R, \hat{H}^{MR}, \hat{H}_{m\sigma}^R, a_{n\sigma}^\dagger, \hat{P}_{j, \sigma'}^{MR}, \hat{Q}_{j, \sigma'}^{MR}, a_{k\sigma'}^\dagger, a_{k\sigma'}, \hat{P}_{mj, \sigma\sigma'}^R, \hat{Q}_{mj, \sigma\sigma'}^{R}, \hat{A}_{nk, \sigma\sigma'}, \hat{B}_{nk, \sigma\sigma'} \}
\end{aligned}$$

Now let

$$\begin{aligned}
\hat{R}_{k\sigma}^{ML/R} &= -2\delta(k \in M) \hat{H}_{k\sigma}^{L/R} + \hat{P}_{k\sigma'}^{ML/R} \\
\hat{S}_{k\sigma}^{ML/R} &= \hat{Q}_{k\sigma'}^{ML/R}
\end{aligned}$$

we have

$$\begin{aligned} L &= \{ \hat{H}^{ML}, \hat{1}^L, a_{j\sigma'}^\dagger, a_{j\sigma'}, \hat{R}_{k\sigma'}^{ML}, \hat{S}_{k\sigma'}^{ML}, \hat{A}_{mj,\sigma\sigma'}, \hat{B}_{mj,\sigma\sigma'}, \hat{P}_{nk,\sigma\sigma'}^L, \hat{Q}_{nk,\sigma\sigma'}^{''L} \} \quad (m, j \in L, n, k \in R) \\ R &= \{ \hat{1}^R, \hat{H}^{MR}, \hat{R}_{j,\sigma'}^{MR}, \hat{S}_{j,\sigma'}^{MR}, a_{k\sigma'}^\dagger, a_{k\sigma'}, \hat{P}_{mj,\sigma\sigma'}^R, \hat{Q}_{mj,\sigma\sigma'}^{''R}, \hat{A}_{nk,\sigma\sigma'}, \hat{B}_{nk,\sigma\sigma'} \} \end{aligned}$$

The total number of operators is

$$N = 2 + 4K + 8K_{ML}K_L + 8K_{MR}K_R$$

Blocking

$$\begin{aligned} \hat{P}_{k\sigma'}^{ML*} &= \sum_{m \in L^*, \sigma} a_{m\sigma}^\dagger \hat{P}_{mk,\sigma\sigma'}^{L*} = \sum_{m \in L^*, \sigma} a_{m\sigma}^\dagger \sum_{j \in L^*} v_{mjkl,\sigma\sigma'} a_{l\sigma'} a_{j\sigma} \\ &= \hat{P}_{k\sigma'}^{ML} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{P}_{k\sigma'}^{M*} + \sum_{m \in *, \sigma} a_{m\sigma}^\dagger \hat{P}_{mk,\sigma\sigma'}^L + \sum_{m \in L, \sigma} a_{m\sigma}^\dagger \sum_{j \in *, l \in L} v_{mjkl,\sigma\sigma'} a_{l\sigma'} a_{j\sigma} + \sum_{m \in L, \sigma} a_{m\sigma}^\dagger \sum_{j \in L, l \in *} v_{mjkl,\sigma\sigma'} a_{l\sigma'} a_{j\sigma} \\ &\quad + \sum_{m \in L, \sigma} a_{m\sigma}^\dagger \hat{P}_{mk,\sigma\sigma'}^* + \sum_{m \in *, \sigma} a_{m\sigma}^\dagger \sum_{j \in *, l \in L} v_{mjkl,\sigma\sigma'} a_{l\sigma'} a_{j\sigma} + \sum_{m \in *, \sigma} a_{m\sigma}^\dagger \sum_{j \in L, l \in *} v_{mjkl,\sigma\sigma'} a_{l\sigma'} a_{j\sigma} \\ &= \hat{P}_{k\sigma'}^{ML} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{P}_{k\sigma'}^{M*} + \sum_{m \in *, \sigma} \hat{P}_{mk,\sigma\sigma'}^L a_{m\sigma}^\dagger + \sum_{m \in L, \sigma} a_{m\sigma}^\dagger \hat{P}_{mk,\sigma\sigma'}^* + \sum_{ml \in L, j \in *, \sigma} v_{mjkl,\sigma\sigma'} a_{m\sigma}^\dagger 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_{m\sigma}^\dagger a_{j\sigma} + \sum_{ml \in *, j \in L, \sigma} v_{mjkl,\sigma\sigma'} a_{j\sigma} a_{m\sigma}^\dagger a_{l\sigma'} \\ &= \hat{P}_{k\sigma'}^{ML} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{P}_{k\sigma'}^{M*} + \sum_{m \in *, \sigma} \hat{P}_{mk,\sigma\sigma'}^L a_{m\sigma}^\dagger + \sum_{m \in L, \sigma} a_{m\sigma}^\dagger \hat{P}_{mk,\sigma\sigma'}^* + \sum_{ml \in L, j \in *, \sigma} v_{mjkl,\sigma\sigma'} a_{m\sigma}^\dagger 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_{m\sigma}^\dagger a_{j\sigma} + \sum_{mj \in *, l \in L, \sigma} v_{mlkj,\sigma\sigma'} a_{l\sigma} a_{m\sigma}^\dagger a_{j\sigma'} \\ &= \hat{P}_{k\sigma'}^{ML} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{P}_{k\sigma'}^{M*} + \sum_{m \in *, \sigma} \hat{P}_{mk,\sigma\sigma'}^L a_{m\sigma}^\dagger + \sum_{m \in L, \sigma} a_{m\sigma}^\dagger \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}$$

$$\begin{aligned}
&= \hat{Q}_{j\sigma'}^{ML} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{Q}_{j\sigma'}^{M*} + \sum_{m \in *, \sigma} a_{m\sigma}^\dagger \hat{Q}_{mj, \sigma\sigma'}^{''L} + \sum_{m \in L, \sigma} a_{m\sigma}^\dagger \hat{Q}_{mj, \sigma\sigma'}^{''*} \\
&\quad - \sum_{ml \in L, k \in *, \sigma} \left(v_{mjkl, \sigma'\sigma} \hat{B}_{ml\sigma'\sigma} a_{k\sigma}^\dagger - v_{mlkj, \sigma\sigma'} \hat{B}_{ml\sigma\sigma'} a_{k\sigma'}^\dagger \right) + \sum_{ml \in L, k \in *, \sigma} \left(v_{mjlk, \sigma'\sigma} \hat{A}_{ml\sigma'\sigma} a_{k\sigma} - v_{mlkj, \sigma\sigma'} \hat{A}_{ml\sigma\sigma'} a_{k\sigma'} \right) \\
&\quad + \sum_{mk \in *, l \in L, \sigma} \left(v_{mjkl, \sigma'\sigma} a_{l\sigma} \hat{A}_{mk\sigma'\sigma} - v_{mlkj, \sigma\sigma'} a_{l\sigma} \hat{A}_{mk\sigma\sigma'} \right) - \sum_{mk \in *, l \in L, \sigma} \left(v_{mjlk, \sigma'\sigma} a_{l\sigma}^\dagger \hat{B}_{mk\sigma'\sigma} - v_{mlkj, \sigma\sigma'} a_{l\sigma'}^\dagger \hat{B}_{mk\sigma\sigma'} \right)
\end{aligned}$$

For P, Q , we have

$$\begin{aligned}
\hat{P}_{ik, \sigma\sigma'}^{L*} &= \sum_{jl \in L*} v_{ijkl, \sigma\sigma'} a_{l\sigma'} a_{j\sigma} = \hat{P}_{ik, \sigma\sigma'}^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{P}_{ik, \sigma\sigma'}^* + \sum_{j \in L, l \in *} v_{ijkl, \sigma\sigma'} a_{l\sigma'} a_{j\sigma} + \sum_{j \in *, l \in L} v_{ijkl, \sigma\sigma'} a_{l\sigma'} a_{j\sigma} \\
&= \hat{P}_{ik, \sigma\sigma'}^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{P}_{ik, \sigma\sigma'}^* - \sum_{j \in L, l \in *} v_{ijkl, \sigma\sigma'} a_{j\sigma} a_{l\sigma'} + \sum_{j \in L, l \in *} v_{ilkj, \sigma\sigma'} a_{j\sigma} a_{l\sigma'} \\
\hat{Q}_{ij, \sigma\sigma'}^{''L*} &= \delta_{\sigma\sigma'} \hat{Q}_{ij\sigma}^{L*} - \hat{Q}_{ij\sigma\sigma'}^{L*} = \delta_{\sigma\sigma'} \sum_{kl \in L*, \sigma''} v_{ijkl, \sigma\sigma''} a_{k\sigma''}^\dagger a_{l\sigma''} - \sum_{kl \in L*} v_{ilkj, \sigma\sigma'} a_{k\sigma'}^\dagger a_{l\sigma} \\
&= \hat{Q}_{ij, \sigma\sigma'}^{''L} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{Q}_{ij, \sigma\sigma'}^{''*} + \delta_{\sigma\sigma'} \sum_{k \in L, l \in *, \sigma''} v_{ijkl, \sigma\sigma''} a_{k\sigma''}^\dagger a_{l\sigma''} - \sum_{k \in L, l \in *} v_{ilkj, \sigma\sigma'} a_{k\sigma'}^\dagger a_{l\sigma} + \delta_{\sigma\sigma'} \sum_{k \in *, l \in L, \sigma''} v_{ijkl, \sigma\sigma''} a_{k\sigma''}^\dagger a_{l\sigma''} \\
&= \hat{Q}_{ij, \sigma\sigma'}^{''L} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{Q}_{ij, \sigma\sigma'}^{''*} + \delta_{\sigma\sigma'} \sum_{k \in L, l \in *, \sigma''} v_{ijkl, \sigma\sigma''} a_{k\sigma''}^\dagger a_{l\sigma''} - \sum_{k \in L, l \in *} v_{ilkj, \sigma\sigma'} a_{k\sigma'}^\dagger a_{l\sigma} - \delta_{\sigma\sigma'} \sum_{k \in L, l \in *, \sigma''} v_{ijkl, \sigma\sigma''} a_{k\sigma''}^\dagger a_{l\sigma''}
\end{aligned}$$

7.1.5 DMRG Quantum Chemistry Hamiltonian in Spin Orbitals

Hamiltonian

The quantum chemistry Hamiltonian is written as follows

$$\hat{H} = \sum_{ij} t_{ij} a_i^\dagger a_j + \frac{1}{2} \sum_{ijkl} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j$$

where $ijkl$ are spin orbital indices, and

$$\begin{aligned}
t_{ij} &= \int d\mathbf{x} \phi_i^*(\mathbf{x}) \left(-\frac{1}{2} \nabla^2 - \sum_a \frac{Z_a}{r_a} \right) \phi_j(\mathbf{x}) \\
v_{ijkl} &= \int d\mathbf{x}_1 d\mathbf{x}_2 \frac{\phi_i^*(\mathbf{x}_1) \phi_k^*(\mathbf{x}_2) \phi_l(\mathbf{x}_2) \phi_j(\mathbf{x}_1)}{r_{12}}
\end{aligned}$$

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

$$\begin{aligned}
t_{i\sigma, j\tau} &= t_{ij} \delta_{\sigma\tau} \\
v_{i\sigma, j\tau, k\mu, l\nu} &= v_{ijkl} \delta_{\sigma\tau} \delta_{\mu\nu}
\end{aligned}$$

For complex orbitals, we have

$$\begin{aligned}
t_{ij} &= t_{ji}^* \\
v_{ijkl} &= v_{klij} = v_{jilk}^* = v_{lkji}^*
\end{aligned}$$

For real orbitals, we have

$$t_{ij} = t_{ji}$$

$$v_{ijkl} = v_{klij} = v_{jilk} = v_{lkji} = v_{jikl} = v_{iljk} = v_{lkij} = v_{klji}$$

Partitioning in Spin Orbitals

The partitioning of Hamiltonian in left (L) and right (R) blocks is given by

$$\hat{H} = \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R + \left(\sum_{i \in L} a_i^\dagger \hat{R}_i^R + h.c. + \sum_{i \in R} a_i^\dagger \hat{R}_i^L + h.c. \right) + \frac{1}{2} \left(\sum_{ik \in L} \hat{A}_{ik}^L \hat{P}_{ik}^R + h.c. \right) + \sum_{ij \in L} \hat{B}_{ij}^L \hat{Q}_{ij}^R$$

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

$$\begin{aligned} & \left(\sum_{i \in L} a_i^\dagger \hat{S}_i^R + h.c. + \sum_{i \in R} a_i^\dagger \hat{S}_i^L + h.c. \right) \\ &= \left(\sum_{i \in L} a_i^\dagger \hat{S}_i^R + \sum_{i \in L} \hat{S}_i^{R\dagger} a_i + \sum_{i \in R} a_i^\dagger \hat{S}_i^L + \sum_{i \in R} \hat{S}_i^{L\dagger} a_i \right) \\ &= \frac{1}{2} \left(\sum_{i \in L, j \in R} t_{ij} a_i^\dagger a_j + \sum_{i \in L, j \in R} t_{ij}^* a_j^\dagger a_i + \sum_{i \in R, j \in L} t_{ij} a_i^\dagger a_j + \sum_{i \in R, j \in L} t_{ij}^* a_j^\dagger a_i \right) \end{aligned}$$

Using $t_{ij}^* = t_{ji}$ and swap the indices ij we have

$$\begin{aligned} \cdots &= \frac{1}{2} \left(\sum_{i \in L, j \in R} t_{ij} a_i^\dagger a_j + \sum_{i \in R, j \in L} t_{ij} a_i^\dagger a_j + \sum_{i \in R, j \in L} t_{ij} a_i^\dagger a_j + \sum_{i \in L, j \in R} t_{ij} a_i^\dagger a_j \right) \\ &= \sum_{i \in L, j \in R} t_{ij} a_i^\dagger a_j + \sum_{i \in R, j \in L} t_{ij} a_i^\dagger a_j \end{aligned}$$

Next consider one of $ijkl$ in left, and three of them in right. These terms are

$$\begin{aligned}\hat{H}_{1L,3R} &= \frac{1}{2} \sum_{i \in L, jkl \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2} \sum_{j \in L, ikl \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2} \sum_{k \in L, ijl \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2} \sum_{l \in L, ijk \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j \\ &= \left[\frac{1}{2} \sum_{i \in L, jkl \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2} \sum_{k \in L, ijl \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j \right] + \frac{1}{2} \sum_{j \in L, ikl \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2} \sum_{l \in L, ijk \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j\end{aligned}$$

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

$$\frac{1}{2} \sum_{j \in L, ikl \in R} v_{ijkl}^* a_j^\dagger a_l^\dagger a_k a_i = \frac{1}{2} \sum_{k \in L, ijl \in R} v_{lkji}^* a_k^\dagger a_i^\dagger a_j a_l = \frac{1}{2} \sum_{k \in L, ijl \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j$$

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 $v_{ijkl} = v_{klji}$

$$\begin{aligned}\hat{H}_{1L,3R} &= \left[\frac{1}{2} \sum_{i \in L, jkl \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2} \sum_{k \in L, ijl \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j \right] + h.c. \\ &= \left[\frac{1}{2} \sum_{i \in L, jkl \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2} \sum_{k \in L, ijl \in R} v_{ijkl} a_k^\dagger a_i^\dagger a_j a_l \right] + h.c. \\ &= \left[\frac{1}{2} \sum_{i \in L, jkl \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2} \sum_{i \in L, jkl \in R} v_{klji} a_i^\dagger a_k^\dagger a_l a_j \right] + h.c. \\ &= \sum_{i \in L, jkl \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j + h.c. \\ &= \sum_{i \in L} a_i^\dagger \sum_{jkl \in R} v_{ijkl} a_k^\dagger a_l a_j + h.c. = \sum_{i \in L} a_i^\dagger R_i^R + h.c.\end{aligned}$$

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,

$$\sum_{ik \in R, jl \in L} a_i^\dagger a_k^\dagger v_{ijkl} a_l a_j = \sum_{jl \in R, ik \in L} a_j^\dagger a_l^\dagger v_{jilk} a_k a_i = \sum_{ik \in L, jl \in R} v_{jilk} a_j^\dagger a_l^\dagger a_k a_i = \sum_{ik \in L, jl \in R} v_{ijkl}^* (a_i^\dagger a_k^\dagger a_l a_j)^\dagger$$

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.

$$\begin{aligned}\hat{H}_{2L,2R} &= \frac{1}{2} \sum_{il \in L, jk \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2} \sum_{ij \in L, kl \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2} \sum_{kl \in L, ij \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j + \frac{1}{2} \sum_{jk \in L, il \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_l a_j \\ &= -\frac{1}{2} \sum_{il \in L, jk \in R} v_{ijkl} a_i^\dagger a_l a_k^\dagger a_j + \frac{1}{2} \sum_{ij \in L, kl \in R} v_{ijkl} a_i^\dagger a_j a_k^\dagger a_l + \frac{1}{2} \sum_{kl \in L, ij \in R} v_{ijkl} a_i^\dagger a_j a_k^\dagger a_l - \frac{1}{2} \sum_{jk \in L, il \in R} v_{ijkl} a_i^\dagger a_l a_k^\dagger a_j\end{aligned}$$

First consider the second and third terms

$$\begin{aligned}
& \frac{1}{2} \sum_{ij \in L, kl \in R} v_{ijkl} a_i^\dagger a_j a_k^\dagger a_l + \frac{1}{2} \sum_{kl \in L, ij \in R} v_{ijkl} a_i^\dagger a_j a_k^\dagger a_l \\
&= \frac{1}{2} \sum_{ij \in L, kl \in R} v_{ijkl} a_i^\dagger a_j a_k^\dagger a_l + \frac{1}{2} \sum_{kl \in L, ij \in R} v_{ijkl} a_i^\dagger a_l a_k^\dagger a_j \\
&= \frac{1}{2} \sum_{ij \in L, kl \in R} v_{ijkl} a_i^\dagger a_j a_k^\dagger a_l + \frac{1}{2} \sum_{ij \in L, kl \in R} v_{klij} a_i^\dagger a_j a_k^\dagger a_l \\
&= \sum_{ij \in L, kl \in R} v_{ijkl} a_i^\dagger a_j a_k^\dagger a_l = \sum_{ij \in L} a_i^\dagger a_j \sum_{kl \in R} v_{ijkl} a_k^\dagger a_l = \sum_{ij \in L} \hat{B}_{ij} \hat{Q}_{ij}^R
\end{aligned}$$

For the other two terms,

$$\begin{aligned}
& -\frac{1}{2} \sum_{il \in L, jk \in R} v_{ijkl} a_i^\dagger a_l a_k^\dagger a_j - \frac{1}{2} \sum_{jk \in L, il \in R} v_{ijkl} a_i^\dagger a_l a_k^\dagger a_j \\
&= -\frac{1}{2} \sum_{il \in L, jk \in R} v_{ijkl} a_i^\dagger a_l a_k^\dagger a_j - \frac{1}{2} \sum_{jk \in L, il \in R} v_{ijkl} a_i^\dagger a_k^\dagger a_j a_l \\
&= -\frac{1}{2} \sum_{il \in L, jk \in R} v_{ijkl} a_i^\dagger a_l a_k^\dagger a_j - \frac{1}{2} \sum_{il \in L, jk \in R} v_{klij} a_i^\dagger a_l a_k^\dagger a_j \\
&= -\sum_{il \in L, jk \in R} v_{ijkl} a_i^\dagger a_l a_k^\dagger a_j \\
&= -\sum_{il \in L} a_i^\dagger a_l \sum_{jk \in R} v_{ijkl} a_k^\dagger a_j = \sum_{il \in L} \hat{B}_{il} \hat{Q}_{il}^R
\end{aligned}$$

Then

$$\hat{Q}_{ij}^R = \hat{Q}_{ij'}^R + \hat{Q}_{ij''}^R = \sum_{kl \in R} (v_{ijkl} - v_{ilkj}) a_k^\dagger a_l$$

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

$$\begin{aligned}
\hat{H}^{NC} &= \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R \\
&+ \sum_{i \in L} \left(a_i^\dagger \hat{R}_i^R - a_i \hat{R}_i^{R\dagger} \right) + \sum_{i \in R} \left(\hat{R}_i^{L\dagger} a_i - \hat{R}_i^L a_i^\dagger \right) \\
&+ \frac{1}{2} \sum_{ik \in L} \left(\hat{A}_{ik} \hat{P}_{ik}^R + \hat{A}_{ik}^\dagger \hat{P}_{ik}^{R\dagger} \right) + \sum_{ij \in L} \hat{B}_{ij} \hat{Q}_{ij}^R
\end{aligned}$$

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

$$\{\hat{H}^L, \hat{1}^L, a_i^\dagger, a_i, \hat{R}_k^{L\dagger}, \hat{R}_k^L, \hat{A}_{ij}, \hat{A}_{ij}^\dagger, \hat{B}_{ij}\} \quad (i, j \in L, k \in R)$$

The operators required in right block are

$$\{\hat{1}^R, \hat{H}^R, \hat{R}_i^R, \hat{R}_i^{R\dagger}, a_k, a_k^\dagger, \hat{P}_{ij}^R, \hat{P}_{ij}^{R\dagger}, \hat{Q}_{ij}^R\} \quad (i, j \in L, k \in R)$$

Assuming that there are K sites in total, and K_L/K_R sites in left/right block (optimally, $K_L \leq K_R$), 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

$$\{\hat{H}^L, \hat{1}^L, a_i^\dagger, a_i, \hat{R}_k^{L\dagger}, \hat{R}_k^L, \hat{P}_{kl}^L, \hat{P}_{kl}^{L\dagger}, \hat{Q}_{kl}^L\} \quad (k, l \in R, i \in L)$$

The operators required in right block are

$$\{\hat{1}^R, \hat{H}^R, \hat{R}_i^R, \hat{R}_i^{R\dagger}, a_k, a_k^\dagger, \hat{A}_{kl}, \hat{A}_{kl}^\dagger, \hat{B}_{kl}\} \quad (k, l \in R, i \in L)$$

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.

$$\begin{aligned} \hat{R}_i^{L*} &= \hat{R}_i^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_i^* + \sum_{j \in L} \left(\sum_{kl \in *} v_{ijkl} a_k^\dagger a_l \right) a_j + \sum_{j \in *} \left(\sum_{kl \in L} v_{ijkl} a_k^\dagger a_l \right) a_j \\ &+ \sum_{k \in L} a_k^\dagger \left(\sum_{jl \in *} v_{ijkl} a_l a_j \right) + \sum_{k \in *} a_k^\dagger \left(\sum_{jl \in L} v_{ijkl} a_l a_j \right) - \sum_{l \in L} a_l \left(\sum_{jk \in *} v_{ijkl} a_k^\dagger a_j \right) - \sum_{l \in *} a_l \left(\sum_{jk \in L} v_{ijkl} a_k^\dagger a_j \right) \\ &= \hat{R}_i^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_i^* + \sum_{j \in L} a_j \left(\sum_{kl \in *} v_{ijkl} a_k^\dagger a_l \right) + \sum_{j \in *} \left(\sum_{kl \in L} v_{ijkl} a_k^\dagger a_l \right) a_j \\ &+ \sum_{k \in L} a_k^\dagger \left(\sum_{jl \in *} v_{ijkl} a_l a_j \right) + \sum_{k \in *} \left(\sum_{jl \in L} v_{ijkl} a_l a_j \right) a_k^\dagger - \sum_{l \in L} a_l \left(\sum_{jk \in *} v_{ijkl} a_k^\dagger a_j \right) - \sum_{l \in *} \left(\sum_{jk \in L} v_{ijkl} a_k^\dagger a_j \right) a_l \end{aligned}$$

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

$$\begin{aligned}
\hat{R}_i^{L*,NC} &= \hat{R}_i^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_i^* + \sum_{j \in L} a_j \hat{Q}_{ij}^* + \sum_{j \in *, kl \in L} (v_{ijkl} - v_{ilkj}) \hat{B}_{kl} a_j + \sum_{k \in L} a_k^\dagger \hat{P}_{ik}^* + \sum_{k \in *, jl \in L} v_{ijkl} \hat{A}_{jl}^\dagger a_k^\dagger \\
&= \hat{R}_i^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_i^* + \sum_{k \in L} a_k^\dagger \hat{P}_{ik}^* + \sum_{j \in L} a_j \hat{Q}_{ij}^* + \sum_{k \in *, jl \in L} v_{ijkl} \hat{A}_{jl}^\dagger a_k^\dagger + \sum_{j \in *, kl \in L} (v_{ijkl} - v_{ilkj}) \hat{B}_{kl} a_j \\
\hat{R}_i^{L*,CN} &= \hat{R}_i^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_i^* + \sum_{j \in L, kl \in *} (v_{ijkl} - v_{ilkj}) a_j \hat{B}_{kl} + \sum_{j \in *} \hat{Q}_{ij}^L a_j + \sum_{k \in L, jl \in *} v_{ijkl} a_k^\dagger \hat{A}_{jl}^\dagger + \sum_{k \in *} \hat{P}_{ik}^L a_k^\dagger \\
&= \hat{R}_i^L \otimes \hat{1}^* + \hat{1}^L \otimes \hat{R}_i^* + \sum_{k \in L, jl \in *} v_{ijkl} a_k^\dagger \hat{A}_{jl}^\dagger + \sum_{j \in L, kl \in *} (v_{ijkl} - v_{ilkj}) a_j \hat{B}_{kl} + \sum_{k \in *} \hat{P}_{ik}^L a_k^\dagger + \sum_{j \in *} \hat{Q}_{ij}^L a_j
\end{aligned}$$

Similarly,

$$\begin{aligned}
\hat{R}_i^{R*,NC} &= \hat{R}_i^* \otimes \hat{1}^R + \hat{1}^* \otimes \hat{R}_i^R + \sum_{k \in *} a_k^\dagger \hat{P}_{ik}^R + \sum_{j \in *} a_j \hat{Q}_{ij}^R + \sum_{k \in R, jl \in *} v_{ijkl} \hat{A}_{jl}^\dagger a_k^\dagger + \sum_{j \in R, kl \in *} (v_{ijkl} - v_{ilkj}) \hat{B}_{kl} a_j \\
\hat{R}_i^{R*,CN} &= \hat{R}_i^* \otimes \hat{1}^R + \hat{1}^* \otimes \hat{R}_i^R + \sum_{k \in *, jl \in R} v_{ijkl} a_k^\dagger \hat{A}_{jl}^\dagger + \sum_{j \in *, kl \in R} (v_{ijkl} - v_{ilkj}) a_j \hat{B}_{kl} + \sum_{k \in R} \hat{P}_{ik}^* a_k^\dagger + \sum_{j \in R} \hat{Q}_{ij}^* a_j
\end{aligned}$$

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 $O(K_{<}^2)$.

$$\begin{aligned}\hat{P}_{ik}^{L,NC \rightarrow CN} &= \sum_{jl \in L} v_{ijkl} a_l a_j = \sum_{jl \in L} v_{ijkl} \hat{A}_{jl}^\dagger \\ \hat{Q}_{ij}^{L,NC \rightarrow CN} &= \sum_{kl \in L} v_{ijkl} a_k^\dagger a_l = \sum_{kl \in L} v_{ijkl} \hat{B}_{kl}\end{aligned}$$

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)

$$\langle n_{p\sigma} n_{q\tau} \rangle = \langle a_{p\sigma}^\dagger a_{p\sigma} a_{q\tau}^\dagger a_{q\tau} \rangle$$

Number of particle correlation (mixed spin)

$$\langle a_{p\sigma}^\dagger a_{p\tau} a_{q\tau}^\dagger a_{q\sigma} \rangle$$

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

$$a_{q\tau}^\dagger a_{p\sigma} = -a_{p\sigma} a_{q\tau}^\dagger + \delta_{pq} \delta_{\sigma\tau}$$

We have

$$\langle a_{p\sigma}^\dagger a_{q\tau}^\dagger a_{q\tau} a_{p\sigma} \rangle = -\langle a_{p\sigma}^\dagger a_{q\tau}^\dagger a_{p\sigma} a_{q\tau} \rangle = \langle a_{p\sigma}^\dagger a_{p\sigma} a_{q\tau}^\dagger a_{q\tau} \rangle - \delta_{pq} \delta_{\sigma\tau} \langle a_{p\sigma}^\dagger a_{q\tau} \rangle$$

Then

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

Therefore,

$$C_{pq} \equiv \sum_{\sigma\tau} \langle n_{p\sigma} n_{q\tau} \rangle = e_{pqqp} + \delta_{pq} E_{pq}$$

Similarly,

$$\begin{aligned} s_{pqqp} &\equiv \sum_{\sigma\tau} (-1)^{1+\delta_{\sigma\tau}} \langle a_{p\sigma}^\dagger a_{q\tau}^\dagger a_{q\tau} a_{p\sigma} \rangle = -\sum_{\sigma\tau} (-1)^{1+\delta_{\sigma\tau}} \langle a_{p\sigma}^\dagger a_{q\tau}^\dagger a_{p\sigma} a_{q\tau} \rangle \\ &= \sum_{\sigma\tau} (-1)^{1+\delta_{\sigma\tau}} \langle a_{p\sigma}^\dagger a_{p\sigma} a_{q\tau}^\dagger a_{q\tau} \rangle - \delta_{pq} \sum_{\sigma} \langle a_{p\sigma}^\dagger a_{q\sigma} \rangle \\ &= \sum_{\sigma\tau} (-1)^{1+\delta_{\sigma\tau}} \langle n_{p\sigma} n_{q\tau} \rangle - \delta_{pq} \sum_{\sigma} \langle a_{p\sigma}^\dagger a_{q\sigma} \rangle \end{aligned}$$

Therefore,

$$S_{pq} \equiv \sum_{\sigma\tau} (-1)^{1+\delta_{\sigma\tau}} \langle n_{p\sigma} n_{q\tau} \rangle = s_{pqpp} + \delta_{pq} E_{pq}$$

Diagonal Spatial Two-Particle Density Matrix (Mixed Spin)

Using anticommutation relation

$$a_{q\tau}^\dagger a_{p\tau} = -a_{p\tau} a_{q\tau}^\dagger + \delta_{pq}$$

we have

$$\begin{aligned} e_{pqpp} &\equiv \sum_{\sigma\tau} \langle a_{p\sigma}^\dagger a_{q\tau}^\dagger a_{p\tau} a_{q\sigma} \rangle = - \sum_{\sigma\tau} \langle a_{p\sigma}^\dagger a_{p\tau} a_{q\tau}^\dagger a_{q\sigma} \rangle + \delta_{pq} \sum_{\sigma\tau} \langle a_{p\sigma}^\dagger a_{q\sigma} \rangle \\ &= - \sum_{\sigma\tau} \langle a_{p\sigma}^\dagger a_{p\tau} a_{q\tau}^\dagger a_{q\sigma} \rangle + 2\delta_{pq} \sum_{\sigma} \langle a_{p\sigma}^\dagger a_{q\sigma} \rangle \end{aligned}$$

Therefore,

$$\sum_{\sigma\tau} \langle a_{p\sigma}^\dagger a_{p\tau} a_{q\tau}^\dagger a_{q\sigma} \rangle = -e_{pqpp} + 2\delta_{pq} E_{pq}$$

INDEX

B

block2::Allocator (C++ *struct*), 278

block2::Allocator::~~Allocator (C++ *function*), 278

block2::Allocator::allocate (C++ *function*), 278

block2::Allocator::Allocator (C++ *function*), 278

block2::Allocator::complex_allocate (C++ *function*), 278

block2::Allocator::complex_deallocate (C++ *function*), 278

block2::Allocator::copy (C++ *function*), 279

block2::Allocator::deallocate (C++ *function*), 278

block2::Allocator::reallocate (C++ *function*), 279

block2::ArchivedSparseMatrix (C++ *struct*), 288

block2::ArchivedSparseMatrix::allocate (C++ *function*), 289

block2::ArchivedSparseMatrix::ArchivedSparseMatrix (C++ *function*), 289

block2::ArchivedSparseMatrix::deallocate (C++ *function*), 289

block2::ArchivedSparseMatrix::filename (C++ *member*), 290

block2::ArchivedSparseMatrix::get_type (C++ *function*), 289

block2::ArchivedSparseMatrix::load_archive (C++ *function*), 289

block2::ArchivedSparseMatrix::offset (C++ *member*), 290

block2::ArchivedSparseMatrix::save_archive (C++ *function*), 289

block2::ArchivedSparseMatrix::sparse_type (C++ *member*), 290

block2::ArchivedTensorFunctions (C++ *struct*), 290

block2::ArchivedTensorFunctions::archive_tensor (C++ *function*), 290

block2::ArchivedTensorFunctions::ArchivedTensorFunctions (C++ *function*), 290

block2::ArchivedTensorFunctions::filename (C++ *member*), 297

block2::ArchivedTensorFunctions::get_type (C++ *function*), 290

block2::ArchivedTensorFunctions::intermediates (C++ *function*), 295

block2::ArchivedTensorFunctions::left_assign (C++ *function*), 291

block2::ArchivedTensorFunctions::left_contract (C++ *function*), 296

block2::ArchivedTensorFunctions::left_rotate (C++ *function*), 294

block2::ArchivedTensorFunctions::numerical_transform (C++ *function*), 295

block2::ArchivedTensorFunctions::offset (C++ *member*), 297

block2::ArchivedTensorFunctions::right_assign (C++ *function*), 291

block2::ArchivedTensorFunctions::right_contract (C++ *function*), 296

block2::ArchivedTensorFunctions::right_rotate (C++ *function*), 294

block2::ArchivedTensorFunctions::tensor_product (C++ *function*), 294

block2::ArchivedTensorFunctions::tensor_product_diagonal (C++ *function*), 293

block2::ArchivedTensorFunctions::tensor_product_multi_m (C++ *function*), 292

block2::ArchivedTensorFunctions::tensor_product_multipl (C++ *function*), 293

block2::ArchivedTensorFunctions::tensor_product_partial

(C++ *function*), 291
 block2::BasicFFT (C++ *struct*), 312
 block2::BasicFFT::BasicFFT (C++ *function*), 312
 block2::BasicFFT::fft (C++ *function*), 312
 block2::BasicFFT::init (C++ *function*), 312
 block2::BasicFFT::pad (C++ *function*), 313
 block2::BasicFFT::r (C++ *member*), 312
 block2::BasicFFT::wb (C++ *member*), 312
 block2::BasicFFT::wf (C++ *member*), 312
 block2::BasicFFT::xw (C++ *member*), 313
 block2::BasicFFT<2> (C++ *struct*), 313
 block2::BasicFFT<2>::BasicFFT (C++ *function*), 313
 block2::BasicFFT<2>::fft (C++ *function*), 313
 block2::BasicFFT<2>::init (C++ *function*), 313
 block2::BasicFFT<2>::pad (C++ *function*), 314
 block2::BasicFFT<2>::r (C++ *member*), 314
 block2::BasicFFT<2>::wb (C++ *member*), 314
 block2::BasicFFT<2>::wf (C++ *member*), 314
 block2::binary_repr (C++ *function*), 298
 block2::BitsCodec (C++ *struct*), 299
 block2::BitsCodec::begin_decode (C++ *function*), 299
 block2::BitsCodec::BitsCodec (C++ *function*), 299
 block2::BitsCodec::buf (C++ *member*), 300
 block2::BitsCodec::d_offset (C++ *member*), 300
 block2::BitsCodec::decode (C++ *function*), 299
 block2::BitsCodec::encode (C++ *function*), 299
 block2::BitsCodec::finish_encode (C++ *function*), 299
 block2::BitsCodec::i_length (C++ *member*), 300
 block2::BitsCodec::i_offset (C++ *member*), 300
 block2::BitsCodec::op_data (C++ *member*), 300
 block2::BluesteinFFT (C++ *struct*), 315
 block2::BluesteinFFT::arx (C++ *member*), 316
 block2::BluesteinFFT::b (C++ *member*), 317
 block2::BluesteinFFT::BluesteinFFT (C++ *function*), 316
 block2::BluesteinFFT::cb (C++ *member*), 316
 block2::BluesteinFFT::cf (C++ *member*), 316
 block2::BluesteinFFT::fft (C++ *function*), 316
 block2::BluesteinFFT::init (C++ *function*), 316
 block2::BluesteinFFT::nn (C++ *member*), 316
 block2::BluesteinFFT::wb (C++ *member*), 316
 block2::BluesteinFFT::wf (C++ *member*), 316
 block2::check_signal_ (C++ *function*), 288
 block2::CompressedVector (C++ *struct*), 303
 block2::CompressedVector::~CompressedVector (C++ *function*), 304
 block2::CompressedVector::arr_len (C++ *member*), 305
 block2::CompressedVector::cache_data (C++ *member*), 305
 block2::CompressedVector::cache_dirty (C++ *member*), 305
 block2::CompressedVector::chunk_size (C++ *member*), 305
 block2::CompressedVector::clear (C++ *function*), 304
 block2::CompressedVector::CompressedVector (C++ *function*), 303
 block2::CompressedVector::cp_data (C++ *member*), 305
 block2::CompressedVector::finalize (C++ *function*), 304
 block2::CompressedVector::fpc (C++ *member*), 305
 block2::CompressedVector::icache (C++ *member*), 305
 block2::CompressedVector::ncache (C++ *member*), 305
 block2::CompressedVector::operator[] (C++ *function*), 304
 block2::CompressedVector::shrink_to_fit (C++ *function*), 304
 block2::CompressedVector::size (C++ *function*), 304

block2::CompressedVectorMT (C++ struct), 305	block2::DataFrame::load_data (C++ function), 283
block2::CompressedVectorMT::cache_datas (C++ member), 306	block2::DataFrame::load_data_from (C++ function), 283
block2::CompressedVectorMT::CompressedVectorMT (C++ function), 305	block2::DataFrame::memory_used (C++ function), 284
block2::CompressedVectorMT::icaches (C++ member), 306	block2::DataFrame::minimal_disk_usage (C++ member), 287
block2::CompressedVectorMT::operator[] (C++ function), 305, 306	block2::DataFrame::minimal_memory_usage (C++ member), 287
block2::CompressedVectorMT::ref_cv (C++ member), 306	block2::DataFrame::mpo_dir (C++ member), 284
block2::CompressedVectorMT::size (C++ function), 306	block2::DataFrame::mps_dir (C++ member), 284
block2::dalloc_ (C++ function), 282	block2::DataFrame::n_frames (C++ member), 285
block2::DataFrame (C++ struct), 282	block2::DataFrame::operator<< (C++ function), 288
block2::DataFrame::_t (C++ member), 286	block2::DataFrame::partition_can_write (C++ member), 285
block2::DataFrame::_t2 (C++ member), 286	block2::DataFrame::peak_used_memory (C++ member), 286
block2::DataFrame::~DataFrame (C++ function), 283	block2::DataFrame::prefix (C++ member), 285
block2::DataFrame::activate (C++ function), 283	block2::DataFrame::prefix_can_write (C++ member), 285
block2::DataFrame::buffer_save_data (C++ function), 287	block2::DataFrame::prefix_distri (C++ member), 285
block2::DataFrame::dallocs (C++ member), 286	block2::DataFrame::present_filenames (C++ member), 286
block2::DataFrame::DataFrame (C++ function), 282	block2::DataFrame::rename_data (C++ function), 283
block2::DataFrame::deallocate (C++ function), 284	block2::DataFrame::reset (C++ function), 283
block2::DataFrame::dsize (C++ member), 285	block2::DataFrame::reset_buffer (C++ function), 283
block2::DataFrame::fp_codec (C++ member), 287	block2::DataFrame::reset_peak_used_memory (C++ function), 284
block2::DataFrame::fpread (C++ member), 286	block2::DataFrame::restart_dir (C++ member), 284
block2::DataFrame::fpwrite (C++ member), 286	block2::DataFrame::restart_dir_optimal_mps (C++ member), 285
block2::DataFrame::i_frame (C++ member), 285	block2::DataFrame::restart_dir_optimal_mps_per_sweep (C++ member), 285
block2::DataFrame::iallocs (C++ member), 286	block2::DataFrame::restart_dir_per_sweep (C++ member), 285
block2::DataFrame::isize (C++ member), 285	block2::DataFrame::save_buffering (C++ member), 287
block2::DataFrame::load_buffering (C++ member), 287	
block2::DataFrame::load_buffers (C++ member), 286	

block2::DataFrame::save_buffers (C++ member), 286
 block2::DataFrame::save_data (C++ function), 284
 block2::DataFrame::save_data_to (C++ function), 284
 block2::DataFrame::save_dir (C++ member), 284
 block2::DataFrame::save_futures (C++ member), 286
 block2::DataFrame::tasync (C++ member), 286
 block2::DataFrame::tread (C++ member), 285
 block2::DataFrame::twrite (C++ member), 286
 block2::DataFrame::update_peak_used_memory (C++ function), 284
 block2::DataFrame::use_main_stack (C++ member), 287
 block2::DFT (C++ struct), 317
 block2::DFT::DFT (C++ function), 317
 block2::DFT::fft (C++ function), 317
 block2::DFT::init (C++ function), 317
 block2::FactorizedFFT (C++ struct), 317, 318
 block2::FactorizedFFT::cooley_tukey (C++ function), 319
 block2::FactorizedFFT::FactorizedFFT (C++ function), 318, 319
 block2::FactorizedFFT::fft (C++ function), 319
 block2::FactorizedFFT::fft_internal (C++ function), 318, 319
 block2::FactorizedFFT::init (C++ function), 319
 block2::FactorizedFFT::max_factor (C++ member), 320
 block2::FactorizedFFT::prime (C++ member), 320
 block2::FFT (C++ type), 320
 block2::FFT2 (C++ type), 320
 block2::FPCodec (C++ struct), 300
 block2::FPCodec::chunk_size (C++ member), 302
 block2::FPCodec::decode (C++ function), 301
 block2::FPCodec::e (C++ member), 303
 block2::FPCodec::encode (C++ function), 301
 block2::FPCodec::FPCodec (C++ function), 301
 block2::FPCodec::m (C++ member), 303
 block2::FPCodec::n_parallel_chunks (C++ member), 302
 block2::FPCodec::ncpsd (C++ member), 302
 block2::FPCodec::ndata (C++ member), 302
 block2::FPCodec::prec (C++ member), 302
 block2::FPCodec::prec_u (C++ member), 302
 block2::FPCodec::read_array (C++ function), 302
 block2::FPCodec::read_chunks (C++ function), 302
 block2::FPCodec::s (C++ member), 303
 block2::FPCodec::write_array (C++ function), 301
 block2::FPCodec::x (C++ member), 303
 block2::FPtraits (C++ struct), 297
 block2::FPtraits::ebits (C++ member), 297
 block2::FPtraits::mbits (C++ member), 297
 block2::FPtraits::U (C++ type), 297
 block2::FPtraits<double> (C++ struct), 298
 block2::FPtraits<double>::ebits (C++ member), 298
 block2::FPtraits<double>::mbits (C++ member), 298
 block2::FPtraits<double>::U (C++ type), 298
 block2::FPtraits<float> (C++ struct), 297
 block2::FPtraits<float>::ebits (C++ member), 298
 block2::FPtraits<float>::mbits (C++ member), 298
 block2::FPtraits<float>::U (C++ type), 298
 block2::frame_ (C++ function), 288
 block2::ialloc_ (C++ function), 282
 block2::KuhnMunkres (C++ struct), 306
 block2::KuhnMunkres::cost (C++ member), 307
 block2::KuhnMunkres::eps (C++ member), 307
 block2::KuhnMunkres::inf (C++ member), 307
 block2::KuhnMunkres::KuhnMunkres (C++ function), 307
 block2::KuhnMunkres::lx (C++ member), 307
 block2::KuhnMunkres::ly (C++ member), 307
 block2::KuhnMunkres::match (C++ function), 307
 block2::KuhnMunkres::n (C++ member), 307

block2::KuhnMunkres::slack (C++ *member*), 308
 block2::KuhnMunkres::solve (C++ *function*), 307
 block2::KuhnMunkres::st (C++ *member*), 308
 block2::Prime (C++ *struct*), 308
 block2::Prime::euler (C++ *function*), 308
 block2::Prime::exgcd (C++ *function*), 309
 block2::Prime::factors (C++ *function*), 308
 block2::Prime::gcd (C++ *function*), 310
 block2::Prime::init_primes (C++ *function*), 308
 block2::Prime::inv (C++ *function*), 310
 block2::Prime::is_prime (C++ *function*), 308
 block2::Prime::miller_rabin (C++ *function*), 311
 block2::Prime::np (C++ *member*), 309
 block2::Prime::pmod (C++ *function*), 309
 block2::Prime::pollard_rho (C++ *function*), 311
 block2::Prime::power (C++ *function*), 310
 block2::Prime::Prime (C++ *function*), 308
 block2::Prime::primes (C++ *member*), 309
 block2::Prime::primitive_root (C++ *function*), 309
 block2::Prime::primitive_roots (C++ *function*), 309
 block2::Prime::quick_multiply (C++ *function*), 310
 block2::Prime::quick_power (C++ *function*), 311
 block2::Prime::sqrt (C++ *function*), 310
 block2::print_trace (C++ *function*), 288
 block2::RaderFFT (C++ *struct*), 314
 block2::RaderFFT::arx (C++ *member*), 315
 block2::RaderFFT::b (C++ *member*), 315
 block2::RaderFFT::cb (C++ *member*), 315
 block2::RaderFFT::cf (C++ *member*), 315
 block2::RaderFFT::fft (C++ *function*), 315
 block2::RaderFFT::init (C++ *function*), 314
 block2::RaderFFT::nn (C++ *member*), 315
 block2::RaderFFT::prime (C++ *member*), 315
 block2::RaderFFT::RaderFFT (C++ *function*), 314
 block2::RaderFFT::wb (C++ *member*), 315
 block2::RaderFFT::wf (C++ *member*), 315
 block2::SeqTypes (C++ *enum*), 274
 block2::SeqTypes::Auto (C++ *enumerator*), 274
 block2::SeqTypes::None (C++ *enumerator*), 274
 block2::SeqTypes::Simple (C++ *enumerator*), 274
 block2::SeqTypes::SimpleTasked (C++ *enumerator*), 275
 block2::SeqTypes::Tasked (C++ *enumerator*), 274
 block2::StackAllocator (C++ *struct*), 279
 block2::StackAllocator::allocate (C++ *function*), 279
 block2::StackAllocator::data (C++ *member*), 280
 block2::StackAllocator::deallocate (C++ *function*), 279
 block2::StackAllocator::operator<< (C++ *function*), 280
 block2::StackAllocator::reallocate (C++ *function*), 280
 block2::StackAllocator::shift (C++ *member*), 280
 block2::StackAllocator::size (C++ *member*), 280
 block2::StackAllocator::StackAllocator (C++ *function*), 279
 block2::StackAllocator::used (C++ *member*), 280
 block2::Threading (C++ *struct*), 275
 block2::Threading::activate_global (C++ *function*), 276
 block2::Threading::activate_global_mkl (C++ *function*), 276
 block2::Threading::activate_normal (C++ *function*), 276
 block2::Threading::activate_operator (C++ *function*), 276
 block2::Threading::activate_quanta (C++ *function*), 276
 block2::Threading::blis_available (C++ *function*), 275
 block2::Threading::complex_available (C++ *function*), 275
 block2::Threading::get_mkl_threading_type (C++ *function*), 275
 block2::Threading::get_mkl_version (C++ *function*), 275

block2::Threading::get_seq_type (C++ *function*), 275
 block2::Threading::get_thread_id (C++ *function*), 275
 block2::Threading::ksymm_available (C++ *function*), 275
 block2::Threading::mkl_available (C++ *function*), 275
 block2::Threading::n_levels (C++ *member*), 277
 block2::Threading::n_threads_global (C++ *member*), 277
 block2::Threading::n_threads_mkl (C++ *member*), 277
 block2::Threading::n_threads_op (C++ *member*), 277
 block2::Threading::n_threads_quanta (C++ *member*), 277
 block2::Threading::openmp_available (C++ *function*), 275
 block2::Threading::operator<< (C++ *function*), 277
 block2::Threading::seq_type (C++ *member*), 277
 block2::Threading::single_precision_available (C++ *function*), 275
 block2::Threading::tbb_available (C++ *function*), 275
 block2::Threading::Threading (C++ *function*), 276
 block2::Threading::type (C++ *member*), 277
 block2::threading_ (C++ *function*), 277
 block2::ThreadingTypes (C++ *enum*), 273
 block2::ThreadingTypes::BatchedGEMM (C++ *enumerator*), 273
 block2::ThreadingTypes::Global (C++ *enumerator*), 274
 block2::ThreadingTypes::Operator (C++ *enumerator*), 273
 block2::ThreadingTypes::OperatorBatchedGEMM (C++ *enumerator*), 274
 block2::ThreadingTypes::OperatorQuanta (C++ *enumerator*), 274
 block2::ThreadingTypes::OperatorQuantaBatchedGEMM (C++ *enumerator*), 274
 block2::ThreadingTypes::Quanta (C++ *enumerator*), 273
 block2::ThreadingTypes::QuantaBatchedGEMM (C++ *enumerator*), 273
 block2::ThreadingTypes::SequentialGEMM (C++ *enumerator*), 273
 block2::VectorAllocator (C++ *struct*), 281
 block2::VectorAllocator::allocate (C++ *function*), 281
 block2::VectorAllocator::copy (C++ *function*), 281
 block2::VectorAllocator::data (C++ *member*), 282
 block2::VectorAllocator::deallocate (C++ *function*), 281
 block2::VectorAllocator::operator<< (C++ *function*), 282
 block2::VectorAllocator::reallocate (C++ *function*), 281
 block2::VectorAllocator::VectorAllocator (C++ *function*), 281
 I
 ialloc (C *macro*), 282
 T
 threading (C *macro*), 277