Using NECI

Contents

1 Using NECI **2**
- 1.1 Getting the Code . . . 2
  - 1.1.1 BitBucket Repository Access . . . 2
- 1.2 NECI Installation . . . 2
- 1.3 HDF5 . . . 3
- 1.4 Documentation . . . 4
- 1.5 Basic NECI Tutorial . . . 4
  - 1.5.1 Anatomy of an Input File . . . 5
  - 1.5.2 Running NECI . . . 6
  - 1.5.3 Checking Convergence and Analysing Results . . . 7
  - 1.5.4 Continuing a NECI Calculation . . . 11
  - 1.5.5 Final Steps . . . 13
- 1.6 Calculation inputs . . . 13
  - 1.6.1 SYSTEM Block . . . 14
  - 1.6.2 CALC Block . . . 20
  - 1.6.3 INTEGRAL Block . . . 31
  - 1.6.4 Development keywords . . . 31
  - 1.6.5 KP-FCIQMC Block . . . 31
  - 1.6.6 REALTIME Block . . . 33
  - 1.6.7 LOGGING Block . . . 34
  - 1.6.8 FCIMCStats output functions . . . 37
- 1.7 Output files . . . 37
  - 1.7.1 FCIMCStats . . . 38
- 1.8 Trial wave functions . . . 40
- 1.9 Sampling excited states . . . 40
- 1.10 Davidson RAS code . . . 41
- 1.11 Performing error analysis . . . 41
- 1.12 Using the NECI pylib . . . 42
- 1.13 Additional Reading with Technical Details . . . 43

1 Using NECI

Here you will find instructions on how to use NECI for FCIQMC calculations. We start with installation instructions, then give a quick overview in the form of a simple tutorial, and then dive into more detail on the individual keywords and capabilities of NECI. See the sidebar (if on the website) for navigation between sections.

If you use NECI, please be sure to cite Ref [7].

1.1 Getting the Code

There are two git repositories for NECI. The stable release is available publicly on github, here. The developer version is on a private repository on bitbucket, for which you need to be invited to see. The Github version is not updated as frequently, so if you wish to use the latest methods, contact the developers.

1.1.1 BitBucket Repository Access

To gain access, contact one of the repository administrators [Oskar Weser (o.weser@fkf.mpg.de) and Werner Dobrautz (w.dobrautz@fkf.mpg.de)] who will invite you. If you already have a bitbucket account let the repository administrators know the email address associated with your account.

You will receive an invitation email. Please accept this invitation, and create a bitbucket account as prompted if necessary.

To gain access to the NECI repository, an ssh key is required. This can be generated on any linux machine using the command[1]

```
ssh-keygen -t rsa -b 2048
```

This will create a private (`~/.ssh/id_rsa`) and a public key file (`~/.ssh/id_rsa.pub`).

The private key must be kept private. On the bitbucket homepage, go to account settings (accessible from the top-right of the main page), and navigate to "SSH keys". Click "Add key" and add the contents of the public key. This will give you access to the repository.

You can now clone the code into a new directory using the command

```
git clone git@bitbucket.org:neci_developers/neci.git [target_dir]
```

1.2 NECI Installation

Installation of NECI requires[2]

- git,
- cmake 3.12 or newer,
- BLAS/LAPACK,
- MPI 3,
- a Fortran compiler supporting Fortran 2003 features, and
- HDF5 (is optional, but strongly recommended and has to be explicitly switched off).

To get started, we must first clone the repository, with

```
git clone https://github.com/ghb24/NECI_STABLE.git
```

for the public release, or

[1] 'ssh-keygen' can also generate DSA keys. Some ssh clients and servers will reject DSA keys longer than 1024 bits, and 1024 bits is currently on the margin of being crackable. As such 2048 bit RSA keys are preferred. Top secret this code is. Probably. Apart from the master branch which hosted for all on github. And in molpro. And anyone that wants it obviously.

[2] If you are on a cluster, you may need to run a command similar to 'module load ifort mpi.intel'.

```
git clone https://<username>@bitbucket.org/neci_developers/neci.git
```

for the developer release (replace `<username>` with your bitbucket username). The directory of this repository will be referred to as "`your_code_directory`".

Next, create a directory (let's call it "`your_build_directory`"), then run cmake and then make:

```
mkdir build
cd build
cmake -DENABLE_HDF5=ON -DENABLE_BUILD_HDF5=ON ${your_code_directory}
make -j
```

> **NOTE**
> If you are making without HDF5, then set the option `-DENABLE_HDF5=OFF` instead.

Sometimes it is necessary to pass in the desired MPI compilers. In particular if several MPI implementations are availabe on the system, then `cmake` might pick up the wrong ones. To explicitly select the correct MPI pass

```
-DCMAKE_Fortran_COMPILER=... -DCMAKE_C_COMPILER=... -DCMAKE_CXX_COMPILER=...
```

with appropiate paths to `cmake`. In the case of GNU compilers that are already in your path it would look like this:

```
-DCMAKE_Fortran_COMPILER=`which mpifort` \
-DCMAKE_C_COMPILER=`which mpicc` \
-DCMAKE_CXX_COMPILER=`which mpicxx`
```

If you want to link against an existing HDF5 library instead of rebuilding it everytime, then follow the instructions in the next section on how to correctly compile HDF5 with parallel support and set `HDF5_ROOT` as environment variable pointing to the installation directory of HDF5. A full command would look like this:

```
HDF5_ROOT=~/lib/hdf5-1.8.20_gfort_mpi \
cmake -DENABLE_HDF5=ON \
-DCMAKE_Fortran_COMPILER=mpifort \
-DCMAKE_C_COMPILER=mpicc \
-DCMAKE_CXX_COMPILER=mpicxx ~/code/neci
```

1.3 HDF5

You may also need to build HDF5 yourself as a shared library, which speeds up the compilation process, since HDF5 does not have to be rebuilt for every new project.

> **NOTE** HDF5 should be built with the same set of compilers as NECI.

In this case, download and extract the program from the HDF5 group website (download link for v1.12), then build with parallel IO and Fortran Support enabled. You may replace the compilers if you wish, for example use GNU `mpifort` or Intel `mpiifort`.

The `cmake` command is:

```
cd your_build_directory
HDF5_SRC= # The HDF5 source
HDF5_ROOT= # Where it should be installed

cmake \
    -DCMAKE_INSTALL_PREFIX:PATH=${HDF5_ROOT} \
    -DHDF5_ENABLE_PARALLEL:BOOL=ON  \
    -DHDF5_BUILD_FORTRAN:BOOL=ON
    -DCMAKE_Fortran_COMPILER:PATH=`which mpifort` \
    -DCMAKE_C_COMPILER:PATH=`which mpicc` \
    ${HDF5_SRC}
make -j
make install
```

The `configure` command for older HDF5 versions is:

```
cd your_build_directory
cd your_build_directory
HDF5_SRC= # The HDF5 source
HDF5_ROOT= # Where it should be installed

FC=mpifort F9X=mpifort CC=mpicc ${HDF5_SRC}/configure \
    --prefix=${HDF5_ROOT} --enable-fortran --enable-fortran2003 --enable-parallel
make
make install
```

where you define `HDF5_SRC` and `HDF5_ROOT` appropriately. Then, before running `cmake` for NECI, run

```
export HDF5_ROOT= # where HDF5 was installed in the previous step
```

and proceed with the NECI installation as before,

```
mkdir build
cd build
cmake -DENABLE_HDF5=ON ${your_code_directory}
make -j
```

1.4 Documentation

If you wish to generate documentation for NECI, based on Ford, then when you build with CMake, you must also include the flag `-DENABLE_DOC=ON`, for example

```
mkdir build
cd build
cmake -DENABLE_DOC=ON ${your_code_directory}
make -j doc
```

Requirements to produce the docs are:

- pandoc,
- latexmk,
- pdflatex,
- biber, and
- a working internet connection (only for the first time, in order to get a custom-build of Ford).

This will not only generate this documentation in the form of a PDF, but also as a website, having in addition to the information in the PDF also automatically generated documentation from comments in the source files.

1.5 Basic NECI Tutorial

FCIQMC is not a blackbox method and as such may be daunting to first approach. In addition, there are many variants which excel in different kinds of problems. NECI has tools to make working with FCIQMC an easier task. While NECI is written predominantly in Fortran, in order to use it you need not know any programming language. In this tutorial, we assume an at least elementary understanding of the FCIQMC algorithm.[3]

The goal of this tutorial is to provide a practical, brief supplement to the full NECI user's guide, which serves us a much more detailed reference. We will use NECI to calculate the ground-state energy of the Nitrogen dimer in a (6,6) active space. We use the equilibrium geometry, 2.074 bohr radii.

NOTE
For this problem, FCIQMC will actually underperform compared to conventional Davidson CI. In addition, Davidson CI has no stochastic error (unlike FCIQMC), so it is strongly preferred for this problem. However, this is a relatively cheap example through which to get comfortable with NECI and FCIQMC. FCIQMC has better scaling behaviour, can be better parallelised, and benefits from sparsity. If you go to (e.g.) a (20, 20) CAS then FCIQMC outshines Davidson, which is not a feasible method for problems of this size.

First, we must generate the FCIDUMP file which contains the information about 1- and 2-electron integrals. NECI is a solver for the CI-problem and *not* a standalone quantum chemistry suite, and cannot do this itself. For this, choose any program that can generate these (e.g. PySCF, Molpro, Molcas). This has been done for you, and you may download the file here.

1.5.1 Anatomy of an Input File

In order to run a NECI calculation, we must create an input file. Here is an example for the FCIDUMP file provided, called `n2_neci.inp`.

```
# comments are given like this

# simple N2 FCIQMC calculation
# for more complex FCIQMC variations, see the keywords for additional options
# such as the Hubbard model, transcorrelated options, or GAS-CI

title

  # read integrals from FCIDUMP
  system read
    electrons 6
    nonuniformrandexcits pchb
  endsys

  calc
    nmcyc 10000
    # for reproducibility
    seed 8

    totalWalkers 50000
    tau 0.01 search

    # use the initiator method
    truncinitiator
    addtoinitiator 3

    methods
      method vertex fcimc
    endmethods

  endcalc

  logging
    hdf5-pops
  endlog

end
```

> **NOTE**
> these keywords are case insensitive.

All these keywords (and plenty more) are explained in the next section. For now, let's break down the structure of the input file.

- Comments can be added in the code with `#`. (A deprecated comment symbol found in legacy inputs is `(`. Please use `#`.)
- First, the actual input starts with `title`, which is mandatory, and must also end with `end` (i.e. wrap the program in this block).
- Next, we have the `system` block, which is also mandatory.
  - The `system` keyword has a mandatory argument which comes directly after it on the same line. Here, we use `read` (as in `system read`), as we are doing the FCIQMC calculation from an FCIDUMP file.
  - We must also specify the number of electrons in the system, with the `electrons` keyword, followed by the number of electrons. Since we are doing a (6,6) calculation, we have `electrons 6`.
  - We must terminate the `system` block with the `endsys` keyword so that NECI knows to stop looking for `system` keywords.

- Then, we have the mandatory `calc` block, which is necessarily terminated with `endcalc`. This block in particular has many options and potential keywords; here we use only a small subset.
  - We specify the number of iterations the FCIQMC calculation will do, with `nmcyc`. Specifically, we specify 10000 iterations.
  - We also specify a seed with the `seed` keyword. FCIQMC is a randomised algorithm, and setting the seed simply ensures we get the same result every time (useful for testing or checking stochastic effects, for example). This keyword is optional.
  - We also must include the `totalWalkers` keyword, followed by the target number of walkers. Once this number is reached, NECI will enter variable-shift mode; that is, the shift will vary so as to keep the number of walkers constant. We wish to have a statistically significant number of iterations where the number of walkers is roughly constant, as we will see. In this case, we have `totalWalkers 50000`.
  - We must also include `tau` which is the size of the time step per iteration. The additional keyword `search` is optional but useful for stability.
  - Another form of FCIQMC is i-FCIQMC, which uses "initiator" walkers to speed up the calculation.[5] This is optional, but generally recommended. Presence of the `truncinitiator` keyword in the `calc` block indicates i-FCIQMC, and `addToInitiator 3` means that any determinant with a population >= 3 will become an initiator.
  - Finally, as a mandatory subblock *inside* the calc block, we have the `methods` subblock, which necessarily ends with `endmethods`. Here, we simply specify what kind of calculation to do. We choose `method vertex fcimc`, which simply means to run an FCIQMC calculation.
- Finally, we have an *optional* `logging` block which ends with `endlog`. By default, NECI will keep track of the population of walkers in a "POPSFILE", which is by default ASCII. Here, we wish to have an HDF5 POPSFILE which is generally better-performing. To do this, inside the `logging` block we have the `hdf5-pops` keyword.

> **NOTE**
> You cannot use the `hdf5-pops` keyword if you did not build NECI with HDF5.

1.5.2 Running NECI

After building, the NECI executable will be in `path/to/neci/build/bin/neci` (e.g. if I installed NECI in my home directory, it would be `~/neci/build/bin/neci`).

NECI can be run directly as any executable:

```
path/to/neci/build/bin/neci n2_neci.inp
```

but parallel execution is usually desired. To run the above input file in parallel, we must use the respective MPI commands (`mpirun`, `mpiexec`, etc.)

```
mpirun -np 4 path/to/neci/build/bin/neci n2_neci.inp
```

where you can replace the `4` with however many processors with which you want to run (4 being a *very* modest number). This will print a lot to standard output, which you may wish to capture, e.g.

```
mpirun -np 4 ~/neci/build/bin/neci n2_neci.inp > n2_neci.out
```

> **NOTE**
> If you made a mistake somewhere in your file (for example, a typo), NECI tries to give useful error messages. However, MPI can sometimes interferes with this. Before doing a full calculation, you may want to run simply without `mpirun`, e.g. simply
>
> ```
> ~/neci/build/bin/neci n2_neci.inp > n2_neci.out
> ```
> If this starts to run without an error, then you may stop it and run with `mpirun`.

NOTE
NECI has a useful utility to dynamically change variables *while it is running*. To do this, create a file called `CHANGEVARS`, and input whatever keyword you wish to change, e.g. if you think `nmcyc 10000` is too small, enter into `CHANGEVARS` the text `nmcyc 20000` (or whatever else). Then, the value will be updated in the next iteration of NECI. This is helpful for interacting with a running simulation.

WARNING
It is generally not advisable to terminate a program with CTRL+C. Instead, use the above `CHANGEVARS` trick, to create a soft exit. Simply open/create `CHANGEVARS`, type in `SOFTEXIT` and save. Alternatively, you can do this in one line on the terminal with

```
echo SOFTEXIT > CHANGEVARS
```

This calculation should produce a few other files in the directory, namely:

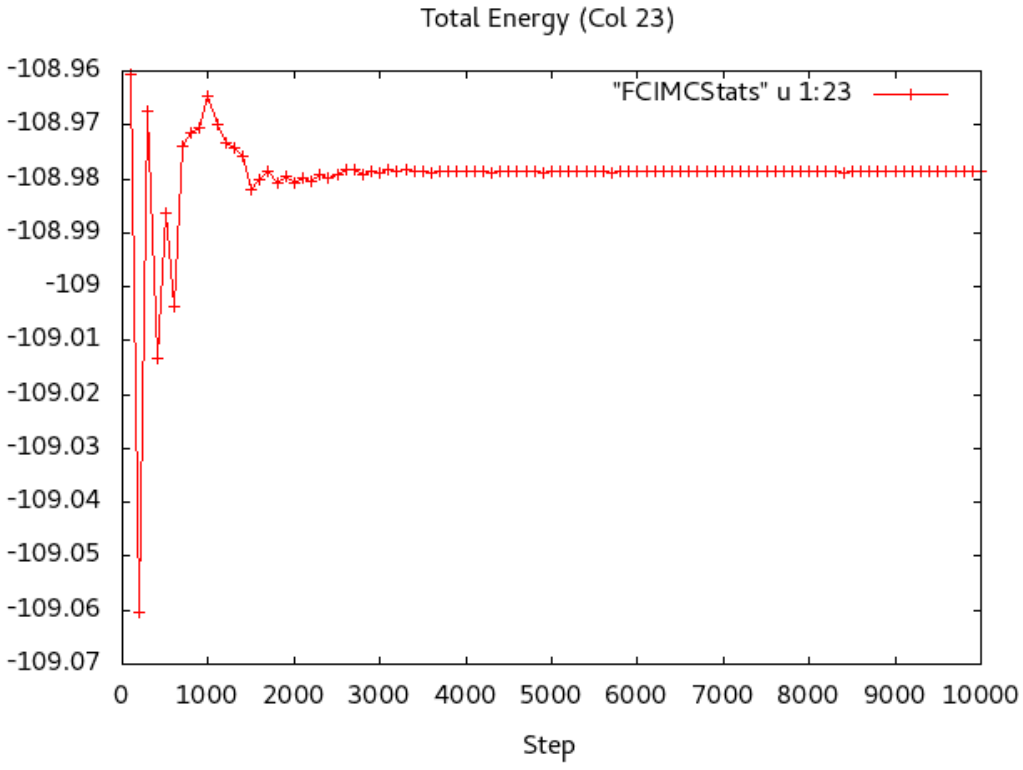
- `popsfile.h5` (or `POPSFILE` if you did not include `hdf5-pops`)
- `INITATORStats`
- `FCIMCStats`

The popsfile will be very useful in case we wish to continue running the simulation. `FCIMCStats` has columns of useful data, which we will explore now.

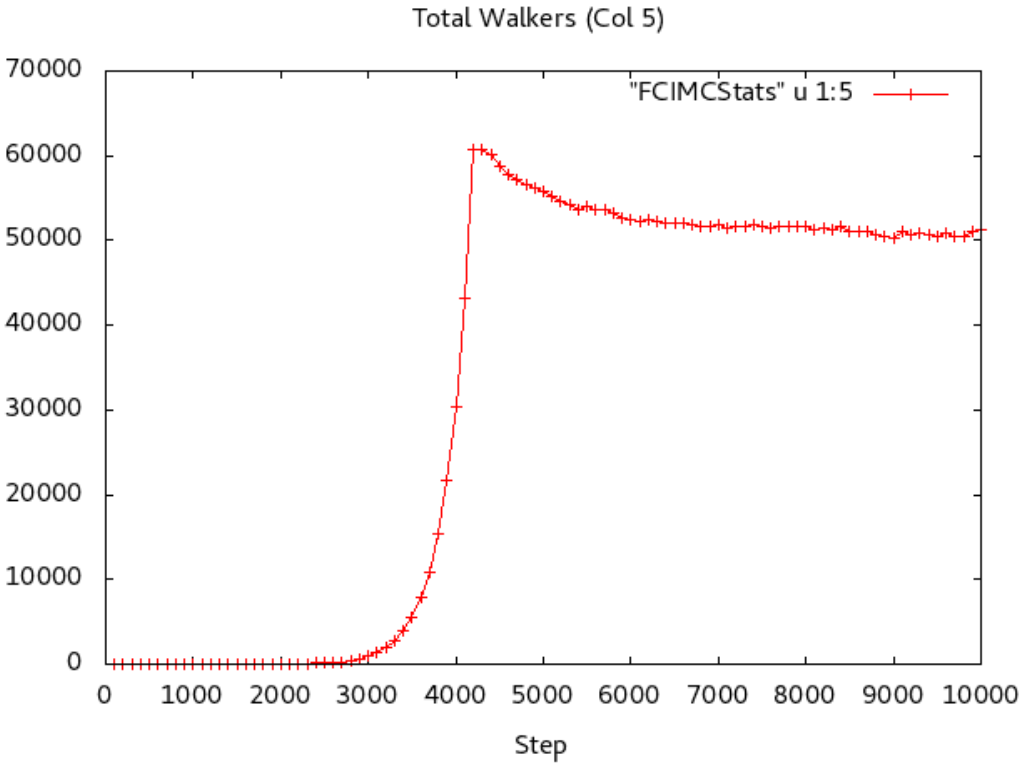
1.5.3 Checking Convergence and Analysing Results

The file `FCIMCStats` has several useful columns which you will want to plot to ensure convergence. To do this in one line, there is a convenience script here, which is run with gnuplot via `gnuplot plot_fcimcstats.plt` and will output plots of the most useful columns to a new `plots/` directory. Your results should look something like this:

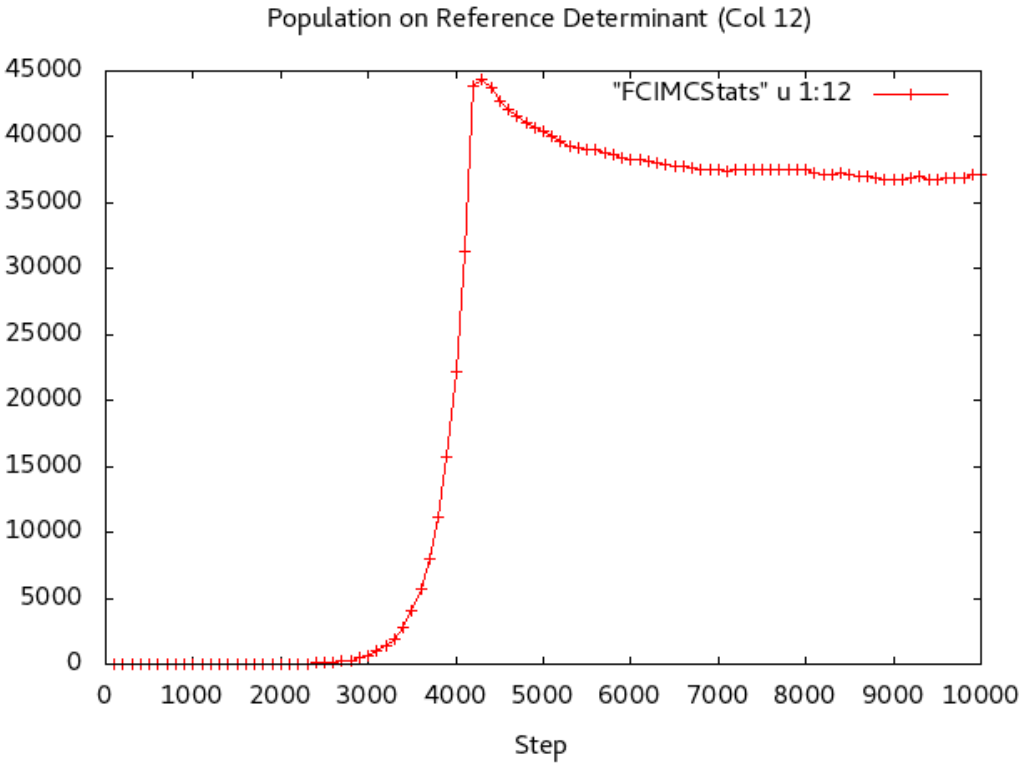
1.

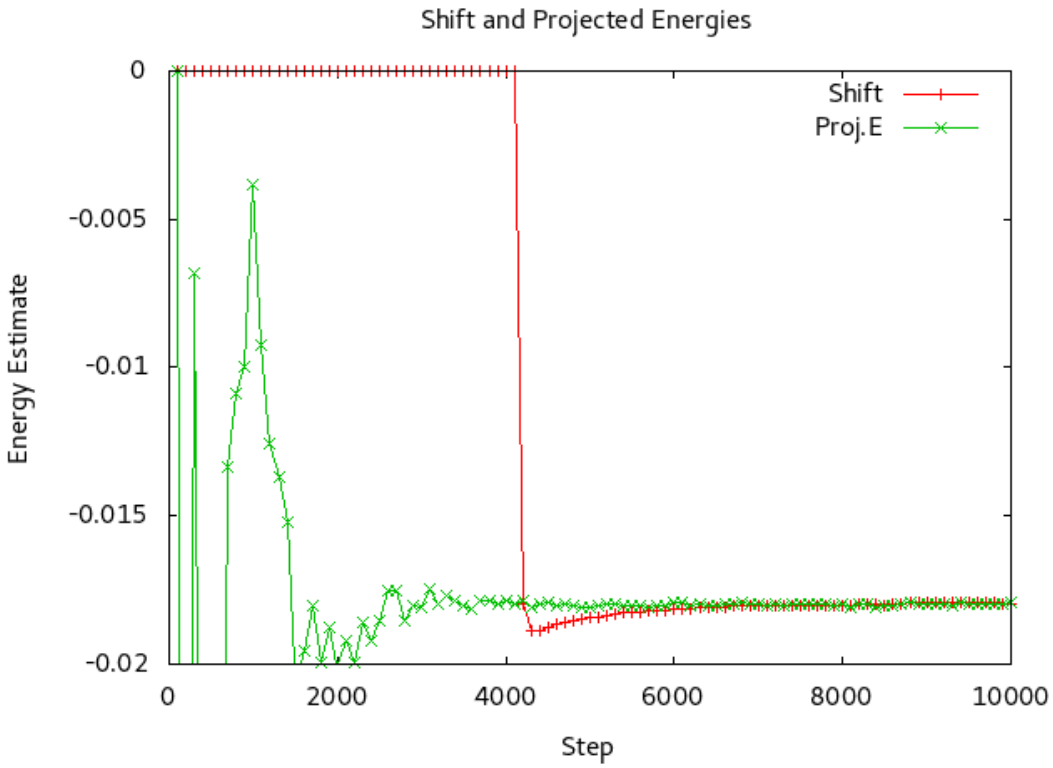


2.

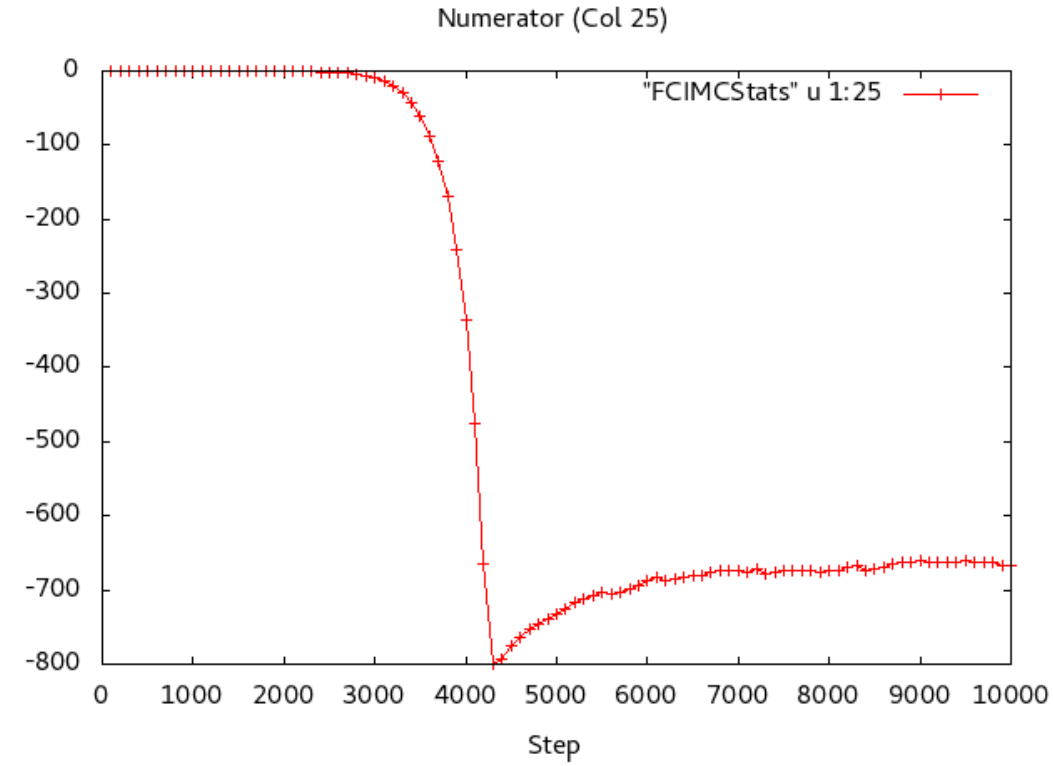

Total Walkers (Col 5)
"FCIMCStats" u 1:5
70000
60000
50000
40000
30000
20000
10000
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Step

3.

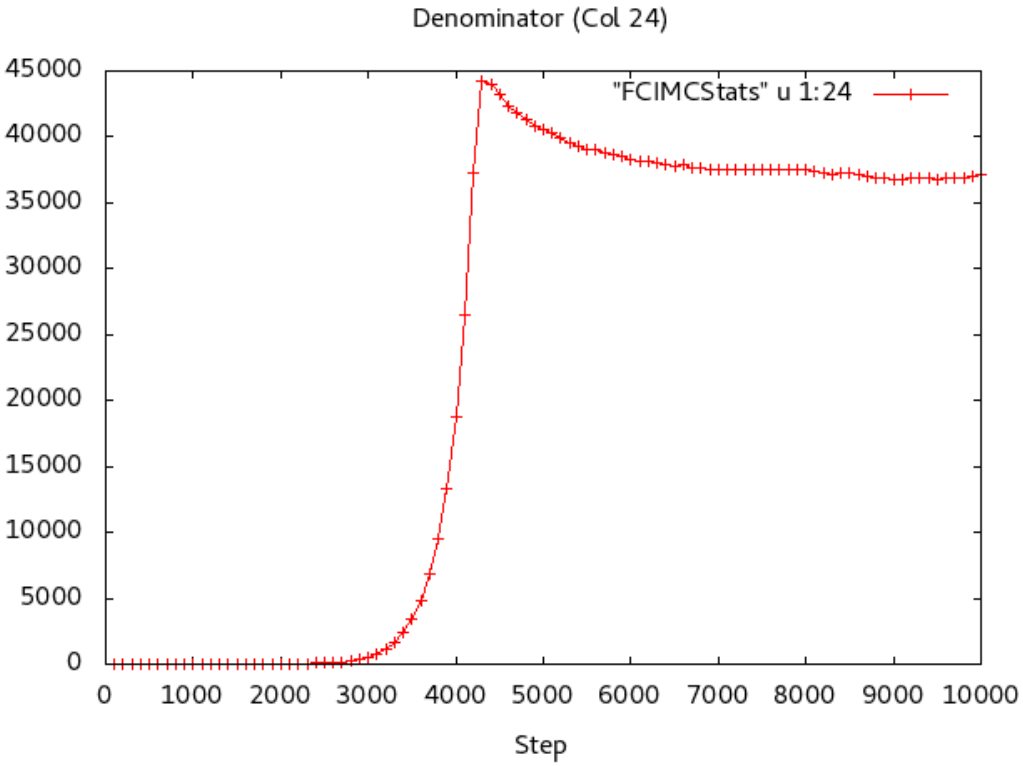

Population on Reference Determinant (Col 12)
"FCIMCStats" u 1:12
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Step


Shift and Projected Energies
Shift
Proj.E
Energy Estimate
Step
0
-0.005
-0.01
-0.015
-0.02
0
2000
4000
6000
8000
10000

4.


Numerator (Col 25)
"FCIMCStats" u 1:25
Step
0
-100
-200
-300
-400
-500
-600
-700
-800
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000

5.



6.

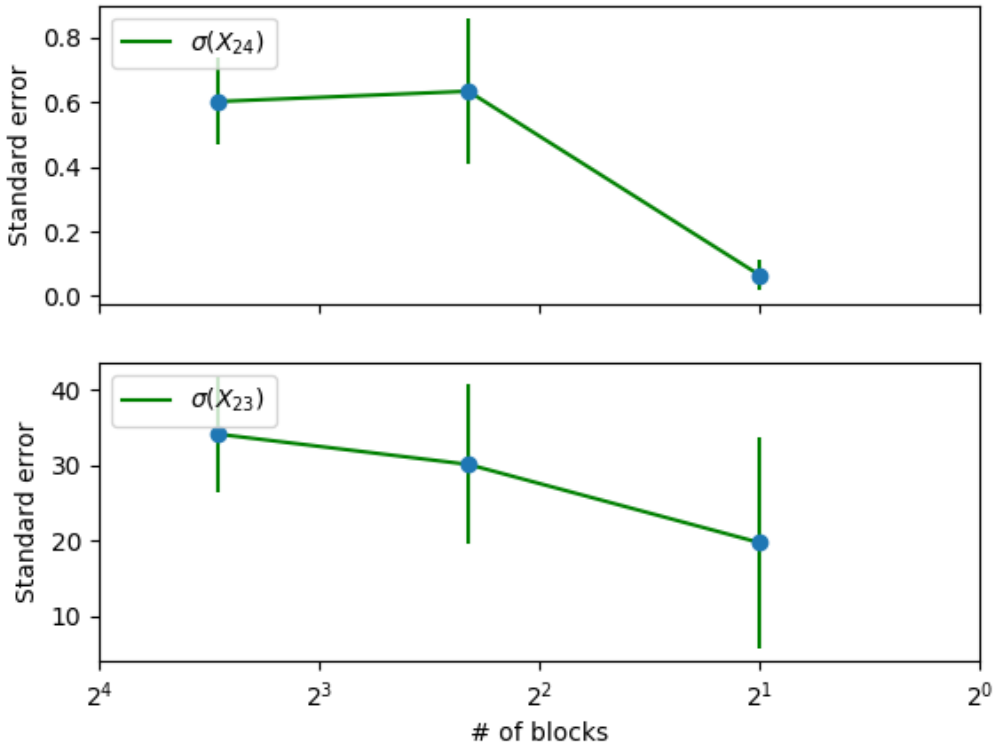
Plot (1) is the most immediately useful plot, as it gives you a quick estimate of the total energy from the calculation (namely, you can see we have around 108.98 Hartree). For all of these, we expect variable behaviour until the total walkers (plot (2)) reaches the target total walkers (as per our input file, 50000 in this case). Then, we want all six of these plots to roughly plateau, as they all do above. Furthermore, FCIQMC has a built-in consistency check whereby the energy can be calculated in two independent ways: namely, by the shift (which is only updated once the target number of walkers is reached) and the projected energy. These are plotted on top of each other in plot (4). As we see, they agree once we have run for a long enough time.

Once we are confident that all these plots exhibit plateaus for sufficiently large step numbers, we proceed with an error analysis. However, since FCIQMC calculations generally have correlated data, we cannot use standard error analysis, and here we use blocking analysis.[6] A script to do blocking analysis is included in the NECI repository: `path/to/neci/utils/blocking.py`.

Running the blocking analysis as

```
path/to/neci/utils/blocking.py -p 'plots/blocking.png' -f <numiter> -d24 -d23 -o/ FCIMCStats > stats
```

will output a blocking plot to the plots subdirectory, starting after `<numiter>` iterations, which should be chosen at a point where the plateaus in plots (5) and (6) (i.e. the numerator and denominator for the error estimate) are both stable. In this example, we might choose `<numiter>` to be 9000. Running this, we get the following plot.



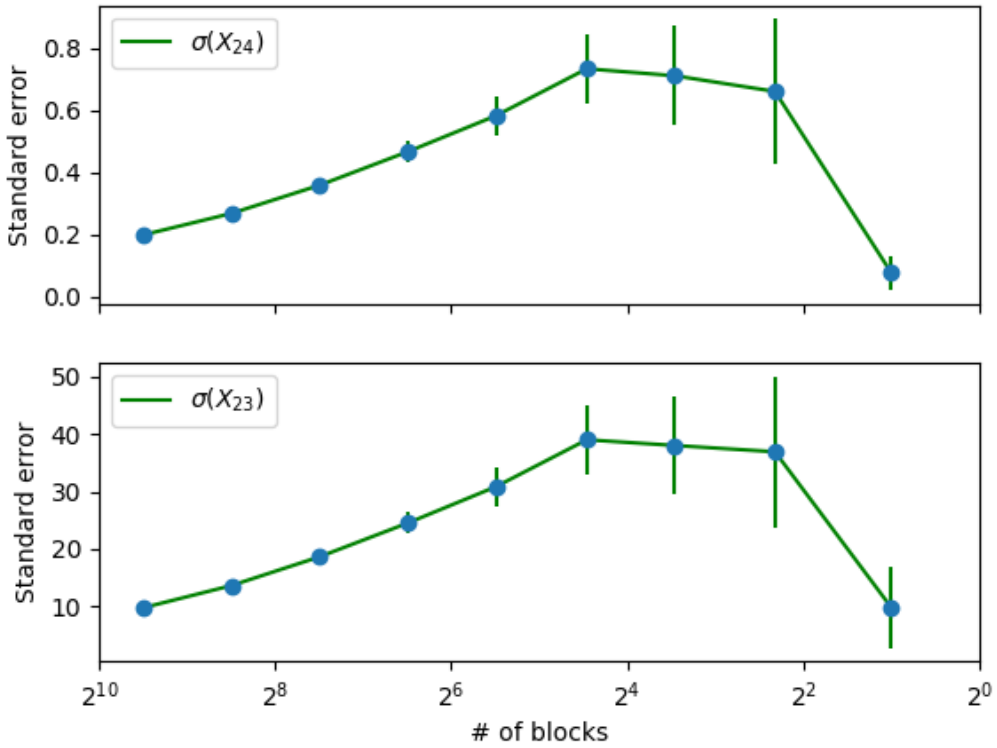
Consisting of only three points, and having no plateau, this indicates that *we have not yet converged our FCIQMC calculation reliably.* That is, if all the above 6 plots indicate convergence but the blocking analysis has no plateau (as in this example), it is most likely that you must continue the calculation to get more data.

1.5.4 Continuing a NECI Calculation

In order to continue a NECI calculation (for example, if like in this example you have done a calculation only to find you do not have enough data), simply take the same NECI input as above, but add into the calc the `readPops` command, which indicates that NECI must read the popsfile previously created. You may also wish to increase the number of iterations `nmcyc`, e.g.

```
title
  ...
  calc
  readPops
  nmcyc 70000
  ...
end
```

This will add data into the previous FCIMCStats. After you have run this the same way as described above, repeat the blocking analysis. The plots will all still have well-defined plateaus, but the blocking analysis will result in something like this:



This time we see a clear plateau. The last point in this example indicates an excessively large blocking length (resulting in the estimate of the standard having too much noise). This is expected behaviour (but only in the blocking analysis; not in the other plots); what is important is that we see a plateau before the onset of this noise. Now, we may be more confident in our FCIQMC calculation. Above, we captured the output of the blocking script into a file called `stats`. In this example, the contents of that file is below:

```
# of blocks mean (X_24)    std.err. (X_24) std.err.err. (X_24)  mean (X_23)
std.err. (X_23) std.err.err. (X_23)  cov(X_23,X_24) mean (X_24/X_23) std.err. (X_24/X_23)
722          -670.049359811 1.99777332e-01  5.26094278e-03        37244.1867452
9.82441101e+00  2.58716360e-01        -1.32220e+03    -0.017990709917  1.946944746518e-06
361          -670.049359811 2.68876787e-01  1.00204462e-02        37244.1867452
1.36690165e+01  5.09414168e-01        -1.28123e+03    -0.017990709917  1.911297631965e-06
180          -670.055219406 3.59855874e-01  1.90189739e-02        37244.4116806
1.86756159e+01  9.87036972e-01        -1.18406e+03    -0.017990758591  2.026001990428e-06
90           -670.055219406 4.67644101e-01  3.50514073e-02        37244.4116806
2.45252068e+01  1.83824197e+00        -1.01830e+03    -0.017990758591  2.124344681041e-06
45           -670.055219406 5.83345340e-01  6.21848221e-02        37244.4116806
3.08250668e+01  3.28596316e+00        -8.00932e+02    -0.017990758591  2.312712518500e-06
22           -670.085373646 7.34544351e-01  1.13342654e-01        37245.7631108
3.89686474e+01  6.01299283e+00        -6.25139e+02    -0.017990915414  2.494555852939e-06
11           -670.085373646 7.12029706e-01  1.59214682e-01        37245.7631108
3.79742226e+01  8.49129431e+00        -2.96473e+02    -0.017990915414  1.687704717679e-06
5            -669.988620008 6.61525494e-01  2.33884581e-01        37240.7990625
3.68826395e+01  1.30399822e+01        -1.21293e+02    -0.017990715475  1.907691051811e-06
2            -670.428455433 7.74026533e-02  5.47319410e-02        37267.1565039
9.84779297e+00  6.96344119e+00        -1.52449e+00    -0.017989793650  2.676810345979e-06
```

We wish to take the energy from the **first** row here in the second-to-last column and its corresponding uncertainty in the last column, i.e.:

```
-0.017990709917 +/- 1.946944746518e-06
```

This is then our estimate for the correlation energy. To get the *total* energy, we must also add the reference energy, which can be found in the standard output of NECI (we called it `n2_neci.out`):

```
Reference Energy set to:       -108.9606713172
```

(search for "Reference Energy"). You'll also find estimates for the correlation energy in the output file. However, this is not as trustworthy as doing a full blocking analysis.

1.5.5 Final Steps

To be completely sure of our FCIQMC calculation, we must again continue from the popsfile with the `readPops` keyword, but change the number of walkers. The goal here is to verify that we have a sufficient number of walkers, and that we are converged with respect to the total number of walkers. Thus, to be really sure of our energy calculation, we must repeat the FCIQMC calculation but varying the number of walkers. The easiest way to do this is to restart from the previous popsfile and increase the total number of walkers. However, since the previous total number of walkers has already been reached, NECI is in variable-shift (or constant-walker-number) mode, and hence we need to tell NECI vary the number of walkers again.

To do this, we keep the `readPops` keyword and add the keyword and add the `walkContGrow` keyword into the `calc` block. We will, of course, also want to increase the total number of walkers (say, to 100000), and from our previous experience above we know we need more data so we can also increase the number of iterations, i.e.

```
title
...
calc
# continue on from previous calculation
readPops
# allow growth from previous calc
walkContGrow

nmcyc 100000
...
end
```

Repeat this as above, do the same convergence analysis as above. Note, however, that since the number of iterations from which to start the blocking analysis (`<numiter>`) will be higher, as you will see by checking the plots (this is just because NECI needs some time to increase the walker number to the new target number and then stabilise).

Once you have done that, you may be much more confident about your calculated correlation energy.

Congratulations on your first FCIQMC calculation with NECI. The software has many more sophisticated options and can be used for bigger systems. The rest of this documentation discusses these in some details, though not in a tutorial format.

1.6 Calculation inputs

The NECI executable takes one input argument, which is the name of an input file containing the instructions for carrying out the calculation. The input file is organized in blocks, with each block being started and terminated by a dedicated keyword. Each block can contain a number of keywords to specify options. Here, a list of the blocks and their respective keywords is given.

The first line of the input is always `title`, the last line is always `end`.

Some keywords are mandatory, those are marked in **red** and are given at the beginning of the description of each paragraph. Then come recommended options, marked in **blue**, followed by further options given in black.

Keywords which are purely for debugging purposes and only interesting for developers are marked in **green**.

Comments can be added in the code with `#`. (A deprecated comment symbol found in legacy inputs is `()` Line continuation is achieved with `\`. (A deprecated line continuation string found in legacy inputs is `+++`.)

1.6.1 SYSTEM Block

The SYSTEM block specifies the properties of the physical system that is considered. The block starts with the `system` keyword and ends with the `endsys` keyword.

- **system**
  Starts the SYSTEM block. Has one mandatory additional argument to specify the type of the system. The options are
  - **read**
    Read in the integrals from a FCIDUMP file, used for ab-initio calculations.
  - **FCIDUMP-name**
    Specify the name of the FCIDUMP file. Defaults to FCIDUMP.
  - **hubbard**
    Uses the Hubbard model Hamiltonian.
    - **k-space**
      In the momentum-space basis (beneficial for low U/t)
    - **real-space**
      In the real-space basis (beneficial for large U/t)
  - **ueg**
    Uses the Hamiltonian of the uniform electron gas in a box.
- **endsys**
  Terminates the SYSTEM block.
- **electrons** n**, nel** n
  Sets the number of electrons to n
- **spin-restrict** $[m]$
  Sets the total S_z quantum number to $\frac{m}{2}$. The argument m is optional and defaults to 0.
- **hphf** s
  Uses a basis of (anti-)symmetric combinations of Slater determinants with respect to global spin-flip. $s = 0$ indicates anti-symmetric combinations, $s = 1$ symmetric combinations. This is useful to exclude unwanted spin configurations. For example, no triplet states can occur for `hphf 0`.
- **guga** S
  Activates the spin-adapted implementation of FCIQMC using CSFs (more precisely Gel'fand-Tsetlin states) and the graphical Unitary Group Approach (GUGA). The total spin S must be specified in multiples of $\hbar/2$. So $S = 0$ is a singlet, $S = 1$ a doublet $S = 2$ a Triplet and so on. This keyword MUST be combined with certain `nonuniformrandexcits` input as described below! Also it is not allowed to use this keyword with the `spin-restrict` or `HPHF` keyword. And the number of electrons must allow the chosen spin (even number for Singlet,Triplet,etc. and odd for Doublet,Quartet,...). Additionally the choice of a reference determinant via the `definedet` keyword of the `CALC` input block (described below) is necessary in most cases, as the automatic reference state setup is not always working in a CSF-based implementation.
- **sym** k_x k_y k_z s
  Specifies the symmetry of the target state. The first three arguments set the momentum (k_x, k_y, k_z) and are only used for Hubbard and ueg-type systems, the last argument s specifies the irrep within d_{2h} and is only used for ab-initio systems. Note that the symmetry label is zero-indexed, so A_{1g} corresponds to 0.
- **lztot**
  Set the total L_s quantum number. Has one mandatory additional argument, which is the value of L_s.

- **useBrillouinTheorem**
  Assume that single excitations have zero matrix elements with the reference. By default, this is determined automatically.
- **noBrillouinTheorem**
  Always assume that single excitations have nozero matrix elements with the reference.
- **umatEpsilon** ϵ
  Defines a threshold value ϵ below which matrix elements of the Hamiltonian are rounded to 0. Defaults to 10^{-8}.
- **diagonaltmat**
  Assume the kinetic operator is diagonal in the given basis set.
- **noSingExcits**
  Assume there is no coupling between single excitations in the Hamiltonian.
- **rohf**
  Use restricted open-shell integrals.
- **read_rofcidump**
  Read the integrals from a ROFCIDUMP file.
- **spinorbs**
  Uses spin orbitals instead of spatial orbtials for addressing the integrals. This can be used if the integrals depend on the spin of the orbitals.
- **molproMimic**
  Use the same orbital ordering as molpro, mimicking the behaviour of calling NECI from molpro. First nelec/2 orbitals sorted by diagonal elements of the Fock matrix are considered to be occupied, and the rest to be virtual. Each sector is then sorted separately by symmetry labels.
- **complexOrbs_realInts**
  The orbitals are complex, but not the integrals. This reduces the symmetry of the 4-index integrals. Only affects `kneci` and `kmneci` calculations.
- **complexWalkers-realInts**
  The integrals and orbitals are real, but the wave function shall be complex. Only affects `kneci` and `kmneci` calculations.
- **system-replicas** n
  Specifies the number of wave functions that shall be evolved in parallel. The argument n is the number of wave functions (replicas). Requires `mneci` or `kmneci`.
- **nonhermitian [1-body] [2-body]**
  Specifies that the input is a non-Hermitian Hamiltonian, but makes no further assumptions. By default it assumes both 1- and 2-body non-Hermiticity. However, it has two optional keywords, if `1-body` is given, only the 1-body components of the Hamiltonian are non-Hermitian; if `2-body` is given then only the 2-body components are non-Hermitian. Note that in the case of transcorrelation, only the 2-body component would be non-Hermitian; hence if you are running a transcorrelated mean-field calculation, use `nonhermitian 2-body`.
- **stoquastize**
  Stoquastize the Hamiltonian. This means that the off-diagonal elements of the original Hamiltonian become negative without changing the magnitudes, i.e. $H_{ij}^{stoq} = H_{ij}\delta_{ij} - |H_{ij}|(1-\delta_{ij})$.

1.6.1.1 Excitation generation options

- **nonUniformRandExcits**
  Use a non-uniform random excitation generator for picking the move in the FCIQMC spawn

step. This can significantly speed up the calculation. Requires an additional argument, that can be chosen from the following

– **pchb**
Generates excitations weighted directly with the matrix elements using pre-computed alias tables. This excitation generator is extremely fast, while maintaining high acceptance rates and is generally recommended when memory is not an issue. The keyword has to be followed by either `LOCALISED`, `DELOCALISED`, or `MANUAL`.

With `LOCALISED` and `DELOCALISED` the fastest PCHB sampler is chosen automatically. Specify `DELOCALISED` for Hartree-Fock like orbitals and `LOCALISED` for localised orbitals. With `MANUAL` one can select the PCHB sampler themselves.

When selecting `MANUAL`, first the singles are specified. The precomputed weight is given by

$$S_I^A = \begin{cases} |h_{AI}| + \sum_R |g_{AIRR} - g_{ARRI}| & I \neq A \\ 0 & \text{else} \end{cases} \tag{1}$$

Note that R runs over all spin-orbitals, not only the occupied. This makes the weighting determinant-independent. The probabilites are then given by:

$$p_1^{\mathrm{PCHB}}(I) = \frac{\sum_C S_I^C}{\sum_{CL} S_L^C} \qquad p_1^{\mathrm{PCHB}}(A|I) = \frac{S_I^A}{\sum_C S_I^C} \quad . \tag{2}$$

Now we can sample weighted without guaranteeing (un-)occupiedness ($p_1^{\mathrm{PCHB}}(I)$ or $p_1^{\mathrm{PCHB}}(A|I)$), weighted with guaranteeing (un-)occupiedness ($p_1^{\mathrm{PCHB}}(I)|_{I\in D_i}$ or $p_1^{\mathrm{PCHB}}(A|I)|_{A\notin D_i}$), or uniformly ($p_1^{\mathrm{uni}}(I)|_{I\in D_i}$ or $p_1^{\mathrm{uni}}(A|I)|_{A\notin D_i}$).

We will write `fast`, `full`, and `unif` for the three cases and separate the particle selection from hole selelection with a colon. This means that e.g. `unif:full` corresponds to sampling via $p_1^{\mathrm{uni}}(I)|_{I\in D_i} \cdot p_1^{\mathrm{PCHB}}(A|I)|_{A\notin D_i}$.

The possible specifications are one of the following `unif:unif`, `unif:fast`, `unif:full`, `full:full`, or `on-the-fly-heat-bath`. Where `on-the-fly-heat-bath` uses the exact on-the-fly calculated matrix element for weighting (which is **slow**).

After the singles the doubles are specified. The following weights are used for the doubles:

$$W_{IJ}^{AB} = \begin{cases} |g_{AIBJ} - g_{AJBI}| & I \neq J \wedge A \neq B \wedge \{I,J\} \cap \{A,B\} = \emptyset \\ 0 & \text{else} \end{cases} \tag{3}$$

and the probalities are given by:

$$\begin{aligned} p_2^{\mathrm{PCHB}}(I) &= \frac{\sum_{LCD} W_{IL}^{CD}}{\sum_{KLCD} W_{KL}^{CD}} & p_2^{\mathrm{PCHB}}(J|I) &= \frac{\sum_{CD} W_{IJ}^{CD}}{\sum_{LCD} W_{IL}^{CD}} \\ p_2^{\mathrm{PCHB}}(A|IJ) &= \frac{\sum_D W_{IJ}^{AD}}{\sum_{CD} W_{IJ}^{CD}} & p_2^{\mathrm{PCHB}}(B|IJA) &= \frac{W_{IJ}^{AB}}{\sum_D W_{IJ}^{AD}} \quad . \end{aligned} \tag{4}$$

Again we can combine different combinations of uniform, fast-, and fully-weighted sampling. We will specify the selection for the first and second particle and then the first and second hole. Again the particle and hole selections are separated by a `:`.

The possible particle selections are `full-full` ($p_2^{\mathrm{PCHB}}(I)|_{I\in D_i} \cdot p_2^{\mathrm{PCHB}}(J|I)|_{J\in D_i}$), `unif-full` ($p_2^{\mathrm{uni}}(I)|_{I\in D_i} \cdot p_2^{\mathrm{PCHB}}(J|I)|_{J\in D_i}$), `unif-fast` ($p_2^{\mathrm{uni}}(I)|_{I\in D_i} \cdot p_2^{\mathrm{PCHB}}(J|I)$), and `unif-unif` ($p_2^{\mathrm{uni}}(I)|_{I\in D_i} \cdot p_2^{\mathrm{unif}}(J|I)|_{J\in D_i}$).

The possible hole selections are `full-full` ($p_2^{\mathrm{PCHB}}(A|IJ)|_{A\notin D_i} \cdot p_2^{\mathrm{PCHB}}(B|IJA)|_{B\notin D_i}$) and `fast-fast` ($p_2^{\mathrm{PCHB}}(A|IJ) \cdot p_2^{\mathrm{PCHB}}(B|IJA)$).

The particle and hole selections can be freely combined among eather, e.g. `unif-full:fast-fast`.

An example input is:

```
nonuniformrandexcits pchb localised
```

An example input for manual specification is

```
nonuniformrandexcits pchb manual \
    unif:full unif-full:full-full
```

- **guga-pchb**
  Uses the pre-computed alias tables for the spin-adapted GUGA implementation. Needs the `guga` keyword in the `System` block. If it is used in conjunction with the histogramming tau-search, which is recommended for GUGA calculations in general, it automatically sets more reasonable defaults than for the usual `mol-guga-weighted` excitation generation option.

- **nosymgen**
  Generate all possible excitations, regardless of symmetry. Might have a low acceptance rate.

- **4ind-weighted**
  Generate exictations weighted by a Cauchy-Schwarz estimate of the matrix element. Has very good acceptance rates, but is comparably slow. Using the `4ind-weighted-2` or `4IND-WEIGHTED-UNBOUND` instead is recommended.

- **4ind-weighted-2**
  Generates excitations using the same Cauchy-Schwary estimate as `4ind-weighted`, but uses an optimized algorithm to pick orbitals of different spin, being faster than the former.

- **4ind-weighted-unbound**
  Generates excitations using the same Cauchy-Schwary estimate as `4-ind-weighted` and optimizations as `4ind-weighted-2`, but uses more accurate estimates, having higher acceptance rates. This excitation generator has high acceptance rates at negligible memory cost.

- **pcpp**
  The pre-computed power-pitzer excitation generator [8]. Has low memory cost and scales only mildly with system size, and can thus be used for large systems.

- **mol-guga-weighted**
  Excitation generator for molecular systems used in the spin-adapted GUGA Approach. The specification of this excitation generator when using GUGA in ab initio systems is necessary!

- **ueg-guga**
  Excitation generator choice when using GUGA in the UEG or k-space/real-space Hubbard model calculations. It is mandatory to specify this keyword in this case!

- **GAS-CI**
  Specify the actual implementation for GAS. Requires a GAS specification via the `GAS-SPEC` keyword.

  * **PCHB**
    This is the default and the fastest implementation, if sufficient memory is available. The double excitations work similar to the FCI precomputed heat bath excitation generator but automatically exclude GAS forbidden excitations. The keyword has two optional sub-keywords, `SINGLES`, `DOUBLES` which allow to tailor the algorithm for your system. (The default choice is usually sufficiently fast.) With **SINGLES** one can select the single excitation algorithm:

    · **PC-WEIGHTED**
      Use precomputed weighted singles, which is the default and recommended sampling scheme. Read at Full CI PCHB for a deeper description. It allows the same sampling schemes `fully-weighted`, `weighted`, and `fast-weighted`.

- **PC-UNIFORM**
  This is the default. It chooses GAS allowed single excitations uniformly.
- **ON-THE-FLY-HEAT-BATH**
  It chooses GAS allowed electrons weighted by their matrix element.

With **doubles** one can select the particle- and hole-selection algorithm for double excitations. It is followed by `particle-selection` or `hole-selection`. Read at Full CI PCHB for a deeper description. It allows the same sampling schemes.

An example input is:

```
nonuniformrandexcits GAS-CI PCHB \
        singles pc-weighted weighted \
        doubles particle-selection weighted \
        doubles hole-selection fully-weighted
```

* **DISCARDING**
  Use a Full CI excitation generator and just discard excitations which are not contained in the GAS space. Currently PCHB is used for Full CI.
* **ON-THE-FLY-HEAT-BATH**
  Use heat bath on the fly general GAS, which is applicable to any GAS specification, but a bit slower than necessary for disconnected spaces.
* **DISCONNECTED**
  Use the disconnected GAS implementations, which assumes disconnected spaces and performs there a bit better than the general implementation.

- **lattice-excitgen**
  Generates uniform excitations using momentum conservation. Requires the `kpoints` keyword.
- **GAS-SPEC**
  Perform a *Generalized Active Spaces* (GAS) calculation and specify the GAS spaces.[13] It is necessary to select the actual implementation with the `GAS-CI` keyword. It is possible to use *local*, *cumulative*, or *flexible* constraints on the particle number. Local constraints define the minimum and maximum particle number per GAS space. Cumulative constraints define cumulative minima and maxima of the cumulative particle number. The flexible constraints allow the user to list the allowed supergroups, i.e. the allowed distribution of particles among the GAS spaces. The specification is first `LOCAL` or `CUMULATIVE` to define the kind of constraints followed by the number of GAS spaces n_{GAS}. The next items are n_{GAS} rows with 3 numbers which are the number of spatial orbitals and (cumulative) minimum and maximum number of particles per GAS space $n_i, N_i^{\min}, N_i^{\max}$. Finally the last row denotes for each spatial orbital to which GAS space it belongs. Instead of `1 1 1 1 1` one can write `5*1`. Two benzenes with single inter-space excitation would be e.g. denoted as:

  ```
  GAS-SPEC LOCAL 2
           6  5  7
           6  5  7
           1  1  1  1  1  1  2  2  2  2  2  2
  ```

  or

  ```
  GAS-SPEC LOCAL  2
           6  5  7
           6  5  7
           6*1 6*2
  ```

  or

  ```
  GAS-SPEC CUMULATIVE 2
           6  5  7
           6 12 12
           6*1 6*2
  ```

  In the given example the local and cumulative constraints are equivalent, but they are not always!

  The flexible constraints start with the keyword `FLEXIBLE` and require the number of GAS spaces and supergroups $n_{\mathrm{GAS}} \quad n_{\mathrm{sg}}$. The next n_{sg} rows list the allowed supergroups. Again the last

row denotes for each spatial orbital to which GAS space it belongs. The previous example of two benzene with single excitations would be

```
GAS-SPEC FLEXIBLE 2 3
        6 6
        5 7
        7 5
        6*1 6*2
```

It is possible to switch off the spin recoupling between different GAS spaces by appending `NO-RECOUPLING`.

- **OUTPUT-GAS-HILBERT-SPACE-SIZE**
  Optional keyword. If a GAS calculation is performed, then output the sizes of the CAS and GAS Hilbert-Space sizes. Note that, depending on the GAS specifications, this operation can be actually expensive, so it is not done by default.

1.6.1.2 Hubbard model and UEG options

- **lattice** *type* l_x l_y [l_Z]
  Defines the basis in terms of a lattice of type *type* with extent $l_x \times l_y \times l_z$. l_x and l_y are mandatory, l_z is optional. *type* can be any of `chain`, `star`, `square`, `rectangle`, `tilted`, `triangular`, `hexagonal`, `kagome`, `ole`.

- **U** U
  Sets the Hubbard interaction strengh to U. Defaults to 4.

- **B** t
  Sets the Hubbard hopping strengh to t. Defaults to -1.

- **twisted-bc** t_1 [t_2]
  Use twisted boundary conditions with a phase of $t_1 \frac{2\Pi}{x}$ applied along x-direction, where x is the lattice size. t_2 is optional and the additional phase along y-direction, in multiples of $\frac{2\Pi}{y}$.

- **open-bc** [*direction*]
  Set the boundary condition in *direction* to open, i.e. no hopping across the cell boundary is possible. *direction* is optional and can be one of `X`, `Y` or `XY`, for open boundary conditions in x-, y- or both directions. If omitted, both directions are given open boundary conditions. Requires a real-space basis.

- **ueg-offset** k_x k_y k_z
  Offset (k_x, k_y, k_z) for the momentum grid used in for the uniform electron gas.

- **bipartite-order** [n]
  Enables a bipartite ordering of bipartite lattice (chain, square for now) Has to be put **before!** the `lattice` keyword. If an additional parameter `n`, which has to equal to the number of lattice sites, is given an arbitrary orbital ordering can be given in the subsequent input line. (It has to be `n` unique numbers from, 1 to n.) An example: `bash bipartite 5 1 3 2 5 4`

1.6.1.3 Transcorrelation options

- **molecular-transcorr**
  Enable the usage of a transcorrelated ab-initio Hamiltonian. This implies the non-hermiticity of the Hamiltonian as well as the presence of 3-body interactions. Requires passing the 3-body integrals in either ASCII or HDF5 format in a `TCDUMP` file, or `tcdump.h5`, respectively. Enables triple excitation generation. When using this option, non-uniform random excitation generator become inefficient, so using `nonUniformRandExcits` is discouraged.

- **adjoint-calculation**
  Instead of calculating H, NECI solves for $H^\dagger$. Note in the case of transcorrelation, this is equivalent to switching the sign of your Jastrow factor: J to $-J$.

- **ueg-transcorr** *mode*
  Enable the usage of a transcorrelated Hamiltonian for the uniform electron gas. This implies the non-hermiticity of the Hamiltonian as well as the presence of 3-body interactions. *mode* can be one of `3-body`, `trcorr-excitgen` or `rand-excitgen`.
- **transcorr** [J]
  Enable the usage of a transcorrelated Hamiltonian for the real-space hubbard mode. This implies the non-hermiticity of the Hamiltonian. The optional parameter J is the transcorrelation parameter and defaults to 1.0.
- **2-body-transcorr** [J]
  Enable the usage of a transcorrelated Hamiltonian for the momentum space hubbard model. This implies the non-hermiticity of the Hamiltonian as well as the presence of 3-body interactions. The optional argument J is the correlation parameter and defaults to 0.25.
- **exclude-3-body-ex**
  Disables the generation of triple excitations, but still takes into account 3-body interactions for all other purposes.
- **adjoint-replicas**
  For multiple replicas (mneci, system-replicas >=2) or a dneci run, evolves the left eigenvector for the even replicas, while still evolving the right eigenvector for the odd replicas. This gives access to stochastically independent $|Psi_R>$ and Ψ_L in the same simulation, for RDM calculation e.g.

1.6.1.4 Spin purification

- **sd-spin-purification** J [**truncate-ladder-operator, only-ladder-operator**]
  Use an adjusted hamiltonian $H + JS^2$ for the dynamic to force antiferromagnetic ordering and ensure pure spin-states in a Slater determinant (SD) basis.

  One can add the optional keyword `only-ladder-operator` after J not to use the full $S^2 = S_z(S_z - 1) + S_+S_-$ operator but a truncated version, which uses only the ladder operator term S_+S_-.

  One can add the optional keyword `truncate-ladder-operator` after J not to use the full $S^2 = S_z(S_z - 1) + S_+S_-$ operator but a truncated version, which uses only the ladder operator term S_+S_- and truncates it by removing its diagonal. This implies that the diagonal counting of open shell α electrons is removed.

1.6.2 CALC Block

The CALC block is used to set options concerning the simulation parameters and modes of FCIQMC. The block starts with the `calc` keyword and ends with the `endcalc` keyword.

- **calc**
  Starts the CALC block
- **endcalc**
  Terminates the CALC block
- **time** t
  Set the maximum time t in minutes the calculation is allowed to run. After t minutes, the calculation will end.
- **nmcyc** n
  Set the maximum number of iterations the calculation is allowed to do. After n iterations, the calculation will end.

- **eq-cyc** n
  Set the number of iterations to perform after reaching variable shift mode. n Iterations after entering the variable shift mode, the calculation will end. If both `eq-cyc` and `nmcyc` are given, the calculation ends when either of the iteration limits is reached.
- **seed** s
  Sets the seed of the random number generator to s. This can be used to specifically probe for stochastic effects, but is generally not required.
- **averageMcExcits** x
  Sets the average number of spawning attempts from each walker to x.
- **rdmSamplingIters** n
  Set the maximum number of iterations used for sampling the RDMs to n. After n iterations of sampling RDMs, the calculation will end.
- **load-balance-blocks [OFF]**
  Distribute the determinants blockwise in a dynamic fashion to maintain equal load for all processors. This is enabled by default and has one optional argument OFF. If given, the load-balancing is disabled.
- **energy**
  Additionally calculate and print the ground state energy using an exact diagonalization technique.
- **averageMcExcits** n
  The number of spawns to attempt per walker. Defaults to 1 and should not be changed without good reason.
- **adjust-averageMcExcits**
  Dynamically update the number of spawns attempted per walker. Can be used if the excitation generator creates a lot of invalid excitations, but should be avoided else.
- **scale-spawns** $[k_{\text{scale-spawn}}]$
  Store the maximum value of $\frac{H_{ij}}{p_{gen}}$ for each determinant. If a bloom with more than $k_{\text{scale-spawn}}$ spawns happens, then several spawning attempts with individual lower probability will happen instead. $k_{\text{scale-spawn}}$ has to be smaller equal than k_{maxbloom} from equations 6 and 7.
- **davidson-max-iters** n
  Set the number of iterations in Davidson's algorithm when this is used. Such algorithm computes a few of the smallest (or largest) eigenvalues of a large sparse real symmetric matrix. This method is used, for instance, in the semi-stochastic implementation or when CI Davidson is used. The default value is 50.
- **davidson-target-tolerance** x
  Set the target convergence tolerance of the residual norm of Davidson diagonalization. The default value is 10^{-7}.

1.6.2.1 Population control options

- **totalWalkers** n
  Sets the targeted number of walkers to n. This means, the shift will be varied to keep the walker number constant once it reaches n.
- **diagShift** S
  Set the initial value of the shift to S. A value of $S < 0$ is not recommended, as it will decrease the population from the beginning.
- **shiftDamp** ζ
  Set the damping factor used in the shift update scheme to ζ. Has to be < 1.0. Defaults to 0.1.

- **target-shiftDamp** ζ [η]
  Uses the shift updating procedure presented in [14] and can be used instead of the shiftDamp keyword. ζ is the usual shift damping factor, η is the additional second shift damping factor. If η is not specified, it is set to $\zeta^2/4$ to achieve critical damping. Both parameters have to be < 1.0 and $\eta < \zeta$.

- **stepsshift** n
  Sets the number of steps per update cycle of the shift to n. Defaults to 10.

- **fixed-n0** n_0
  Instead of varying the shift to fix the total number of walkers, keep the number of walkers at the reference fixed at n_0. Automatically sets `stepsSft 1` and overwrites any `stepssft` options given.

- **targetGrowRate** *grow walks*
  When the number of walkers in the calculation exceeds *walk*, the shift is iteratively adjusted to maintain a fixed grow rate *grow* until reaching the requested number of total walkers.

- **jump-shift** [**OFF**]
  When entering the variable shift mode, the shift will be set to the current projected energy. This is enabled by default. There is an optional argument OFF that disables this behaviour.

- **pops-jump-shift**
  Reset the shift when restarting a previous calculation to the current projected energy instead of using the shift from the previous calculation.

- **trunc-nopen** n
  Restrict the Hilbert space of the calculation to those determinants with at most n unpaired electrons.

- **avGrowthRate** [**OFF**]
  Average the change in walker number used to calculate the shift. This is enabled by default and has one optional argument OFF, which, when given, turns the option off.

1.6.2.2 Real walker coefficient options

- **allRealCoeff**
  Allow determinants to have non-integer population. There is a minimal population below which the population of a determinant will be rounded stochastically. This defaults to 1.

- **realSpawnCutoff (x | OFF | ON)**
  Continuous real spawning will be performed, unless the spawn has weight less than x. In this case, the weight of the spawning will be stochastically rounded up to x or down to zero, such that the average weight of the spawning does not change. This is a method of removing very low weighted spawnings from the spawned list, which require extra memory, processing and communication. This keyword is on by default with a value of $x = 0.95$. It can be explicitly turned off via `OFF`, explicitly turned on via `ON` (using the default value then), or it can read a user-supplied value for x.

- **realCoeffbyExcitLevel** n
  Allow all determinants up to an excitation level of n to have non-integer population.

- **setOccupiedThresh** x
  Set the value for the minimum walker weight in the main walker list. If, after all annihilation has been performed, any determinants have a total weight of less than x, then the weight will be stochastically rounded up to x or down to zero such that the average weight is unchanged. Defaults to 1, which should only be changed with good reason.

- **energy-scaled-walkers** [*mode* α β]
  Scales the occupied threshold with the energy of a determinant. Has three optional arguments and requires `allRealCoeff`. The argument *mode* can be one of EXPONENTIAL, POWER,

EXP-BOUND or NEGATIVE and defaults to POWER. Both α and β default to 1 and `realspawncutoff` β is implied.

1.6.2.3 Time-step options

- **tau-values**
  This keyword is mandatory. It is followed by "start", "min", or "max" and their respective sub-keywords. It is necessary to define the source of the starting value of $\Delta\tau$, this means that `start` is required.
  - **start**
    This defines the source of the initial $\Delta\tau$.

    Has to be followed by one of the following sub-keywords:
    - **user-defined** $\Delta\tau$
      Has to be followed by a real number that is the initial $\Delta\tau$.
    - **from-popsfile**
      Use the value from a popsfile. Requires `readpops`.
    - **tau-factor**
      Use `tau-factor` times the connections from the reference determinant as starting guess.
    - **refdet-connections**
      Use information about the connections from the reference determinant as starting guess.
    - **deterministic**
      Use the deterministic time-step:
      $$\tau = \frac{1}{E_{\max} - E_0} \tag{5}$$
      where $E_{\max} - E_0$ is approximated by the spread of diagonal elements of $\hat{H}$.
    - **not-needed**
      Say explicitly that $\Delta\tau$ is not needed. This is only relevant for fully deterministic calculations, e.g. Lanczos CI.
  - [**min** $\tau_{\mathbf{min}}$]
    Optional keyword. It defines a minimum for the initial value of $\Delta\tau$ and following $\Delta\tau$-searches.
  - [**max** $\tau_{\mathbf{max}}$]
    Optional keyword. It defines a maximum for the initial value of $\Delta\tau$ and following $\Delta\tau$-searches.
  - [**readpops-but-tau-not-from-popsfile**]
    Optional keyword. If `readpops` is switched on, but $\Delta\tau$ should not be read from a popsfile then it is necessary to explicitly state that it is indeed wanted.
- **tau-search**
  The $\Delta\tau$-search is off by default, but can be switched on with two different algorithms. It can also be switched off again when certain stop conditions are reached, since it is an unnecessary expensive operation when $\Delta\tau$ has reached a stable value. Has the following sub-keywords:
  - [**algorithm**]
    Optional keyword. This defines the algorithm of the $\Delta\tau$-search.

    Has to be followed by one of the following sub-keywords:

* **conventional**
  Adjusts $\Delta\tau$ such that:

$$\Delta\tau = k_{\text{maxbloom}} \cdot \min_{i,j}\left(\frac{p_{\text{gen}}(i,j)}{H_{ij}}\right) \quad . \tag{6}$$

  The prefactor k_{maxbloom} is changed via `MaxWalkerBloom`.

* **histogramming** [$(1-c)$ $n_{\textbf{bins}}$ b]
  Update the time-step based on histogramming of the ratio $\frac{H_{ij}}{p_{\text{gen}}(i|j)}$.

$$\Delta\tau = k_{\text{maxbloom}} \cdot \left(\text{argmin}_t \left| c - \int_0^t p(x)\,\mathrm{d}x \right| \right)^{-1} \quad . \tag{7}$$

  Where p is the probability distribution of $\frac{H_{ij}}{p_{\text{gen}}(i|j)}$ which is obtained numerically by binning. The three arguments $1-c$, n_{bins} and b are optional. $0 < c < 1$ is the fraction of the histogram used for determining the new timestep, n_{bins} the number of bins in the histogram and b is the maximum value of $\frac{H_{ij}}{p_{\text{gen}}(i|j)}$ to be stored.
  For spin-adapted GUGA calculations this option is *highly* recommended! Otherwise the time-step can become quite small in these simulations. Note that for $c = 1$ the conventional and histogramming time-search are equivalent. The prefactor k_{maxbloom} is changed via `MaxWalkerBloom`.

– [**stop-condition**]
  Optional keyword. Defines a stop-condition for the $\Delta\tau$-search. The default is `var-shift`, i.e. the search ends, when variable shift mode is reached.

  Has to be followed by one of the following sub-keywords:

  * **off**
    No stop-condition, i.e. run $\Delta\tau$-search until the calculation ends.

  * **no-change** i
    The $\Delta\tau$-search is switched off, if there was no change of $\Delta\tau$ in the last i iterations.

  * **max-iter** i
    The $\Delta\tau$-search is switched off after the i-th iteration.

  * **max-eq-iter** i
    The $\Delta\tau$-search is switched off after the i-th iteration counting from variable shift mode.

  * **n-opts** i
    The $\Delta\tau$-search is switched off after the i-th optimization of $\Delta\tau$.

  * **var-shift**
    The $\Delta\tau$-search is switched off if variable shift is reached.

– [**off**]
  Switch the tau-search explicitly off. (Equivalent to not having the `tau-search` keyword at all.) Note that this keyword is incompatible with other options (e.g. `maxWalkerBloom`).

– [**scale-tau-to-death**]
  Optional keyword. Off by default. If the $\Delta\tau$-search is off, still scale $\Delta\tau$ such that the death probability is smaller than 1.0.

– **maxWalkerBloom** $k_{\textbf{maxbloom}}$
  The time step is scaled such that at most k_{maxbloom} walkers are spawned in a single attempt, with the scaling being guessed from previous spawning attempts. Changes the prefactor in equations 6 and 7.

- **truncate-spawns** [n **UNOCC**]
  Truncate spawns which are larger than a threshold value n. Both arguments are optional, n defaults to 3. If UNOCC is given the truncation is restricted to spawns onto unoccupied. Useful in combination with hist-tau-search.

1.6.2.3.1 Example inputs for $\Delta\tau$

The following input start with $\Delta\tau = 0.002 \frac{\mathrm{I}\cdot\hbar}{E_\mathrm{h}}$ and keeps its value between $0.001 \frac{\mathrm{I}\cdot\hbar}{E_\mathrm{h}}$ and $0.003 \frac{\mathrm{I}\cdot\hbar}{E_\mathrm{h}}$. The conventional $\Delta\tau$-search that is stopped if there was no change of $\Delta\tau$ for 1000 iterations.

```
tau-values \
    start user-defined 0.002 \
    min 0.001 \
    max 0.003

tau-search \
    algorithm conventional \
    stop-condition no-change 1000
```

The following input start with $\Delta\tau$ from a popsfile. The histogramming $\Delta\tau$-search is performed with $c = 0.9999, n_\mathrm{bins} = 1000, b = 2000$ and is stopped after 10000 iterations.

```
tau-values \
    start from-popsfile

tau-search \
    algorithm histogramming 1e-4 1000 2000 \
    stop-condition max-iter 10000
```

1.6.2.4 Wave function initialization options

- **walkContGrow**
  When reading in a wave function from a file, do not set the shift or enter variable shift mode.
- **defineDet** *det*
  Sets the reference determinant of the calculation to *det*. If no other initialisation is specified, this will also be the initial wave function. The format can either be a comma-separated list of spin orbitals, a range of spin orbitals (like 12-24) or a combination of both.
  For spin-adapted calculations using the `GUGA` keyword defining a starting reference CSF manually is *highly* encouraged, as the automatic way often fails. It works in similar ways as for SDs, however odd numbered singly occupied orbitals indicate a positive spin coupled orbital in GUGA CSFs and even numbered singly occupied orbitals negatively spin coupled. So one needs to be careful to not define a unphysical CSF with an negative intermediate total spin $S_i < 0$. E.g. a CSF like:
  `definedet 1 2 4 5`
  would cause the calculation to crash, as the first singly occupied orbital (4) would cause the total spin to be negative $S_i < 0$. When defining the starting CSF one also needs to ensure that the defined CSFs satisfies the total number of electrons and total S defined in the `System` block of the input with the keywords `nel` and `guga`. As an example a triplet (`guga 2`) CSF with 4 electrons (`nel 4`) would be
  `definedet 1 2 3 5`
  with the first spatial orbital doubly occupied and 2 open-shell orbitals (positively spin coupled, hence odd numbers).
- **readPops**
  Read in the wave function from a file and use the read-in wave function for initialisation. In addition to the wave function, also the time-step and the shift are read in from the file. This starts the calculation in variable shift mode, maintaining a constant walker number, unless `walkContGrow` is given.
- **readpops-changeref**
  Allow the reference determinant to be updated after reading in a wave function.

- **startSinglePart** [n]
  Initialise the wave function with n walkers on the reference only unless specified differently. The argument n is optional and defaults to 1.
- **proje-changeRef** [$frac$ min]
  Allow the reference to change if a determinant obtains $frac$ times the population of the current reference and the latter has a population of at least min. Both arguments are optional and default to 1.2 and 50, respectively. This is enabled by default.
- **no-changeref**
  Never change the reference.

1.6.2.5 Initiator options

- **truncInitiator**
  Use the initiator method [5].
- **addToInitiator** x
  Sets the initiator threshold to x, so any determinant with more than x walkers will be an initiator.
- **senior-initiators** [age]
  Makes any determinant that has a half-time of at least age iterations an initiator. age is optional and defaults to 1.
- **superInitiators** [n]
  Create a list of n superinitiators, from which all connected determinants are set to be initiators. The superinitiators are chosen according to population. n is optional and defaults to 1.
- **coherent-superInitiators** [$mode$]
  Apply a restriction on the sign-coherence between a determinant and any connected superinitiator to determine whether it becomes an initiator due to connection. The optional argument $mode$ can be chosen from STRICT, WEAK, XI, AV and OFF. The default is WEAK and is enabled by default if the `superInitiators` keyword is given.
- **dynamic-superInitiators** n
  Updates the list of superinitiators every n steps. This is enabled by default with $n = 100$ if the `superInitiators` keyword is given. A value of 0 indicates no update. This implies the `dynamic-core n` option (with the same n) unless specified otherwise.
- **allow-signed-spawns** $mode$
  Never abort spawns with a given sign, regardless of initiators. $mode$ can be either POS or NEG, indicating the sign to keep.
- **initiator-space**
  Define all determinants within a given initiator space as initiators. The space is specified through one of the following keywords
  - **doubles-initiator**
    Use the reference determinant and all single and double excitations from it to form the initiator space.
  - **cas-initiator cas1 cas2**
    Use a CAS space to form the initiator space. The parameter cas1 specifies the number of electrons in the cas space and cas2 specifies the number of virtual spin orbitals (the cas2 highest energy orbitals will be virtuals).
  - **ras-initiator ras1 ras2 ras3 ras4 ras5**
    Use a RAS space to form the initiator space. Suppoose the list of spatial orbitals are split into three sets, RAS1, RAS2 and RAS 3, ordered by their energy. ras1, ras2 and ras3 then specify the number of spatial orbitals in RAS1, RAS2 and RAS3. ras4 specifies the minimum number of electrons in RAS1 orbitals. ras5 specifies the maximum number

of electrons in RAS3 orbitals. These together define the RAS space used to form the initiator space.

- **optimised-initiator**
  Use the iterative approach of Petruzielo *et al.* (see PRL, 109, 230201). One also needs to use either optimised-initiator-cutoff-amp or optimised-initiator-cutoff-num with this option.
- **optimised-initiator-cutoff-amp** $x1$**,** $x2$**,** $x3\ldots$
  Perform the optimised initiator option, and in iteration i, choose which determinants to keep by choosing all determinants with an amplitude greater than xi in the ground state of the space (see PRL 109, 230201). The number of iterations is determined by the number of parameters provided.
- **optimised-initiator-cutoff-num** $n1$**,** $n2$**,** $n3\ldots$
  Perform the optimised initiator option, and in iteration i, choose which determinants to keep by choosing the ni most significant determinants in the ground state of the space (see PRL 109, 230201). The number of iterations is determined by the number of parameters provided.
- **fci-initiator**
  Use all determinants to form the initiator space. A fully deterministic projection is therefore performed with this option.
- **pops-initiator** n
  When starting from a POPSFILE, this option will use the n most populated determinants from the popsfile to form the initiator space.
- **read-initiator**
  Use the determinants in the INITIATORSPACE file to form the initiator space. A INITIATORSPACE file can be created by using the write-initiator option in the LOGGING block.

1.6.2.6 Adaptive shift options

- **auto-adaptive-shift** [t α c]
  Scale the shift per determinant based on the acceptance rate on a determinant. Has three optional arguments. The first is the threshold value t which is the minimal number of spawning attempts from a determinant over the full calculation required before the shift is scaled, with a default of 10. The second is the scaling exponent α with a default of 1 and the last is the minimal scaling factor, which uses $\frac{1}{\text{HF conn.}}$ as default.
- **linear-adaptive-shift** [σ f_1 f_2]
  Scale the shift per determinant linearly with the population of a determinant. All arguments are optional and define the function used for scaling. σ gives the minimal walker number required to have a shift and defaults to 1, f_1 the shift fraction to be applied at σ with a default of 0 and f_2 is the shift fraction to be applied at the initiator threshold, defaults to 1. Every initiator is applied the full shift.
- **core-adaptive-shift**
  By default, determinants in the corespace are always applied the full shift. Using this option also scales the shift in the corespace.
- **aas-matele2**
  Uses the matrix elements for determining the scaling factor in the `auto-adaptive-shift`. The recommended option to scale the shift.

1.6.2.7 Multi-replica options

- **multiple-initial-refs**
  Define a reference determinant for each replica. The following n lines give the reference determinants as comma-separated lists of orbitals, where n is the number of replicas.
- **orthogonalise-replicas**
  Orthogonalise the replicas after each iteration using Gram Schmidt orthogonalisation. This will converge each replica to another state in a set of orthogonal eigenstates. Can be used for excited state search.
- **orthogonalise-replicas-symmetric**
  Use the symmetric Löwdin orthonaliser instead of Gram Schmidt for orthogonalising the replicas.
- **replica-single-det-start**
  Starts each replica from a different excited determinant.

1.6.2.8 Semi-stochastic options

- **semi-stochastic**
  Turn on the semi-stochastic adaptation.
- **pops-core** n
  This option will use the n most populated determinants to form the core space. This keyword cannot be used with `pops-core-proportion`. Note that core-space configurations behave like initiators and are treated as such by default. See also the `core-inits` keyword.
- **pops-core-proportion** f
  This option will use a fraction f of most populated initiator determinants to form the core space. For example, about 50% of most populated initiator determinants are chosen if $f = 0.5$. This keyword cannot be used with `pops-core` and requires `semi-stochastic`.
- **doubles-core**
  Use the reference determinant and all single and double excitations from it to form the core space.
- **cas-core cas1 cas2**
  Use a CAS space to form the core space. The parameter cas1 specifies the number of electrons in the cas space and cas2 specifies the number of virtual spin orbitals (the cas2 highest energy orbitals will be virtuals).
- **ras-core ras1 ras2 ras3 ras4 ras5**
  Use a RAS space to form the core space. Suppoose the list of spatial orbitals are split into three sets, RAS1, RAS2 and RAS 3, ordered by their energy. ras1, ras2 and ras3 then specify the number of spatial orbitals in RAS1, RAS2 and RAS3. ras4 specifies the minimum number of electrons in RAS1 orbitals. ras5 specifies the maximum number of electrons in RAS3 orbitals. These together define the RAS space used to form the core space.
- **optimised-core**
  Use the iterative approach of Petruzielo *et al.* (see PRL, 109, 230201). One also needs to use either optimised-core-cutoff-amp or optimised-core-cutoff-num with this option.
- **optimised-core-cutoff-amp** $x1$**,** $x2$**,** $x3$**...**
  Perform the optimised core option, and in iteration i, choose which determinants to keep by choosing all determinants with an amplitude greater than xi in the ground state of the space (see PRL 109, 230201). The number of iterations is determined by the number of parameters provided.
- **optimised-core-cutoff-num** $n1$**,** $n2$**,** $n3$**...**
  Perform the optimised core option, and in iteration i, choose which determinants to keep by choosing the ni most significant determinants in the ground state of the space (see PRL 109, 230201). The number of iterations is determined by the number of parameters provided.

- **fci-core**
  Use all determinants to form the core space. A fully deterministic projection is therefore performed with this option. This option requires information about spin(-projection) and spatial symmetry, so the keywords `sym`, and `spin-restrict` for a diagonalization in a Slater Determinant basis or `guga` for a diagonalization in a Gelfand-Tsetlin basis are required.
- **read-core**
  Use the determinants in the CORESPACE file to form the core space. A CORESPACE file can be created by using the write-core option in the LOGGING block.
- **dynamic-core** n
  Update the core space every n iterations, where n is optional and defaults to 400. This is enabled by default if the `superinitiators` option is given.
- **core-inits [{OFF, ON}]**
  Declare all determinants in the core-space as initiators, independent from their population. Since core-space determinants behave like initiators regardless of their population this option is enabled by default. There is an optional keyword which is either ON, or OFF. If the optional keyword is ommitted it defaults to ON.

1.6.2.9 Trial wave function options

- **trial-wavefunction** [n]
  Use a trial wave function to obtain an estimate for the energy, as described in 1.4. The argument n is optional, when given, the trial wave function will be initialised n iterations after the variable shift mode started, else, at the start of the calculation. The trial wave function is defined through one of the following keywords
  - **pops-trial** n
    When starting from a POPSFILE, this option will use the n most populated determinants from the popsfile to form the trial space.
  - **doubles-trial**
    Use the reference determinant and all single and double excitations from it to form the trial space.
  - **cas-trial cas1 cas2**
    Use a CAS space to form the trial space. The parameter cas1 specifies the number of electrons in the cas space and cas2 specifies the number of virtual spin orbitals (the cas2 highest energy orbitals will be virtuals).
  - **ras-trial ras1 ras2 ras3 ras4 ras5**
    Use a RAS space to form the trial space. Suppoose the list of spatial orbitals are split into three sets, RAS1, RAS2 and RAS 3, ordered by their energy. ras1, ras2 and ras3 then specify the number of spatial orbitals in RAS1, RAS2 and RAS3. ras4 specifies the minimum number of electrons in RAS1 orbitals. ras5 specifies the maximum number of electrons in RAS3 orbitals. These together define the RAS space used to form the trial space.
  - **optimised-trial**
    Use the iterative approach of Petruzielo *et al.* (see PRL, 109, 230201). One also needs to use either optimised-trial-cutoff-amp or optimised-trial-cutoff-num with this option.
  - **optimised-trial-cutoff-amp** $x1$**,** $x2$**,** $x3$**...**
    Perform the optimised trial option, and in iteration i, choose which determinants to keep by choosing all determinants with an amplitude greater than xi in the ground state of the space (see PRL 109, 230201). The number of iterations is determined by the number of parameters provided.
  - **optimised-trial-cutoff-num** $n1$**,** $n2$**,** $n3$**...**
    Perform the optimised trial option, and in iteration i, choose which determinants to keep

by choosing the ni most significant determinants in the ground state of the space (see PRL 109, 230201). The number of iterations is determined by the number of parameters provided.

- **fci-trial**
  Use all determinants to form the trial space. A fully deterministic projection is therefore performed with this option.

1.6.2.10 Memory options

- **memoryFacPart** x
  Sets the factor between the allocated space for the wave function and the required memory for the specified number of walkers to x. Defaults to 10.
- **memoryFacSpawn** x
  Sets the factor between the allocated space for new spawns and the estimate of required memory for the spawns of the specified number of walkers on a single processor to x. The memory required for spawns increases, the more processors are used, so when running with few walkers on relatively many processors, a large factor might be needed. Defaults to 3.
- **prone-determinants**
  Instead of terminating when running out of memory, randomly delete determinants with low population and few spawns.
- **store-dets**
  Employ extra memory to store additional information on the determinants that had to be computed on the fly else. Trades in memory for faster iterations.

1.6.2.11 Reduced density matrix (RDM) options

- **rdmSamplingIters** n
  Set the number of iterations for sampling the RDMs to n. After n iterations of sampling, the calculation ends.
- **inits-rdm**
  Only take into account initiators when calculating RDMs. By default, only restricts to initiators for the right vector used in RDM calculation. This makes the RDMs non-variational, and the resulting energy is the projected energy on the initiator space.
- **strict-inits-rdm**
  Require both sides of the inits-rdm to be initiators.
- **no-lagrangian-rdms**
  This option disables the correction used for RDM calculation for the adaptive shift. Use this only for debugging purposes, as the resulting RDMs are flawed.

1.6.2.12 METHODS Block The METHODS block is a subblock of CALC, i.e. it is specified inside the CALC block. It sets the main algorithm to be used in the calculation. The subblock is started with the `methods` keyword and terminated with the `endmethods` keyword.

- **methods**
  Starts the METHODS block
- **endmethods**
  Terminates the METHODS block.
- **method** *mode*
  Sets the algorithm to be executed. The relevant choice for *mode* is VERTEX FCIMC to run an FCIQMC calculation. Alternative choices are DETERM-PROJ to run a deterministic calculation and SPECTRAL-LANCZOS to calculate a spectrum using the lanczos algorithm.

1.6.3 INTEGRAL Block

The INTEGRAL block can be used to freeze orbitals and set properties of the integrals. The block is started with the `integral` keyword and terminated with the `endint` keyword.

- **integral**
  Starts the INTEGRAL block.
- **endint**
  Terminates the INTEGRAL block.
- **freeze** n m
  Freeze n core and m virtual orbitals which are not to be considered active in this calculation. The orbitals are selected according to orbital energy, the n lowest and m highest orbitals in energy are frozen.
- **freezeInner** n m
  Freeze n core and m virtual orbitals which are not to be considered active in this calculation. The orbitals are selected according to orbital energy, the n highest and m lowest orbitals in energy are frozen.
- **partiallyFreeze** $n_{\mathbf{orb}}$ $n_{\mathbf{holes}}$
  Freeze n_{orb} core orbitals partially. This means at most n_{holes} holes are now allowed in these orbitals.
- **partiallyFreezeVirt** $n_{\mathbf{orb}}$ $n_{\mathbf{els}}$
  Freeze n_{orb} virtual orbitals partially. This means at most n_{els} electrons are now allowed in these orbitals.
- **hdf5-integrals**
  Read the 3-body integrals for a transcorrelated ab-initio Hamiltonian from an HDF5 file. Ignored when the `molecular-transcorrelated` keyword is not given. Requires compiling with HDF5.
- **sparse-lmat**
  Store the 3-body integrals in a sparse format to save memory. Initialisation and iterations might be slower. Requires `hdf5-integrals`.

1.6.4 Development keywords

- **UNIT-TEST-PGEN**
  Test the pgens for the n(most populated) configurations of an existing pops-file. Perform the tests n(iterations) times for each configuration. The order of arguments is n(most populated), n(iterations).

1.6.5 KP-FCIQMC Block

This block enables the Krylov-projected FCIQMC (KPFCIQMC) method [1] which is fully implemented in NECI. It requires `dneci` or `mneci` to be run. When specifying the KP-FCIQMC block, the METHODS block should be omitted. This block is started with the `kp-fciqmc` keyword and terminated with the `end-kp-fciqmc` keyword.

- **kp-fciqmc**
  Starts the KP-FCIQMC block
- **end-kp-fciqmc**
  Terminates the KP-FCIQMC block.
- **num-krylov-vecs** N
  N specifies the total number of Krylov vectors to sample.

- **num-iters-between-vecs** N
  N specifies the (constant) number of iterations between each Krylov vector sampled. The first Krylov vector is always the starting wave function.
- **num-iters-between-vecs-vary** i_{12}**,** i_{23}**,** i_{34}**...**
  $i_{n,n+1}$ specifies the number of iterations between the nth and (n+1)th Krylov vectors. The number of parameters input should be the number of Krylov vectors asked for minus one. The first Krylov vector is always the starting wave function.
- **num-repeats-per-init-config** N
  N specifies the number repeats to perform for each initial configuration, i.e. the number of repeats of the whole evolution, from the first sampled Krylov vector to the last. The projected Hamiltonian and overlap matrix estimates will be output for each repeat, and the averaged values of these matrices used to compute the final results.
- **averagemcexcits-hamil** N
  When calculating the projected Hamiltonian estimate, an FCIQMC-like spawning is used, rather than calculating the elements exactly, which would be too computationally expensive. Here, N specifies the number of spawnings to perform from each walker from each Krylov vector when calculating this estimate. Thus, increasing N should improve the quality of the Hamiltonian estimate.
- **finite-temperature**
  If this option is included then a finite-temperature calculation is performed. This involves starting from several different random configurations, whereby walkers are distributed on random determinants. The number of initial configurations should be specified with the num-init-configs option.
- **num-init-configs** N
  N specifies the number of initial configurations to perform the sampling over. An entire FCIQMC calculation will be performed, and an entire subspace generated, for each of these configurations. This option should be used with the finite-temperature option, but is not necessary for spectral calculations where one always starts from the same initial vector.
- **memory-factor** x
  This option is used to specify the size of the array allocated for storing the Krylov vectors. The number of slots allocated to store unique determinants in the array holding all Krylov vectors will be equal to ABx, where here A is the length of the main walker list, B is the number of Krylov vectors, and x is the value input with this option.
- **num-walkers-per-site-init** x
  For finite-temperature jobs, x specifies the number of walkers to place on a determinant when it is chosen to be occupied.
- **exact-hamil**
  If this option is specified then the projected Hamiltonian will be calculated exactly for each set of Krylov vectors sampled, rather than randomly sampling the elements via an FCIQMC-like spawning dynamic.
- **fully-stochastic-hamil**
  If this option is specified then the projected Hamiltonian will be estimated without using the semi-stochastic adaptation. This will decrease the quality of the estimate, but may be useful for debugging or analysis of the method.
- **init-correct-walker-pop**
  For finite-temperature calculations on multiple cores, the initial population may not be quite as requested. This is because the quickest (and default) method involves generating determinants randomly and sending them to the correct processor at the end. It is possible in this process that walkers will die in annihilation. However, if this option is specified then each processor will throw away spawns to other processors, thus allowing the correct total number of walkers to be spawned.

- **init-config-seeds seed1, seed2...**
  If this option is used then, for finite-temperature calculations, at the start of each calculation over an initial configuration, the random number generator will be re-initialised with the corresponding input seed. The number of seeds provided should be equal to the number of initial configurations.
- **all-sym-sectors**
  If this option is specified then the FCIQMC calculation will be run in all symmetry sectors simultaneously. This is an option relevant for finite-temperature calculations.
- **scale-population**
  If this option is specified then the initial population will be scaled to the populaton specified with the 'totalwalkers' option in the Calc block. This is relevant for spectral calculations when starting from a perturbed `POPSFILE` wave function, where the initial population is not easily controlled.

In spectral calculations, one also typically wants to consider a particular perturbation operator acting on the ground state wave functions. Therefore, you must first perform an FCIQMC calculation to evolve to the ground state and output a `POPSFILE`. You should then start the KP-FCIQMC calculation from that `POPSFILE`. To apply a perturbation operator to the `POPSFILE` wave function as it is read in, use the pops-creation and pops-annihilate options. These allow operators such as

$$\hat{V} = \hat{c}_i \hat{c}_j + \hat{c}_k \hat{c}_l \hat{c}_m$$

to be applied to the `POPSFILE` wave function. The general form is `pops-annihilate n_sum orb1 orb2...` ... where n_{sum} is the number of terms in the sum for $\hat{V}$ (2 in the above example), and *orbi* specify the spin orbital labels to apply. The number of lines of such orbitals provided should be equal to n_{sum}. The first line provides the orbital labels for the first term in the sum, the second line for the second term, etc...

1.6.6 REALTIME Block

The REALTIME block enables the calculation of the real-time evolution of a given state and is only required for real-time calculations. Real-time evolution strictly requires `kneci` or `kmneci` and using `kmneci` is strongly recommended. This block is started with the `realtime` keyword and terminatd with the `endrealtime` keyword.

- **realtime**
  Starts the realtime block. This automatically enables `readpops`, and reads from a file named `<popsfilename>.0`.
- **endrealtime**
  Terminates the REALTIME block.
- **single** i a
  Applies a single excitation operator exciting from orbital i to orbital a to the initial state before starting the calculation.
- **lesser** i j
  Calculates the one-particle Green's function $< c_i^\dagger(t) c_j >$. Requires using one electron less than in the popsfile.
- **greater** i j
  Calculates the one-particle Green's function $< c_i(t) c_j^\dagger >$. Requires using one electron more than in the popsfile.
- **start-hf**
  Do not read in a popsfile but start from a single determinant.
- **rotate-time** α
  Calculates the evolution along a trajectory $e^{i\alpha}t$ instead of pure real-time trajectory.

- **dynamic-rotation** [η]
  Determine a time-dependent α on the fly for `rotate-time`. η is optional and a damping parameter, defaults to 0.05. α is then chosen such that the walker number remains constant.
- **rotation-threshold** N
  Grow the population to N walkers before starting to adjust α.
- **stepsalpha** n
  The number of timesteps between two updates of the α parameter when using `dynamic-rotation`.
- **log-trajectory**
  Output the time trajectory to a separate file. This is useful if the same calculation shall be reproduced.
- **read-trajectory**
  Read the time trajectory from disk, using a file created by NECI with the `log-trajectory` keyword.
- **live-trajectory**
  Read the time trajectory from disk, using a file which is currently being created by NECI with the `log-trajectory` keyword. Can be used to create additional data for the same trajectory while the original calculation is still running.
- **noshift**
  Do not apply a shift during the real-time evolution. Strongly recommended.
- **stabilize-walkers** [S]
  Use the shift to stabilize the walker number if it drops below 80% of the peak value. S Is optional and is an asymptotic value used to fix the shift.
- **energy-benchmark** E
  Set the energy origin to E by applying a global, constant shift to the Hamiltonian. Can be chosen arbitrarily, but a reasonable selection can greatly help efficiency.
- **rt-pops**
  A second popsfile is supplied containing a time evolved state created with the `realtime` keyword whose time evolution is to be continued. In this case, the original popsfile is still required for calculating properties, so two popsfiles will be read in.

1.6.7 LOGGING Block

The LOGGING block specifies the output of the calculation and which status information of the calculation shall be collected. This block is started with the `logging` keyword and terminated with the `endlog` keyword.

- **logging**
  Starts the LOGGING block.
- **endlog**
  Terminates the LOGGING block.
- **hdf5-pops**
  Sets the format to read and write the wave function to HDF5. Requires building with the `ENABLE-HDF5` cmake option.
- **popsfile** n
  Save the current wave function on disk at the end of the calculation. Can be used to initialize subsequent calculations and continue the run. This is enabled by default. n is optional and, when given, specifies that every n iteration, the wave function shall be saved. Setting $n = -1$ disables this option.

- **popsFileTimer** n
  Write out a the wave function to disk every n hours, each time renaming the last `<popsfile>` to `<popsfile>.bk`.
- **hdf5-pops-write**
  Sets the format to write the wave function to HDF5. Requires building with the `ENABLE-HDF5` cmake option.
- **hdf5-pops-read**
  Sets the format to read the wave function to HDF5. Requires building with the `ENABLE-HDF5` cmake option.
- **highlyPopWrite** n
  Print out the n most populated determinants at the end of the calculation. Is enabled by default with $n = 15$.
- **replicas-popwrite** [n]
  Have each replica print out its own highly populated determinants in a separate block instead of one collective output of the on average most important determinants. Optionally, n is the numbers of determinants to print, this is the same n as in `highlypopwrite`.
- **inits-exlvl-write** n
  Sets the excitation level up to which the number of initiators is logged to n. Defaults to $n = 8$.
- **binarypops**
  Sets the format to write the wave function to binary.
- **nomcoutput**
  Suppress the printing of iteration information to stdout. This data is still written to disk.
- **stepsOutput** n
  Write the iteration data to stdout/disk every n iterations. Defaults to the number of iterations per shift cycle. Setting $n = 0$ disables iteration data output to stdout and uses the shift cycle for disk output.
- **fval-energy-hist**
  Create a histogram of the scaling factor used for the auto-adaptive shift over the energy of a determinant. Only has an effect if `auto-adaptive-shift` is used.
- **fval-pop-hist**
  Create a histogram of the scaling factor used for the auto-adaptive shift over the population of a determinant. Only has an effect if `auto-adaptive-shift` is used.
- **local-spin**
  Activates the cumulative local spin, $\hat{S}_i^2$, measurement for spin-adapted GUGA calculations. Needs two replicas for an unbiased measurement. Since this is a diagonal property for the GUGA, there is no additional cost.
- **biased-RDMs**
  Only relevant for (k)-neci runs. By default the calculation stops with an error if RDMs are sampled with (k)-neci to prevent user error. With this keyword the user can explicitly say that they want to sample RDMs without replica.
- **ci-coefficients** [n *excitation*]
  Enables the collection of CI coefficients and their average over a number of iterations. The outputs are printed in separate ASCII files named `ci_coeff_*_av`. Additional files named `ci_coeff_*` are printed in a sorted manner that can directly be fed into Molpro [12] for tailored Coupled/Distinguishable Cluster calculations [10]-[11]. The optional argument n is the number of iterations for averaging the CI coefficients and defaults to 1000. This is done in the last iterations of the FCIQMC run (i.e. if NMCYC = 10000 and $n = 1000$, the CI coefficients collection will start at iteration 9001). However, the collection can begin after the NECI

run reaches the preset number of walkers, but it should only take place when the projected correlation energy is already converged. The second optional argument is the *excitation* level of the CI coefficients to be collected and defaults to 2 (i.e., only singles and doubles). CI coefficients up to triples (i.e. setting *excitation* = 3) are available. The semi-stochastic approach is recommended, in order to reach a lower stochastic error for equal time averaging of the CI coefficients.

1.6.7.1 Semi-stochastic output options

- **write-core**
  When performing a semi-stochastic calculation, adding this option to the Logging block will cause the core space determinants to be written to a file called CORESPACE. These can then be further read in and used in subsequent semi-stochastic options using the `read-core` option in the CALC block.
- **write-most-pop-core-end** n
  At the end of a calculation, output the n most populated determinants to a file called CORESPACE. This can further be read in and used as the core space in subsequent calculations using the `read-core` option.
- **print-core-hamil**
  Prints the semi-stochastic Hamiltonian and the semistochastic basis states to the output file `semistoch-hamil` and `semistoch-basis`. Caution with the size of the semistochastic space!
- **print-core-vec**
  Prints the semi-stochastic "eigenvector" to the output files `determ_vecs` and `determ_vecs_av`. Caution with the size of the semistochastic space!

1.6.7.2 RDM output options These options control how the RDMs are printed. For a description of how the RDMs are calculated and the content of the files, please see section 1.7.

- **calcRdmOnfly** *i step start*
  Calculate RDMs stochastically over the course of the calculation. Starts sampling RDMs after *start* iterations, and outputs an average every *step* iterations. *i* indicates whether only 1-RDMs (1), only 2-RDMs (2) or both are produced.
- **rdmLinSpace** *start n step*
  A more user friendly version of `calcrdmonfly` and `rdmsamplingiters`, this samples both 1- and 2-RDMs starting at iteration *start*, outputting an average every *step* iterations n times, then ending the calculation.
- **diagFlyOneRdm**
  Diagonalise the 1-RDMs, yielding the occupation numbers of the natural orbitals.
- **printOneRdm**
  Always output the 1-RDMs to a file, regardless of which RDMs are calculated. May compute the 1-RDMs from the 2-RDMs.
- **writeRdmsToRead off**
  The presence of this keyword overrides the default. If the OFF word is present, the unnormalised `TwoRDM_POPS_a***` files will definitely not be printed, otherwise they definitely will be, regardless of the state of the `popsfile/binarypops` keywords.
- **readRdms**
  This keyword tells the calculation to read in the `TwoRDM_POPS_a***` files from a previous calculation. The restarted calc then continues to fill these RDMs from the very first iteration regardless of the value put with the `calcRdmOnFly` keyword. The calculation will crash if one of the `TwoRDM_POPS_a***` files are missing. If the `readRdms` keyword is present, but the calc is doing a `StartSinglePart` run, the `TwoRDM_POPS_a***` files will be ignored.

- **noNormRdms**
  This will prevent the final, normalised `TwoRDM_a***` matrices from being printed. These files can be quite large, so if the calculation is definitely not going to be converged, this keyword may be useful.
- **writeRdmsEvery** *iter*
  This will write the normalised `TwoRDM_a***` matrices every *iter* iterations while the RDMs are being filled. At the moment, this must be a multiple of the frequency with which the energy is calculated. The files will be labelled with incrementing values - `TwoRDM_a***.1` is the first, and then next `TwoRDM_a***.2` etc.
- **write-spin-free-rdm**
  Output the spin-free 2-RDMs to disk at the end of the calculation.
- **printRoDump**
  Output the integrals of the natural orbitals to a file.
- **print-molcas-rdms**
  It is now possible to calculate stochastic spin-free RDMs with the GUGA implementation. This keyword is necessary if one intends to use this feature in conjunction with `Molcas` to perform a spin-free Stochastic-CASSCF. It produces the three files `DMAT, PAMAT` and `PSMAT`, which are read-in by `Molcas`.
- **full-core-rdms**
  In the "normal" RDM only entries, e.g. for the 1-RDM, ρ_{ij}, which are actually connected by some Hamiltonian matrix element are sampled. This has no negative effect on the energy calculation, but for properties, e.g. spin-spin correlation functions. This option activates a full sampling of RDMs, at least in the semi-stochastic space. This option does increase the cost though.
- **print-hdf5-rdms**
  Output the density matrices in HDF5 format to a file called `fciqmc.rdms.<statenumber>.h5`. Currently only pure state RDMs are supported. This keyword needs to be used in conjunction with `write-spin-free-rdm` for the 2RDM and `printonerdm` for the 1RDM respectively.

1.6.8 FCIMCStats output functions

- **instant-s2-full** [x]
  Calculate an instantaneous value for $\hat{S}^2$, and output it to the relevant column (28) in the `FCIMCStats` file.

  The second optional parameter is a multiplier such that we only calculate $\hat{S}^2$ once for every n update cycles (it must be on an update cycle such that $|\Psi|^2$ is correct)
- **instant-s2-init** [x]
  Calculate an instantaneous value for $\hat{S}^2$, considering only the initiators, and output it to the relevant column (29) in the `FCIMCStats` file.

  The second optional parameter is a multiplier such that we only calculate $\hat{S}^2$ once for every n update cycles (it must be on an update cycle such that $|\Psi|^2$ is correct)

1.7 Output files

Apart from the output that is printed to standard out there are some other useful files created by `NECI`. Some of them are only created, when certain keywords are given.

1.7.1 FCIMCStats

This file contains whitespace delimited data that is written every 10nth iteration. Currently (2021-06-08) there are 35 columns. The information in this file is useful in virtually every way of using `NECI`.

The columns are listed in the following. Important and often used columns are highlighted red, rarely used columns are greyed out.

1 Steps (Step)
The number of iterations.

2 Shift (Shift)
The shift S for population control. Equals the correlation energy in the equilibrium. Shift + Reference energy should equal the projected energy in equilibrium.

3 Walker Change (WalkerCng)
Absolute change of the walker number. (Since there are fractional walkers, this can be non-integer.)

4 Growth Rate (GrowRate)
Relative change of walker number.

5 Total Walkers (TotWalkers)
The number of total walkers.

6 Annihilation (Annihil)
The number of annihilated walkers.

7 Number of died walkers (NoDied)
The number of died walkers (from annihilation or diagonal death step).

8 Number of born walkers (NoBorn)
The number of born walkers.

9 Projected averaged correlation energy (Proj.E)
Averaged correlation energy as calculated from the projected energy expression. The unaveraged values are given in column 11.

10 Average Shift (Av.Shift)
Averaged shift. Note that it should not be used, when the average was taken over a period that not in stationary conditions.

11 Projected instantaneous correlation energy (Proj.E.ThisCyc)
Instantaneous (not averaged) correlation energy as calculated from the projected energy expression.

$$E_{\text{proj}} - H_{00} = \frac{\langle D_0|H|\Psi\rangle}{\langle D_0\Psi\rangle} - H_{00} = \sum_{j\neq 0} H_{0j}\frac{C_j}{C_0} \quad . \tag{8}$$

The averaged values are given in column 9. The total energy (correlation + reference energy) is given in column 23)

12 Number of walkers at reference determinant D_0 (NoatHF)
The number of walkers at the reference determinant. (Since there are fractional walkers, this can be non-integer.)

13 Number of walkers at doubles (NoatDoubs)
Number of walkers that are a double excitation away from the reference.

14 Acceptance rate (AccRat)
Probability that a spawn gets accepted. This is **not** the condition probability, but the absolute one.

15 Uniqe determinants (UniqueDets)
Number of unique configurations (determinants/CSFs).

16 Iteration time (IterTime)
This is the time averaged over the last 10 iterations.

17 (FracSpawnFromSing)
TODO

18 (WalkersDiffProc)
TODO

19 Total imaginary time τ (TotImagTime)
The elapsed **imaginary** time since start of the dynamics.

20 (ProjE.ThisIter)
TODO

21 (HFInstShift)
TODO

22 (TotInstShift)
TODO

23 Projected instantaneous total energy (Tot-Proj.E.ThisCyc)
Instantaneous (not averaged) total energy as calculated from the projected energy expression

$$E_{\text{proj}} = \frac{\langle D_0|H|\Psi\rangle}{\langle D_0\Psi\rangle} = H_{00} + \sum_{j\neq 0} H_{0j}\frac{C_j}{C_0} \quad . \tag{9}$$

The instantaneous correlation energy is given in column 11. The nominator and denominator are given in columns 24 and 25 and should be used for statistical analysis of errors. It is averaged over the last 10 iterations.

24 Instantaneous denominator of projected energy (HFContribtoE)
This is the instantaneous denominator of the projected energy C_0 and equivalent to the reference weight (column 12). Column 24 and 25 are used to evaluate the other projected energy columns (9, 11, and 23). It is averaged over the last 10 iterations.

25 Instantaneous numerator of projected energy (NumContribtoE)
This is the instantaneous nominator of the projected energy

$$\sum_{j\neq 0} H_{0j}C_j \tag{10}$$

Column 24 and 25 are used to evaluate the other projected energy columns (9, 11, and 23). It is averaged over the last 10 iterations.

26 Amplitude of reference (HF weight)
Unlike the name suggests, this is the **amplitude** weight of the reference determinant given by:

$$\frac{|c_0|}{|\Psi|} = \frac{|c_0|}{\sqrt{\sum_i |c_i|^2}} \quad . \tag{11}$$

This means that this column can be obtained by dividing column 12 by column 27.

27 $|\Psi|$ (|Psi|)
Instantaneous L2-norm of the current wavefunction.

$$|\Psi| = \sqrt{\sum_i |c_i|^2} \tag{12}$$

28 Expectation value of S^2 operator over all determinants (Inst S^2)
Requires the `instant-s2-full` keyword in the logging block.

29 Expectation value of S^2 operator over initiator determinants (Inst S^2)
Requires the `instant-s2-init` keyword in the logging block.

30 Absolute instantaneous projected correlation energy (AbsProjE)
L1-norm of projected correlation energy expression.

$$\sum_{j \neq 0} |H_{0j}| \frac{|C_j|}{|C_0|} \quad . \tag{13}$$

31 (PartsDiffProc)
Uninitialised Garbage.

32 Weight of semistochastic space (|Semistoch|/|Psi|)
If S_C is the semistochastic or core space, then we have:

$$\frac{\sum_{j \in S_C} |c_j|^2}{\sum_i |c_i|^2} \quad . \tag{14}$$

33 Largest spawn per iteration (MaxCycSpawn)
This is the largest spawn per iteration.

34 The number of discarded excitations (InvalidExcits)
This is the number of discarded excitations averaged over the last 10 iterations.

35 The number of valid excitations (ValidExcits)
This is the number of valid excitations averaged over the last 10 iterations.

1.8 Trial wave functions

By default, NECI uses a single reference determinant, $|D_0\rangle$, in the projected energy estimator, or potentially a linear combination of two determinants if the the HPHF code is being used.

$$E_0 = \frac{\langle D_0|\hat{H}|\Psi\rangle}{\langle D_0|\Psi\rangle}.$$

This estimator can be improved by using a more accurate estimate of the true ground state, a trial wave function, $|\Psi^T\rangle$,

$$E_0 = \frac{\langle \Psi^T|\hat{H}|\Psi\rangle}{\langle \Psi^T|\Psi\rangle}.$$

Such a trial wave function can be used in NECI using by adding the `trial-wavefunction` option to the Calc block. You must also specify a trial space. The trial wave function used will be the ground state of the Hamiltonian projected into this trial space.

The trial spaces available are the same as the core spaces available for the semi-stochastic option. However, you must replace `core` with `trial`. For example, to use all single and double excitations of the reference determinant, one should use the 'doubles-trial' option.

1.9 Sampling excited states

As well as sampling the ground state, NECI can be used to estimate excited-state properties using an orthogonalisation procedure. Specifically, by performing m FCIQMC simulations simultaneously, the lowest m energy states can be sampled.

To do this, one must use the mneci compilation. Using dneci is not sufficient.

To specify how many states are to be sampled, one should use the `system-replicas` option in the System block of the input file.

Then, the `orthogonalise-replicas` option should be included in the Calc section of the input file. This will tell NECI to orthogonalise the FCIQMC wave functions against each other. States representing high-energy wave functions will be orthogonalised against those representing low-energy wave functions. This prevents higher-energy states being projected to the ground state, and instead allows excited states to be converged upon.

Also, one must tell NECI how to initialise each FCIQMC wave function. It is a bad idea to start from single determinants, as many of these will be poor estimates to the desired excited states, and so convergence will be slow. Instead, one should start from trial estimates to the desired excited states. These trial states are generated by calculating the lowest-energy states within a subspace. Thus, one must simply tell NECI what subspace to use. The options available are the same as for semi-stochastic and trial spaces (see above).

For example, if one wants to initialise from the lowest-energy CISD states, one should use the `doubles-init` option in the Calc block. If one wants to start from the lowest-energy states in a $(10, 10)$ CAS space, you should put `cas-init 10 10` in the Calc block.

Also, single-determinant energy estimators can be give poor results for excited states. Instead, trial wave function-based estimators should be used. This should be done exactly as for the ground state – see above for more details on this. For example, to use CISD wave functions in the trial energy estimators, include both the `trial-wavefunction` and `doubles-trial` options in the Calc block of the input file.

For some example excited-state calculations with NECI, see the test_suite/mneci/excited_state directory and the tests therein.

1.10 Davidson RAS code

NECI has an option to find the ground state of a RAS space using a direct CI davidson approach, which does not require the Hamiltonian to be stored. This code is particularly efficient for FCI and CAS spaces, but is less efficient for CI spaces.

To perform a davidson calculation, put

```
davidson ras1 ras2 ras3 ras4 ras5
```

in the Methods block, inside the Calc block. The parameters ras1-ras5 define the RAS space that will be used. These are defined as follows. First, split all of the spatial orbitals into theree sets, RAS1, RAS2 and RAS3, so that RAS1 contains the lowest energy orbitals, and RAS3 the highest. Then, ras1, ras2 and ras3 define the the number of spatial orbitals in RAS1, RAS2 and RAS3. ras4 defines the minimum number of electrons in RAS1. ras4 defines the maximum number of electrons in RAS3. These 5 parameters define the ras space.

This method will allocate space for up to 25 Krylov vectors. It will iterate until the norm of the residual vector is less than 10^{-7}. If this is not achieved in 25 iterations, the calculation will simply stop and output whatever the current best estimate at the ground state is.

This code should be able to perform FCI or CAS calculations for spaces up to around 5×10^6 or so, but will probably struggle for spaces much larger than this.

The method has only been implemented with RHF calculations and with $M_s = 0$.

1.11 Performing error analysis

Data from an FCIQMC calculation is usually correlated. As a result, standard error analysis for uncorrelated data cannot be used. Instead we perform a so-called blocking analysis (JCP 91, 461). In this, data is grouped into blocks of increasing size until the data in subsequent blocks becomes uncorrelated, to a good approximation.

A blocking analysis can be performed in NECI in one of two ways. Firstly, a rough blocking analysis is performed automatically after a job is finished. The final result is output to standard output and further information about the blocking analysis at various block sizes is output to separate files, such as `Blocks_num` and `Blocks_denom`. This should only be used as a rough and quick estimate as there are issues with this approach. For example, the analysis starts as soon as the shift is turned on. This is before the population has stabilised, and so unusual results can occur in the analysis of the denominator and numerator. Also, data is not taken from the optimal block size.

A better approach for a more careful analysis is to use the blocking script in the utils directory, called blocking.py. The key command is

```
./blocking.py -f start_iter -d24 -d23 -o/ FCIMCStats
```

This will perform a blocking analysis starting from iteration `start_iter`. The analysis should be started only once the energy estimate, (column 11 in `FCIMCStats`) and the numerator and denominator (columns 24 and 25) have stabilised and are fluctuating about some final value. Just because the energy looks stable, it does not mean that the populations is not still growing!

`-d24 -d23'` tells the script to perform the blocking on columns 25 and 24 of the `FCIMCStats` file, which correspond to the numerator and denominator of the energy estimator, respectively. `-o/` tells the script to also provide data for the results of dividing columns 25 and 24, which gives the energy estimate that we want.

Running this will produce a graph of the errors for both the numerator and denominator as a function of the number of blocks (and therefore of the block size). As the block size increases, the error estimates should increase, tending towards the true values. Eventually the estimates will plateau. This indicates that, at this block length, the data in the blocks are uncorrelated to a good approximation, and the error estimate calculated is accurate. The data from this block length should therefore be used.

Each estimate of the error will also have an error on it. As the block length increases this 'error on the error' will increase. One should therefore use the *first* block length where the plateau is reached, so as to minimise the error on the final error estimate.

If no plateau is seen in the plot then the simulation has not been run for long enough, and needs to be continued by restarting from the `POPSFILE`. It can take on the order of $10^5 - 10^6$ iterations to perform an accurate blocking analysis.

The `blocking.py` script will also output the final estimates on the energy at the different block lengths. You should find the blocking length where the errors plateau and read of the final estimates (the rightmost columns) from here.

More information (including example plots, similar to those that `blocking.py` produces) is available in Ref [6].

1.12 Using the NECI pylib

A python frontend is available in the `neci_guga` python library, which is based on the `neci` build target (i.e. plain neci, without complex or multi-replica support) and is built by executing

```
make neci_guga_pylib
```

in the neci build directory. This will create a python3 library `neci_guga.<build-specifier>.so`, which is installed in the `python/` subdirectory of the neci build directory.

To use it, load the `python` subdirectory of the neci build directory into the library path of python, either by

```
export PYTHONPATH=<neci_build>/python:$PYTHONPATH
```

or by adding

```
import sys
sys.path.append('<neci_build>/python')
```

to the calling python module or script.

The `neci_guga` python module can then be loaded in calling python code with `import neci_guga` and provides the following functionality:

- `neci_guga.init_guga(fcidump_path, S)` Takes the desired total spin `S` and initializes the GUGA functionality of neci by reading in an existing FCIDUMP (`fcidump_path`) file.
- `neci_guga.clear_guga()` Clears all memory and deletes all objects initialized by `init_guga`, returns 0 on success, 1 else.
- `neci_guga.csf_matel(D_i, D_j)` Returns the matrix element between `D_i` and `D_j`, passed as an array of the size of the number of electrons in the DefineDet format.
- `neci_guga.run_neci(perm)` Reads a neci input file `neci.inp` and an FCIDUMP file in the current directory, using an orbital permutation `perm` to re-order the orbitals used for the calculation (with respect to the FCIDUMP file). Then, a neci calculation with the specified input is run and the weight of the leading CSF is returned. The permutation is given by specifying the new position for each orbital, i.e. a permutation `python perm = [3, 4, 2, 1]` would put orbital 1 in the third position, orbital 2 in the fourth position, orbital 3 in the second position and orbital 4 in the first position.

1.13 Additional Reading with Technical Details

A list of references can be found in the references section. Here we simply point to some of them of which may be of interest to the reader.

- The original FCIQMC method is in Ref [3].
- Quite a bit of symmetries and some details on the initiator method that is actually implemented is in Ref [4].
- Linear scaling algorithm, (uniform) excitation generation and overall algorithm of FCIQMC is in Ref [2].
- Density matrices, real walker weights and sampling bias is in Ref [9].
- The KP-FCIQMC method is in Ref [1].
- Error (blocking) analysis is described in Ref [6].

Bibliography

[1] N. S. Blunt, Ali Alavi, and George H. Booth. “Krylov-Projected Quantum Monte Carlo”. In: *Phys. Rev. Lett.* 115.5 (2015).

[2] George H. Booth, Simon D. Smart, and Ali Alavi. “Linear-Scaling and Parallelizable Algorithms for Stochastic Quantum Chemistry”. In: *Mol. Phys.* 112.14 (2014), pp. 1855–1869.

[3] George H. Booth, Alex J. W. Thom, and Ali Alavi. “Fermion Monte Carlo without Fixed Nodes: A Game of Life, Death, and Annihilation in Slater Determinant Space”. In: *J. Chem. Phys.* 131.5 (2009), p. 054106.

[4] George H. Booth et al. “Breaking the Carbon Dimer: The Challenges of Multiple Bond Dissociation with Full Configuration Interaction Quantum Monte Carlo Methods”. In: *J. Chem. Phys.* 135.8 (2011), p. 084104.

[5] Deidre Cleland, George H. Booth, and Ali Alavi. “Communications: Survival of the Fittest: Accelerating Convergence in Full Configuration-Interaction Quantum Monte Carlo”. In: *J. Chem. Phys.* 132.4 (2010), p. 041103.

[6] H. Flyvbjerg and H. G. Petersen. “Error Estimates on Averages of Correlated Data”. In: *J. Chem. Phys.* 91.1 (1989), pp. 461–466.

[7] Kai Guther et al. “NECI: N-Electron Configuration Interaction with an Emphasis on State-of-the-Art Stochastic Methods”. In: *J. Chem. Phys.* 153.3 (2020), pp. 34107–34131.

[8] Verena A. Neufeld and Alex J. W. Thom. “Exciting Determinants in Quantum Monte Carlo: Loading the Dice with Fast, Low-Memory Weights”. In: *J. Chem. Theory Comput.* 15.1 (2019), pp. 127–140.

[9] Catherine Overy et al. “Unbiased Reduced Density Matrices and Electronic Properties from Full Configuration Interaction Quantum Monte Carlo”. In: *J. Chem. Phys.* 141.24 (2014), p. 244117.

[10] E. Vitale, Ali Alavi, and Daniel Kats. “FCIQMC-Tailored Distinguishable Cluster Approach”. In: *J. Chem. Theory Comput.* 16 (2020), pp. 5621–5634.

[11] Eugenio Vitale et al. “FCIQMC-Tailored Distinguishable Cluster Approach: Open-Shell Systems”. In: *J. Chem. Theory Comput.* 18 (2022), p. 3427.

[12] Hans-Joachim Werner et al. “The Molpro Quantum Chemistry Package”. In: *J. Chem. Phys.* 152.14 (2020), p. 144107.

[13] Oskar Weser et al. “Chemical Insights into the Electronic Structure of Fe(II) Porphyrin Using FCIQMC, DMRG, and Generalized Active Spaces”. In: *Int. J. Quantum Chem.* 121.3 (2021), pp. 26454–26467.

[14] Mingrui Yang, Elke Pahl, and Joachim Brand. “Improved walker population control for full configuration interaction quantum Monte Carlo”. In: *The Journal of Chemical Physics* 153.17 (2020). Publisher: American Institute of Physics, p. 174103.