block2

Huanchen Zhai

Jun 22, 2022

USER GUIDE

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/

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

CHAPTER

ONE

CONTRIBUTORS

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

CHAPTER

TWO

FEATURES

- **State symmetry**
  - U(1) particle number symmetry
  - SU(2) or U(1) spin symmetry (spatial orbital)
  - No spin symmetry (general spin orbital)
  - Abelian point group symmetry
  - Translational (K point) / Lz symmetry
- **Sweep algorithms (1-site / 2-site / 2-site to 1-site transition)**
  - **Ground-State DMRG**
    - Decomposition types: density matrix / SVD
    - Noise types: wavefunction / density matrix / perturbative
  - **Multi-Target Excited-State DMRG**
    - State-averaged / state-specific
  - MPS compression / addition
  - Expectation
  - **Imaginary / real time evolution**
    - Hermitian / non-Hermitian Hamiltonian
    - Time-step targeting method
    - Time dependent variational principle method
  - Green's function
- Finite-Temperature DMRG (ancilla approach)
- Low-Temperature DMRG (partition function approach)
- **Particle Density Matrix (1-site / 2-site)**
  - 1PDM / 2PDM
  - Transition 1PDM
  - 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 overlap (non-spin-adapted only)
- Determinant/CSF overlap sampling
- **Multi-level parallel DMRG**
  - Parallelism over sites (2-site only)
  - Parallelism over sum of MPOs (non-spin-adapted only)
  - Parallelism over operators (distributed/shared memory)
  - Parallelism over symmetry sectors (shared memory)
  - Parallelism within dense matrix multiplications (MKL)
- DMRG-CASSCF (pyscf interface)
- Stochastic perturbative DMRG
- **Uncontracted dynamic correlation**
  - DMRG Multi-Reference Configuration Interaction (MRCI) of arbitrary order
  - DMRG Multi-Reference Averaged Quadratic Coupled Cluster (AQCC)/ Coupled Pair Functional (ACPF)
  - DMRG NEVPT2/3/. . . , REPT2/3/. . . , MR-LCC, . . .
- **Orbital Reordering**
  - Fiedler
  - Genetic algorithm
- **MPS Transformation**
  - SU2 to SZ mapping
  - Point group mapping
  - Orbital basis rotation

CHAPTER

THREE

USER GUIDE

3.1 Installation

3.1.1 Using `pip`

One can install `block2` using `pip`:

- OpenMP-only version (no MPI dependence)

```
pip install block2
```

- Hybrid openMP/MPI version (requiring openMPI 4.0.x and `mpi4py` based on the same openMPI library installed)

```
pip install block2-mpi
```

- Binary format are prepared via `pip` for python 3.7, 3.8, and 3.9 with macOS (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.

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.

You may need to add the `build` directory to your environment

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

3.1.3 Options

MKL

If `-DUSE_MKL=ON` is not given, `blas` and `lapack` are required (with limited support for multi-threading).

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 5 minutes, need 7 to 10 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 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 `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=...`.

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/4.0.4
module load gcc/9.2.0
```

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

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

Sometimes, the above procedure may still give the `undefined symbol: ompi_mpi_logical8` error. Then it is possible that the `mpi4py` is still linked to the `mpich` (version 3 or lower) library installed in `anaconda`. If this is the case, one should first `conda uninstall mpich` and then `python -m pip -v install`

`--no-binary :all: mpi4py` and if the installation is successful, we can `ldd $(python -c 'from mpi4py import MPI;print(MPI.__file__)')` to check the linkage of the `libmpi.so`. Ideally it should points to the `openmpi/4.0.4` library or any other version 4.0 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.

Supported operating systems and compilers

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

Using `block2` together with other python extensions

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

Using C++ Interpreter cling

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

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

3.2 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`.

3.2.1 Preparation

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

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

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

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

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

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

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

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

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

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

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

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

3.2.2 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 pyscf script:

```
from pyscf import gto, scf, mcscf, dmrgscf
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)
dmrgscf.dryrun(mc)
```

3.2.3 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
```

Note: Note that the integral file C2.CAS.PVDZ.FCIDUMP.ORIG should be in the working direcotry. By default, the orbitals will be reordered using the fiedler method.

Note: 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`.

3.2.4 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
```

3.2.5 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.[?] The number of states (roots) and the weight of each state can be specified using keywords `nroots` and `weights`, respectively. `block2` will then try to find the low energy states within the space of targets formed by all combintaions of the given values of `nelec`, `spin` and `irrep`.

Note: In `StackBlock`, state-averaged calculation can only be done for states with the same `nelec`, `spin` and `irrep`. In `block2`, targetting multiple `nelec`, `spin` or `irrep` may cause the calculation hard to converge to the lowest energy states. Typically, one needs larger `nroots` than the number of states actually needed, to make sure that the low energy states are converged.

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

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

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

nelec 8
spin 0
irrep 1
nroots 2
weights 0.5 0.5

hf_occ integral
schedule default
maxM 500
maxiter 30
```

This will generate the following output:

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

3.2.6 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.[?]

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

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

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

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

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

hf_occ integral
schedule default
maxM 500
maxiter 30
```

This will generate the following output:

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

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

Level Shift Approach

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

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

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

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

hf_occ integral
schedule default
maxM 500
maxiter 30
```

This will generate the following output:

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

Without State-Average

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

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

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

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

nelec 8
spin 0
irrep 1

schedule default
maxM 500
maxiter 30
mps_tags KET1
```

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

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

nelec 8
spin 0
irrep 1

schedule default
maxM 500
maxiter 30
mps_tags KET2

proj_mps_tags KET1
proj_weights 5
```

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

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

nelec 8
spin 0
irrep 1

schedule default
maxM 500
maxiter 30
mps_tags KET3

proj_mps_tags KET1 KET2
proj_weights 5 5
```

And so on.

This will generate the following output:

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

Mixed with State-Average

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

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

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

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

nelec 8
spin 0
irrep 1
nroots 2
weights 0.5 0.5

schedule default
maxM 500
maxiter 30
mps_tags KET

copy_mps
split_states
```

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

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

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

schedule default
maxM 500
maxiter 30
mps_tags EXKET

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

(continues on next page)

(continued from previous page)

```
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
```

3.2.7 n-Particle Reduced Density Matrix

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

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` (in the scratch folder), respectively, starting with `n = 0`. If there is only one root, the files are named `1pdm.npy`, `2pdm.npy`, 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
```

(continues on next page)

(continued from previous page)

```
hf_occ integral
schedule default
maxM 500
maxiter 30

twopdm
```

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

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

where the three components with indicies $[:, p, q, r, s]$ are for $\langle a^\dagger_{p\alpha} a^\dagger_{q\alpha} a_{r\alpha} a_{s\alpha} \rangle$, $\langle a^\dagger_{p\alpha} a^\dagger_{q\beta} a_{r\beta} a_{s\alpha} \rangle$, and $\langle a^\dagger_{p\beta} a^\dagger_{q\beta} a_{r\beta} a_{s\beta} \rangle$, respectively.

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

hf_occ integral
schedule default
maxM 500
maxiter 30

twopdm
```

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)
```

3.2.8 n-Particle Transition Reduced Density Matrix

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

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` (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

```

(continues on next page)

(continued from previous page)

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

hf_occ integral
schedule default
maxM 500
maxiter 30

tran_twopdm
```

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

hf_occ integral
schedule default
maxM 500
maxiter 30

tran_twopdm
```

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

3.2.9 Restart DMRG Energy Calculation

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

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

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

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

The following input file restarts an interrupted calculation:

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

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

fullrestart
```

3.2.10 Load MPS for Density Matrix Calculation

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

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

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

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

restart_twopdm
```

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

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

nelec 8
spin 0
irrep 1
nroots 2
weights 0.5 0.5

hf_occ integral
schedule default
maxM 500
maxiter 30

restart_tran_twopdm
```

Now we explain how to compute 2-particle transition density matrix for bra and ket states belonging to different point

group irreducible representations. We consider the A_{1g} (bra) and B_{3u} (ket) states.

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

```
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
```

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
```

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
mps_tags BRA KET

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

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.[?]

3.2.11 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`.[?] The time cost for diagonal 2pdm is roughly 2 times of the cost of 1pdm.

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

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

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

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

```
>>> import numpy as np
>>> _2pdm = np.load('./nodex/2pdm.npy')
>>> print(_2pdm.shape)
(3, 26, 26, 26, 26)
>>> _e_pqqp = np.load('./nodex/e_pqqp.npy')
>>> _e_pqpq = np.load('./nodex/e_pqpq.npy')
>>> _2pdm_spat = _2pdm[0] + 2 * _2pdm[1] + _2pdm[2]
>>> _2pdm_spat_pqqp = np.einsum('pqqp->pq', _2pdm_spat)
>>> _2pdm_spat_pqpq = np.einsum('pqpq->pq', _2pdm_spat)
>>> print(np.linalg.norm(_e_pqqp - _2pdm_spat_pqqp))
3.28666776770176e-14
>>> print(np.linalg.norm(_e_pqpq - _2pdm_spat_pqpq))
1.6947732597975102e-14
```

3.2.12 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
```

(continues on next page)

(continued from previous page)

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

3.2.13 Sweep Energy Extrapolation

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

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

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

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

nelec 8
spin 0
irrep 1

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

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

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

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

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

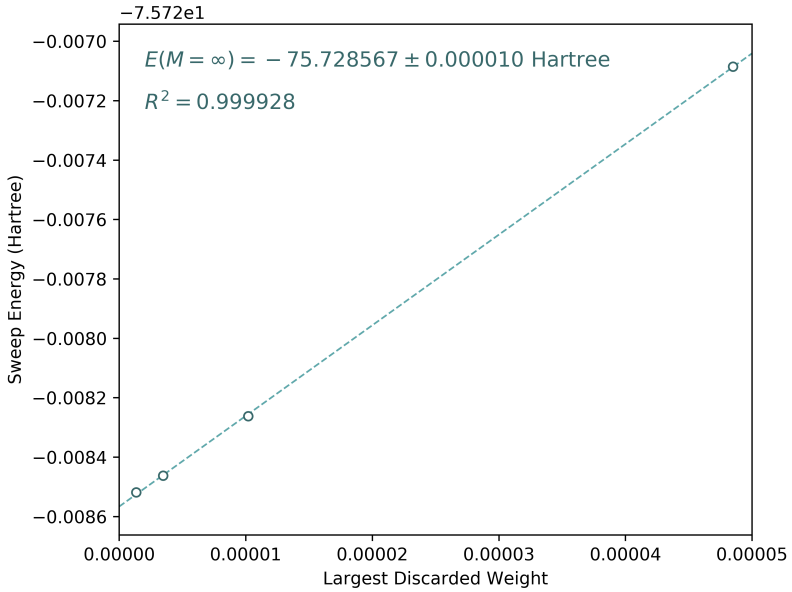
(continues on next page)

(continued from previous page)

```
import scipy.stats

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

3.3 Advanced Usage

3.3.1 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
```

(continues on next page)

(continued from previous page)

```
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
```

(continues on next page)

(continued from previous page)

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

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

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

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

```
sym d2h
orbitals C2.NAT.FCIDUMP

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

mps_tags BRA
restart_oh
restart_onepdm
noreorder
```

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

We have the following output:

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

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

3.3.2 MPS Transform

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

Limitations:

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

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

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

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
mps_tags KET
```

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

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

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
mps_tags KET
```

(continues on next page)

(continued from previous page)

```
restart_copy_mps ZKET
trans_mps_to_sz
```

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

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

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

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
mps_tags ZKET
restart_oh
nonspinadapted
```

Some reference outputs for this example:

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

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

3.3.3 Singlet Embedding

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

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

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

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

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

nelec 8
spin 2
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
mps_tags KET
```

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

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

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

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

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

nelec 8
spin 2
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
```

(continues on next page)

(continued from previous page)

```
mps_tags SEKET
singlet_embedding
```

When singlet embedding is used, the output is:

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

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

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

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

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

nelec 8
spin 2
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
mps_tags SEKET
restart_copy_mps TKET
trans_mps_from_singlet_embedding
```

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

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

nelec 8
spin 2
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
mps_tags TKET
restart_oh
```

With the outputs:

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

(continues on next page)

(continued from previous page)

```
$ grep Energy dmrg-4.out
OH Energy =  -75.423879916245824
```

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

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

nelec 8
spin 2
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

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

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

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

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

nelec 8
spin -2
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30

irrep_reorder
mps_tags ZKETM2
restart_oh
nonspinadapted
```

Some reference outputs for this example:

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

(continues on next page)

(continued from previous page)

```
$ grep Energy dmrg-6.out
OH Energy =  -75.423879916245909
```

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

3.3.4 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 2000000000020200000020000000  =    0.828657540546610
CSF           1 2020000000000200000020000000  =   -0.330323898091116
CSF           2 20+00000000+0200000-000-00  =   -0.140063445607095
CSF           3 20+00000000+0-0-0002000000  =   -0.140041987646036
... ...
CSF          16 200000000002000+0-02000000  =    0.050020205617060
```

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

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

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

3.3.5 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> will be compressed. The tag of the input MPS is given by `read_mps_tags`, and the tag of the output MPS is given by `mps_tags`.

Some reference outputs for this example:

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

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

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

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

for the target MPS. Otherwise, the generated output MPS may be inaccurate.

3.3.6 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))
        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]
```

(continues on next page)

(continued from previous page)

```
    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]
            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.flatten()
_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.flatten()

fcidump_tol = 1E-13
na = nb = mol.nelectron // 2
```

(continues on next page)

(continued from previous page)

```
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)

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

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

fcidump.write('FCIDUMP')
```

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

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

```
sym d2h
orbitals FCIDUMP
k_symmetry
k_irrep 0

nelec 12
spin 0
irrep 1

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

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

Some reference outputs for this input file:

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

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

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

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

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

nelec 12
spin 0
irrep 1

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

Some reference outputs for this input file:

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

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

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

3.3.7 Initial Guess with Occupation Numbers

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

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

```
sym d2h
orbitals FCIDUMP

nelec 12
spin 0
irrep 1
```

(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
onepdm
```

Some reference outputs for this input file:

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

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

```
sym d2h
orbitals FCIDUMP
k_symmetry
k_irrep 0
warmup occ
occ 2.000 2.000 1.957 1.626 1.870 1.870 0.360 0.098 0.098 0.006 0.008 0.008 0.008 0.
↪013 0.014 0.014 0.011 0.006 0.006 0.006 0.005 0.005 0.002 0.002 0.002 0.001 0.001 0.
↪001
cbias 0.2

nelec 12
spin 0
irrep 1

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.

3.3.8 Time Evolution

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

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

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

Real Time Evolution

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

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

hf_occ integral
schedule default
maxM 500
maxiter 30

noreorder
```

Note that the orbital reordering is disabled. The output:

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

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

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

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

hf_occ integral
schedule default
maxM 500
maxiter 30

restart_copy_mps
```

(continues on next page)

(continued from previous page)

```
split_states
trans_mps_to_complex
noreorder
```

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

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

The output looks like :

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

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

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

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

hf_occ integral
schedule
0 500 0 0
end
maxiter 10

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

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

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

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

The tag of the input MPS (old MPS) is given by `read_mps_tags`. The tag of the output MPS (new MPS) is given

by `mps_tags`. The two tags cannot be the same. They should (better) not have common prefix. For example, `KET` and `KET-1` may not be used together, as `-1` may be used by the code internally which will lead to confusion.

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

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

The output :

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

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

We can do the same for the excited state:

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

hf_occ integral
schedule
0 500 0 0
end
maxiter 10

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

The output :

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

The energy is close to the DMRG value -75.63717415.

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

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

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

hf_occ integral
schedule
0 500 0 0
end
maxiter 10

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

The output is:

```
$ grep 'OH' dmrg-5.out
OH Energy    0 -    0 = RE    1.000000000000002 + IM    0.000000000000000
OH Energy    1 -    0 = RE   -0.845792004408687 + IM   -0.533433527528264
OH Energy    1 -    1 = RE    0.999986418355938 + IM    0.000000000000000
OH Energy    2 -    0 = RE   -0.000000000000000 + IM    0.000000000000000
OH Energy    2 -    1 = RE   -0.000000827506956 + IM   -0.000000742303613
OH Energy    2 -    2 = RE    1.000000000000004 + IM    0.000000000000000
OH Energy    3 -    0 = RE    0.000001731091412 + IM   -0.000000316659748
OH Energy    3 -    1 = RE   -0.000001122421894 + IM    0.000002348984005
OH Energy    3 -    2 = RE   -0.836158473098047 + IM   -0.548435696470209
OH Energy    3 -    3 = RE    0.999988252849993 + IM    0.000000000000000
```

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

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

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

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

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

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

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

Which is close to the printed value.

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

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

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

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

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

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

3.4 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.

3.4.1 Global Settings

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

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

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

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

model Optional. Can be used to perform calculations for some simple model Hamiltonian and the `orbitals` keyword can be skipped. For example, `model hubbard 16 1 2` will calculate ground state for 1-dimensional non-periodic Hubbard model with 16 sites and nearest-neighbor interaction, t = 1 and U = 2. `model hubbard_periodic 16 1 2` will do the calculation for the periodic Hubbard model. `model hubbard_kspace 16 1 2` will do the calculation for the periodic Hubbard model in the momentum space.

One can then use this together with `k_symmetry` to utilize the translational symmetry or not use it if the keyword `k_symmetry` is not given. `model hubbard 16 1 2 per-site` will print the energy for each site.

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

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

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

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

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

mem_ratio Optional. Followed by a float number (0.0 ~ 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 >= 80.

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

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

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

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

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

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

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

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

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

the average diagonal of the one-electron integral. This should only be used when the particle number of the calculation is a constant (namely, `nelec` contains only one number).

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

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

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

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

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

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

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

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

3.4.2 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.

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

diag_twopdm / restart_diag_twopdm Diagonal two-particle density matrix calculation.

correlation / restart_correlation Spin and charge correlation function.

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

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

orbital_rotation Orbital rotation of an MPS to generate another MPS.

compression MPS compression.

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

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

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

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

3.4.3 Calculation Modifiers

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

expt_algo_type Optional. Followed by a string `auto`, `fast`, or `lowmem`. Default is `auto`. This keyword can only be used with density matrix or transition density matrix calculations. The default is `auto`. `lowmem` uses less memory, but the complexity can be higher.

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

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

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

complex_mps Optional keyword with no associated value. If given, complex expectation values will be computed for MPS with complex wavefunction tensor and real rotation matrices (in non-complex mode). Should be used

together with `pdm`, `oh`, or (complex) `delta_t` type calculations. In complex mode, this should not be used as everything is complex.

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

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

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

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

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

3.4.4 Uncontracted Dynamic Correlation

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

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

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

mrci *Same as* `mrcisd`.

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

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

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

nevpt2 *Same as* `nevpt2sd`.

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

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

mrrept2 *Same as* `mrrept2sd`.

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

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

3.4.5 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 two-site DMRG algorithm. The sweep iteration is counted from zero.

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

store_wfn_spectra Optional with no associated value. If given, the singular values at each left-right partition during the last 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.

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.

davidson_soft_max_iter Optional. Maximal number of iterations in Davidson. Default is -1. 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. Defualt is 2. This keyword only has effect when used with `delta_t` and when `te_type` is `rk4`.

3.4.6 System Definition

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

3.4.7 Orbital Reordering

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

noreorder The order of orbitals is not changed.

nofiedler *Same as* `noreorder`.

gaopt Genetic algorithm for orbital ordering. Followed by (optionally) the configuration file for the `gaopt` subroutine. Default parameters for the genetic algorithm will be used if no configuration file is given.

fiedler Default. Fiedler orbital reordering.

irrep_reorder Group orbitals with the same irrep together.

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

3.4.8 Unused Keywords

hf_occ integral Optional. For StackBlock compatibility only.

3.5 DMRGSCF

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

3.5.1 Preparation

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

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

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

3.5.2 DMRGSCF (serial)

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

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

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

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

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

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

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

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

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

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

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

This will generate the following output:

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

3.5.3 DMRGSCF (distributed parallel)

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

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

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

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

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

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

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

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

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

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

This will generate the following output:

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

3.5.4 CASSCF Reference

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

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

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

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

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

This will generate the following output:

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

3.5.5 State-Average with Different Spins

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

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

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

This will generate the following output:

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

3.5.6 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

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):
```

(continues on next page)

(continued from previous page)

```
    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)

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

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

    import numpy as np

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

    file2pdm = "2pdm-%d-%d.npy" % (state, state) if DMRGCI.nroots > 1 else "2pdm.npy"
    dm2 = np.load(os.path.join(DMRGCI.scratchDirectory, "node0", file2pdm))
    dm2 = dm2.transpose(0, 1, 4, 2, 3)
```

(continues on next page)

(continued from previous page)

```
    dm1a = np.einsum('ikjj->ki', dm2[0]) / (na - 1)
    dm1b = np.einsum('ikjj->ki', dm2[2]) / (nb - 1)

    return (dm1a, dm1b), dm2

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

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

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

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

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

This will generate the following output:

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

3.5.7 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 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.

3.6.1 Boost Installation

One can download the most recent version of boost in https://www.boost.org/users/download/ Assuming the downloaded file is named boost_1_76_0.tar.gz, stored in ~/program/boost-1.76 (you can choose any other directory). One can then install boost in the following way. Please make sure the correct version of C++ compiler is set in the environment. The same C++ compiler should be used for compiling boost and block2, pyblock and StackBlock.

```
$ mkdir ~/program/boost-1.76
$ cd ~/program/boost-1.76
$ PREFIX=$PWD
$ wget https://boostorg.jfrog.io/artifactory/main/release/1.76.0/source/boost_1_76_0.
↪tar.gz
$ tar zxf boost_1_76_0.tar.gz
$ cd boost_1_76_0
$ gcc --version
gcc (GCC) 9.2.0
$ bash bootstrap.sh
$ 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>/....

3.6.2 StackBlock Installation

The default vresion of `StackBlock` has some bugs for some non-popular features. It is recommended to use this fork, which can be compiled using `cmake`. First, make sure a working MPI library such as `openmpi 4.0` can be found in the system, a C++ compiler with the correct version can be found, and `BOOSTROOT` is set. Then `StackBlock` can be compiled in the following way (starting from the cloned `StackBlock` repo directory):

```
export STACKBLOCK=$PWD
mkdir build
cd build
cmake ..
make -j 10
rm CMakeCache.txt
cmake .. -DBUILD_OH=ON
make -j 10
rm CMakeCache.txt
cmake .. -DBUILD_READROT=ON
make -j 10
rm CMakeCache.txt
cmake .. -DBUILD_GAOPT=ON
make -j 10
```

This will generate four executables in the build direcotry.

`block.spin_adapted` is the main `StackBlock` program. One can optionally add the following to `~/.bashrc`:

```
export PATH=${STACKBLOCK}/build:$PATH
```

`OH` is the program to compute the expectation value on an MPS (or between two MPSs), of Hamiltonian and/or the identity operator.

`read_rot` is the program to translate the intermediate rotation matrix format (from the spin-projected `zmpo_dmrg` code) to the `StackBlock` rotation matrix and wavefunction format.

`gaopt` is the program for orbital reordering using genetic algorithm. Note that for this purpose, the `block2` driver `${BLOCK2HOME}/pyblock2/driver/gaopt.py` should provide much better performance.

3.6.3 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
```

3.6.4 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
maxiter 30
prefix ./tmp
noreorder
```

The following command can be used to run `StackBlock` with this input file:

```
mkdir ./tmp
${STACKBLOCK}/build/block.spin_adapted dmrg.conf > dmrg.out
```

The DMRG ground-state energy can be obtained from the output file:

```
$ grep 'Sweep Energy' dmrg.out | tail -1
M = 500     state = 0     Largest Discarded Weight = 0.000000000000  Sweep Energy = -
↪75.728442606745
```

The energy for the MPS that will be translated is the energy at the last site of the last sweep:

```
$ grep 'sweep energy' dmrg.out | tail -1
Finished Sweep with 500 states and sweep energy for State [ 0 ] with Spin [ 0 ] :: -
↪75.728442606745
```

Since in the default schedule the one-site algorithm is used for the last sweep. This two energies are identical.

Now the MPS in `StackBlock` format is stored in the scratch folder `./tmp/node0`. We will only need files in this folder with file names `Rotation-*`, `StateInfo-*`, `wave-*`. The other files `Block-b-*` and `Block-f-*` (with renormalized operators stored) are not part of the MPS, which can be deleted.

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

```
$ PYTHONPATH=${BLOCK2HOME}/build:$PYTHONPATH
$ PYTHONPATH=${PYBLOCKHOME}:$PYTHONPATH
$ PYTHONPATH=${PYBLOCKHOME}/build:$PYTHONPATH
$ READWFN=${BLOCK2HOME}/pyblock2/driver/readwfn.py
```

(continues on next page)

(continued from previous page)

```
$ python3 $READWFN dmrg.conf -expect
-75.72844260674495
```

Note: Here we use a special build of `block2` python extension, which was built using the `cmake` option `-DTBB=OFF` (the default is `OFF`). On some systems `-DUSE_MKL=OFF -OMP_LIB=SEQ` may be required. This is to solve the conflicts for importing `pyblock` and `block2` in the same script.

Note that `-expect` option is optional. With this option, the energy of the translated MPS will be evaluted in `block2` and printed. We can see that the printed `block2` energy is almost exactly the same as the one obtained from `StackBlock`. By default, the translated `block2` MPS will be put in the output directory named `./out` with the tag `KET`.

3.6.5 Export MPS from block2

Now we show how to translate a `block2` MPS to `StackBlock` MPS.

We will first perform a DMRG ground-state calculation using the following input file `dmrg2.conf`:

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

nelec 8
spin 0
irrep 1

hf_occ integral
schedule default
maxM 500
maxiter 30
prefix ./tmp2
noreorder
```

Note that the only difference between `dmrg.conf` and `dmrg2.conf` is the `prefix`. The following command can be used to run `block2` with this input file:

```
${BLOCK2HOME}/pyblock2/driver/block2main dmrg2.conf > dmrg2.out
```

The energy for the MPS that will be translated is the energy at the last site of the last sweep:

```
$ grep 'DW' dmrg2.out | tail -1
Time elapsed =      3.883 | E =     -75.7284436933 | DE = -3.85e-07 | DW = 3.76e-16
```

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

```
$ PYTHONPATH=${BLOCK2HOME}/build:$PYTHONPATH
$ PYTHONPATH=${PYBLOCKHOME}:$PYTHONPATH
$ PYTHONPATH=${PYBLOCKHOME}/build:$PYTHONPATH
$ WRITEWFN=${BLOCK2HOME}/pyblock2/driver/writewfn.py
$ python3 $WRITEWFN dmrg2.conf -out out2
load MPSInfo from ./../tmp2/KET-mps_info.bin
SRRRRRRRRRRRRRRRRRRRRRRRR -> LLLLLLLLLLLLLLLLLLLLLLLR 24
```

From the print we can see that the canonical form of MPS has been changed, which may cause some small error in the translated MPS. The translated MPS in StackBlock format is now stored in the out2 directory. We can now evaluate the energy of the translated MPS using the OH program in StackBlock:

```
$ sed -i "s|^prefix.*|prefix ./out2|" dmrg2.conf
$ ${STACKBLOCK}/build/OH dmrg2.conf | grep -A 1 'printing hamiltonian' | tail -1
-75.7284436933
```

We can see that the printed StackBlock energy is exactly the same as the one obtained from block2.

Note: The OH program in StackBlock can only evalute the onedot MPS (namely, MPS used in 1-site DMRG algorithm). The MPS can be spin-adapted or non-spin-adapted. If you use the OH in the default standard version of StackBlock, the non-spin-adapted MPS is not supported and you need an extra argument for a file including the MPS ids. For example, you should use /path/to/default/StackBlock/OH dmrg2.conf wavenum where a file named wavenum should be set with contents 0 (or any space-separated list of integers, if you have multiple MPSs).

Alternatively, we can also translate back to block2 and evaluate the energy:

```
$ sed -i "s|^prefix.*|prefix ./out2|" dmrg2.conf
$ READWFN=${BLOCK2HOME}/pyblock2/driver/readwfn.py
$ python3 $READWFN dmrg2.conf -dot 1 -expect -out out3
-75.72844369332921
```

Which also prints the same energy.

3.7 References

A detailed description of the parallel DMRG algorithm implemented in **block2** can be found in the following paper

- Zhai, H., Chan, G. K. Low communication high performance ab initio density matrix renormalization group algorithms. *The Journal of Chemical Physics* 2021, **154**, 224116.

Please cite this paper if you find **block2** useful in your research. 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.

3.7.1 Qauntum Chemisty DMRG

- Chan, G. K.-L.; Head-Gordon, M. Highly correlated calculations with a polynomial cost algorithm: A study of the density matrix renormalization group. *The Journal of Chemical Physics* 2002, **116**, 4462–4476. doi: 10.1063/1.1449459
- Sharma, S.; Chan, G. K.-L. Spin-adapted density matrix renormalization group algorithms for quantum chemistry. *The Journal of Chemical Physics* 2012, **136**, 124121. doi: 10.1063/1.3695642
- Wouters, S.; Van Neck, D. The density matrix renormalization group for ab initio quantum chemistry. *The European Physical Journal D* 2014, **68**, 272. doi: 10.1140/epjd/e2014-50500-1

3.7.2 Parallelization

- Chan, G. K.-L. An algorithm for large scale density matrix renormalization group calculations. *The Journal of Chemical Physics* 2004, **120**, 3172–3178. doi: 10.1063/1.1638734
- Chan, G. K.-L.; Keselman, A.; Nakatani, N.; Li, Z.; White, S. R. Matrix product operators, matrix product states, and ab initio density matrix renormalization group algorithms. *The Journal of Chemical Physics* 2016, **145**, 014102. doi: 10.1063/1.4955108
- Stoudenmire, E.; White, S. R. Real-space parallel density matrix renormalization group. *Physical Review B* 2013, **87**, 155137. doi: 10.1103/PhysRevB.87.155137
- Zhai, H., Chan, G. K. Low communication high performance ab initio density matrix renormalization group algorithms. *The Journal of Chemical Physics* 2021, **154**, 224116. doi: 10.1063/5.0050902

3.7.3 Spin-Orbit Coupling

- Sayfutyarova, E. R., Chan, G. K. L. A state interaction spin-orbit coupling density matrix renormalization group method. *The Journal of Chemical Physics* 2016, **144**, 234301. doi: 10.1063/1.4953445
- Sayfutyarova, E. R., Chan, G. K. L. Electron paramagnetic resonance g-tensors from state interaction spin-orbit coupling density matrix renormalization group. *The Journal of Chemical Physics* 2018, **148**, 184103. doi: 10.1063/1.5020079

3.7.4 Green's Function

- Ronca, E., Li, Z., Jimenez-Hoyos, C. A., Chan, G. K. L. Time-step targeting time-dependent and dynamical density matrix renormalization group algorithms with ab initio Hamiltonians. *Journal of Chemical Theory and Computation* 2017, **13**, 5560-5571. doi: 10.1021/acs.jctc.7b00682

3.7.5 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
- 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

3.7.6 Linear Response

- Sharma, S., Chan, G. K. Communication: A flexible multi-reference perturbation theory by minimizing the Hylleraas functional with matrix product states. *Journal of Chemical Physics* 2014, **141**, 111101. doi: 10.1063/1.4895977

3.7.7 Perturbative Noise

- White, S. R. Density matrix renormalization group algorithms with a single center site. *Physical Review B* 2005, **72**, 180403. doi: 10.1103/PhysRevB.72.180403
- Hubig, C., McCulloch, I. P., Schollwöck, U., Wolf, F. A. Strictly single-site DMRG algorithm with subspace expansion. *Physical Review B* 2015, **91**, 155115. doi: 10.1103/PhysRevB.91.155115

3.7.8 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

3.7.9 Multi-Reference Correlation Theories

- Szalay, P. G.; Müller, T.; Gidofalvi, G.; Lischka, H.; Shepard, R. Multiconfiguration Self-Consistent Field and Multireference Configuration Interaction Methods and Applications. *Chemical Reviews* 2012, **112**, 108-181. doi: 10.1021/cr200137a
- Gdanitz, R. J., Ahlrichs, R. The Averaged Coupled-Pair Functional (ACPF): A Size-Extensive Modification of MR CI(SD). *Chemical Physics Letters* 1988, **143**, 413-420. doi: 10.1016/0009-2614(88)87388-3
- Szalay, P. G., Bartlett, R. J. Multi-Reference Averaged Quadratic Coupled-Cluster Method: A Size-Extensive Modification of Multi-Reference CI. *Chemical Physics Letters* 1993, **214**, 481-488. doi: 10.1016/0009-2614(93)85670-J
- Laidig, W. D.; Bartlett, R. J. A Multi-Reference Coupled-Cluster Method for Molecular Applications. *Chemical Physics Letters* 1984, **104**, 424-430. doi: 10.1016/0009-2614(84)85617-1
- Laidig, W. D., Saxe, P., Bartlett, R. J. The Description of N 2 and F 2 Potential Energy Surfaces Using Multireference Coupled Cluster Theory. *The Journal of Chemical Physics* 1987, **86**, 887-907. doi: 10.1063/1.452291
- Angeli, C., Cimiraglia, R., Evangelisti, S., Leininger, T., Malrieu, J.-P. Introduction of N-Electron Valence States for Multireference Perturbation Theory. *J. Chem. Phys.* 2001, **114**, 10252–10264. doi: 10.1063/1.1361246
- Angeli, C., Pastore, M., Cimiraglia, R. New Perspectives in Multireference Perturbation Theory: The n-Electron Valence State Approach. *Theor Chem Account* 2007, **117**, 743–754. doi: 10.1007/s00214-006-0207-0
- Fink, R. F. The Multi-Reference Retaining the Excitation Degree Perturbation Theory: A Size-Consistent, Unitary Invariant, and Rapidly Convergent Wavefunction Based Ab Initio Approach. *Chemical Physics* 2009, **356**, 39-46. doi: 10.1016/j.chemphys.2008.10.004
- Fink, R. F. Two New Unitary-Invariant and Size-Consistent Perturbation Theoretical Approaches to the Electron Correlation Energy. *Chemical Physics Letters* 2006, **428**, 461–466. doi: 10.1016/j.cplett.2006.07.081
- Sharma, S., Chan, G. K.-L. Communication: A Flexible Multi-Reference Perturbation Theory by Minimizing the Hylleraas Functional with Matrix Product States. *The Journal of Chemical Physics* 2014, **141**, 111101. doi: 10.1063/1.4895977
- Sharma, S., Alavi, A. Multireference Linearized Coupled Cluster Theory for Strongly Correlated Systems Using Matrix Product States. *The Journal of Chemical Physics* 2015, **143**, 102815. doi: 10.1063/1.4928643
- Sharma, S., Jeanmairet, G., Alavi, A. Quasi-Degenerate Perturbation Theory Using Matrix Product States. *The Journal of Chemical Physics* 2016, **144**, 034103. doi: 10.1063/1.4939752
- Larsson, H. R., Zhai, H., Gunst, K., Chan, G. K. L. Matrix product states with large sites. *Journal of Chemical Theory and Computation* 2022, **18**, 749-762. doi: 10.1021/acs.jctc.1c00957

3.7.10 Determinant Coefficients

- Lee, S., Zhai, H., Sharma, S., Umrigar, C. J., Chan, G. K. Externally Corrected CCSD with Renormalized Perturbative Triples (R-ecCCSD (T)) and the Density Matrix Renormalization Group and Selected Configuration Interaction External Sources. *Journal of Chemical Theory and Computation* 2021, **17**, 3414-3425. doi: 10.1021/acs.jctc.1c00205

3.7.11 Perturbative DMRG

- Guo, S., Li, Z., Chan, G. K. L. Communication: An efficient stochastic algorithm for the perturbative density matrix renormalization group in large active spaces. *The Journal of chemical physics* 2018, **148**, 221104. doi: 10.1063/1.5031140
- Guo, S., Li, Z., Chan, G. K. L. A perturbative density matrix renormalization group algorithm for large active spaces. *Journal of chemical theory and computation* 2018, **14**, 4063-4071. doi: 10.1021/acs.jctc.8b00273

3.7.12 Orbital Reordering

- Olivares-Amaya, R.; Hu, W.; Nakatani, N.; Sharma, S.; Yang, J.;Chan, G. K.-L. The ab-initio density matrix renormalization group in practice. *The Journal of Chemical Physics* 2015, **142**, 034102. doi: 10.1063/1.4905329

CHAPTER

FOUR

DEVELOPER GUIDE

4.1 DMRG Options

4.1.1 me->dot

- **Values:** 1 or 2.
- **Meaning:** Select 2-site or 1-site algorithm, unless affected by `last_site_1site`.

4.1.2 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`.

4.1.3 last_site_svd

- **Meaning:** If `true`, for *affected* sites: - Davidson step will be skipped - *and* decomposition method will be changed to SVD - *and* if `noise_type = DensityMatrix`, it will be changed to `Wavefunction`.
- **Default:** `false`.
- **Affected sites:** only affect site `n - 1` for backward sweep.
- **Restriction:** Only active for `me->dot = 1` (or when `me->dot = 2` but `last_site_1site = true`).
- **Accuracy:** If `true`: - Skipping davidson step should not affect accuracy. - Using SVD instead of density matrix may decrease accuracy.
- **Efficiency:** `last_site_svd = true` provides faster speed.
- **Indicator:** When `last_site_svd = true`, the affected site will print `Ndav = 0 E = 0.0`.
- **Requirement:** Need `DMRGSCI`.

4.1.4 last_site_1site

- **Meaning:** If true, for *affected* sites: - In forward sweep, 2-site iteration for sites n - 2 and n - 1 will be changed to 1-site iteration for site n - 1. - In backward sweep, 2-site iteration for sites n - 2 and n - 1 will be changed to 1-site iteration for site n - 1.
- **Default:** false.
- **Affected sites:** only affect site n - 2 and n - 1 for forward and backward sweep.
- **Side effect:** MPS bond between site n - 2 and site n - 1 will not be updated in forward sweep.
- **Restriction:** Only active for me->dot = 2.
- **Accuracy:** If true: - Accuracy decreased because of 1-site algorithm. - Accuracy decreased because of the side effect.
- **Efficiency:** last_site_1site = true provides faster speed.
- **Indicator:** When last_site_1site = true, the affected site will print Site = <n - 1> LAST.
- **Requirement:** Need DMRGSCI.

4.1.5 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.

4.2 MPS Orbital Rotation

In this part we explain how to tranform an MPS with one orbital basis to another orbital basis. For the case when the new basis is the one with natural orbitals, please see *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*.

4.2.1 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
```

(continues on next page)

(continued from previous page)

```
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


# sort orbitals by irrep
def irrep_sort(coeff):
    optimal_reorder = [0, 6, 3, 5, 7, 1, 4, 2]  # d2h
    orb_sym = casci_symm.label_symmetry_(mc, mo_coeff_act).orbsym
    orb_opt = [optimal_reorder[x] for x in orb_sym]
    idx = np.argsort(orb_opt)
    return coeff[:, idx], orb_sym[idx]


# HF orbitals (old basis)
mo_coeff_act = mc.mo_coeff[:, mc.ncore:mc.ncore + mc.ncas].copy()
mo_coeff_act, mo_orb_sym = irrep_sort(mo_coeff_act)

# Symmetrized localized orbitals (new basis)
lmo_coeff_act = mo_coeff_act.copy()
for isym in set(mo_orb_sym):
    mask = np.array(mo_orb_sym) == isym
    lmo_coeff_act[:, mask] = scdm(
        mo_coeff_act[:, mask], mol.intor('cint1e_ovlp_sph'))
```

where `mo_coeff_act` represents the AO to MO coefficients for the old orbitals, and `lmo_coeff_act` represents the AO to MO coefficients for the new orbitals. The two sets of orbitals share the same irrep labels `mo_orb_sym`.

It is not necessary that the orbitals should be sorted according to irrep. But if orbitals with the same irrep are far from each other, the orbital rotation may be likely non-local.

Next, we construct the rotation matrix between the two sets of orbitals:

```
# orbital transform rot[old, new]
orb_rot = np.linalg.pinv(mo_coeff_act) @ lmo_coeff_act
assert np.linalg.norm(orb_rot.T - np.linalg.inv(orb_rot)) < 1E-12
assert np.linalg.norm(lmo_coeff_act - mo_coeff_act @ orb_rot) < 1E-12
```

To make the transformation as local as possible (so that the required MPS bond dimension for time evolution can be lower), we need to do some premutation and flipping of signs in the rotation matrix and consequently the new orbitals:

```
# change det sign and reorder rot within each irrep
def regularize_rot_mat(rot, orb_sym, iprint=False):
    rot = rot.copy()
    for isym in set(orb_sym):
        mask = np.array(orb_sym) == isym
        # orbital matching (reordering within irrep)
        kmidx = scipy.optimize.linear_sum_assignment(
            1 - rot[mask, :][:, mask] ** 2)[1]
        if iprint:
            print("overlap before matching = ", np.sum(
```

(continues on next page)

(continued from previous page)

```
                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
    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.flatten()
    g2e = ao2mo.restore(8, mc.get_h2cas(), mc.ncas)
    h1e[np.abs(h1e) < tol] = 0
```

(continues on next page)

(continued from previous page)

```
    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 = DataFrame(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)
```

(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("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)
```

The following script can be used to transform the ground-state MPS to the new basis:

```
# Hamiltonain for orbital transform
hamil_kappa = HamiltonianQC(vacuum, n_sites, orb_sym, fd_kappa)

# MPO (anti-Hermitian)
mpo_kappa = MPOQC(hamil_kappa, QCTypes.Conventional)
mpo_kappa = SimplifiedMPO(mpo_kappa, AntiHermitianRuleQC(RuleQC()),
    True, True, OpNamesSet((OpNames.R, OpNames.RD)))

# Time Step
dt = 0.05
# Target time
tt = 1.0
n_steps = int(abs(tt) / abs(dt) + 0.1)
assert np.abs(abs(n_steps * dt) - abs(tt)) < 1E-10
print("Time Evolution NSTEPS = %d" % n_steps)
me_kappa = MovingEnvironment(mpo_kappa, mps, mps, "DMRG")
me_kappa.delayed_contraction = OpNamesSet.normal_ops()
me_kappa.cached_contraction = True
me_kappa.init_environments(True)

# Time Evolution (anti-Hermitian)
# te_type can be TETypes.RK4 or TETypes.TangentSpace (TDVP)
```

(continues on next page)

(continued from previous page)

```
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" %
            ((i + 1) * dt, te.energies[-1], te.normsqs[-1]))
```

Note that when constructing MPO, `AntiHermitianRuleQC` has to be used. Also `te.hermitian` must be set to `False` for anti-Hermitian "Hamiltonian", otherwise it will be assumed Hermitian.

Note: `TimeEvolution` can support both one-site and two-site algorithm, but we highly recommend the two-site algorithm as there is no noise, and the one-site algorithm may have severe problem with losing quantum numbers.

Since every step in time evolution is a unitary transform, the "energy" expectation should always be zero, and the "norm" of the MPS should be close to one. Normally, a too large discarded weight or "norm" far from 1 indicates that the error during the transform is too large.

Finally, we can check the energy expectation of the transformed MPS in the new basis:

```
# Hamiltonain in new basis
hamil_new = HamiltonianQC(vacuum, n_sites, orb_sym, fd_new)

# MPO
mpo_new = MPOQC(hamil_new, QCTypes.Conventional)
mpo_new = SimplifiedMPO(mpo_new, RuleQC(), True, True, OpNamesSet((OpNames.R, OpNames.
↪RD)))

# Energy Expectation
me_new = MovingEnvironment(mpo_new, mps, mps, "OVL")
me_new.delayed_contraction = OpNamesSet.normal_ops()
me_new.cached_contraction = True
me_new.init_environments(True)

expect = Expect(me_new, mps.info.bond_dim, mps.info.bond_dim)
ener_new = expect.solve(False, mps.center == 0)

print('Energy expectation = %20.15f' % ener_new)
```

Some reference outputs for this example:

```
D2H ORB SYM =  VectorUInt8[ 0 0 0 0 0 0 5 5 5 5 5 5 7 7 7 2 2 2 6 6 6 3 3 3 1 4 ]
DMRG Energy =  -75.728487321653233
Time Evolution NSTEPS = 20
T =    0.05000 <E> =   -0.000000000000000 <Norm^2> =    0.999999979398520
T =    0.10000 <E> =   -0.000000000000000 <Norm^2> =    0.999999926838107
... ...
T =    0.95000 <E> =    0.000000000000000 <Norm^2> =    0.999996763879923
Time elapsed =      5.738 | E =       0.0000000000 | Norm^2 =       0.9999964412 | DW␣
↪= 3.83e-08
```

(continues on next page)

(continued from previous page)

```
T =    1.00000 <E> =    0.000000000000000 <Norm^2> =    0.999996441150652
Energy expectation =  -75.728011987963555
```

4.2.2 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)
```

4.2.3 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 = DataFrame(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)
    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:
```

(continues on next page)

(continued from previous page)

```
            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")
me.delayed_contraction = OpNamesSet.normal_ops()
me.cached_contraction = True
me.init_environments(True)
dmrg = DMRG(me, VectorUBond([250, 500]), VectorDouble([1E-5] * 5 + [1E-6] * 5 + [0]))
dmrg.noise_type = NoiseTypes.ReducedPerturbativeCollected
dmrg.davidson_conv_thrds = VectorDouble([1E-6] * 5 + [1E-7] * 5)
ener = dmrg.solve(20, mps.center == 0, 1E-8)
print('MRCI DMRG Energy = %20.15f' % ener)


# Hamiltonain for orbital transform
hamil_kappa = HamiltonianQC(vacuum, n_sites, orb_sym, fd_kappa)

# MPO (anti-Hermitian)
mpo_kappa = MPOQC(hamil_kappa, QCTypes.Conventional)
mpo_kappa = create_big_site(hamil_kappa, mpo_kappa)
mpo_kappa = SimplifiedMPO(mpo_kappa, AntiHermitianRuleQC(RuleQC()), True, True,␣
↪OpNamesSet((OpNames.R, OpNames.RD)))

# Time Step
dt = 0.05
# Target time
```

(continues on next page)

(continued from previous page)

```
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" %
            ((i + 1) * dt, te.energies[-1], te.normsqs[-1]))


# Hamiltonain in new basis
hamil_new = HamiltonianQC(vacuum, n_sites, orb_sym, fd_new)

# MPO
mpo_new = MPOQC(hamil_new, QCTypes.Conventional)
mpo_new = create_big_site(hamil_new, mpo_new)
mpo_new = SimplifiedMPO(mpo_new, RuleQC(), True, True, OpNamesSet((OpNames.R, OpNames.
↪RD)))

# Energy Expectation
me_new = MovingEnvironment(mpo_new, mps, mps, "OVL")
me_new.delayed_contraction = OpNamesSet.normal_ops()
me_new.cached_contraction = True
me_new.init_environments(True)

expect = Expect(me_new, mps.info.bond_dim, mps.info.bond_dim)
ener_new = expect.solve(False, mps.center == 0)

print('Energy expectation = %20.15f' % ener_new)
```

where the big-site MPO is created using the function `create_big_site`, where the right-boundary sites in the MPO are folded to a big site using the `FusedMPO` class. Other more efficient methods for creating a big site can be used, but note that, the big site in the three MPOs `mpo`, `mpo_kappa`, and `mpo_new` must be created using the same method. This is to ensure that the quantum number fusing order is consistent among different MPOs. This is required because the same MPS is used with all these MPOs.

Also note that `SeqTypes.Nothing` (instead of `SeqTypes.Tasked`) should be used for big-site with CSR matrices.

Some reference outputs for this example:

```
D2H ORB SYM =  VectorUInt8[ 0 0 0 0 0 0 5 5 5 5 5 5 7 7 7 2 2 2 6 6 6 3 3 3 1 4 ]
MRCI DMRG Energy =  -75.727859086194130
Time Evolution NSTEPS = 20
T =    0.05000 <E> =    0.000000000000000 <Norm^2> =    0.999999980443349
T =    0.10000 <E> =   -0.000000000000000 <Norm^2> =    0.999999930992521
... ...
T =    0.95000 <E> =    0.000000000000000 <Norm^2> =    0.999996944337650
Time elapsed =      6.035 | E =      -0.0000000000 | Norm^2 =       0.9999966399 | DW␣
↪= 3.84e-08
T =    1.00000 <E> =   -0.000000000000000 <Norm^2> =    0.999996639941846
Energy expectation =  -75.727409014459965
```

4.3 Point Group Mapping

Here we discuss how to transform an MPS with a point group (PG) (such as D_{2h}) to an MPS with another PG (such as C_{2v} or C_s).

Limitations:

- Can transform from high-order PG (ket) to low-order PG (bra) or low-order PG (ket) to high-order PG (bra);
- The mapping between the two PG must be a homomorphism. As long as it is a homomorphism, any mapping can be used.
- For normal MPS, the transformation only have a tiny fitting error. For MPS with big site, the matrix elements in the big-site tensor in the MPS will be artificially reordered. (Because the "fusing order" inside the big site can have an influence on the order of states within each symmetry block. Since "fusing order" in the big site can be arbitrary, the mapping code itself cannot figure out the correct mapping for the order of states within each symmetry block. For normal site, there is only one state for each symmetry block so there is no problem.) So the transformed big-site tensor will not be accurate (but the normal site tensors in a big-site MPS will still be accurate).
- The integral (FCIDUMP) with the two PG must be exactly the same. Consider the following case: if you generate the integral from `pyscf`, you calculate one integral with `D2h` symmetry in the molecule, and another integral with `C2v` symmetry in the molecule. Then there can be small (or big) changes in the integral. Then you cannot use this feature, since point group symmetry is not the only thing that changed.

4.3.1 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 = DataFrame(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)
```

(continues on next page)

(continued from previous page)

```
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)
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
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.
```

(continues on next page)

(continued from previous page)

Some reference output for this example:

```
D2H ORB SYM =  VectorUInt8[ 5 0 6 5 3 5 0 0 5 0 3 6 5 0 3 6 7 2 7 2 7 2 1 4 0 5 ]
<-- Site =    0-   1 .. Mmps =    3 Ndav =   1 E =    -75.7284493902 Error = 1.14e-16␣
↪FLOPS = 8.66e+05 Tdav = 0.00 T = 0.01
DMRG Energy =  -75.728475543752168
C2V ORB SYM =  VectorUInt8[ 2 0 3 2 1 2 0 0 2 0 1 3 2 0 1 3 3 1 3 1 3 1 0 2 0 2 ]
Norm =    1.000000000000001
C2V Energy =  -75.728449390238850
```

4.3.2 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 = DataFrame(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')

# 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)
```

(continues on next page)

(continued from previous page)

```
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
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)))
```

(continues on next page)

(continued from previous page)

```
# D2H Expectation
me = MovingEnvironment(mpo_d2h, mps_d2h, mps_d2h, "DMRG")
me.delayed_contraction = OpNamesSet.normal_ops()
me.cached_contraction = True
me.init_environments(True)
ex = Expect(me, 500, 500)
ener_d2h = ex.solve(False) / norm ** 2
print('D2H Energy = %20.15f' % ener_d2h)
```

Some reference outputs for this example:

```
C2V ORB SYM =  VectorUInt8[ 2 0 3 2 1 2 0 0 2 0 1 3 2 0 1 3 3 1 3 1 3 1 0 2 0 2 ]
--> Site =   24-  25 .. Mmps =    3 Ndav =   1 E =    -75.7284490538 Error = 1.62e-19␣
↪FLOPS = 3.87e+05 Tdav = 0.00 T = 0.01
DMRG Energy =  -75.728475021520978
D2H ORB SYM =  VectorUInt8[ 5 0 6 5 3 5 0 0 5 0 3 6 5 0 3 6 7 2 7 2 7 2 1 4 0 5 ]
Norm =    0.999999999998821
D2H Energy =  -75.728449053829152
```

4.3.3 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.

4.4 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.

4.4.1 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 = DataFrame(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)
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
```

(continues on next page)

(continued from previous page)

```
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

Global.frame = DataFrame(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'
```

(continues on next page)

(continued from previous page)

```
mps_info.set_bond_dimension(250)
mps = MPS(n_sites, 0, 2)
mps.initialize(mps_info)
mps.random_canonicalize()
mps.save_mutable()
mps_info.save_mutable()

# DMRG
me = MovingEnvironment(mpo, mps, mps, "DMRG")
me.delayed_contraction = OpNamesSet.normal_ops()
me.cached_contraction = True
me.init_environments(True)
dmrg = DMRG(me, VectorUBond([250, 500]), VectorDouble([1E-5] * 5 + [1E-6] * 5 + [0]))
dmrg.noise_type = NoiseTypes.ReducedPerturbativeCollected
dmrg.davidson_conv_thrds = VectorDouble([1E-6] * 5 + [1E-7] * 5)
ener = dmrg.solve(20, mps.center == 0, 1E-8)
print('DMRG Energy = %20.15f' % ener)
```

Some reference outputs (the memory information can be different for each run):

```
$ grep 'usage\|Energy' dmrg-2.out
 pre-load-mpo memory usage =    42.6 MB
post-load-mpo memory usage =    53.5 MB
DMRG Energy =  -75.728475321395166
```

So the reloaded MPO object is smaller, which needs only 10.9 MB memory. The DMRG takes 70.581 seconds.

Loading a Serial MPO with Minimal Memory

One can change the line in the above script:

```
mpo.load_data('mpo.bin')
```

to:

```
mpo.load_data('mpo.bin', minimal=True)
```

Then rerun the script. Now the MPO is loaded in the minimal memory mode.

Some reference outputs (the memory information can be different for each run):

```
$ grep 'usage\|Energy' dmrg-2.out
 pre-load-mpo memory usage =    40.7 MB
post-load-mpo memory usage =    43.0 MB
DMRG Energy =  -75.728475329694518
```

Now the reloaded MPO object occupies only 2.3 MB memory before the DMRG calculation. The DMRG takes 70.688 seconds (which is not greatly affected by dynamically reloading MPO parts).

Saving Parallelized MPO

For distributed calculations, we can still reload the serial MPO and parallelize it. But this way is only compatible to the non-minimal-memory mode. To save the memory for distributed calculations, we need to save the parallelized MPO. The parallelization script for MPO does not have to be run in parallel (but you still can run it in parallel, which has a lower wall time cost but a higher memory cost).

The following script generates and saves the parallelized MPO for 7 mpi processsors (note that this script should be run in serial, namely, no `mpirun`):

```
from block2 import *
from block2.su2 import *
import numpy as np
import psutil
import os

Global.frame = DataFrame(10 * 1024 ** 2, 10 * 1024 ** 3, "nodex")

mpo = MPO(0)
mpo.load_data('mpo.bin')

# size, rank, root
comm = ParallelCommunicator(7, 0, 0)
prule = ParallelRuleQC(comm)

for irank in range(comm.size):
    comm.rank = irank
    para_mpo = ParallelMPO(mpo, prule)
    para_mpo.save_data('mpo.bin.%d' % irank)
    fsize = os.path.getsize('mpo.bin.%d' % irank)
    print("mpo.%d size = %10s" % (irank, Parsing.to_size_string(fsize)))
```

Here we assume a serial MPO `mpo.bin` has already been saved in the disk. The `ParallelCommunicator` is a fake object for distributed parallelism. We can manually change the `rank` of `ParallelCommunicator` to generate parallelized MPOs for different ranks.

Some reference outputs:

```
mpo.0 size =    2.74 MB
mpo.1 size =    2.75 MB
mpo.2 size =    2.73 MB
mpo.3 size =    2.74 MB
mpo.4 size =    2.77 MB
mpo.5 size =    2.78 MB
mpo.6 size =    2.77 MB
```

Note that each parallelized MPO is larger than the serial MPO. Actually, each of them includes both the "local" part and "global" part. The "global" part then has the same size as the serial MPO. (For big site code the "global" part for parallelized MPO can be smaller than the full MPO).

Reloading Parallelized MPO

The following script is used for parallel DMRG with 7 mpi processsors (namely, `mpirun -n 7 --bind-to none python -u dmrg.py`, for example):

```
from block2 import *
from block2.su2 import *
import numpy as np
import psutil
import os
SX = SU2

MPI = MPICommunicator()

Global.frame = DataFrame(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
dmrg.davidson_conv_thrds = VectorDouble([1E-6] * 5 + [1E-7] * 5)
```

(continues on next page)

(continued from previous page)

```
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
MRG Energy =  -75.728475151371001
MRG Energy =  -75.728475151371001
MRG Energy =  -75.728475151371001
MRG Energy =  -75.728475151371001
MRG Energy =  -75.728475151371001
MRG Energy =  -75.728475151371001
MRG Energy =  -75.728475151371001
```

We can see that the memory usage after loading MPO is smaller, compared to the non-minimal-memory-usage mode.

4.5 Debugging Hints

Here we list some of common assertion failure, errors, wrong outputs, and the solutions.

4.5.1 Ground State Calculation

[2021-05-09]

Assertion:

```
block2/parallel_mpi.hpp:330: void block2::MPICommunicator<S>::reduce_sum(double*,␣
↪size_t, int) [with S = block2::SU2Long; size_t = long unsigned int]: Assertion␣
↪`ierr == 0' failed.
```

Conditions: More than one MPI processors, `QCTypes.Conventional`, `Random.rand_seed(0)`, and `gaopt`. Random assertion failure.

Reason: A different gaopt reordering was used in different mpi processors. Then the error happens during the initialization of environments. Then there will be an array-size mismatching due to the difference in integrals.

Solution: Broadcast the orbital reordering indices before reordering the integral.

[2021-05-10]

Assertion:

```
block2/operator_functions.hpp:185: void block2::OperatorFunctions<S>::tensor_
↪rotate(const std::shared_ptr<block2::SparseMatrix<S> >&, const std::shared_ptr
↪<block2::SparseMatrix<S> >&, const std::shared_ptr<block2::SparseMatrix<S> >&,␣
↪const std::shared_ptr<block2::SparseMatrix<S> >&, bool, double) const [with S =␣
↪block2::SZLong]: Assertion `a->get_type() == SparseMatrixTypes::Normal && c->get_
↪type() == SparseMatrixTypes::Normal && rot_bra->get_type() ==␣
↪SparseMatrixTypes::Normal && rot_ket->get_type() == SparseMatrixTypes::Normal'␣
↪failed.
```

Conditions: Loaded MPO, CSR.

Reason: The non-CSR `OperatorFunctions` is used for calculation requiring CSR matrices, after loading MPO.

Solution: Change `csr_opf = OperatorFunctions(cg)` to `csr_opf = CSROperatorFunctions(cg)`.

[2021-05-11]

Output:

```
Sweep =   15 | Direction = backward | Bond dimension = 2000 | Noise =  1.00e-07 | Dav␣
↪threshold =  1.00e-08
<-- Site =   11-  12 .. Mmps =    3 Ndav =   1 E =    -36.8356589402 Error = 0.00e+00␣
↪FLOPS = 4.12e+06 Tdav = 0.02 T = 0.17
<-- Site =   10-  11 .. Mmps =   10 Ndav =   1 E =    -36.8356589402 Error = 0.00e+00␣
↪FLOPS = 2.91e+08 Tdav = 0.02 T = 0.18
<-- Site =    9-  10 .. Mmps =   35 Ndav =   1 E =    -36.8356589402 Error = 0.00e+00␣
↪FLOPS = 9.70e+09 Tdav = 0.02 T = 0.20
```

(continues on next page)

(continued from previous page)

```
<-- 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

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
```

(continues on next page)

(continued from previous page)

```
| Tread = 0.550 | Twrite = 0.624 | Tfpread = 0.504 | Tfpwrite = 0.092 | Tasync = 0.000
| Trot = 0.385 | Tctr = 0.039 | Tint = 0.023 | Tmid = 2.480 | Tdctr = 0.052 | Tdiag =␣
↪0.323 | Tinfo = 0.035
| Teff = 1.734 | Tprt = 2.656 | Teig = 12.197 | Tblk = 19.563 | Tmve = 3.667 | Tdm =␣
↪0.000 | Tsplt = 0.000 | Tsvd = 1.508
```

Conditions: More than one MPI processors, and `QCTypes.Conventional`.

Reason: We see from the output that the energy jumps between two values even in very large bond dimension. If only one MPI is used, there is no such behavior. This is because the input integrals `h1e` and `g2e` are not synchronized. In `QCTypes.Conventional`, communication between MPI procs only happens at the middle site. After this communication, the inconsistentcy between integrals can cause an artificial change in energy. Note that inside `block2`, we do not explicitly synchronize integral. In future, for larger systems, the integral can even be distributed, such that synchronization will not be meaningful.

Solution: Synchronizing the input integrals `h1e` and `g2e` can solve this problem.

[2021-05-12 | 2021-06-07]

Error Message: (note that this problem in `block2main` has been fixed in commit 4f87784)

```
Traceback (most recent call last):
File "block2/pyblock2/driver/block2main", line 302, in <module>
    mps.load_data()
RuntimeError: MPS::load_data on '/central/scratch/.../F.MPS.KET.-1' failed.
```

or

```
Traceback (most recent call last):
File "block2/pyblock2/driver/block2main", line 313, in <module>
    mps.load_mutable()
RuntimeError: SparseMatrix:load_data on '/central/scratch/.../F.MPS.KET.14' failed.
```

or

```
Traceback (most recent call last):
File "block2/pyblock2/driver/block2main", line 313, in <module>
    mps.load_mutable()
ValueError: cannot create std::vector larger than max_size()
```

Conditions: More than one MPI processors, python driver, happening with a very low probability.

Reason A: The problematic code is:

```
mps.load_data()
if mps.dot != dot and nroots == 1:
    mps.dot = dot
    mps.save_data()
```

And the non-root MPI proc can load data before or after the root proc saves the data. The wrong loaded data can cause the subsequent `mps.load_mutable()` to fail.

Solution A: Adding `MPI.barrier()` around `mps.save_data()`.

Reason B: The problematic code is:

```
mps.load_mutable()
mps.save_mutable()
```

And the non-root MPI proc can load data before or after the root proc saves the data. This may cause simultaneously reading and writing into the same file (with a very low probability).

Solution B: Adding `MPI.barrier()` between `mps.load_mutable()` and `mps.save_mutable()`.

4.5.2 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.

4.5.3 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)
```

4.6 Notes

4.6.1 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==3.17 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
```

4.6.2 Installing `block2` using python virtual environment

This guide shows how one can manually build `block2` with openmpi without using Anaconda and Intel oneapi.

First we assume a suitable python3, openmpi library, and gcc compiler can be found in the system. For example, I have

```
$ which gcc
/opt/gcc/11.2.0/bin/gcc
$ which mpirun
/opt/openmpi/4.1.2/gnu/bin/mpirun
$ which python3
```

First we install the python virtualenv

```
$ python3 -m pip install --user --upgrade pip
$ python3 -m pip install --user virtualenv
```

Then we create a virtualenv called `base` and activate it, which will create a folder called `base` in the current folder

```
$ python3 -m venv base
$ source base/bin/activate
```

Then we install necessary packages in this virtualenv

```
$ pip install mkl mkl-include pybind11 numpy scipy psutil
$ pip install mpi4py --no-binary mpi4py
```

Then we can build block as the following

```
$ mkdir build
$ cd build
$ export MKLROOT=/path/to/base
$ export PATH=/path/to/base:$PATH
$ cmake .. -DUSE_MKL=ON -DBUILD_LIB=ON -DMPI=ON -DLARGE_BOND=ON -DTBB=ON -DUSE_
↪DMRG=ON -DUSE_BIG_SITE=ON -DUSE_SP_DMRG=ON -DUSE_IC=ON -DUSE_KSYMM=ON -DUSE_
↪COMPLEX=ON
```

We can test that `mpi4py` is working

```
$ mpirun -n 2 python -c 'from mpi4py import MPI;print(MPI.COMM_WORLD.rank)'
```

CHAPTER

FIVE

API REFERENCE

5.1 Global Settings

In **block2**, we try to minimize the use of global variables. Two global variables 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.

Warning: doxygendefine: Cannot find define "frame" in doxygen xml output for project "block2" from directory: ../build/doxygenxml/

threading
: Global variable containing information for shared-memory parallelism schemes and number of threads used for each threading layer.

5.1.1 Threading

enum `block2::`**ThreadingTypes**
: 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 `block2::SeqTypes`
Method of GEMM (dense matrix multiplication) parallelism. For CSR matrix multiplication, the only possbile case is `SeqTypes::None`, but one can still use `SeqTypes::Simple` and it will only parallelize dense matrix multiplication.

Values:

enumerator None
GEMM are not parallelized. Parallelism may happen inside each GEMM, if a threaded version of MKL is linked.

enumerator Simple
GEMM written to the different outputs are parallelized, otherwise they are executed in sequential. With this mode, the code will sort and divide GEMM to several groups (batches). Inside each batch, the output addresses are guarenteed to be different. The `cblas_dgemm_batch` is invoked to compute each batch.

enumerator Auto
DGEMM automatically divided into several batches only when there are data dependency. Conflicts of output are automatically resolved by introducing temporary arrays. The `cblas_dgemm_batch` is invoked to compute each batch. This option normally requires a large amount of time for preprocessing and it will introduce a large number of temporary arrays, which is not memory friendly.

enumerator Tasked
GEMM will be evenly divided into `n_threads` groups, Different groups are executed in different threads. Since different threads may write into the same output array, there is an additional reduction step after all GEMM finishes. This mode is mainly implemented for Davidson matrix-vector step (`tensor_product_multiply`), where the size of the output array (wavefunction) is small compared to that of all input arrays. For blocking/rotation step, `SeqTypes::Tasked` has no effect and it is equivalent to `SeqTypes::None`. The `cblas_dgemm_batch` is not used in this mode.

enumerator SimpleTasked
This is the same as `SeqTypes::Tasked` for the Davidson matrix-vector step, and the same as `SeqTypes::Simple` for other steps.

struct `block2::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 **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 friend ostream & operator<< (ostream &os, const Threading &th)
Print threading information.

inline shared_ptr<*Threading*> &`block2::threading_()`
Implementation of the `threading` global variable.

5.1.2 Allocators

template<typename **T**>
struct `block2::Allocator`
Abstract memory allocator.

tparam 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 block2::**StackAllocator** : **public** block2::*Allocator*<*T*>
Stack memory allocator.

tparam T The type of the element in the array.

Public Functions

inline StackAllocator(*T* *ptr, size_t *max_size*)
Constructor.

Parameters

- **ptr** – Pointer to the first elemenet in the stack. The stack should be pre-allocated.
- **max_size** – Total size of the stack (in number of elements).

inline StackAllocator()
Default constructor.

inline virtual *T* ***allocate**(size_t *n*) **override**
Allocate a length n array.

Parameters **n** – Number of elements in the array.

Returns The allocated pointer.

inline virtual void **deallocate**(void **ptr*, size_t *n*) **override**
Deallocate a length n array. Must be invoked in the reverse order of allocation.

Parameters

- **ptr** – The pointer to be deallocated.
- **n** – Number of elements in the array.

inline virtual *T* ***reallocate**(*T* **ptr*, size_t *n*, size_t *new_n*) **override**
Change the allocated size in middle of stack memory and introduce a shift for moving memory after it.

Parameters

- **ptr** – The allocated pointer.
- **n** – Number of elements in original allocation.
- **new_n** – Number of elements in the new allocation.

Returns The new pointer.

Public Members

size_t **size**
Total size of the stack (in number of elements).

size_t **used**
Occupied size of the stack (in number of elements).

size_t **shift**
Temporary shift introduced due to deallocation in the middle of the stack.

T ***data**
Pointer to the first elemenet in the stack.

Friends

inline friend 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 block2::**VectorAllocator** : **public** block2::*Allocator*<*T*>
Vector memory allocator.

tparam 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 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.

Warning: doxygendefine: Cannot find define "dalloc" in doxygen xml output for project "block2" from directory: ../build/doxygenxml/

5.1.3 Data Frame

template<typename **FL**>
struct block2::**DataFrame**
DataFrame includes several (n_frames = 2) frames. Each frame includes one integer stack memory and one double stack memory. The two frames are used alternatively to avoid data copying.

Public Functions

inline DataFrame(size_t *isize* = 1 << 28, size_t *dsize* = 1 << 30, **const** string &*save_dir* = "node0", double *dmain_ratio* = 0.7, double *imain_ratio* = 0.7, int *n_frames* = 2)
Constructor.

Parameters

- **isize** – Max size (in bytes) of all integer stacks.
- **dsize** – Max size (in bytes) of all double stacks.
- **save_dir** – Scartch folder for renormalized operators.
- **dmain_ratio** – The fraction of stack space occupied by the main double stacks.
- **imain_ratio** – The fraction of stack space occupied by the main integer stacks.
- **n_frames** – Number of data frames.

inline ~DataFrame()
Destructor.

inline void **activate**(int *i*)
Activate one data frame.

Parameters **i** – The index of the data frame to be activated.

inline void **reset**(int *i*)
Reset one data frame, marking all stack memory as unused.

Parameters **i** – The index of the data frame to be reset.

inline void **reset_buffer**(int *i*)
Reset saving and loading buffers for one data frame. Contents in the loading buffer will be deleted. Unsaved contents in the saving buffer will be saved in disk.

Parameters **i** – The index of the data frame.

inline void **rename_data**(**const** string &*old_filename*, **const** string &*new_filename*) **const**
Rename one scratch file.

Parameters

- **old_filename** – original filename.
- **new_filename** – new filename.

inline void **load_data_from**(int *i*, istream &*ifs*) **const**
Load one data frame from input stream.

Parameters

- **i** – The index of the data frame.
- **ifs** – The input stream.

inline void **load_data**(int *i*, **const** string &*filename*) **const**
Load one data frame from disk.

Parameters

- **i** – The index of the data frame.
- **filename** – The filename for the data frame.

inline void **save_data_to**(int *i*, ostream &*ofs*) **const**
Save one data frame into output stream.

Parameters

- **i** – The index of the data frame.
- **ofs** – The output stream.

inline void **save_data**(int *i*, **const** string &*filename*) **const**
Save one data frame to disk.

Parameters

- **i** – The index of the data frame.
- **filename** – The filename for the data frame.

inline void **deallocate**()
Deallocate the memory allocated for all stacks. Note that this method is automatically invoked at deconstruction.

inline size_t **memory_used**() **const**
Return the current used memory in all stacks.

Returns The current used memory in Bytes.

inline void **update_peak_used_memory**() **const**
Update prak used memory statistics.

inline void **reset_peak_used_memory**() **const**
Reset prak used memory statistics to zero.

Public Members

string **save_dir**
Scartch folder for renormalized operators.

string **mps_dir**
Scartch folder for MPS (default is the same as save_dir).

string **mpo_dir**
Scartch folder for MPO (default is the same as save_dir, only used when minimal_memory_usage is true).

string **restart_dir** = ""
If not empty, save MPS to this dir after each sweep.

string **restart_dir_per_sweep** = ""
if not empty, save MPS to this dir with sweep index as suffix, so that MPS from all sweeps will be kept in individual dirs.

string **restart_dir_optimal_mps** = ""
If not empty, save MPS to this dir whenever an optimal solution is reached in one sweep. For DMRG, this is the MPS with the lowest energy. Note that if the best solution from the current sweep is worse than the best solution from the previous sweep (for example in a reverse schedule), the best solution from the current sweep is saved.

string **restart_dir_optimal_mps_per_sweep** = ""
If not empty, save the optimal MPS from each sweep to this dir with sweep index as suffix.

string **prefix** = "F"
Filename prefix for common scratch files (such as MPS tensors).

string **prefix_distri** = "F0"
Filename prefix for distributed scratch files (such as renormalized operators). When distributed parallelization is used, different procs will have different values for this data.

bool **prefix_can_write** = true
Whether this proc should be able to write common scratch files (such as MPS tensors).

bool **partition_can_write** = true
Whether this proc should be able to write renormalized operators.

size_t **isize**
Max number of elements in all integer stacks.

size_t **dsize**
Max number of elements in all double stacks.

int **n_frames**
Total number of data frames.

int **i_frame**
The index of Current activated data frame.

mutable double **tread** = 0
IO Time cost for reading scratch files.

mutable double **twrite** = 0
IO Time cost for writing scratch files.

mutable 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.

mutable double **fpwrite** = 0
IO Time cost for writing scratch files with floating-point compression.

mutable Timer **_t**
Temporary timer.

mutable 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 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.

5.1.4 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.

5.2 Sparse Matrix

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

5.2.1 Archived Sparse Matrix

template<typename **S**, typename **FL**>
struct block2::**ArchivedSparseMatrix** : **public** block2::SparseMatrix<*S*, *FL*>
Block-sparse Matrix associated with disk storage, representing sparse operator.

tparam S Quantum label type.

tparam 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.

5.3 Tensor Functions

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

5.3.1 Archived Tensor Functions

template<typename **S**, typename **FL**>
struct block2::**ArchivedTensorFunctions** : **public** block2::TensorFunctions<*S*, *FL*>
Operations for operator tensors with internal data stored in disk file.

tparam S Quantum label type.

tparam FL float point type.

Public Functions

inline **ArchivedTensorFunctions**(**const** shared_ptr<OperatorFunctions<*S*, *FL*>> &*opf*)
Constructor.

Parameters **opf** – Sparse matrix algebra driver.

inline **virtual** TensorFunctionsTypes **get_type**() **const** **override**
Get the type of this driver for tensor functions.

Returns Type of this driver for tensor functions.

inline void **archive_tensor**(**const** shared_ptr<OperatorTensor<*S*, *FL*>> &*a*) **const**
Save the content of an operator tensor into disk, transforming its internal representation to sparse matrices with internal data stored in disk file, and deallocating its memory data.

Parameters **a** – Operator tensor with ordinary memory storage.

inline **virtual** void **left_assign**(**const** shared_ptr<OperatorTensor<*S*, *FL*>> &*a*, shared_ptr<OperatorTensor<*S*, *FL*>> &*c*) **const** **override**
Left assignment (copy) operation: c = a. This is the edge case for the left blocking step. Left assignment means that the operator tensor is a row vector of symbols.

Parameters

- **a** – Operator a (input).
- **c** – Operator c (output).

inline virtual void **right_assign**(**const** shared_ptr<OperatorTensor<*S*, *FL*>> &*a*, shared_ptr<OperatorTensor<*S*, *FL*>> &*c*) **const override**

Right assignment (copy) operation: c = a. This is the edge case for the right blocking step. Right assignment means that the operator tensor is a column vector of symbols.

Parameters

- **a** – Operator a (input).
- **c** – Operator c (output).

inline virtual void **tensor_product_partial_multiply**(**const** shared_ptr<OpExpr<*S*>> &*expr*, **const** shared_ptr<OperatorTensor<*S*, *FL*>> &*lopt*, **const** shared_ptr<OperatorTensor<*S*, *FL*>> &*ropt*, bool *trace_right*, **const** shared_ptr<SparseMatrix<*S*, *FL*>> &*cmat*, **const** vector<pair<uint8_t, *S*>> &*psubsl*, **const** vector<vector<shared_ptr<**typename** SparseMatrixInfo<*S*>::ConnectionInfo>>> &*cinfos*, **const** vector<*S*> &*vdqs*, **const** shared_ptr<SparseMatrixGroup<*S*, *FL*>> &*vmats*, int &*vidx*, int *tvidx*, bool *do_reduce*) **const override**

Partial tensor product multiplication operation: vmat = expr[L part | R part] x cmat. This is used only for perturbative noise, where only left or right block part of the Hamiltonian expression is multiplied by the wavefunction cmat, to get a perurbed wavefunction vmat.

Parameters

- **expr** – Symbolic expression in form of sum of tensor products.
- **lopt** – Symbol lookup table for left operands in the tensor products.
- **ropt** – Symbol lookup table for right operands in the tensor products.
- **trace_right** – If true, the left operands in the tensor products are used. The right operands are treated as identity. Otherwise, the right operands in the tensor products are used.
- **cmat** – Input "vector" operand (wavefunction).
- **psubsl** – Vector of transpose pattern and delta quantum of the "matrix" operand (in the same order as cinfos).
- **cinfos** – Vector of sparse matrix connection info (in the same order as cinfos).
- **vdqs** – Vector of quantum number of each vmat (for lookup).
- **vmats** – Vector of output "vectors" (perurbed wavefunctions).
- **vidx** – If -1, there is only one perurbed wavefunction for each target quantum number (used in NoiseTypes::ReducedPerturbative). Otherwise, one vmat is created for each ten-

sor product (used in NoiseTypes::Perturbative), and vidx is used as an incremental index in vmats.

- **tvidx** – If -1, vmats is copied to every thread, which is the high memory mode but may be more load-balanced, the multi-thread parallelization is over different terms in expr for this case. If -2, every thread works on a single vmat, which is the low memory mode (used in NoiseTypes:: NoiseTypes::LowMem), the multi-thread parallelization is over different vmats for this case. Otherwise, if >= 0, only the specified vmat is handled, which is used only internally.
- **do_reduce** – If true, the output vmats are accumulated to the root processor in the distributed parallel case.

inline virtual void **tensor_product_multi_multiply**(**const** shared_ptr<OpExpr<*S*>> &*expr*, **const** shared_ptr<OperatorTensor<*S*, *FL*>> &*lopt*, **const** shared_ptr<OperatorTensor<*S*, *FL*>> &*ropt*, **const** shared_ptr<SparseMatrixGroup<*S*, *FL*>> &*cmats*, **const** shared_ptr<SparseMatrixGroup<*S*, *FL*>> &*vmats*, **const** unordered_map<*S*, shared_ptr<**typename** SparseMatrixInfo<*S*>::ConnectionInfo>> &*cinfos*, *S opdq*, *FL factor*, bool *all_reduce*) **const override**

Tensor product multiplication operation (multi-root case): vmats = expr x cmats. Both cmats and vmats are wavefunctions with multi-target components.

Parameters

- **expr** – Symbolic expression in form of sum of tensor products.
- **lopt** – Symbol lookup table for left operands in the tensor products.
- **ropt** – Symbol lookup table for right operands in the tensor products.
- **cmats** – Input "vector" operand (multi-target wavefunction).
- **vmats** – Output "vector" result (multi-target wavefunction).
- **cinfos** – Lookup table of sparse matrix connection info, where the key is the combined quantum number of the vmat and cmat.
- **opdq** – The delta quantum number of expr.
- **factor** – Sacling factor applied to the results.
- **all_reduce** – If true, the output result is accumulated and broadcast to all processors.

inline virtual void **tensor_product_multiply**(**const** shared_ptr<OpExpr<*S*>> &*expr*, **const** shared_ptr<OperatorTensor<*S*, *FL*>> &*lopt*, **const** shared_ptr<OperatorTensor<*S*, *FL*>> &*ropt*, **const** shared_ptr<SparseMatrix<*S*, *FL*>> &*cmat*, **const** shared_ptr<SparseMatrix<*S*, *FL*>> &*vmat*, *S opdq*, bool *all_reduce*) **const override**

Tensor product multiplication operation (single-root case): vmat = expr x cmat.

Parameters

- **expr** – Symbolic expression in form of sum of tensor products.
- **lopt** – Symbol lookup table for left operands in the tensor products.
- **ropt** – Symbol lookup table for right operands in the tensor products.
- **cmat** – Input "vector" operand (wavefunction), assuming the cinfo is already attached.
- **vmat** – Output "vector" result (wavefunction).
- **opdq** – The delta quantum number of expr.
- **all_reduce** – If true, the output result is accumulated and broadcast to all processors.

inline virtual void **tensor_product_diagonal**(**const** shared_ptr<OpExpr<*S*>> &*expr*, **const** shared_ptr<OperatorTensor<*S*, *FL*>> &*lopt*, **const** shared_ptr<OperatorTensor<*S*, *FL*>> &*ropt*, **const** shared_ptr<SparseMatrix<*S*, *FL*>> &*mat*, *S opdq*) **const override**

Extraction of diagonal of a tensor product expression: mat = diag(expr).

Parameters

- **expr** – Symbolic expression in form of sum of tensor products.
- **lopt** – Symbol lookup table for left operands in the tensor products.
- **ropt** – Symbol lookup table for right operands in the tensor products.
- **mat** – Output "vector" result (diagonal part).
- **opdq** – The delta quantum number of expr.

inline virtual void **tensor_product**(**const** shared_ptr<OpExpr<*S*>> &*expr*, **const** unordered_map<shared_ptr<OpExpr<*S*>>, shared_ptr<SparseMatrix<*S*, *FL*>>> &*lop*, **const** unordered_map<shared_ptr<OpExpr<*S*>>, shared_ptr<SparseMatrix<*S*, *FL*>>> &*rop*, shared_ptr<SparseMatrix<*S*, *FL*>> &*mat*) **const override**

Direct evaluation of a tensor product expression: mat = eval(expr).

Parameters

- **expr** – Symbolic expression in form of sum of tensor products.
- **lop** – Symbol lookup table for left operands in the tensor products.
- **rop** – Symbol lookup table for right operands in the tensor products.
- **mat** – Output "vector" result (the sparse matrix value of the expression).

inline virtual void **left_rotate**(**const** shared_ptr<OperatorTensor<*S*, *FL*>> &*a*, **const** shared_ptr<SparseMatrix<*S*, *FL*>> &*mpst_bra*, **const** shared_ptr<SparseMatrix<*S*, *FL*>> &*mpst_ket*, shared_ptr<OperatorTensor<*S*, *FL*>> &*c*) **const override**

Rotation (renormalization) of a left-block operator tensor: c = mpst_bra.T x a x mpst_ket. In the above expression, [x] means multiplication. Note that the row and column of mpst_bra and mpst_ket are for system and environment indices, respectively. Left rotation means that system indices are contracted.

Parameters

- **a** – Input operator tensor a (as a row vector of symbols).
- **mpst_bra** – Rotation matrix (bra MPS tensor) for row of sparse matrices in a.
- **mpst_ket** – Rotation matrix (ket MPS tensor) for column of sparse matrices in a.
- **c** – Output operator tensor c (as a row vector of symbols).

inline virtual void **right_rotate**(**const** shared_ptr<OperatorTensor<*S*, *FL*>> &*a*, **const** shared_ptr<SparseMatrix<*S*, *FL*>> &*mpst_bra*, **const** shared_ptr<SparseMatrix<*S*, *FL*>> &*mpst_ket*, shared_ptr<OperatorTensor<*S*, *FL*>> &*c*) **const override**

Rotation (renormalization) of a right-block operator tensor: c = mpst_bra x a x mpst_ket.T. In the above expression, [x] means multiplication. Note that the row and column of mpst_bra and mpst_ket are for system and environment indices, respectively. Right rotation means that environment indices are contracted.

Parameters

- **a** – Input operator tensor a (as a column vector of symbols).
- **mpst_bra** – Rotation matrix (bra MPS tensor) for row of sparse matrices in a.
- **mpst_ket** – Rotation matrix (ket MPS tensor) for column of sparse matrices in a.
- **c** – Output operator tensor c (as a column vector of symbols).

inline virtual void **intermediates**(**const** shared_ptr<Symbolic<*S*>> &*names*, **const** shared_ptr<Symbolic<*S*>> &*exprs*, **const** shared_ptr<OperatorTensor<*S*, *FL*>> &*a*, bool *left*) **const override**

Compute the intermediates to speed up the tensor product operations in the next blocking step. Intermediates are formed by collecting terms sharing the same left or right operands during the MPO simplification step.

Parameters

- **names** – Operator names (symbols, only used in distributed parallelization).
- **exprs** – Tensor product expressions (expressions of symbols).
- **a** – Symbol lookup table for symbols in the tensor product expressions (updated).
- **left** – Whether this is for left-blocking or right-blocking.

inline virtual void **numerical_transform**(**const** shared_ptr<OperatorTensor<*S*, *FL*>> &*a*, **const** shared_ptr<Symbolic<*S*>> &*names*, **const** shared_ptr<Symbolic<*S*>> &*exprs*) **const override**

Numerical transform from normal operators to complementary operators near the middle site.

Parameters

- **a** – Symbol lookup table for symbols in the tensor product expressions (updated).
- **names** – List of complementary operator names (symbols).
- **exprs** – List of symbolic expression of complementary operators as linear combination of normal operators.

inline virtual void **left_contract**(**const** shared_ptr<OperatorTensor<*S*, *FL*>> &*a*, **const** shared_ptr<OperatorTensor<*S*, *FL*>> &*b*, shared_ptr<OperatorTensor<*S*, *FL*>> &*c*, **const** shared_ptr<Symbolic<*S*>> &*cexprs* = nullptr, OpNamesSet *delayed* = OpNamesSet()) **const override**

Tensor product operation in left blocking: c = a x b.

Parameters

- **a** – Operator a (left block tensor).
- **b** – Operator b (dot block single-site tensor).
- **c** – Operator c (enlarged left block tensor).
- **cexprs** – Symbolic expression for the tensor product operation. If nullptr, this is automatically contructed from expressions in a and b.
- **delayed** – The set of operator names for which the tensor product operation should be delayed. The delayed tensor product will not be performed here. Instead, it will be evaluated as three-tensor operations later.

inline virtual void **right_contract**(**const** shared_ptr<OperatorTensor<*S*, *FL*>> &*a*, **const** shared_ptr<OperatorTensor<*S*, *FL*>> &*b*, shared_ptr<OperatorTensor<*S*, *FL*>> &*c*, **const** shared_ptr<Symbolic<*S*>> &*cexprs* = nullptr, OpNamesSet *delayed* = OpNamesSet()) **const override**

Tensor product operation in right blocking: c = b x a.

Parameters

- **a** – Operator a (right block tensor).
- **b** – Operator b (dot block single-site tensor).
- **c** – Operator c (enlarged right block tensor).
- **cexprs** – Symbolic expression for the tensor product operation. If nullptr, this is automatically contructed from expressions in b and a.
- **delayed** – The set of operator names for which the tensor product operation should be delayed. The delayed tensor product will not be performed here. Instead, it will be evaluated as three-tensor operations later.

Public Members

string **filename** = ""

The name of the associated disk file.

mutable int64_t **offset** = 0

Byte offset in the file (where to read/write the content).

5.4 Tools

5.4.1 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 `block2::FPtraits`
Floating-point number representation details.

tparam 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 `block2::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 `block2::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 block2::**BitsCodec**
Read/write bits from/into a floating-point number array.

tparam T The floating type for the elemenet of the array.

tparam 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 block2::**FPCodec**
Codec for compressing/decompressing array of floating-point numbers.

tparam T Floating type to implement.

tparam U The corresponding integer type.

tparam mbits Number of bits in significand.

tparam ebits Number of bits in exponent.

Public Functions

inline **FPCodec** ()
Default constructor.

inline **FPCodec** (*T prec*)
Constructor.

Parameters **prec** – Floating-point number precision.

inline **FPCodec** (*T prec*, size_t *chunk_size*)
Constructor.

Parameters

- **prec** – Floating-point number precision.

- **chunk_size** – Length of the array elements that should be processed at one time.

inline size_t **decode**(*T* *ip_data, size_t *len*, *T* *op_data) **const**
Decode data.

Parameters

- **ip_data** – The compressed floating-point array.
- **len** – Length of the original floating-point array.
- **op_data** – Output array for storing original data. Memory should be pre-allocated with length = len.

Returns Length of the array for the compressed data.

inline size_t **encode**(*T* *ip_data, size_t *len*, *T* *op_data) **const**
Encode data.

Parameters

- **ip_data** – The original floating-point array.
- **len** – Length of the original floating-point array.
- **op_data** – Output array for storing compressed data. Memory should be pre-allocated with length >= len + 1.

Returns Length of the array for the compressed data.

inline void **write_array**(ostream &*ofs*, *T* *data, size_t *len*) **const**
Compress array of floating-point data and write into file stream.

Parameters

- **ofs** – Output stream.
- **data** – The original floating-point array.
- **len** – The length of the original floating-point array.

inline void **read_array**(istream &*ifs*, *T* *data, size_t *len*) **const**
Read from file stream and deompress the data.

Parameters

- **ifs** – Input stream.
- **data** – The floating-point array for storing the original data.
- **len** – The length of the original floating-point array.

inline void **read_chunks**(istream &*ifs*, size_t *len*, vector<vector<*T*>> &*chunks*, size_t &*chunk_size*) **const**
Read from file stream (but not decompress the data).

Parameters

- **ifs** – Input stream.
- **len** – The length of the original floating-point array.
- **chunks** – For storing chunks of the compressed data.
- **chunk_size** – Number of original array elements in each chunk (output).

Public Members

T **prec**
Precision for compression.

U **prec_u**
Integer representation of the precision.

mutable size_t **ndata** = 0
Length of the array of the data that has been compressed.

mutable 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 block2::**CompressedVector**
An array with data stored in compressed form. Only support single thread.

tparam T Elemenet type.

Subclassed by *block2::CompressedVectorMT< T >*

Public Functions

inline CompressedVector(size_t *arr_len*, *T prec*, size_t *chunk_size*, int *ncache* = 4)
Constructor.

Parameters

- **arr_len** – Size of the array.
- **prec** – Precision for compression.
- **chunk_size** – Number of array elements in each chunk.
- **ncache** – Number of cached decompressed chunks.

inline CompressedVector(istream &*ifs*, size_t *arr_len*, *T prec*, int *ncache* = 4)
Constructor.

Parameters

- **ifs** – Input stream, from which the compressed data is obtained.
- **arr_len** – Size of the array.
- **prec** – Precision for compression.
- **ncache** – Number of cached decompressed chunks.

inline **CompressedVector**(*T* *arr, size_t arr_len, *T* prec, int ncache = 4)
Constructor.

Parameters

- **arr** – The compressed data as an array.
- **arr_len** – Size of the array.
- **prec** – Precision for compression.
- **ncache** – Number of cached decompressed chunks.

virtual **~CompressedVector**() = default
Deconstructor.

inline void **clear**()
Set all array elements to zero.

inline void **shrink_to_fit**()
Minimize the storage cost of the compressed data (after the data has been changed).

inline void **finalize**()
Write all cached data into compressed form.

inline **virtual** *T* &**operator[]**(size_t *i*)
Write one element into the array.

Parameters **i** – Array index.

Returns The array elemenet.

inline **virtual** *T* **operator[]**(size_t *i*) **const**
Read one element from the array.

Parameters **i** – Array index.

Returns The array elemenet.

inline size_t **size**() **const**
Get the size of the array.

Public Members

size_t **arr_len**
Size of the array.

size_t **chunk_size**
Number of array elements in each chunk.

int **ncache**
Number of cached decompressed chunks.

mutable int **icache**
Index of next available cache (head of the circular queue).

mutable vector<vector<*T*>> **cp_data**
Chunks of compressed data.

mutable vector<pair<size_t, vector<*T*>>> **cache_data**
Cached data of decompressed chunks.

mutable vector<bool> **cache_dirty**
Whether each cached chunk has been changed.

FPCodec<*T*> **fpc**
Floating-point number compression driver.

template<typename **T**>
struct block2::**CompressedVectorMT** : **public** block2::*CompressedVector*<*T*>
A read-only array with data stored in compressed form, with multi-threading support.

tparam T Elemenet type.

Public Functions

inline **CompressedVectorMT**(**const** shared_ptr<*CompressedVector*<*T*>> &*ref_cv*, int *ntg*)
Constructor.

Parameters

- **ref_cv** – Pointer to the single-thread compressed array.
- **ntg** – Number of threads.

inline **virtual** *T* &**operator[]**(size_t *i*)
Write one element into the array (not supported, will cause assertion failure).

Parameters **i** – Array index.

Returns The array elemenet.

inline **virtual** *T* **operator[]**(size_t *i*) **const**
Read one element from the array.

Parameters **i** – Array index.

Returns The array elemenet.

inline size_t **size**() **const**
Get the size of the array.

Public Members

mutable vector<int> **icaches**
Index of next available cache (head of the circular queue) for each thread.

mutable vector<vector<pair<size_t, vector<*T*>>>> **cache_datas**
Cached data of decompressed chunks for each thread.

shared_ptr<*CompressedVector*<*T*>> **ref_cv**
Pointer to the single-thread compressed array.

5.4.2 Kuhn-Munkres Algorithm

The `KuhnMunkres` class implements the Kuhn-Munkres algorithm for finding the best matching in a bipartite with the lowest cost.

struct `block2::KuhnMunkres`
Kuhn-Munkres algorithm for finding the best matching with lowest cost. Complexity: O(n^3).

Public Functions

inline **KuhnMunkres**(**const** vector<double> &*cost_matrix*, int *m*, int *n* = 0)
Constructor.

Parameters

- **cost_matrix** – Flattened matrix of cost.
- **m** – Number of rows.
- **n** – Number of columns.

inline bool **match**(vector<int> &*x*, int *u*)
Find an augmenting alternating path in the equality subgraph. If an equality subgraph has a perfect matching, the it is a maximum-weight matching in the graph.

Parameters

- **x** – Current matching.
- **u** – Starting vertex.

Returns `true` if an augmenting alternating path is found.

inline pair<double, vector<int>> **solve**()
Find the lowest cost and a matching.

Returns the lowest cost and an index array x, with x[i] = j meaning that column index i should be matched to row index j.

Public Members

vector<double> **cost**
Flattened n x n matrix of cost.

int **n**
Number of rows or columns.

double **inf**
Infinity (constant).

double **eps**
Machine precision (constant).

vector<double> **lx**
Left feasible labeling (working array).

vector<double> **ly**
Right feasible labeling (working array).

vector<double> **slack**
Slack working array.

vector<char> **st**
Candidate augmenting alternating path (working array).

5.4.3 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 `block2::Prime`
Number theory algorithms for prime numbers and primitive root.

Public Functions

inline **Prime**()
Default constructor.

inline void **init_primes**(int *np* = 50000)
Initialize set of small primes, using sieve of Eratosthenes.

Parameters **np** – Maximal number considered for finding primes.

inline void **factors**(LL *x*, vector<pair<LL, int>> &*pp*)
Integer factorization.

Parameters

- **x** – Integer x.
- **pp** – (output) Set of prime factors and their occurrence.

inline LL **euler**(LL *n*)
Euler's totient function.

Parameters **n** – Integer n (not necessarily prime).

Returns phi(n), which counts the positive integers up to the given integer n that are relatively prime to n.

inline bool **is_prime**(LL *n*)
Prime testing.

Parameters **n** – Integer n to be tested (n < 2^63 - 1).

Returns `true` if n is a prime, `false` if n is not a prime.

inline int **primitive_root**(LL *p*)
Find one of primitive roots modulo n. A number g is a primitive root modulo n if every number a coprime to n is congruent to a power of g modulo n.

Parameters **p** – Modulus p.

Returns a primitive root g.

inline void **primitive_roots**(LL *p*, vector<LL> &*gg*)
Find all primitive roots modulo n.

Parameters

- **p** – Modulus p.
- **gg** – (output) All primitive roots modulo n.

Public Members

int **np**
Maximal number considered for generation of set of small primes.

vector<int> **primes**
Set of small primes.

Public Static Functions

static inline int **pmod**(LL *x*, LL *n*)
Return positive x mod n.

static inline void **exgcd**(LL *a*, LL *b*, LL &*d*, LL &*x*, LL &*y*)
Extended Euclidean algorithm. Find integers x and y such that a x + b y = gcd(a, b).

Parameters

- **a** – Integer a.
- **b** – Integer b.
- **d** – (output) Greatest Common Divisor gcd(a, b).
- **x** – (output) Integer x.
- **y** – (output) Integer y.

static inline LL **gcd**(LL *a*, LL *b*)
Euclidean algorithm. Find Greatest Common Divisor gcd(a, b).

Parameters

- **a** – Integer a.
- **b** – Integer b.

Returns Greatest Common Divisor gcd(a, b).

static inline LL **inv**(LL *a*, LL *n*)
Find modular multiplicative inverse of a (mod n). Using extended Euclidean algorithm.

Parameters

- **a** – Integer a.
- **n** – Modulus n.

Returns a^{-1} (mod n) if it exists, otherwise -1.

static inline LL **power**(LL *n*, int *i*)
Find n to the power of i using binary algorithm.

Parameters

- **n** – Integer n.
- **i** – Integer i (i >= 0).

Returns n^i.

static inline LL **sqrt**(LL *x*)
Find the largest number r such that r * r <= x. Using binary algorithm.

Parameters **x** – Integer x.

Returns floor(sqrt(x)).

static inline LL **quick_multiply**(LL *x*, LL *y*, LL *p*)
Find (x * y) mod p. Note that the expression x * y % p may overflow if the intermediate x * y > 2^63 - 1. This function will not overflow (if p <= 2^62 - 1)

Parameters

- **x** – Integer x.
- **y** – Integer y.
- **p** – Modulus p.

Returns (x * y) mod p.

static inline LL **quick_power**(LL *n*, LL *i*, LL *p*)
Find (n ^ i) mod p using binary algorithm.

Parameters

- **n** – Integer n.
- **i** – Integer i (i >= 0).
- **p** – Modulus p.

Returns (n ^ i) mod p.

static inline bool **miller_rabin**(LL *a*, LL *n*)
Miller-Rabin primality test.

Note: If n < 3215031751, it is enough to test a = 2, 3, 5, and 7; if n < 341550071728321, it is enough to test a = 2, 3, 5, 7, 11, 13, and 17.

Parameters

- **a** – Base number a.
- **n** – The number to be tested for prime (n >= 3).

Returns `true` if n is likely to be a prime. `false` if n is not a prime.

static inline LL **pollard_rho**(LL *n*)
Pollard's rho algorithm. Find a factor of integer n.

Parameters **n** – Integer n.

Returns A (not necessarily prime) factor of n.

5.4.4 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 ** 3` and `5929742 = 2 * 7 * 13 * 31 * 1051`.

template<int **base**>
struct `block2::BasicFFT`
Fast Fourier Transform (FFT) with array length of base^k (k >= 0). The complexity is O(n log_base n * base^2).

tparam base The radix (base = 2 is specialized)

Public Functions

inline BasicFFT()
Constructor.

inline void **init**(size_t *n*)
Precompute for array length n for both forward and backward FFT.

Parameters **n** – The array length, must be a power of `base`.

inline void **fft**(complex<double> **arr*, size_t *n*, bool *forth*)
Perform inplace FFT.

Parameters

- **arr** – A pointer to the array of complex numbers.
- **n** – Number of elements in the array, must be a power of `base`.
- **forth** – Whether this is forward transform.

Public Members

vector<size_t> **r**
The index permutation array.

vector<complex<double>> **wb**
The precomputed primitive nth root of 1 exp(i2pi k/n) for backward FFT.

vector<complex<double>> **wf**
The precomputed primitive nth root of 1 exp(-i2pi k/n) for forward FFT.

array<array<complex<double>, *base*>, *base*> **xw**[2]
The precomputed primitive `base`-th root of 1 exp(-/+ i2pi jk/base) for forward/backward FFT.

Public Static Functions

static inline size_t **pad**(size_t *n*)
Find smallest number x such that x = base^k >= n.

Parameters **n** – The array length.

Returns The padded array length.

template<>
struct `block2::`**BasicFFT**<2>
Radix-2 Fast Fourier Transform (FFT) with complexity O(nlog n).

Public Functions

inline BasicFFT()
Constructor.

inline void **init**(size_t *n*)
Precompute for array length n for both forward and backward FFT.

Parameters **n** – The array length must be a power of 2.

inline void **fft**(complex<double> **arr*, size_t *n*, bool *forth*)
Perform inplace FFT.

Parameters

- **arr** – A pointer to the array of complex numbers.
- **n** – Number of elements in the array, must be a power of 2.
- **forth** – Whether this is forward transform.

Public Members

vector<size_t> **r**
The index permutation array.

vector<complex<double>> **wb**
The precomputed primitive nth root of 1 exp(i2pi k/n) for backward FFT.

vector<complex<double>> **wf**
The precomputed primitive nth root of 1 exp(-i2pi k/n) for forward FFT.

Public Static Functions

static inline size_t **pad**(size_t *n*)
Find smallest number x such that x = 2^k >= n.

Parameters **n** – The array length.

Returns The padded array length.

template<typename **B** = *BasicFFT*<2>>
struct block2::**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.

tparam B The backend FFT for computing the padded FFT.

Public Functions

inline RaderFFT()
Default constructor.

inline RaderFFT(**const** shared_ptr<*Prime*> &*prime*)
Constructor.

Parameters **prime** – Instance for prime number algorithms.

inline void **init**(size_t *n*)
Precompute for array length n for both forward and backward FFT.

Parameters **n** – The array length, which must be a prime number.

inline void **fft**(complex<double> *arr, size_t *n*, bool *forth*)
Perform inplace FFT.

Parameters

- **arr** – A pointer to the array of complex numbers.
- **n** – Number of elements in the array, which must be a prime number.
- **forth** – Whether this is forward transform.

Public Members

vector<LL> **wb**
Precomputed inverse of the power of primitive root g^{-k} mod n for k = 0, 1, . . . , n - 1.

vector<LL> **wf**
Precomputed power of primitive root g^k mod n for k = 0, 1, . . . , n - 1.

vector<complex<double>> **cb**
FFT transformed exp(i2pi g^{-k}/n).

vector<complex<double>> **cf**
FFT transformed exp(-i2pi g^{-k}/n).

vector<complex<double>> **arx**
Working space for padded array.

size_t **nn**
Padded array length.

B **b**
The backend FFT instance.

shared_ptr<*Prime*> **prime**
Instance for prime number algorithms.

template<typename **B** = *BasicFFT*<2>>
struct block2::**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.

tparam B The backend FFT for computing the padded FFT.

Public Functions

inline **BluesteinFFT**()
Default constructor.

inline **BluesteinFFT**(**const** shared_ptr<*Prime*> &*prime*)
Constructor.

Parameters **prime** – Instance for prime number algorithms (ignored).

inline void **init**(size_t *n*)
Precompute for array length n for both forward and backward FFT.

Parameters **n** – The array length.

inline void **fft**(complex<double> *arr, size_t *n*, bool *forth*)
Perform inplace FFT.

Parameters

- **arr** – A pointer to the array of complex numbers.
- **n** – Number of elements in the array.
- **forth** – Whether this is forward transform.

Public Members

vector<complex<double>> **wb**
The precomputed primitive nth root of 1 exp(i2pi k/n) for backward FFT.

vector<complex<double>> **wf**
The precomputed primitive nth root of 1 exp(-i2pi k/n) for forward FFT.

vector<complex<double>> **cb**
FFT transformed exp(i2pi [((k - n) - (k - n)(k - n)) / 2]/n).

vector<complex<double>> **cf**
FFT transformed exp(-i2pi [((k - n) - (k - n)(k - n)) / 2]/n).

vector<complex<double>> **arx**
Working space for padded array.

size_t **nn**
Padded array length.

B **b**
The backend FFT instance.

struct block2::**DFT**
Naive Discrete Fourier Transform (*DFT*) algorithm with complexity O(n^2).

Public Functions

inline **DFT**()
Default constructor.

inline **DFT**(**const** shared_ptr<*Prime*> &*prime*)
Constructor.

Parameters **prime** – Instance for prime number algorithms (ignored).

inline void **init**(size_t *n*)
Precompute for array length n for both forward and backward FFT. For *DFT* this method does nothing.

Parameters **n** – The array length.

inline void **fft**(complex<double> *arr, size_t *n*, bool *forth*)

Perform inplace DFT.

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

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

Parameters

- **arr** – A pointer to the array of complex numbers.
- **n** – Number of elements in the array.
- **forth** – Whether this is forward transform.

template<typename **F**, int **P**, int... **Q**>
struct block2::**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.

tparam F The prime number FFT backend.

tparam P Using Radix-P FFT backend.

tparam 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 block2::**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.

tparam F The prime number FFT backend.

tparam P Using Radix-P FFT backend.

Public Functions

inline FactorizedFFT()
Default constructor.

inline FactorizedFFT(int *max_factor*)
Constructor.

Parameters **max_factor** – Maximal radix that should be checked for radix based FFT.

inline void **init**(size_t *n*)
Precompute for array length n for both forward and backward FFT. For *FactorizedFFT* this method does nothing.

Parameters **n** – The array length.

inline virtual void **fft_internal**(complex<double> **arr*, size_t *p*, size_t *q*, bool *forth*, int *b*)
Perform independent FFTs for p arrays, each with length q.

Parameters

- **arr** – A pointer to the array of complex numbers (as a matrix).
- **p** – Number of rows (FFTs).
- **q** – Number of columns (length of each FFT).
- **forth** – Whether this is forward transform.
- **b** – Radix. Zero if radix based FFT should not be used.

inline void **cooley_tukey**(complex<double> **arr*, size_t *n*, bool *forth*, **const** size_t **pr*, **const** int **b*, size_t *np*)
Cooley-Tukey FFT algorithm for FFT with array length being a composite number.

Parameters

- **arr** – A pointer to the array of complex numbers.
- **n** – Number of elements in the array.
- **forth** – Whether this is forward transform.
- **pr** – A pointer to the array of factors in n;
- **b** – A pointer to the array of radices for each factor. pr should be multiple of b, if b is not zero.
- **np** – Number of factors.

inline void **fft**(complex<double> **arr*, size_t *n*, bool *forth*)

Perform inplace FFT.

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

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

Parameters

- **arr** – A pointer to the array of complex numbers.
- **n** – Number of elements in the array.
- **forth** – Whether this is forward transform.

Public Members

const int **max_factor** = *P*
Maximal radix number.

shared_ptr<*Prime*> **prime**
Instance for prime number algorithms.

typedef *FactorizedFFT*<*RaderFFT*<>, 2, 3, 5, 7, 11> block2::**FFT2**
FFT with small prime factorization implemeted using Rader's algorithm.

typedef *FactorizedFFT*<*BluesteinFFT*<>, 2, 3, 5, 7, 11> block2::**FFT**
FFT with small prime factorization implemeted using Bluestein's algorithm.

CHAPTER

SIX

THEORY

6.1 DMRG Hamiltonian

6.1.1 DMRG Quantum Chemistry Hamiltonian in Spatial Orbitals

Hamiltonian

The quantum chemistry Hamiltonian is written as follows

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

where

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

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

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

Partitioning in Spatial Orbitals

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

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

where the normal and complementary operators are defined by

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

Note that we need to move all on-site interaction into local Hamiltonian, so that when construction interaction terms in Hamiltonian, operators anticommute (without giving extra constant terms).

Derivation

First consider one-electron term. ij indices have only two possibilities: i left, j right, or i right, j left. Index i must be associated with creation operator. So the second case is the Hermitian conjugate of the first case. Namely,

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

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

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

where the terms in bracket equal to first and third terms in left-hand-side. Outside the bracket are second, forth terms.

The conjugate of third term in rhs is second term in rhs

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

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

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

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

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

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,

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

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.

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

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

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

Note that no minus sign for Hermitian conjugate terms with A, P because these are not Fermion operators.

Also note that

$$\sum_{i\in L,\sigma} a_{i\sigma}^\dagger \hat{S}_{i\sigma}^R = \sum_{i\in L, j\in R,\sigma} t_{ij} a_{i\sigma}^\dagger a_{j\sigma} = \sum_{j\in R,\sigma} S_{j\sigma}^{L\dagger} a_{j\sigma}$$

Define

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

we have

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

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

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

The operators required in right block are

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

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

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

Now the operators required in left block are

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

The operators required in right block are

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

The total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$N_{CN} = 1 + 1 + 4K_R + 4K_L + 8K_R^2 + K_R^2 + 4K_R^2 = 13K_R^2 + 4K + 2$$

Blocking

The enlarged left/right block is denoted as $L*/R*$. Make sure that all L operators are to the left of $*$ operators.

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

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

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

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

Similarly,

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

Number of terms

$$\begin{aligned}
N_{R',NC} &= (2 + 5K_L + 5K_L^2)K_R + (2 + 5 + 5K_R)K_L = 5K_L^2 K_R + 10K_L K_R + 2K + 5K_L \\
N_{R',CN} &= (2 + 5K_L + 5)K_R + (2 + 5K_R^2 + 5K_R)K_L = 5K_R^2 K_L + 10K_R K_L + 2K + 5K_R
\end{aligned}$$

Blocking of other complementary operators is straightforward

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

and

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

and

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

Middle-Site Transformation

When the sweep is performed from left to right, passing the middle site, we need to switch from NC partition to CN partition. The cost is $O(K^4/16)$. This happens only once in the sweep. The cost of one blocking procedure is $O(K_<^2 K_>)$, but there are K blocking steps in one sweep. So the cost for blocking in one sweep is $O(KK_<^2 K_>)$. Note that the most expensive part in the program should be the Hamiltonian step in Davidson, which scales as $O(K_<^2)$.

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

6.1.2 Spin-Adapted DMRG Quantum Chemistry Hamiltonian

Partitioning in SU(2)

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

$$\begin{aligned}
(\hat{H})^{[0]} =\ & \big(\hat{H}^L\big)^{[0]} \otimes_{[0]} \big(\hat{1}^R\big)^{[0]} + \big(\hat{1}^L\big)^{[0]} \otimes_{[0]} \big(\hat{H}^R\big)^{[0]} \\
& + \sqrt{2} \sum_{i\in L} \Big[\big(a^\dagger_i\big)^{[\frac{1}{2}]} \otimes_{[0]} \big(\hat{S}^R_i\big)^{[\frac{1}{2}]} + h.c. \Big] \\
& + 2 \sum_{i\in L} \Big[\big(a^\dagger_i\big)^{[\frac{1}{2}]} \otimes_{[0]} \big(\hat{R}^R_i\big)^{[\frac{1}{2}]} + h.c. \Big] + 2 \sum_{i\in R} \Big[\big(a^\dagger_i\big)^{[\frac{1}{2}]} \otimes_{[0]} \big(\hat{R}^L_i\big)^{[\frac{1}{2}]} + h.c. \Big] \\
& - \frac{1}{2} \sum_{ik\in L} \Big[\sqrt{3} \big(\hat{A}_{ik}\big)^{[1]} \otimes_{[0]} \big(\hat{P}^R_{ik}\big)^{[1]} + \big(\hat{A}_{ik}\big)^{[0]} \otimes_{[0]} \big(\hat{P}^R_{ik}\big)^{[0]} + h.c. \Big] \\
& + \sum_{ij\in L} \Big[\big(\hat{B}_{ij}\big)^{[0]} \otimes_{[0]} \Big(2 \big(\hat{Q}^R_{ij}\big)^{[0]} - \big(\hat{Q}'^R_{ij}\big)^{[0]} \Big) + \sqrt{3} \big(\hat{B}'_{ij}\big)^{[1]} \otimes_{[0]} \big(\hat{Q}'^R_{ij}\big)^{[1]} \Big]
\end{aligned}$$

where the normal and complementary operators are defined by

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

Derivation

CG Factors

From $j_2 = 1/2$ CG factors

$$\left\langle j_1 \ \left(M - \frac{1}{2}\right) \ \frac{1}{2} \ \frac{1}{2} \middle| \left(j_1 \pm \frac{1}{2}\right) \ M \right\rangle = \pm \sqrt{\frac{1}{2}\left(1 \pm \frac{M}{j_1 + \frac{1}{2}}\right)}$$

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

and symmetry relation

$$\langle j_1 \ m_1 \ j_2 \ m_2 | J \ M \rangle = (-1)^{j_1 + j_2 - J} \langle j_2 \ m_2 \ j_1 \ m_1 | J \ M \rangle$$

and

$$(-1)^{j_1 + \frac{1}{2} - j_1 \mp \frac{1}{2}} = (-1)^{\frac{1}{2} \mp \frac{1}{2}} = \pm 1$$

we have

$$\left\langle \frac{1}{2} \ \frac{1}{2} \ j_1 \ \left(M - \frac{1}{2}\right) \middle| \left(j_1 \pm \frac{1}{2}\right) \ M \right\rangle = \sqrt{\frac{1}{2}\left(1 \pm \frac{M}{j_1 + \frac{1}{2}}\right)}$$

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

let $j_1 = 1$, we have

$$\langle \tfrac{1}{2} \ \tfrac{1}{2} \ 1 \ (M - \tfrac{1}{2}) | \tfrac{1}{2} \ M \rangle = \sqrt{\tfrac{1}{2}(1 - \frac{M}{\frac{3}{2}})}$$

$$\langle \tfrac{1}{2} \ (-\tfrac{1}{2}) \ 1 \ (M + \tfrac{1}{2}) | \tfrac{1}{2} \ M \rangle = -\sqrt{\tfrac{1}{2}(1 + \frac{M}{\frac{3}{2}})}$$

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

$$[\tfrac{1}{2} + 0 = \tfrac{1}{2}] = \sqrt{\tfrac{1}{3}}, \quad [-\tfrac{1}{2} + 1 = \tfrac{1}{2}] = -\sqrt{\tfrac{2}{3}}$$

$$[\tfrac{1}{2} + (-1) = -\tfrac{1}{2}] = \sqrt{\tfrac{2}{3}}, \quad [-\tfrac{1}{2} + 0 = -\tfrac{1}{2}] = -\sqrt{\tfrac{1}{3}}$$

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

$$[0 + \tfrac{1}{2} = \tfrac{1}{2}] = -\sqrt{\tfrac{1}{3}}, \quad [1 - \tfrac{1}{2} = \tfrac{1}{2}] = \sqrt{\tfrac{2}{3}}$$

$$[(-1) + \tfrac{1}{2} = -\tfrac{1}{2}] = -\sqrt{\tfrac{2}{3}}, \quad [0 - \tfrac{1}{2} = -\tfrac{1}{2}] = \sqrt{\tfrac{1}{3}}$$

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

Tensor Product Formulas

Singlet

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

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

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

Triplet

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

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

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

Doublet times singlet/triplet

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

$$= \begin{pmatrix} -\frac{1}{\sqrt{2}} \frac{1}{\sqrt{3}} a_{p\alpha}^\dagger \left(a_{r\beta} a_{s\alpha} + a_{r\alpha} a_{s\beta}\right) - \frac{\sqrt{2}}{\sqrt{3}} a_{p\beta}^\dagger a_{r\beta} a_{s\beta} \\ \frac{\sqrt{2}}{\sqrt{3}} a_{p\alpha}^\dagger a_{r\alpha} a_{s\alpha} + \left(-\frac{1}{\sqrt{3}}\right)\left(-\frac{1}{\sqrt{2}}\right) a_{p\beta}^\dagger \left(a_{r\beta} a_{s\alpha} + a_{r\alpha} a_{s\beta}\right) \end{pmatrix}^{[1/2]} = \frac{1}{\sqrt{6}} \begin{pmatrix} -a_{p\alpha}^\dagger a_{r\beta} a_{s\alpha} - a_{p\alpha}^\dagger a_{r\alpha} a_{s\beta} - 2 a_{p\beta}^\dagger a_{r\beta} a_{s\beta} \\ 2 a_{p\alpha}^\dagger a_{r\alpha} a_{s\alpha} + a_{p\beta}^\dagger a_{r\beta} a_{s\alpha} + a_{p\beta}^\dagger a_{r\alpha} a_{s\beta} \end{pmatrix}^{[1/2]}$$

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

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

Therefore,

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

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

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

Another case

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

$$= \begin{pmatrix} \frac{1}{\sqrt{2}}\frac{1}{\sqrt{3}}(-a_{r\beta})\left(a_{p\alpha}^\dagger a_{q\alpha} - a_{p\beta}^\dagger a_{q\beta}\right) + \frac{\sqrt{2}}{\sqrt{3}} a_{r\alpha} a_{p\alpha}^\dagger a_{q\beta} \\ -\frac{\sqrt{2}}{\sqrt{3}} a_{r\beta} a_{p\beta}^\dagger a_{q\alpha} - \frac{1}{\sqrt{2}}\frac{1}{\sqrt{3}} a_{r\alpha}\left(a_{p\alpha}^\dagger a_{q\alpha} - a_{p\beta}^\dagger a_{q\beta}\right) \end{pmatrix}^{[1/2]} = \frac{1}{\sqrt{6}} \begin{pmatrix} -a_{r\beta} a_{p\alpha}^\dagger a_{q\alpha} + a_{r\beta} a_{p\beta}^\dagger a_{q\beta} + 2a_{r\alpha} a_{p\alpha}^\dagger a_{q\beta} \\ -2a_{r\beta} a_{p\beta}^\dagger a_{q\alpha} - a_{r\alpha} a_{p\alpha}^\dagger a_{q\alpha} + a_{r\alpha} a_{p\beta}^\dagger a_{q\beta} \end{pmatrix}^{[1/2]}$$

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

$$= \frac{1}{\sqrt{2}} \begin{pmatrix} -a_{r\beta} a_{p\alpha}^\dagger a_{q\alpha} - a_{r\beta} a_{p\beta}^\dagger a_{q\beta} \\ a_{r\alpha} a_{p\alpha}^\dagger a_{q\alpha} + a_{r\alpha} a_{p\beta}^\dagger a_{q\beta} \end{pmatrix}^{[1/2]}$$

Therefore,

$$\sqrt{3}S^{[1/2]} - T^{[1/2]} = \frac{1}{\sqrt{6}} \begin{pmatrix} -a_{r\beta} a_{p\alpha}^\dagger a_{q\alpha} + a_{r\beta} a_{p\beta}^\dagger a_{q\beta} + 2a_{r\alpha} a_{p\alpha}^\dagger a_{q\beta} \\ -2a_{r\beta} a_{p\beta}^\dagger a_{q\alpha} - a_{r\alpha} a_{p\alpha}^\dagger a_{q\alpha} + a_{r\alpha} a_{p\beta}^\dagger a_{q\beta} \end{pmatrix}^{[1/2]} - \frac{1}{\sqrt{2}} \begin{pmatrix} -a_{r\beta} a_{p\alpha}^\dagger a_{q\alpha} - a_{r\beta} a_{p\beta}^\dagger a_{q\beta} \\ a_{r\alpha} a_{p\alpha}^\dagger a_{q\alpha} + a_{r\alpha} a_{p\beta}^\dagger a_{q\beta} \end{pmatrix}^{[1/2]}$$

$$= \frac{1}{\sqrt{2}} \begin{pmatrix} -a_{r\beta} a_{p\alpha}^\dagger a_{q\alpha} + a_{r\beta} a_{p\beta}^\dagger a_{q\beta} + 2a_{r\alpha} a_{p\alpha}^\dagger a_{q\beta} + a_{r\beta} a_{p\alpha}^\dagger a_{q\alpha} + a_{r\beta} a_{p\beta}^\dagger a_{q\beta} \\ -2a_{r\beta} a_{p\beta}^\dagger a_{q\alpha} - a_{r\alpha} a_{p\alpha}^\dagger a_{q\alpha} + a_{r\alpha} a_{p\beta}^\dagger a_{q\beta} - a_{r\alpha} a_{p\alpha}^\dagger a_{q\alpha} - a_{r\alpha} a_{p\beta}^\dagger a_{q\beta} \end{pmatrix}^{[1/2]}$$

$$= \sqrt{2} \begin{pmatrix} a_{r\beta} a_{p\beta}^\dagger a_{q\beta} + a_{r\alpha} a_{p\alpha}^\dagger a_{q\beta} \\ -a_{r\beta} a_{p\beta}^\dagger a_{q\alpha} - a_{r\alpha} a_{p\alpha}^\dagger a_{q\alpha} \end{pmatrix}^{[1/2]}$$

Triplet times triplet

$$X^{[0]} = \left[\left(a_p^\dagger\right)^{[1/2]} \otimes_{[1]} \left(a_q^\dagger\right)^{[1/2]}\right] \otimes_{[0]} \left[\left(a_r\right)^{[1/2]} \otimes_{[1]} \left(a_s\right)^{[1/2]}\right]$$

$$= \begin{pmatrix} a_{p\alpha}^\dagger a_{q\alpha}^\dagger \\ \frac{1}{\sqrt{2}}\left(a_{p\alpha}^\dagger a_{q\beta}^\dagger + a_{p\beta}^\dagger a_{q\alpha}^\dagger\right) \\ a_{p\beta}^\dagger a_{q\beta}^\dagger \end{pmatrix}^{[1]} \otimes_{[0]} \begin{pmatrix} a_{r\beta} a_{s\beta} \\ -\frac{1}{\sqrt{2}}\left(a_{r\beta} a_{s\alpha} + a_{r\alpha} a_{s\beta}\right) \\ a_{r\alpha} a_{s\alpha} \end{pmatrix}^{[1]}$$

$$= \frac{1}{\sqrt{3}}\left(a_{p\alpha}^\dagger a_{q\alpha}^\dagger a_{r\alpha} s_{s\alpha} + \frac{1}{2}\left(a_{p\alpha}^\dagger a_{q\beta}^\dagger + a_{p\beta}^\dagger a_{q\alpha}^\dagger\right)\left(a_{r\beta} a_{s\alpha} + a_{r\alpha} a_{s\beta}\right) + a_{p\beta}^\dagger a_{q\beta}^\dagger a_{r\beta} a_{s\beta}\right)$$

$$Y^{[0]} = \left[\left(a_p^\dagger\right)^{[1/2]} \otimes_{[0]} \left(a_q^\dagger\right)^{[1/2]}\right] \otimes_{[0]} \left[\left(a_r\right)^{[1/2]} \otimes_{[0]} \left(a_s\right)^{[1/2]}\right]$$

$$= \frac{1}{\sqrt{2}}\left(a_{p\alpha}^\dagger a_{q\beta}^\dagger - a_{p\beta}^\dagger a_{q\alpha}^\dagger\right)^{[0]} \otimes_{[0]} \frac{1}{\sqrt{2}}\left(-a_{r\beta} a_{s\alpha} + a_{r\alpha} a_{s\beta}\right)^{[0]}$$

$$= \frac{1}{2}\left(a_{p\alpha}^\dagger a_{q\beta}^\dagger - a_{p\beta}^\dagger a_{q\alpha}^\dagger\right)\left(-a_{r\beta} a_{s\alpha} + a_{r\alpha} a_{s\beta}\right)$$

Using

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

we have

$$\sqrt{3}X^{[0]} + Y^{[0]} = a_{p\alpha}^\dagger a_{q\alpha}^\dagger a_{r\alpha} s_{s\alpha} + a_{p\beta}^\dagger a_{q\beta}^\dagger a_{r\beta} a_{s\beta} + a_{p\alpha}^\dagger a_{q\beta}^\dagger a_{r\alpha} a_{s\beta} + a_{p\beta}^\dagger a_{q\alpha}^\dagger a_{r\beta} a_{s\alpha}$$

$$= \sum_{\sigma\sigma'} a_{p\sigma}^\dagger a_{q\sigma'}^\dagger a_{r\sigma} s_{s\sigma'}$$

Another case

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

Using

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

we have

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

S Term

From second singlet formula we have

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

R Term

This is the same as the S term. Note that in the expression for $\hat{R}$, we have a $\otimes_{[0]}$, this is because in the original spatial expression there is a summation over σ. Then there is a $[0] \otimes_{[1/2]} [1/2]$, which will not produce any extra coefficients.

AP Term

Using definition

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

We have

$$\sum_{ik\in L}\left[\sqrt{3}\left(\hat{A}_{ik}\right)^{[1]}\otimes_{[0]}\left(\hat{P}^R_{ik}\right)^{[1]}+\left(\hat{A}_{ik}\right)^{[0]}\otimes_{[0]}\left(\hat{P}^R_{ik}\right)^{[0]}\right]$$

$$=\sum_{ik\in L,jl\in R}v_{ijkl}\left[\sqrt{3}\left[\left(a_i^\dagger\right)^{[\frac{1}{2}]}\otimes_{[1]}\left(a_k^\dagger\right)^{[\frac{1}{2}]}\right]\otimes_{[0]}\left[\left(a_j\right)^{[\frac{1}{2}]}\otimes_{[1]}\left(a_l\right)^{[\frac{1}{2}]}\right]+\left[\left(a_i^\dagger\right)^{[\frac{1}{2}]}\otimes_{[0]}\left(a_k^\dagger\right)^{[\frac{1}{2}]}\right]\otimes_{[0]}\left[\left(a_j\right)^{[\frac{1}{2}]}\otimes_{[0]}\left(a_l\right)^{[\frac{1}{2}]}\right]\right]$$

$$=\sum_{ik\in L,jl\in R}v_{ijkl}\left[\sum_{\sigma\sigma'}a^\dagger_{i\sigma}a^\dagger_{k\sigma'}a_{j\sigma}a_{l\sigma'}\right]=-\sum_{ik\in L,jl\in R,\sigma\sigma'}v_{ijkl}a^\dagger_{i\sigma}a^\dagger_{k\sigma'}a_{l\sigma'}a_{j\sigma}$$

Note that in last step, we can anticommute $a_{l\sigma'}, a_{j\sigma}$ because it's assumed that in the σ summation, when $j=l, \sigma\neq\sigma'$. Otherwise there will be two a operators acting on the same site and the contribution is zero.

BQ Term

In spatial expression, this term is $BQ-B'Q'$. Now $-\sqrt{3}Z^{[0]}+W^{[0]}$ gives $B'Q'$. And $2W^{[0]}$ gives BQ. Therefore,

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

This looks like $\hat{A}\hat{P}$ term, but without $\frac{1}{2}$ and $h.c.$. But this is not correct, because the definition of Q, Q' is not equivalent due to the index order in v_{ijkl}. So they will give different $W^{[0]}$. Instead we have (note that $\left(\hat{B}_{ij}\right)^{[0]}=\left(\hat{B}'_{ij}\right)^{[0]}$)

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

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

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

Normal/Complementary Partitioning

Note that

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

Therefore,

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

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

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

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

So define

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

Here $\frac{\sqrt{2}}{4}$ should be understood as $\frac{1}{2}\cdot\frac{1}{\sqrt{2}}$. The $\frac{1}{2}$ is the same as spatial case, and $\frac{1}{\sqrt{2}}$ is because the expected $\sqrt{2}$ factor is not added for the $\hat{R}$ term.

Operator Exchange factors

Here we consider fermion and SU(2) exchange factors together. From $j_2 = 1/2$ CG factors

$$\left\langle j_1 \ \left(M - \frac{1}{2}\right) \ \frac{1}{2} \ \frac{1}{2} \middle| \left(j_1 \pm \frac{1}{2}\right) \ M \right\rangle = \pm \sqrt{\frac{1}{2}\left(1 \pm \frac{M}{j_1 + \frac{1}{2}}\right)}$$

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

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

$$\left\langle \frac{1}{2} \ \left(-\frac{1}{2}\right) \ \frac{1}{2} \ \frac{1}{2} \middle| \left(\frac{1}{2} \pm \frac{1}{2}\right) \ 0 \right\rangle = \pm \sqrt{\frac{1}{2}}$$

$$\left\langle \frac{1}{2} \ \frac{1}{2} \ \frac{1}{2} \ \left(-\frac{1}{2}\right) \middle| \left(\frac{1}{2} \pm \frac{1}{2}\right) \ 0 \right\rangle = \sqrt{\frac{1}{2}}$$

The exchange factor formula is

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

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

$$\mathrm{P}^{\mathrm{exchange}}(\tfrac{1}{2}, \tfrac{1}{2}, 0) = (-1) \frac{\left\langle \frac{1}{2} \ \frac{1}{2} \ \frac{1}{2} \ \left(-\frac{1}{2}\right) \middle| 0 \ 0 \right\rangle}{\left\langle \frac{1}{2} \ \left(-\frac{1}{2}\right) \ \frac{1}{2} \ \frac{1}{2} \middle| 0 \ 0 \right\rangle} = (-1) \frac{\sqrt{\frac{1}{2}}}{-\sqrt{\frac{1}{2}}} = 1$$

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

$$\mathrm{P}^{\mathrm{exchange}}(\tfrac{1}{2}, \tfrac{1}{2}, 1) = (-1) \frac{\left\langle \frac{1}{2} \ \frac{1}{2} \ \frac{1}{2} \ \left(-\frac{1}{2}\right) \middle| 1 \ 0 \right\rangle}{\left\langle \frac{1}{2} \ \left(-\frac{1}{2}\right) \ \frac{1}{2} \ \frac{1}{2} \middle| 1 \ 0 \right\rangle} = (-1) \frac{\sqrt{\frac{1}{2}}}{\sqrt{\frac{1}{2}}} = -1$$

From CG factors

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

we have

$$\mathrm{P}^{\mathrm{exchange}}(1, 1, 0) = (+1) \frac{\left\langle 1 \ 1 \ \ 1 \ -1 \middle| 0 \ 0 \right\rangle}{\left\langle 1 \ -1 \ 1 \ 1 \middle| 0 \ 0 \right\rangle} = (+1) \frac{\frac{(-1)^0}{\sqrt{3}}}{\frac{(-1)^2}{\sqrt{3}}} = 1$$

we have

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

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

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

The operators required in right block are

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

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.

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

Now the operators required in left block are

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

The operators required in right block are

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

The total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$
N_{CN} = 1 + 1 + 2K_L + 2K_R + 4K_R^2 + 2K_R^2 = 6K_R^2 + 2K + 2
$$

Blocking

The enlarged left/right block is denoted as $L*/R*$. Make sure that all L operators are to the left of $*$ operators. (The exchange factor for this is -1 for doublet $\otimes$ triplet and +1 doublet $\otimes$ singlet.)

First we have

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

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

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

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

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

We have

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

After reordering of terms

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

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)

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

we have

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

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

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

Similarly,

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

Number of terms

$$
\begin{aligned}
N_{R^\prime,NC} &= (2 + 4K_L + 4K_L^2)K_R + (2 + 4 + 4K_R)K_L = 4K_L^2 K_R + 8K_L K_R + 2K + 4K_L \\
N_{R^\prime,CN} &= (2 + 4K_L + 4)K_R + (2 + 4K_R^2 + 4K_R)K_L = 4K_R^2 K_L + 8K_R K_L + 2K + 4K_R
\end{aligned}
$$

Blocking of other complementary operators is straightforward

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

and

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

and

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

Middle-Site Transformation

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

6.1.3 DMRG Quantum Chemistry Hamiltonian in Unrestricted Spatial Orbitals

Hamiltonian

The quantum chemistry Hamiltonian is written as follows

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

where

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

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

Partitioning in Spatial Orbitals

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

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

where the normal and complementary operators are defined by

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

Note that we need to move all on-site interaction into local Hamiltonian, so that when construction interaction terms in Hamiltonian, operators anticommute (without giving extra constant terms).

Define

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

Then we have

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

Normal/Complementary Partitioning

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

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

The operators required in right block are

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

Assuming that there are K sites in total, and K_L/K_R sites in left/right block (optimally, $K_L \le K_R$), the total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$N_{NC} = 1 + 1 + 4K_L + 4K_R + 8K_L^2 + 2K_L^2 + 4K_L^2 = 14K_L^2 + 4K + 2$$

Complementary/Normal Partitioning

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

Now the operators required in left block are

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

The operators required in right block are

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

The total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$N_{CN} = 1 + 1 + 4K_R + 4K_L + 8K_R^2 + 2K_R^2 + 4K_R^2 = 14K_R^2 + 4K + 2$$

Blocking

The enlarged left/right block is denoted as $L*/R*$. Make sure that all L operators are to the left of $*$ operators.

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

Now there are two possibilities. In NC partition, in L we have $A, A^\dagger, B, B'$ and in $*$ we have $P, P^\dagger, Q, Q'$. In CN partition, the opposite is true. Therefore, we have

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

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

Simplified Form

Define

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

we have N/C form

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

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

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

The operators required in right block are

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

Assuming that there are K sites in total, and K_L/K_R sites in left/right block (optimally, $K_L \le K_R$), the total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$N_{NC} = 1 + 1 + 4K_L + 4K_R + 8K_L^2 + 4K_L^2 = 12K_L^2 + 4K + 2$$

and C/N form

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

Now the operators required in left block are

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

The operators required in right block are

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

The total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$N_{CN} = 1 + 1 + 4K_R + 4K_L + 8K_R^2 + 4K_R^2 = 12K_R^2 + 4K + 2$$

Then for blocking

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

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

6.1.4 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

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

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: $LLL, LRR, RLR, RRL, LLR, LRL, RLL, RRR$.

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

Let

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

we have

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

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

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

where

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

For $m \in R$:

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

In summary

$$\hat{H} = \hat{H}^{ML} \otimes \hat{1}^R + \sum_{m\in L,\sigma} a^\dagger_{m\sigma} \otimes \hat{H}^R_{m\sigma} + \frac{1}{2}\sum_{mj\in L,\sigma\sigma'} \hat{A}_{mj,\sigma\sigma'}\hat{P}^R_{mj,\sigma\sigma'} + \frac{1}{2}\sum_{mj\in L,\sigma\sigma'} \hat{B}_{mj,\sigma\sigma'}\hat{Q}''^R_{mj,\sigma\sigma'} + \frac{1}{2}\sum_{k\in R,\sigma'} \hat{P}^{ML}_{k\sigma'}a^\dagger_{k\sigma'} + \frac{1}{2}\sum_{k\in R,\sigma}$$
$$- \sum_{n\in R,\sigma} \hat{H}^L_{n\sigma} \otimes a^\dagger_{n\sigma} + \hat{1}^L \otimes \hat{H}^{MR} - \frac{1}{2}\sum_{j\in L,\sigma'} a^\dagger_{j\sigma'}\hat{P}^{MR}_{j,\sigma'} - \frac{1}{2}\sum_{j\in L,\sigma'} a_{j\sigma'}\hat{Q}^{MR}_{j,\sigma'} + \frac{1}{2}\sum_{nk\in R,\sigma\sigma'} \hat{P}^L_{nk,\sigma\sigma'}\hat{A}_{nk,\sigma\sigma'} + \frac{1}{2}\sum_{nk\in R,\sigma\sigma'} \hat{Q}''^L_{nk,\sigma\sigma'}$$

The operators required in left block are

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

The total number of operators is

$$N = 1 + 2K_{ML} + 4K_{ML}K_L + 4K_{ML}K_L + 2K_R + 2K_R + 2K_{MR} + 1 + 2K_L + 2K_L + 4K_{MR}K_R + 4K_{MR}K_R$$
$$= 2 + 2K_M + 4K + 8K_{ML}K_L + 8K_{MR}K_R$$

Reordered left and right block operators

$$L = \left\{\hat{H}^{ML}, \hat{1}^L, a^\dagger_{m\sigma}, \hat{H}^L_{n\sigma}, a^\dagger_{j\sigma'}, a_{j\sigma'}, \hat{P}^{ML}_{k\sigma'}, \hat{Q}^{ML}_{k\sigma'}, \hat{A}_{mj,\sigma\sigma'}, \hat{B}_{mj,\sigma\sigma'}, \hat{P}^L_{nk,\sigma\sigma'}, \hat{Q}''^L_{nk,\sigma\sigma'}\right\} \quad (m, j \in L,\ n, k \in R)$$
$$R = \left\{\hat{1}^R, \hat{H}^{MR}, \hat{H}^R_{m\sigma}, a^\dagger_{n\sigma}, \hat{P}^{MR}_{j,\sigma'}, \hat{Q}^{MR}_{j,\sigma'}, a^\dagger_{k\sigma'}, a_{k\sigma'}, \hat{P}^R_{mj,\sigma\sigma'}, \hat{Q}''^R_{mj,\sigma\sigma'}, \hat{A}_{nk,\sigma\sigma'}, \hat{B}_{nk,\sigma\sigma'}\right\}$$

Now let

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

we have

$$L = \left\{\hat{H}^{ML}, \hat{1}^L, a^\dagger_{j\sigma'}, a_{j\sigma'}, \hat{R}^{ML}_{k\sigma'}, \hat{S}^{ML}_{k\sigma'}, \hat{A}_{mj,\sigma\sigma'}, \hat{B}_{mj,\sigma\sigma'}, \hat{P}^L_{nk,\sigma\sigma'}, \hat{Q}''^L_{nk,\sigma\sigma'}\right\} \quad (m, j \in L,\ n, k \in R)$$
$$R = \left\{\hat{1}^R, \hat{H}^{MR}, \hat{R}^{MR}_{j,\sigma'}, \hat{S}^{MR}_{j,\sigma'}, a^\dagger_{k\sigma'}, a_{k\sigma'}, \hat{P}^R_{mj,\sigma\sigma'}, \hat{Q}''^R_{mj,\sigma\sigma'}, \hat{A}_{nk,\sigma\sigma'}, \hat{B}_{nk,\sigma\sigma'}\right\}$$

The total number of operators is

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

Blocking

$$\hat{P}^{ML*}_{k\sigma'} = \sum_{m\in L*,\sigma} a^\dagger_{m\sigma}\hat{P}^{L*}_{mk,\sigma\sigma'} = \sum_{m\in L*,\sigma} a^\dagger_{m\sigma} \sum_{jl\in L*} v_{mjkl,\sigma\sigma'}a_{l\sigma'}a_{j\sigma}$$
$$= \hat{P}^{ML}_{k\sigma'} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{P}^{M*}_{k\sigma'} + \sum_{m\in *,\sigma} a^\dagger_{m\sigma}\hat{P}^L_{mk,\sigma\sigma'} + \sum_{m\in L,\sigma} a^\dagger_{m\sigma} \sum_{j\in *,l\in L} v_{mjkl,\sigma\sigma'}a_{l\sigma'}a_{j\sigma} + \sum_{m\in L,\sigma} a^\dagger_{m\sigma} \sum_{j\in L,l\in *} v_{mjkl,\sigma\sigma'}a_{l\sigma'}a_{j\sigma}$$
$$+ \sum_{m\in L,\sigma} a^\dagger_{m\sigma}\hat{P}^*_{mk,\sigma\sigma'} + \sum_{m\in *,\sigma} a^\dagger_{m\sigma} \sum_{j\in *,l\in L} v_{mjkl,\sigma\sigma'}a_{l\sigma'}a_{j\sigma} + \sum_{m\in *,\sigma} a^\dagger_{m\sigma} \sum_{j\in L,l\in *} v_{mjkl,\sigma\sigma'}a_{l\sigma'}a_{j\sigma}$$
$$= \hat{P}^{ML}_{k\sigma'} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{P}^{M*}_{k\sigma'} + \sum_{m\in *,\sigma} \hat{P}^L_{mk,\sigma\sigma'}a^\dagger_{m\sigma} + \sum_{m\in L,\sigma} a^\dagger_{m\sigma}\hat{P}^*_{mk,\sigma\sigma'} + \sum_{ml\in L,j\in *,\sigma} v_{mjkl,\sigma\sigma'}a^\dagger_{m\sigma}a_{l\sigma'}a_{j\sigma} - \sum_{mj\in L,l\in *,\sigma} v_{mjkl}$$
$$- \sum_{mj\in *,l\in L,\sigma} v_{mjkl,\sigma\sigma'}a_{l\sigma'}a^\dagger_{m\sigma}a_{j\sigma} + \sum_{ml\in *,j\in L,\sigma} v_{mjkl,\sigma\sigma'}a_{j\sigma}a^\dagger_{m\sigma}a_{l\sigma'}$$
$$= \hat{P}^{ML}_{k\sigma'} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{P}^{M*}_{k\sigma'} + \sum_{m\in *,\sigma} \hat{P}^L_{mk,\sigma\sigma'}a^\dagger_{m\sigma} + \sum_{m\in L,\sigma} a^\dagger_{m\sigma}\hat{P}^*_{mk,\sigma\sigma'} + \sum_{ml\in L,j\in *,\sigma} v_{mjkl,\sigma\sigma'}a^\dagger_{m\sigma}a_{l\sigma'}a_{j\sigma} - \sum_{ml\in L,j\in *,\sigma} v_{mlkj}$$
$$- \sum_{mj\in *,l\in L,\sigma} v_{mjkl,\sigma\sigma'}a_{l\sigma'}a^\dagger_{m\sigma}a_{j\sigma} + \sum_{mj\in *,l\in L,\sigma} v_{mlkj,\sigma\sigma'}a_{l\sigma}a^\dagger_{m\sigma}a_{j\sigma'}$$
$$= \hat{P}^{ML}_{k\sigma'} \otimes \hat{1}^* + \hat{1}^L \otimes \hat{P}^{M*}_{k\sigma'} + \sum_{m\in *,\sigma} \hat{P}^L_{mk,\sigma\sigma'}a^\dagger_{m\sigma} + \sum_{m\in L,\sigma} a^\dagger_{m\sigma}\hat{P}^*_{mk,\sigma\sigma'}$$
$$+ \sum_{ml\in L,j\in *,\sigma} v_{mjkl,\sigma\sigma'}\hat{B}_{ml,\sigma\sigma'}a_{j\sigma} - \sum_{ml\in L,j\in *,\sigma} v_{mlkj,\sigma\sigma'}\hat{B}_{ml,\sigma\sigma}a_{j\sigma'} + \sum_{mj\in *,l\in L,\sigma} v_{mlkj,\sigma\sigma'}a_{l\sigma}\hat{B}_{mj,\sigma\sigma'} - \sum_{mj\in *,l\in L,\sigma} v_{mjk}$$

and

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

and

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

after simplification

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

For P, Q, we have

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

6.1.5 DMRG Quantum Chemistry Hamiltonian in Spin Orbitals

Hamiltonian

The quantum chemistry Hamiltonian is written as follows

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

where $ijkl$ are spin orbital indices, and

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

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

$$\begin{aligned}
t_{ij} &= t_{ji} \\
v_{ijkl} &= v_{klij} = v_{jilk} = v_{lkji} = v_{jikl} = v_{ijlk} = v_{lkij} = v_{klji}
\end{aligned}$$

Partitioning in Spin Orbitals

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

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

where the normal and complementary operators are defined by

$$\begin{aligned}
\hat{R}_i^{L/R} &= \frac{1}{2} \sum_{j \in L/R} t_{ij} a_j + \sum_{jkl \in L/R} v_{ijkl} a_k^\dagger a_l a_j, \\
\hat{A}_{ik} &= a_i^\dagger a_k^\dagger, \\
\hat{B}_{ij} &= a_i^\dagger a_j, \\
\hat{P}_{ik}^R &= \sum_{jl \in R} v_{ijkl} a_l a_j, \\
\hat{Q}_{ij}^R &= \sum_{kl \in R} \big(v_{ijkl} - v_{ilkj} \big) a_k^\dagger a_l
\end{aligned}$$

Note that we need to move all on-site interaction into local Hamiltonian, so that when construction interaction terms in Hamiltonian, operators anticommute (without giving extra constant terms).

Derivation

First consider one-electron term. ij indices have only two possibilities: i left, j right, or i right, j left. Index i must be associated with creation operator. So the second case is the Hermitian conjugate of the first case. Namely, consider $\hat{S}_i^{L/R}$ as the one-body part of $\hat{R}_i^{L/R}$, we have

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

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

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

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

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

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

$$\frac{1}{2}\sum_{j\in L,ikl\in R} v^*_{ijkl}a^\dagger_j a^\dagger_l a_k a_i = \frac{1}{2}\sum_{k\in L,ijl\in R} v^*_{lkji}a^\dagger_k a^\dagger_i a_j a_l = \frac{1}{2}\sum_{k\in L,ijl\in R} v_{ijkl}a^\dagger_i a^\dagger_k a_l a_j$$

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

$$\frac{1}{2}\sum_{l\in L,ijk\in R} v^*_{ijkl}a^\dagger_j a^\dagger_l a_k a_i = \frac{1}{2}\sum_{i\in L,jkl\in R} v^*_{lkji}a^\dagger_k a^\dagger_i a_j a_l = \frac{1}{2}\sum_{i\in L,jkl\in R} v_{ijkl}a^\dagger_i a^\dagger_k a_l a_j$$

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

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

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,

$$\sum_{ik\in R,jl\in L} a^\dagger_i a^\dagger_k v_{ijkl}a_l a_j = \sum_{jl\in R,ik\in L} a^\dagger_j a^\dagger_l v_{jilk}a_k a_i = \sum_{ik\in L,jl\in R} v_{jilk}a^\dagger_j a^\dagger_l a_k a_i = \sum_{ik\in L,jl\in R} v^*_{ijkl}\Big(a^\dagger_i a^\dagger_k a_l a_j\Big)^\dagger$$

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.

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

First consider the second and third terms

$$\begin{aligned}
&\frac{1}{2}\sum_{ij\in L,kl\in R} v_{ijkl}a^\dagger_i a_j a^\dagger_k a_l + \frac{1}{2}\sum_{kl\in L,ij\in R} v_{ijkl}a^\dagger_i a_j a^\dagger_k a_l \\
=&\frac{1}{2}\sum_{ij\in L,kl\in R} v_{ijkl}a^\dagger_i a_j a^\dagger_k a_l + \frac{1}{2}\sum_{kl\in L,ij\in R} v_{ijkl}a^\dagger_k a_l a^\dagger_i a_j \\
=&\frac{1}{2}\sum_{ij\in L,kl\in R} v_{ijkl}a^\dagger_i a_j a^\dagger_k a_l + \frac{1}{2}\sum_{ij\in L,kl\in R} v_{klij}a^\dagger_i a_j a^\dagger_k a_l \\
=&\sum_{ij\in L,kl\in R} v_{ijkl}a^\dagger_i a_j a^\dagger_k a_l = \sum_{ij\in L} a^\dagger_i a_j \sum_{kl\in R} v_{ijkl}a^\dagger_k a_l = \sum_{ij\in L} \hat{B}_{ij}\hat{Q}^R_{ij'}
\end{aligned}$$

For the other two terms,

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

Then

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

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

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

The operators required in right block are

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

Assuming that there are K sites in total, and K_L/K_R sites in left/right block (optimally, $K_L \le K_R$), the total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$
N_{NC} = 1 + 1 + 2K_L + 2K_R + 2K_L^2 + K_L^2 = 3K_L^2 + 2K + 2
$$

Complementary/Normal Partitioning

$$\hat{H}^{CN} = \hat{H}^L \otimes \hat{1}^R + \hat{1}^L \otimes \hat{H}^R + \sum_{i\in L}\left(a_i^\dagger \hat{R}_i^R - a_i \hat{R}_i^{R\dagger}\right) + \sum_{i\in R}\left(\hat{R}_i^{L\dagger} a_i - \hat{R}_i^L a_i^\dagger\right) + \frac{1}{2}\sum_{jl\in R}\left(\hat{P}_{jl}^L \hat{A}_{jl} + \hat{P}_{jl}^{L\dagger}\hat{A}_{jl}^\dagger\right) + \sum_{kl\in R}\hat{Q}_{kl}^L \hat{B}_{kl}$$

Now the operators required in left block are

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

The operators required in right block are

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

The total number of operators (and also the number of terms in Hamiltonian with partition) in left or right block is

$$N_{CN} = 1 + 1 + 2K_R + 2K_L + 2K_R^2 + K_R^2 = 3K_R^2 + 2K + 2$$

Blocking

The enlarged left/right block is denoted as $L*/R*$. Make sure that all L operators are to the left of $*$ operators.

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

Now there are two possibilities. In NC partition, in L we have $A, A^\dagger, B, B'$ and in $*$ we have $P, P^\dagger, Q, Q'$. In CN partition, the opposite is true. Therefore, we have

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

Similarly,

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

Number of terms

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

Blocking of other complementary operators is straightforward

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

and

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

Middle-Site Transformation

When the sweep is performed from left to right, passing the middle site, we need to switch from NC partition to CN partition. The cost is $O(K^4/16)$. This happens only once in the sweep. The cost of one blocking procedure is $O(K_<^2 K_>)$, but there are K blocking steps in one sweep. So the cost for blocking in one sweep is $O(K K_<^2 K_>)$. Note that the most expensive part in the program should be the Hamiltonian step in Davidson, which scales as $O(K_<^2)$.

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

6.1.6 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,

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

Similarly,

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

Therefore,

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

Diagonal Spatial Two-Particle Density Matrix (Mixed Spin)

Using anticommutation relation

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

we have

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

Therefore,

$$\boxed{\sum_{\sigma\tau} \langle a^\dagger_{p\sigma} a_{p\tau} a^\dagger_{q\tau} a_{q\sigma} \rangle = -e_{pqpq} + 2\delta_{pq} E_{pq}}$$

INDEX

B

I

T