MOKIT Manual

version 1.2.3 Sep 13, 2021

jxzou

1 Introduction

1.1 Introduction to MOKIT

The full name of MOKIT is **Molecular Orbital KIT**. MOKIT offers various utilities and modules to transfer MOs among various quantum chemistry software packages. Besides, the **AutoMR** program in MOKIT can set up and run common multi-reference calculations in a block-box way.

With MOKIT, one can perform multi-reference calculations in a quite simple way, and utilize the best modules of each program, e.g.

   UHF(UNO) -> CASSCF -> CASPT2

   Gaussian   PySCF   OpenMolcas

or

   UHF(UNO) -> GVB   -> CASSCF -> NEVPT2

   Gaussian   GAMESS   PySCF   PySCF

RHF   ->   GVB   -> CASSCF   -> ic-MRCISD+Q
Gaussian   GAMESS   PySCF   OpenMolcas

Negligible energy loss (usually<1e-6 a.u., for the same wave function method in two programs) are ensured during transferring MOs, since the basis order of angular momentum up to H(i.e. l=5) is explicitly considered.

Note that although MOKIT aims to make the multi-reference calculations block-box, the users are still required to have practical experiences of quantum chemistry computations (e.g. familiar with routine DFT calculations in Gaussian). You are encouraged to learn how to use Gaussian first if you are a fresh hand. See Appendix A2 for more information.

1.2 Citation

If you use any module or utility of MOKIT in your work, you **MUST CITE** the GitLab page of MOKIT, just like

[1] Jingxiang Zou, Molecular Orbital Kit (MOKIT), https://gitlab.com/jxzou/mokit (accessed month day, year)

If any of the keyword ist=0,1,3 (see Section 4.4.4) in program AutoMR is used, please **ALSO CITE** the following two papers

[2] J. Chem. Theory Comput. 2019, 15, 141-153. DOI: 10.1021/acs.jctc.8b00854
[3] J. Phys. Chem. A 2020. DOI: 10.1021/acs.jpca.0c05216.

Besides, you need to **CITE** all quantum chemistry software packages called in your AutoMR job(s). For example, the file
                    mokit/examples/automr/00-h2o_cc-pVDZ_1.5.gjf
is a CASSCF job. 3 software packages will be called:
(1) Gaussian will be called to perform RHF/UHF;
(2) GAMESS will be called to perform GVB;
(3) PySCF will be called to perform CASSCF.

Therefore, in this case you should also **cite** corresponding references of Gaussian, GAMESS and PySCF, as well as possible references of electronic-structure methods and basis sets.

In the future version of MOKIT, there will be a Citation.txt file generated after your computations terminated. You can simply copy citation information in that plain text file.

2 Installation

The installation of MOKIT does not require any root privilege.
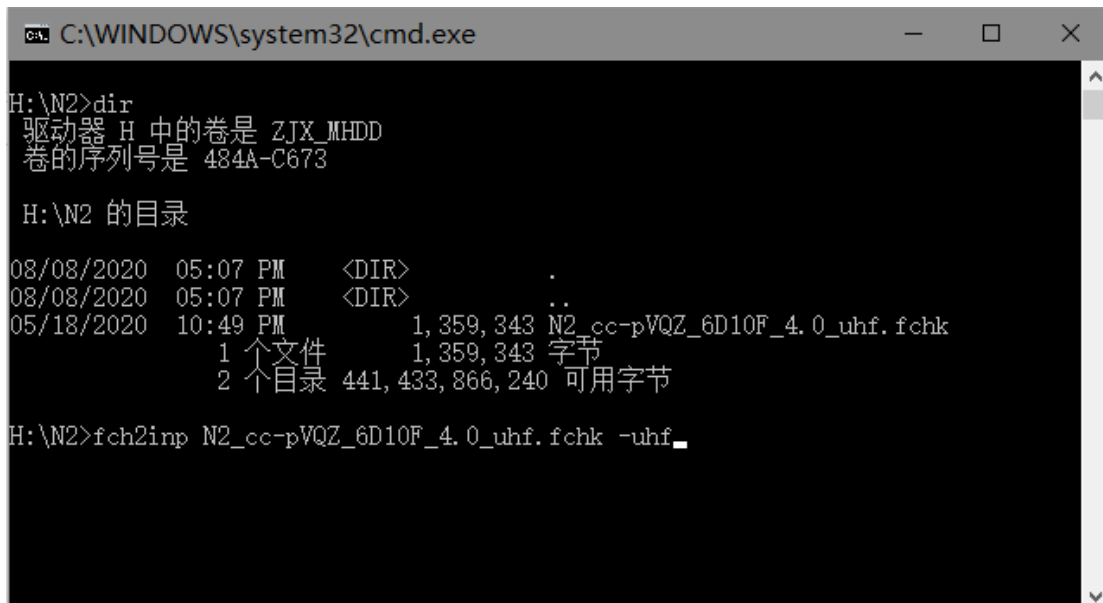
2.1 Pre-built Windows executables

If you only want to use some utilities of MOKIT (e.g. transfer MOs among various programs, generate input files of various programs), the most convenient way is probably to use the pre-built (or pre-compiled) Windows executables.

Only 23 out of all utilities in MOKIT are provided. These executables are compressed into a .7z file and can be downloaded at Releases. Download, uncompress it, and set the environment variables, then these utilities can used in any directory.

To set the environment variables, search 'environment' ("环境变量" in Chinese) in the Windows search bar, then press Enter and click Edit to edit the PATH variables. Create a new path and type the path of MOKIT \bin into it. For example, on my computer the path is H:\Software\MOKIT\mokit\bin.

Press the combination keys Win+R on the keyboard, type 'cmd' to prompt CMD, and **change into the directory where your .fch(k) file holds**, simply run like

2.2 Installation under Linux

2.2.1 Prerequisite

1. Intel compiler (for ifort compiler and Math Kernel Library(MKL))

2. [Anaconda Python3](#) (for python, f2py, etc)

It is strongly recommended to install **Intel compiler** and **Anaconda Python3** on your computer/node. Although these two packages may be large, they meet all prerequisites of compiling MOKIT. Note that the Intel compiler is free of charge for academic use.

If you do not have the ifort compiler and MKL on your node/computer, you may want to download and install them. There are several versions recommended, you can choose any one of them:
(1) Intel Parallel Studio XE 2017~2020 are all OK (2019 or 2020 preferred).
(2) Since 2021, there is no Parallel Studio XE, but only the Intel® oneAPI. You should download and install both [HPC Toolkit](#) and [MKL](#) if you want to use 2021 version.

If you want to use any Fortran compiler other than ifort (e.g. gfortran), you need to install the MKL library (or equivalently, LAPACK/BLAS, OpenBLAS, etc) on your node. And then **uncomment** the first few lines in file mokit/src/Makefile for gfortran settings. And of course, use '#' to comment lines of ifort in that Makefile. The recommended version of gfortran is >= 4.8.5 but < 8.0. Gfortran <= 4.7 is outdated and thus not recommended. Even if you can successfully compile all utilities using gfortran<=4.7, they may not work normally or correctly. If you had compiled MOKIT before, and assuming you want to use another compiler, REMEMBER to run 'make clean' before compiling.

If you want to use [Python3](#) or [Miniconda](#) instead of Anaconda Python3, you need to ensure that the **f2py** compiler/wrapper is valid. For example, run 'which f2py' in Shell to see whether f2py exists. If you are using f2py from PSI4 package, you may encounter errors when compiling MOKIT. In that case, you should comment/deactivate environment variables of PSI4 and use local python (e.g. Anaconda Python3) temporarily. After successfully compiling MOKIT, you can again activate/load environment variables of PSI4.
For trouble shooting during compiling, please read Section A1.

The AutoMR program/executable in MOKIT is an integrated interface program connecting common quantum chemistry software packages. The AutoMR itself does not contain any code for computing electron integrals, it just transfers MOs among software, sometimes read one-electron integrals from some package, and call various packages to do specified computations. Therefore, the users are assumed to have successfully installed these common software packages, which are [Gaussian](#), [PySCF](#), [GAMESS](#), [OpenMolcas](#), [Molpro](#), [ORCA](#), [BDF](#), [PSI4](#), and [Dalton](#). Some recommended versions are shown below

Gaussian: >= g09 (>=g16 preferred)
PySCF >= 1.7.4
GAMESS >= 2017 (>= 2019 preferred)
ORCA >= 4.2.1
OpenMolcas >= v21.02
Molpro >= 2015 (>= 2019 preferred)

BDF >= 0.9.8

PSI4 >= 1.3.2 (>= 1.4 preferred)

Dalton >= 2020.0

For DMRG related packages:

Block >= 1.5.3

QCMaquis >= 3.0.3

CheMPS2 >= 1.8.9

Older versions are not recommended, since (1) they are possibly not tested by the author jxzou, (2) they had been tested but some functionality had not been (correctly) implemented at that time. So that they may work or may not. OpenMolcas v18.09 has also been tested, but you need to take care of a problem, see Section A1, Q5 in Appendix.

Of course, not all packages will be called in an AutoMR job. It depends on the job type and the program specified by users (see Section 4.4.9 ~ 4.4.19 for details). Usually three software packages (Gaussian, PySCF and GAMESS) are extensively used in routine computations. Thus you are recommended to install at least these three packages. If you have any difficulty in installing software, please read their corresponding manuals carefully. Or you can find answers from their official websites, forums, GitHub/GitLab pages:

Gaussian: http://gaussian.com/help

PySCF: https://github.com/pyscf/pyscf/issues

ORCA: https://orcaforum.kofo.mpg.de

OpenMolcas: https://gitlab.com/Molcas/OpenMolcas/-/issues

Molpro: https://groups.google.com/d/forum/molpro-user

GAMESS: https://github.com/gms-bbg/gamess-issues/issues

PSI4: http://forum.psicode.org

Dalton: https://gitlab.com/dalton/dalton

If you can read Chinese, the following installation instructions or tutorials of common software packages on the WeChat Official Accounts 'quantumchemistry' (微信公众号"量子化学") are strongly recommended:

1. 《Linux 下安装 Intel oneAPI》
2. 《Linux 下 Gaussian 16 安装教程》
3. 《ORCA 软件安装教程》
4. 《GAMESS 编译教程》
5. 《离线安装 PySCF-1.7.6》
6. 《PSI4 程序安装及运行》
7. 《离线编译 OpenMolcas+QCMaquis》
8. 《Block-1.5 的编译和安装》
9. 《Boost.MPI 的编译》
10. 《安装基于 openmpi 的 mpi4py》
11. 《自动做多参考态计算的程序 MOKIT》

Note that the original GAMESS source code can only deal with GVB up to 12 pairs. To go beyond that (which is routine type of calculation in AutoMR of MOKIT), please read Section 4.4.10 carefully. After re-compiling GAMESS (followed by instructions in Section 4.4.10), an executable 'gamess.01.x' will be generated. This is the executable to be called in automr.

2.2.2 How to compile

Assuming you've already downloaded the MOKIT .zip package, just run

unzip mokit-master.zip

to uncompress it. Note that do not uncompress the package in Windows* OS, but uncompress it in Linux* OS. Then simply run the following three command lines in Shell

mv mokit-master mokit
cd mokit/src
make all

to compile MOKIT. This will take about 2 minutes. There is no need to 'make install'. If you do not need automr and only want to compile one or several modules, e.g. fch2inp (for Gaussian -> GAMESS orbital transferring), then you simply need to run

make fch2inp

Be careful with hints on the screen, some modules depend on other modules, thus a compilation of two or three modules is necessary sometimes.

2.2.3 Environment variables

After successful compilation, you need to add the following environment variables into your ~/.bashrc file:

export MOKIT_ROOT=/home/$USER/software/mokit
export PATH=$MOKIT_ROOT/bin:$PATH
export PYTHONPATH=$MOKIT_ROOT/lib:$PYTHONPATH
export GMS=/home/$USER/software/games/rungms
export ORCA=/home/$USER/software/orca-4.2.1/orca   # optional
export BDF=/home/$USER/software/bdf-pkg/sbin/run.sh   # optional

Note that do not mistake 'PYTHONPATH' for 'LD_LIBRARY_PATH'. Please modify the above paths to **suit your situations**. Since PySCF is run by python, OpenMolcas is run by pymolcas, Molpro is run by molpro, PSI4 is run by psi4 and Dalton is run by dalton, there is no extra environment variable to be exported here.

The environment variables ORCA and BDF are optional, please do not write them if you do not want to use ORCA or BDF. For Dalton, currently only the MKL version of Dalton is supported. The MPI parallel version may be supported in the future.

The configuration file 'program.info' is no longer used since MOKIT 1.2.1. After writing environment variables, a logout and re-login on your terminal is strongly recommended.

The scratch directory (in Chinese, 临时文件存放目录) is not determined by MOKIT or AutoMR, but by each quantum chemistry package (Gaussian, OpenMolcas, GAMESS, etc). Please do not submit two computations with the same input filename (even in different work directories) at the same time, since their temporary files may be overwritten by each other. For example, GAMESS, OpenMolcas and Molpro jobs are sensitive to the files in the scratch directory, while Gaussian, PySCF and ORCA are not that sensitive.

2.2.4 Test

To see the version and help information, you can run
automr -v (or -V, --version)
automr -h (or -help, --help)

To test whether the automr program works, please change into the example directory, and pick up a .gjf file. For example,

cd $MOKIT_ROOT/examples/automr/
automr 00-h2o_cc-pVDZ_1.5.gjf >& 00-h2o_cc-pVDZ_1.5.out &

If you are running calculations on Ubuntu* OS, where the default sh is probably /bin/dash, not /usr/bin/bash, then you should submit a job like

automr 00-h2o_cc-pVDZ_1.5.gjf >00-h2o_cc-pVDZ_1.5.out 2>&1 &

If you encounter any errors in output, please search your problem in Section A1 FAQ of Appendix firstly. Some common questions have be listed. For bug reporting, please read Section A3.

It is strongly recommended to create a new directory and put in the input file. Because during computation many files will be generated by AutoMR and those common software packages.

2.3 Installation on Cluster

Installing MOKIT on a cluster (in Chinese, 集群或超算) is somewhat troublesome (also for other software packages). If you can write environment variables freely into ~/.bashrc, then there is nothing different with installation on your local Linux computer.

However, sometimes you are not allowed, or unwilling to modify ~/.bashrc. And usually environment variables of various packages (e.g. ifort, python, etc) are controlled by the 'module'

function. For example, if you need some version of python (other than system default), you have to run 'module load python/…' to make it accessible. In such case, you have to firstly load proper versions of Intel compiler and Python before installing MOKIT. For example,

```
module avai                 # find which version you can use
module load intel/2019u5    # load the version you want to use
module load python/3.7.6    # load the version you want to use
make all                    # compile MOKIT
```

The first line finds which versions of software packages you can choose. The 2nd and 3rd lines load the corresponding software. And the 4th line is just the installing of MOKIT. This is just a simple example telling you how to load modules on a cluster. For detailed questions, please consult the administrator of your node, or the cluster. Moreover, a good choice to running automr is that you should create a Shell script, e.g. run.sh, with

```
module load intel/2019u5
module load python/3.7.6
gjfname=$1
outname=${gjfname%.*}.out
automr $gjfname >& $outname
```

written in this script. If you are allowed to submit jobs to the current node, then you can run

```
chmod +x run.sh
./run.sh 00-h2o_cc-pVDZ_1.5.gjf &
```

However, usually you are not allowed to submit any job in the main node or login node. And there exists a queue/batch system on your cluster, e.g. 'bsub' in LSF batch or 'qsub' in PBS batch, which distributes your job to a proper computation node. Taking 'bsub' as an example, you should write a Shell script like

```
#!/bin/sh
#BSUB -q …      # queue name
#BSUB -n …      # number of cores
module load intel/2019u5
module load python/3.7.6
gjfname=$1
outname=${gjfname%.*}.out
automr $gjfname >& $outname
```

Note that do not write '&' symbol after the outname. Then you can run

```
bsub run.sh 00-h2o_cc-pVDZ_1.5.gjf
```

to submit the job. If you still encounter problems of installing or running MOKIT on your cluster, please consult the administrator of your cluster. This is beyond the scope of this manual.

3 Quickstart for MOKIT

See examples in $MOKIT_ROOT/examples/. More details to be added.

3.1 Which method should I use?

For practical use, e.g. to properly compare results with those from experiments, NEVPT2, CASPT2, MC-PDFT or MRCISD is recommended. The GVB and CASSCF methods are often qualitatively correct. For high accuracy results, dynamic correlation is necessary. Additional general recommendations are provided below:

(1) NEVPT2 is free of the intruder-state problem, thus is preferable to CASPT2.

(2) If you find NEVPT2/CASPT2 is too expensive for your system, consider the MC-PDFT method.

(3) If you find the computation cost of NEVPT2/CASPT2 is acceptable for your system and you want higher accuracy, consider ic-MRCISD in OpenMolcas/Molpro. For very small systems such as two or three atoms, the uncontracted MRCISD in ORCA is a good choice.

3.2 Which basis set should I use?

For testing or debug, you can use any type of basis sets. You can, of course, use a small basis set like def2-SVP to test whether AutoMR works. However, for practical use or to obtain publishable data, please use at least triple-zeta basis set, e.g. cc-pVTZ, def-TZVP, def2-TZVP or def2-TZVPP. Results obtained from combinations like MRCISD/def2SVP, NEVPT2/6-31G(d) are almost useless. Additional general recommendations are provided below:

(1) If your system is large (>600 basis functions), you should consider properly simplify your system or use mixed basis sets for different elements/atoms. For example, replace unimportant methyl group(s) by hydrogen atoms, use cc-pVTZ for important atoms and cc-pVDZ (or even 6-31G(d)) for other atoms, etc. Also/Alternatively you can turn on the RI approximation (see Section 4.4.29) in CASSCF and CASSCF-NEVPT2 computations.

(2) Pople-type basis set like 6-311G(d,p) or 6-31G(d,p) is less recommended, but can be used for unimportant atoms.

(3) If pseudopotential (PP) is desired, better to use cc-pVTZ-PP, def2-TZVP. Be careful with the built-in basis set (with PP) SDD in Gaussian software for elements >= the 4th period. Often there is no d, f polarization functions in the built-in SDD basis set of Gaussian (although the PP part seem pretty good), but the d, f polarization functions are usually important for high-accuracy computations. If you insist on using SDD, you should search in previous papers the data of d, f polarization functions of SDD for your atoms, and add the data to the .gjf file manually.

(4) If your molecule is an anion, e.g. MnO_4^-, it is strongly recommended to use basis set with diffuse functions, such as aug-cc-pVTZ, ma-def2-TZVP. Similar to (1), if your system is large, you can use basis set with diffuse functions for atoms with significant negative charges, and use no diffuse functions for other atoms. If the atoms involving significant negative charges are not so important in your study, you can just use aug-cc-pVDZ or ma-def2-SVP.

(5) If all-electron relativistic calculations are desired (DKH2 or X2C), do remember to use basis sets like cc-pVTZ-DK, x2c-TZVPall or ANO-RCC series. All-electron relativistic computations using CASSCF/cc-pVTZ or cc-pVTZ-PP with DKH2 is almost non-sense.

(6) Remember that def2-TZVP is used formally in papers, but def2TZVP is used as the name of basis set in Gaussian syntax.

(7) If you want to compute the NMR shielding constants, basis sets like pcSseg-1 or pcSseg-2 are strongly recommended. Large basis sets like pcSseg-3, def2-QZVP or cc-pVQZ would be better but they are very time-consuming.

Currently AutoMR cannot read user-defined basis sets or pseudopotentials in .gjf file, so you have to provide a Gaussian .fch(k) file and use one of the three options 'readrhf', 'readuhf', 'readno'. See Section 4.4.1~4.4.3 for related information.

3.3 Do I need to specify the active space?

AutoMR can automatically determine the active space, thus you need not specify that. If you do not want the automatically determined one, you can manually specify it, such as CASSCF(m,n) or NEVPT2(m,n), with m, n being the number of active electrons and active orbitals, respectively. Currently only $m=n$ is supported (this is very reasonable, see DOI: 10.1021/acs.jctc.0c00123). While for GVB, you may specify GVB(n), where n is the number of pairs. **Manually specifying is only recommended for experienced users**.

Note that usually there is no need to write DMRGCI or DMRGSCF, the users can just write CASCI or CASSCF. Once AutoMR detects the size of active space is larger than (15,15), it will switch from CASCI/CASSCF to DMRGCI/DMRGSCF automatically. Similarly, the NEVPT2/CASPT2/MC-PDFT will be automatically switched into DMRG-NEVPT2/DMRG-CASPT2/DMRG-PDFT when the active space is larger than (15,15). The only exception is that the users just want to perform a DMRG calculation with active space smaller than (15,15), then the DMRGCI or DMRGSCF must be specified.

Usually the automatically determined active space is reasonable. The algorithm in MOKIT is designed to automatically find the minimum active space for a given molecule. However, when you are studying a potential energy curve/surface, the automatically determined active space may be not the same size for each geometry. For example, for N_2 molecule at d(N-N) = 1.15 Å, the CAS(4,4) may be automatically determined by AutoMR, but for for d(N-N) = 4.0 Å, the active space turns into CAS(6,6). Thus, if you want to keep the size to be CAS(6,6), you need to specify CASSCF(6,6), NEVPT2(6,6), MRCISD(6,6), etc.

3.4 Are my computation (results) reasonable?

(1) If you perform the CASSCF (or CASSCF-based methods like NEVPT2, CASPT2, etc), there would be a file with suffix *CASSCF_NO.fch generated, in which the CASSCF natural orbitals and corresponding occupation numbers are held. You can use GaussView or Multiwfn to open this file and visualize whether these active orbitals are reasonable, if you know your molecule fairly well.

If you perform CASCI (or CASCI-based jobs), the CASCI NOs are held in a *CASCI_NO.fch file. If you only perform GVB computation, the GVB NOs are held in the *gvb2_s.fch file.

(2) If you are comparing relative energies of two target molecules (e.g. two local minima of the same molecule), be sure that you are using the same size of active space for two molecules. For example, it is incorrect(or unfair) to compare two NEVPT2 energies if one is based on CAS(4,4) and the other one is based on CAS(8,8). If, unfortunately, you encounter such special case, you have two options: (i) explicitly specify the active space in the former calculation, i.e. NEVPT2(8,8) in the input file; (ii) if you find there are 4 orbitals not so active in the latter calculation (meaning occupation numbers close to 0 or 2), you can specify NEVPT2(4,4) in the input file.

(3) If you are calculating the bond dissociation energy, you should check whether the sum or combination of active space from two fragments equals to the active space of the original molecule. You are supposed to analyze main components of the active space. For example, assuming the original molecule has a CAS(6,6) active space, and assuming it contains 2 singly occupied orbitals from fragment 1, two bonding orbitals and two anti-bonding orbitals constructed by two fragments. Then quintuplet CAS(4,4) should be used for fragment 1 and triplet CAS(2,2) for fragment 2.

(4) For the rigid/unrelaxed scanning of a chemical bond, you are always recommended to scan the bond from long distance to short distance, rather than from short to long. And you are recommended to write the keyword 'scan' in the Gaussian keyword #p line. This will perform the rigid/unrelaxed scan automatically. It is not recommended to calculate each scanned geometry manually.

4 User's Guide

4.1 Syntax Rules of AutoMR

Syntax rules of the input file of AutoMR is almost identical to those of Gaussian software. Therefore, it takes little time to become familiar with the usage of AutoMR. A simple input example is shown below

```
---------------------------------------------------------------
%mem=4GB
%nprocshared=4
#p CASSCF/cc-pVTZ
```

```
mokit{ist=1,readuhf='N2_cc-pVTZ_4.0_uhf.fchk' }
------------------------------------------------------------
```

For the memory and parallel settings, please read Section **4.2**. For the theoretical method and basis set, please read Section **4.3**. Here we focus on the keywords in mokit{}.

1. All keywords must be written in the curly bracket of 'mokit{}' in the Title Card line of a .gjf file.

2. Using upper or lower case of keywords in AutoMR makes no difference. For example, the following two lines have identical meanings:

(1) MOKIT{readuhf='a,fchk',ist=1,LocalM=Boys}
(2) mokit{readuhf='a,fchk',ist=1,localm=boys}

3. If 'readrhf', 'readuhf', or 'readno' keyword is used, a filename of the provided .fch(k) file must be included in a pair of single quotation marks ''. Do not use double quotation marks "" or no quotation marks.

4. If pure Cartesian functions of basis set are used in the provided .fch(k) file, you need to write keyword 'cart'. The default is pure spherical harmonic functions. And you do not need to write any keyword about this if you provide a .fch(k) file in which pure spherical harmonic functions are used.

5. Use a comma to separate different keywords. Do not use other symbols (like spacing).

6. User-defined basis sets (i.e. gen) and pseudopotentials (i.e. genecp) are not supported in the .gjf file currently. To use that, you can write one of the keywords 'readrhf', 'readuhf', or 'readno', and provided a .fch(k) file (in which the data of basis sets and pseudopotentials are recorded).

4.2 Memory and Parallel Settings

The settings of memory and the number of processors are identical to that in Gaussian. For example, the following syntax

```
%chk=16GB
%nprocshared=8
```

requests a AutoMR calculation use 16GB memory and 8 processors. Note that only the unit GB and %nprocshared is supported. Some DON'T things are listed below:
(1) Do not use unit MB, MW, GW, or %cpu..
(2) Do not write %chk since it is useless for AutoMR.

Since multireference computations often require large memory, it is recommended to use at least %mem=1GB for even a small molecule. Note that memory and parallel settings of OpenMolcas is controlled by variables in your ~/.bashrc (export MOLCAS_NPROCS, export MOLCAS_MEM),

not by AutoMR. You should refer to (Open)Molcas manual for details. And the memory and parallel settings of the BDF program is controlled by variables in the Shell script bdf-pkg/sbin/run.sh, you should properly modify the script before doing computations.

The memory and parallel settings are passed into various programs called by AutoMR (e.g. PySCF, Gaussian, etc). The AutoMR itself only needs negligible memory and usually run in a serial mode.

4.3 Keywords of supported methods in AutoMR

Currently, the keywords of all supported methods in AutoMR are: **GVB, CASCI, CASSCF, DMRGCI, DMRGSCF, NEVPT2, CASPT2, SDSPT2, MRMP2, MRCISD, MCPDFT**, **DFTCI**. More multi-configurational and multi-reference methods will be supported in the future. These terms should be written in the '#p …' line in the .gjf file.

There must be a '/' symbol between the method and the basis set, e.g. CASSCF/cc-pVTZ. AutoMR does not allow the use of a spacing to separate the method and basis set (which is allowed in Gaussian). When 'readrhf', 'readuhf', or 'readno' is used in mokit{}, the basis set after '/' symbol is usually useless, since the geometry and basis set data will be read from the given .fch(k) file. Note that, however, the user still needs to provide a basis set name, although it is not used in this case.

There is also an exception that the basis set after '/' symbol matters. If RI (see Section 4.4.29) approximation is turned on, the auxiliary basis set will be automatically determined (by AutoMR) according to the basis set. And if F12 (see Section 4.4.31) is turned on, the F12-CABS will be automatically determined (by AutoMR) according to the basis set, too.

4.3.1 GVB

Generalized Valence Bond theory.

Note that the original GAMESS source code can only deal with GVB up to 12 pairs. To go beyond that (which is routine type of calculation in AutoMR of MOKIT), you need to modify the GAMESS source code, please read Section 4.4.10 carefully.

4.3.2 CASSCF

Complete Active Space Self-consistent Field.

Currently excited state computations are not supported. Please read Section A2.3 for comments on excited state computations.

Please read related keyword CASSCF_prog in Section 4.4.12.

4.3.3 CASCI

Complete Active Space Configuration Interaction.

CASCI can be viewed as the 0-th step of the CASSCF step, i.e. CASSCF without orbital

optimization. It is always recommended to perform CASSCF rather than CASCI, expect for those who wish to compare the quality of different initial guesses, and/or do not want the CASSCF result.

Please read related keyword CASCI_prog in Section 4.4.11.

4.3.4 DMRGSCF

Here the term DMRGSCF actually means the DMRG-CASSCF method. Please also read Section 4.4.14 for related information.

4.3.5 DMRGCI

Here the term DMRGCI actually means the DMRG-CASCI method. Please also read Section 4.4.13 for related information.

DMRGCI can be viewed as the 0-th step of the DMRGSCF step, i.e. DMRGSCF without orbital optimization. It is always recommended to perform DMRGSCF rather than DMRGCI, expect for those who wish to compare the quality of different initial guesses, and/or do not want the DMRGSCF result.

4.3.6 NEVPT2

Second order N-Electron Valence state Perturbation Theory based on the CASSCF reference.

Please read related keyword NEVPT2_prog in Section 4.4.16.

4.3.7 CASPT2

Second order Perturbation Theory based on CASSCF reference.

Please read related keyword CASPT2_prog in Section 4.4.15.

Generally speaking, NEVPT2 is more recommended than CASPT2 since there is no need for IP-EA shift, real or imaginary shift in NEVPT2.

4.3.8 CASPT3

Third order Perturbation Theory based on CASSCF reference.

Only Molpro program can be called to perform CASPT3, and it is default.

4.3.9 SDSPT2

Static-dynamic-static multi-state multi-reference second-order perturbation theory (SDS-MS-MRPT2, or SDSPT2 for short). There are several restrictions when you use this method:

(1) only the single (electronic) state version can be used in automr, since only the ground state is calculated.

(2) SDSPT2 based on the CASCI reference is not supported currently. CASSCF-SDSPT2 is supported.

(3) background point charges are not supported.

This version of SDSPT2 is performed by the Xi'an CI module of BDF program. So you are assumed to have successfully installed BDF. You should cite this paper DOI: 10.1080/00268976.2017.1308029 if you use this method.

4.3.10 MRMP2

Second order Multi-reference Perturbation Theory based on CASSCF reference. After CASSCF converged, AutoMR will call the GAMESS program to perform MRMP2 calculation. No other program is supported. Also, DMRG-MRMP2 is not supported and AutoMR will abort in that case.

4.3.11 OVBMP2

Orthogonal valence bond Møller-Plesset 2 based on CASSCF reference. After CASSCF converged, AutoMR will call the Gaussian program to perform OVB-MP2 calculation. No other program is supported. Also, DMRG-OVB-MP2 is not supported and AutoMR will abort in that case.

Note that OVBMP2 is a keyword in AutoMR but OVB-MP2 is the method name. Do not mix them up. Some literature might call this method as "CASSCF MP2" but that is actually misleading (those authors were fooled by keywords "CASSCF MP2"in Gaussian .gjf file).

4.3.12 MRCISD

Multi-reference Configuration Interaction Singles and Doubles, based on the CASCI/CASSCF reference.

Please read related keyword MRCISD_prog in Section 4.4.17.

4.3.13 MCPDFT

Multi-configurational Pair Density Functional Theory, based on CASSCF reference. Note that MCPDFT is a keyword in AutoMR but MC-PDFT is the method name. Do not mix them up.

Note that if the active space is larger than (15,15), the MC-PDFT will be automatically switched to DMRG-PDFT. In this special case you need to install the QCMaquis package (interfaced with OpenMolcas) for DMRG computations. DMRG-PDFT is not supported in GAMESS currently.

Please read related keyword MCPDFT_prog in Section 4.4.19. Also note that in GAMESS, the MC-PDFT is only supported for version >= 2019(R2).

4.3.14 DFTCI

The DFT/MRCI method by Stefan Grimme, Mirko Waletzke, and Martin Kleinschmidt *et. al*. Note that the name of this method is DFT/MRCI, but you should write the keyword DFTCI in the input file of AutoMR, in order to perform DFT/MRCI computations.

Currently, to perform DFT/MRCI computations, you should have ORCA and DFT/MRCI program installed on your node/machine.

Note: full implementation of this keyword has not been finished yet.

4.3.15 MRCC

Multi-reference Coupled Cluster theory, based on the CASCI/CASSCF reference. Currently only the FIC-MRCC in ORCA(>=5.0.0) is supported.

4.4 List of AutoMR Keywords

If any of the 'readrhf', 'readuhf', and 'readno' keywords is used, there is no need to write Cartesian coordinates in .gjf file (in fact AutoMR will not read coordinates in such case), since the geometry is already provided in the specified .fch(k) file.

4.4.1 readrhf

Read RHF (or ROHF) orbitals from a specified .fch file. Do not provide a UHF-type .fch file using this keyword. This keyword is usually used along with another keyword ist=3 (see Section 4.4.4 ist).

4.4.2 readuhf

Read UHF orbitals from a specified .fch(k) file. AutoMR will firstly check the difference between alpha and beta MOs. If the difference is tiny, the wave function in .fch file will be identified as a RHF one and will call utility fch_u2r to generate a RHF-type .fch file (in which all beta information is deleted). Otherwise (i.e. truly UHF), AutoMR will generate UHF natural orbitals (UNO) using input UHF orbitals. It is strongly recommended to check the stability of UHF wave function (using keyword 'stable=opt' in Gaussian). An instable or not-the-lowest UHF solution sometimes leads to improper GVB or CASSCF results.

The author jxzou does not recommend the usage of UDFT orbitals. But in case you have to do that (e.g. forced by your Boss or driven by your curiosity), remember to add an extra keyword 'no10cycle' (see 4.4.24 for no10cycle).

This keyword is usually used along with another keyword ist=1 or 2 (see Section 4.4.4 ist). Note that do not provide a UNO .fch file with this keyword. If you want to use any type of NOs as the initial guess, see 'readno' in the following section.

4.4.3 readno

Read natural orbital occupation numbers (NOON) and natural orbitals (NO) from a specified .fch file. In this case, you must ensure that the 'Alpha Orbital Energies' section in the .fch file contains occupation numbers, not energy levels or something else, since the size of the active space will be determined according to the (threshold of the) occupation numbers.

With this keyword, you may try different NOs as the initial guess, like MP2 NOs, CCSD NOs, or some type of NOs generated by your own program. You may also use UNOs from UHF as the initial guess. In this case, it is equivalent to use the keyword 'readuhf' to read in the UHF orbitals and generate UNOs by MOKIT.

This keyword is usually used along with another keyword ist=5 (see Section 4.4.4 ist).

4.4.4 ist

Request the use of the i-th *st*rategy. Default is 0. This means: (1) if the spin of the target molecule is singlet, MOKIT will call the Gaussian software to perform RHF and UHF computations, then determine whether to change 'ist' to 1 or 3. If the $E_{UHF} = E_{RHF}$, ist will be changed to 3. If $E_{UHF} < E_{RHF}$, ist will become 1. (2) if not singlet, ist will be changed to 1 immediately.

For simple organic molecules which have multireference characters (like diradicals), the UHF performed by MOKIT (calling Gaussian) can usually find the lowest (and stable) UHF solution. But for complicated systems like binuclear transition metal complex, there often exist multiple UHF solutions. And the UHF solution found by MOKIT is not necessarily the lowest one. In this case you are recommended to do UHF computations by yourself and use ist=1 to read in the Gaussian .fch file. See a practical guide for advanced UHF computations on http://gaussian.com/afc/. If you can read Chinese, you are recommended to read Sobereva's blog http://sobereva.com/82.

Currently, there are 6 allowed values for ist:
0: meaning that if RHF wave function is stable, use strategy 3; otherwise use strategy 1
1: UHF -> UNO -> associated rotation -> GVB -> CASCI/CASSCF -> ...
2: UHF -> UNO -> (associated rotation ->) CASCI/CASSCF -> ...
3: RHF -> virtual orbital projection -> localization -> pairing -> GVB -> CASCI/CASSCF -> ...
4: RHF -> virtual orbital projection -> CASCI/CASSCF -> ...
5: NOs -> CASCI/CASSCF -> ...
The value 0 is recommended, if you do not know which one to choose.

4.4.5 LocalM

Specify the orbital localization method. Only the Boys (also called Foster-Boys) localization and Pipek-Mezey (PM) localization method are supported. The corresponding keywords are 'LocalM=Boys' and 'LocalM=PM'. By default, the PM localization is used.

Note: the Boys method will mix σ and π orbitals, while the PM method tends to keep them separated. These two methods make no difference when the target molecule contains only σ bonds (and possibly a few isolated π bonds). But if you are dealing with multiple π bonds or conjugated π systems like oligoacene(benzene, naphthalene, etc), or if you want the active space to contain only π orbitals, better use the PM method. The GVB and CASSCF optimized orbitals will be affected by the localization method sometimes.

For people who are keen on comparing initial guesses generated from different methods/algorithms, 'Local=Boys' is strongly recommended to be taken into consideration, to see whether a lower GVB, CASCI or CASSCF energy occurs.

4.4.6 CIonly

Skip the CASSCF orbital optimization in the CASPT2 or NEVPT2 job. Obviously, this keyword only applies to CASPT2 or NEVPT2 case. Writing CIonly means a CASCI -> CASPT2 or CASCI -> NEVPT2 job. In fact, the CASSCF orbital optimization is always recommended to be performed, unless it is too time-consuming, or you happen to want this type of result.

Note: if you simply need a CASCI computation, do not use CIonly in a CASSCF job, but simply write CASCI/basis_set in the keyword line of .gjf file.

4.4.7 Force

Request a calculation of the analytical nuclear gradient. Currently this keyword only applies to CASSCF. Geometry optimization is currently not supported, but you can use the generated files to perform optimization using corresponding software. Numerical gradient is not supported.

Note that this keyword do not have any attribute value, i.e. do not write 'force=.True.' but only 'force' in {}. Also keep in mind that force is negative gradient.

4.4.8 Cart

Request the use of Cartesian type atomic basis functions. The default basis type in AutoMR (i.e. spherical harmonic functions) will then be disabled. These two types of basis functions correspond to '6D 10F' (Cartesian functions) and '5D 7F' (spherical harmonic functions) in Gaussian. It is strongly recommended to use spherical harmonic functions, especially in all-electron relativistic computations (DKH2, X2C, etc).

Note that for computations involving the ORCA program, this keyword cannot be used since ORCA only support spherical harmonic functions.

If you don't know the meaning of 5D or 6D, you are referred to Schlegel and Frisch's paper (DOI: 10.1002/qua.560540202), and a good explanation from Chemissian http://www.chemissian.com/ch5. If you can read Chinese, you are recommended to read Sobereva's blog http://sobereva.com/51.

4.4.9 HF_prog

Specify the program for performing Hartree-Fock (HF) calculations. Supported programs are Gaussian(default), PySCF, PSI4 and ORCA. If the input molecule is singlet, RHF and broken symmetry UHF (plus wave function stability analysis) will be successively performed. If not singlet, only UHF (plus wave function stability analysis) will be performed.

It is strongly recommended to use the default HF_prog (i.e. Gaussian), since the SCF algorithm in Gaussian is the most effect in almost all cases, among all quantum software packages. The only case when you are recommended to use PySCF/PSI4/ORCA, is that if your studied molecule is large and cannot be further simplified ($N_{basis} > 1500$), you can consider turn on the RI approximation of HF to accelerate computations in PySCF, PSI4 or ORCA. For example, in this special case you are supposed to write

mokit{RI,HF_prog=PySCF} (or PSI4, ORCA if you wish)

in the input file of AutoMR. For small to medium-size molecules ($N_{basis} < 1500$), better not use PySCF/PSI4/ORCA since their SCF is usually slower or harder to converge than that of Gaussian.

Note: implementation of this keyword for PySCF/PSI4/ORCA has not been finished yet.

4.4.10 GVB_prog

Specify the GVB program. Supported programs are GAMESS(default) and Gaussian. The second-order SCF (SOSCF) algorithm in GAMESS is used to converge the GVB wave function. The GAMESS version >=2017 is strongly recommended. Older versions of GAMESS may work or may not, since they are not tested by the author jxzou. It is always recommended to use the default program GAMESS, rather than Gaussian (due to poor convergence for transition metal).

Note the original GAMESS can only do GVB up to 12 pairs. Nowadays we can do a black-box GVB computation with hundreds of pairs. So, to go beyond 12 pairs, you need to modify and re-compile the source code of GAMESS.

MOKIT offers a Shell script to help you automatically handle this. Assuming you've compiled GAMESS before (i.e. all *.o files are still in **gamess/object/** directory of GAMESS), now you simply need to copy two files (modify_GMS1.sh and modify_GMS2.f90) from mokit/src/ into **gamess/source/** directory, and run './modify_GMS1.sh'. The script will modify source code and re-compile GAMESS, taking about 2 minutes. The linked GAMESS executable will be gamess.01.x. If you already had an executable named gamess.01.x, please rename it to another filename. Otherwise it will be destroyed and replaced during re-compilation.

Besides, for GAMESS earlier than version 2021-R1, it only supports 32 CPU cores. If your machine has more cores (and if you want to use >32 cores), you need to modify the variable MAXCPUS in file gamess/ddi/compddi. See a simple guide in file src/modify_GMS_beyond32CPU.txt. Since GAMESS 2021, the MAXCPUS is already set to 128, so modification is not needed.

It is strongly recommended to check whether the GAMESS runs normally after modifying. Such check can be done by running './runall 01' in gamess/ directory, where 01 means checking the executable gamess.01.x. This is same as checking for version 00. All tests are supposed to be passed successfully.

4.4.11 CASCI_prog

Specify the program for performing the CASCI calculation, e.g. CASCI_prog=PySCF. Supported programs are PySCF(default), Molpro, GAMESS, OpenMolcas, Gaussian, ORCA, BDF, PSI4 and Dalton.

If you use some old versions of BDF as the CASCI_prog, you may find the NOONs are zero. In that case you should update to the latest version of BDF.

4.4.12 CASSCF_prog

Specify the program for performing the CASSCF calculation, e.g. CASSCF_prog=PySCF. Supported programs are PySCF(default), Molpro, GAMESS, OpenMolcas, Gaussian, ORCA, BDF, PSI4 and Dalton. If you want to use a CASSCF program instead of PySCF, I recommend Molpro, OpenMolcas or GAMESS.

If you use some old versions of BDF as the CASSCF_prog, you may find the NOONs are zero. In that case you should update to the latest version of BDF.

Currently automr cannot run excited state calculations. Please read related comments in Section A2.3.

4.4.13 DMRGCI_prog

Specify the program for performing DMRG-CASCI calculation, e.g. DMRGCI_prog=PySCF. Currently only PySCF is supported and it is the default program.

Note that in fact the DMRG-CASCI calculations are performed by Block-1.5 and PySCF, not only by PySCF. Thus you should install the Block-1.5 program. And you should also cite corresponding reference of the Block-1.5 program.

Also note that the MPI version used by Block is probably contradicted with MPI version used by ORCA, thus you have to comment one version of MPI environment variables at a time. Or you can use a Shell script to submit the automr job, in which your desired MPI environment variables are written.

4.4.14 DMRGSCF_prog

Specify the program for performing DMRG-CASSCF calculation, e.g. DMRGSCF_prog=PySCF. Currently only PySCF is supported and it is the default program.

Note that in fact the DMRG-CASSCF calculations are performed by Block-1.5 and PySCF, not only by PySCF. Thus you should install the Block-1.5 program. And you should also cite corresponding reference of the Block-1.5 program.

Also note that the MPI version used by Block is probably contradicted with MPI version used by ORCA, thus you have to comment one version of MPI environment variables at a time. Or you can use a Shell script to submit the automr job, in which your desired MPI environment variables are written.

4.4.15 CASPT2_prog

Specify the program for performing CASPT2 calculation, e.g. CASPT2_prog=OpenMolcas. Currently only OpenMolcas(default), Molpro and ORCA are supported. All core orbitals are not frozen. Note that a default IP-EA shift 0.25 a.u. will be applied and it cannot be modified. If your CASPT2 results are sensitive to the IP-EA shift, it implies that CASPT2 is not suitable to your problem. Another two types of shift (the real or imaginary shift) is not supported.

Generally speaking, NEVPT2 is more recommended than CASPT2 since there is no need for IP-EA shift, real or imaginary shift in NEVPT2.

4.4.16 NEVPT2_prog

Specify the program for performing NEVPT2 calculation, e.g. NEVPT2_prog=PySCF. Currently supported programs are PySCF(default), Molpro, ORCA, OpenMolcas and BDF. All core orbitals are not frozen.

Note that there exist at least two variants of the NEVPT2: SC-NEVPT2 and FIC-NEVPT2(aka PC-NEVPT2). By default, for NEVPT2_prog=PySCF/Molpro/ORCA/OpenMolcas, SC-NEVPT2 is chosen; while for NEVPT2_prog=BDF, FIC-NEVPT2 is chosen. To turn on the FIC-NEVPT2 when using PySCF/Molpro/ORCA/OpenMolcas, please read Section 4.4.34. Also note that

(1) When you specify NEVPT2_prog=Molpro or OpenMolcas, both of the SC-NEVPT2 and FIC-NEVPT2 energies are actually printed in Molpro/OpenMolcas output (.out file). You can open the output file and read it manually, if you need that energy.

(2) If you specify NEVPT2_prog=OpenMolcas, it actually turns into a DMRG-NEVPT2 computation, no matter how large/small the size of active space is. In this special case, you need to install the QCMaquis package (interfaced with OpenMolcas) for DMRG computations.

4.4.17 MRCISD_prog

Specify the program for performing MRCISD calculation. By default, MRCISD_prog=OpenMolcas. You MUST also specify a contraction type, please read Section 4.4.20 carefully.

Currently, AutoMR supports the interfaces of three MRCISD variants:

(1) uncontracted MRCISD

(2) internally contracted MRCISD (ic-MRCISD)

(3) fully internally contracted MRCISD (FIC-MRCISD)

where the computational cost and accuracy is (1)>(2)>(3). The ic- and FIC-MRCISD are both approximations of uncontracted MRCISD. If the Davidson size-consistency correction energy is added, then the method should be denoted as MRCISD+Q. It is recommended to use ic-MRCISD+Q or FIC-MRCISD+Q in practical calculations since the uncontracted MRCISD is often too expensive.

AutoMR is able to call OpenMolcas/Molpro/ORCA/Gaussian/GAMESS/PSI4/Dalton programs to perform MRCISD. Currently all core orbitals are not frozen. MRCISD based on the DMRG reference is not supported currently.

If MRCISD_prog=OpenMolcas, only variants (1) and (2) are supported. The Davidson correction for size-consistency can only be calculated for (2).

If MRCISD_prog=ORCA, only (1) and (3) are supported. The Davidson correction can be provided for both methods. However, only spherical harmonic functions of basis sets are supported in ORCA. But this is often not a problem, since it is recommended to use spherical harmonic functions than Cartesian functions.

Note that you should use ORCA>=5.0.0 for FIC-MRCISD+Q computations since older versions have a tiny bug in the Davidson correction.

If MRCISD_prog=Gaussian, PSI4 or Dalton, only (1) is supported and no Davidson correction is given.

If MRCISD_prog=GAMESS, only (1) is supported. The Davidson correction energy will also be printed.

If MRCISD_prog=Molpro, only (2) is supported. The Davidson correction energy will also be printed. Note that the MRCIC program of Molpro will be called to perform ic-MRCISD. This ic-MRCISD is not exactly identical to that of OpenMolcas, so their electronic energies are different with (but close to) each other. If you want to compare (relative) electronic energies of two molecules using ic-MRCISD method, please choose the same type, i.e. both using Molpro or both using OpenMolcas.

4.4.18 MRMP2_prog

Specify the program for performing MRMP2 calculation. Only GAMESS is supported and this is the default.

4.4.19 MCPDFT_prog

Specify the program for performing MC-PDFT calculation. Only OpenMolcas(default) and GAMESS are supported.

Note that if the active space is larger than (15,15), the MC-PDFT will be automatically switched to DMRG-PDFT. In this special case you need to install the QCMaquis package (interfaced with OpenMolcas) for DMRG computations. DMRG-PDFT is not supported in GAMESS currently.

Also note that in GAMESS, the MC-PDFT is only supported for version >= 2019(R2), and currently it can only be run in serial.

4.4.20 CtrType

Specify the contraction type of the MRCISD method. The default value is 0. When you specify the MRCISD method and the MRCISD_prog, you must assign an integer for this variable, where
1 for uncontracted MRCISD
2 for ic-MRCISD
3 for FIC-MRCISD.

Generally, the ic- and FIC-MRCISD methods are recommended. If your calculation involves Cartesian-type basis functions, then you cannot use FIC-MRCISD in ORCA. In such case, you can choose ic-MRCISD in OpenMolcas instead.

4.4.21 MaxM

Specify the bond dimension MaxM in DMRG-related calculations. The default values is 1000 (e.g. MaxM=1000). When maxM increases, the DMRG-CASCI energy will become closer to the CASCI energy, but the computational cost increases as well. The value 1000 is suitable for common cases. But do check whether it is valid for your system. For example, three computations using different MaxM (e.g. 500, 1000, 1500) may be conducted to study whether the energy converges with MaxM.

4.4.22 hardwfn

This option can only be applied to CASCI/CASSCF calculations using PySCF, OpenMolcas, GAMESS or PSI4. By specifying 'hardwfn', AutoMR will add extra keywords into the CAS input files to ensure a better convergence. Note that normally you do not need this keyword, and it is useless if you specify other programs as the CAS solver.

4.4.23 crazywfn

This option can only be applied to CASCI/CASSCF calculations using PySCF, OpenMolcas,

GAMESS or PSI4. By specifying 'crazywfn', AutoMR will add extra keywords (more than those of 'hardwfn') into the CAS input files to ensure a better convergence. Note that usually you do not need this keyword, and it is useless if you specify other programs as the CAS solver.

For example, when the N_2 molecule is stretched to d(N-N) = 4.0 Å, this is a system which features strong correlation and requires a CAS(6,6) active space. The Davidson iterative diagonalization in determinant CASCI (using GAMESS) may not find the singlet state in the lowest 5 states. In this case, specifying 'crazywfn' will increase the NSTATE to 10, so that the singlet state can be found.

4.4.24 no10cycle

Skip the 10 cycles of SCF calculations after importing MOs. By default, AutoMR will do several (up to 10) cycles of RHF or UHF after importing MOs from .fch file. This has two advantages: (1) check whether the SCF converges immediately. If not, the MO in .fch file may be wrong, or there exists some bug of fch2py (which has never been observed by the author). (2) it will slightly improve the orthonormality of the input MOs, although the orthonormality is already good. Thus usually you do not need to write this keyword.

If you start with MOs which are not RHF or UHF MOs (e,g, NOs, UDFT MOs, etc), you must specify 'no10cycle' in mokit{}. Because any iteration of input MOs will change them, which is not what you want.

4.4.25 charge

This keyword has identical meaning with the same keyword in Gaussian software, i.e. including background point charges in calculations. This keyword is supported for almost all methods in AutoMR. Methods which are incompatible with background point charges will signal errors immediately. The charge-charge and charge-nuclei interaction energies are both included in all electronic energies printed (UHF, GVB, CASSCF, NEVPT2, etc).

The including of background point charges is useful for QM/MM calculations or fragmentation-based linear scaling methods (like GEBF, Many-body expansion, etc).

Note: please write this keyword in mokit{}. **DO NOT WRITE** this keyword in the Route Section of .gjf file (i.e. '#p …' line).

4.4.26 OtPDF

The choice of the on-top pair density functional. This keyword has identical meaning with the keyword KSDFT in (Open)Molcas software. Currently available functionals are tPBE(default), tBLYP, tLSDA, trevPBE, tOPBE, ftPBE, ftBLYP, ftLSDA, ftrevPBE and ftOPBE. For more details please refer to the Molcas manual. Note that the available functionals depends on your version of OpenMolcas or GAMESS. Old versions may not support some of the functionals.

Note that in GAMESS, the MC-PDFT is only supported for version >= 2019(R2).

4.4.27 DKH2

Request the 2nd order scalar relativistic Douglas–Kroll–Hess (DKH2) correction to the one-electron Hamiltonian. Note that: (1) the two keywords DKH2 and X2C are mutually exclusive; (2) you should use all-electron basis sets like 'cc-pVTZ-DK', 'x2c-TZVPall' or 'ANO-RCC-VDZ' for all-electron relativistic calculations; (3) it is strongly not recommended to use Cartesian-type function of basis set (severe numerical instability observed), please just use the default spherical harmonic functions.

Currently only the point nuclei charge distribution is supported.

4.4.28 X2C

Request the scalar relativistic X2C (eXact-two-Component) corrections to the one-electron Hamiltonian. Note that: (1) the two keywords DKH2 and X2C are mutually exclusive; (2) you should use all-electron basis sets like 'cc-pVTZ-DK', 'x2c-TZVPall' or 'ANO-RCC-VDZ' for all-electron relativistic calculations; (3) it is strongly not recommended to use Cartesian-type function of basis set (severe numerical instability observed), please just use the default spherical harmonic functions.

Also note that the RHF/UHF is performed using Gaussian called by AutoMR, and GVB is performed using GAMESS called by AutoMR. Since Gaussian and GAMESS do not support X2C, in these steps the X2C will be replaced by DKH2. According to the limited tests of the author jxzou, MOs resulting from DKH2 and X2C are similar and often converge in few cycles.

Currently only the point nuclei charge distribution is supported.

4.4.29 RI

Request to turn on the RI-JK approximation for two-electron integrals in CASSCF. Default is off. Please just write 'mokit{RI}', do not write 'mokit{RI=True}' or 'mokit{RI on}'. The other two types of RI approximations RI-J and RIJCOSX are not supported.

This option currently can only be used in CASSCF computations conducted by PySCF, OpenMolcas, Molpro, ORCA, or PSI4 programs. To learn more about the auxiliary basis set used in RI-JK approximation, see the following Section 4.4.30.

4.4.30 RIJK_bas

Specify an auxiliary basis set for RI-JK approximation in CASSCF computations conducted by ORCA. Usually you do not need to specify this, since the automr program will automatically assign a proper auxiliary basis set according to the basis set (e.g. def2/JK for def2TZVP, cc-pVTZ/JK for cc-pVTZ). You can simply open the output file of automr and see what auxiliary basis set is assigned.

In the current version of OpenMolcas, there is no auxiliary basis set in it. The AutoMR program

in MOKIT will automatically transformed the needed basis set file and put that into $MOLCAS/basis_library/jk_Basis/.

4.4.31 F12

Request to turn on the F12 technique in NEVPT2 computations conducted by ORCA. F12 is not used by default. But if you turn on F12, RI (see Section 4.4.29) will be turned on as a byproduct.

This option currently can only be used in CASSCF and CASSCF-NEVPT2 computations conducted by ORCA program, i.e. you need to specify

<div align="center">CASSCF_prog=ORCA,NEVPT2_prog=ORCA,F12</div>

in mokit{}. To learn more about the near-complete auxiliary basis set used in F12 technique, see the following Section 4.4.32.

4.4.32 F12_cabs

Specify a near-complete auxiliary basis set for the F12 technique in NEVPT2 computations conducted by ORCA. Usually you do not need to specify this, since the automr program will automatically assign a proper auxiliary basis set according to the basis set (e.g. cc-pVTZ-F12-CABS for cc-pVTZ-F12). You can simply open the output file of automr and see what CABS is assigned.

4.4.33 DLPNO

Request to turn on the DLPNO technique in NEVPT2 computations conducted by ORCA. DLPNO is not used by default. But if you turn on DLPNO, RI (see 4.4.29) and FIC (see 4.4.34) will be turned on as byproducts.

This option currently can only be used in CASSCF and CASSCF-NEVPT2 computations conducted by ORCA program, i.e. you need to specify

<div align="center">CASSCF_prog=ORCA,NEVPT2_prog=ORCA,DLPNO</div>

in mokit{}. Of course it can be combined with F12 to perform RI-DLPNO-FIC-NEVPT2-F12 computations for large systems, where the keywords should be

<div align="center">CASSCF_prog=ORCA,NEVPT2_prog=ORCA,DLPNO,F12</div>

in mokit{}.

4.4.34 FIC

Request the FIC- variant of NEVPT2 (i.e. FIC-NEVPT2) to be used. By default SC-NEVPT2 is invoked if you specify NEVPT2 in route section (#p NEVPT2/…) and use PySCF/Molpro/OpenMolcas/ORCA program as NEVPT2_prog. But if you specify NEVPT2_prog=BDF, this option is turned on as a byproduct and FIC-NEVPT2 will then be performed. SC-NEVPT2 is not supported in BDF. FIC-NEVPT2 is not supported in PySCF.

4.4.35 ON_thres

The threshold of UNO occupation numbers for determining the number of pairs in GVB computation. The default value is 0.99999 and it means all unoccupied UNOs with occupation numbers >1e-5 will be chosen (as well as their corresponding occupied UNOs) for subsequent orbital localization. Similarly, setting ON_thres=0.98 corresponds to the fact that any unoccupied UNOs with occupation numbers >0.02 will be chosen. DO NOT modify the default value unless you are an experienced user.

4.4.36 NMR

Request the calculation of nuclear shielding constants. Currently only the CASSF method is supported, and CASSCF_prog must be set to Dalton. Interfaces to other methods such as RASSCF may be supported in the future. Gauge-Independent Atomic Orbital (GIAO) method is used to compute the NMR shielding tensors.

Note that:

(1) This keyword should be written in mokit{}, not in the Gaussian keyword line (#p …).

(2) The chemical shift of an atom or element is the difference of nuclear shielding constants between the studied molecule and the standard reference molecule. For example, hydrogen atoms in tetramethylsilane(TMS) is usually used as the reference for chemical shifts in ^{1}H-NMR.

(3) For practical computations of chemical shifts, it is recommended to use specially designed basis sets like pcSseg-1 or pcSseg-2. Larger basis sets like pcSseg-3, def2-QZVP or cc-pVQZ would be better but often too expensive.

4.5 List of Utilities in MOKIT

The utilities of transferring MOs are summarized in the following figure:
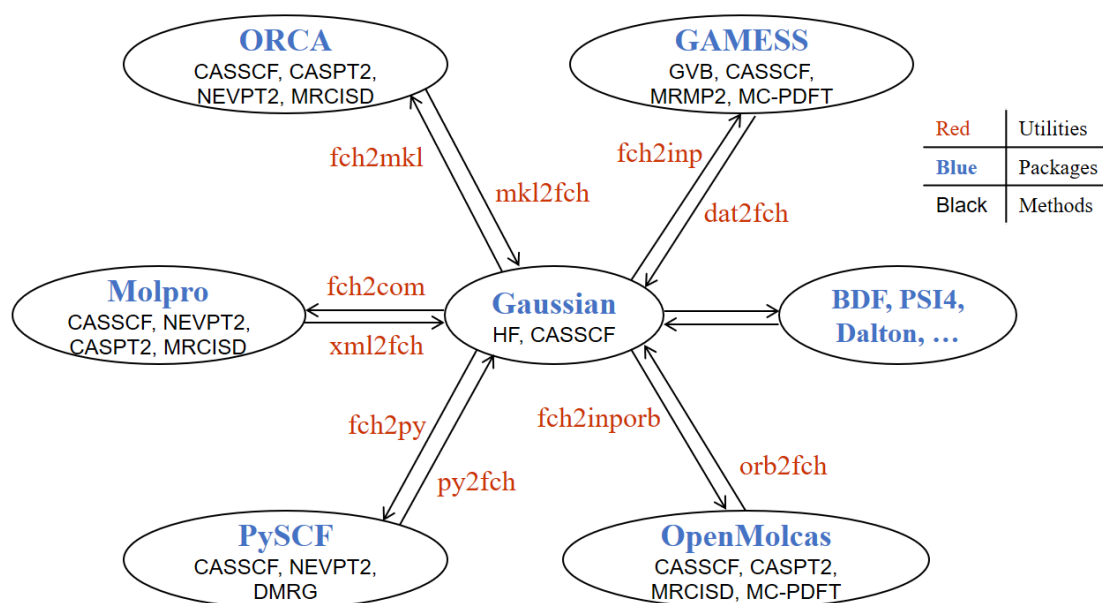
For detailed explanations of all utilities, please read the following subsections.

4.5.1 add_bgcharge_to_inp

This utility is designed to add background charges to the input file of various software packages. If you do not use background charges in your computation, you can skip this section. But if you do use them (e.g. in subsystems of fragmentation-based or embedding methods), they are not recorded in any .fch(k) file. This can be viewed as a defect of the .fch(k) file. Therefore, the generated input file by utilities fch2com, fch2inp, fch2iporb, fch2psi or bas_fch2py will contain no background charges either.

To add background charges into the input file, you have to provide a .chg file which contains information of those charges. An example of such file is shown below

```
2
4.0     0.0     0.0     0.1
-4.0    0.0     0.0     0.1
```

The first line holds the number of point charges. While the charges are written starting from the second line, with format *x, y, z, charge*.

Note that in all computations of the AutoMR program, this situation is explicitly considered. This utility will be automatically called if needed. The only situation when you need this utility is merely using utilities fch2com, fch2inp, fch2iporb, fch2psi or bas_fch2py (and of course, with background charges).

4.5.2 bas_fch2py

Generate a PySCF .py file from a Gaussian .fch file. The Cartesian coordinates and basis set

data are written in the .py file. Note: AutoMR does not use any built-in basis sets of PySCF, but always generates the basis set data from .fch file. This procedure ensures an (almost) exactly identical setting of basis set in Gaussian and PySCF.

This utility is in fact a wrapper of two utilities 'fch2inp' and 'bas_gms2py'. So if you only want to compile this utility, you have to compile 'fch2inp' and 'bas_gms2py' additionally.

Note that if you use background charges in your studied system, the background charges are not recorded in the .fch(k) file. So there are no background charges in the generated .py file, either. To add background charges, you need to use the utility add_bgcharge_to_inp (see Section 4.5.1).

4.5.3 bas_gau2molcas

Transform a basis set file in Gaussian format to another in (Open)Molcas format. If you turn on RI (see Section 4.4.29) and use OpenMolcas as the CASSCF_prog, there is no RI-JI auxiliary basis set file in current version of OpenMolcas package. Therefore, this utility will be called automatically to transform the auxiliary basis set file in $MOKIT_ROOT/basis/ directory to the (Open)Molcas syntax. And the transformed file will normally be in $MOLCAS/basis_library/jk_Basis/.

You can, of course, use this utility by yourself. An example is shown below

bas_gau2molcas def2-universal-jkfit

Assuming the basis set file def2-universal-jkfit (in Gaussian format) has been put in the current directory, this command will generate a basis set file named DEF2-UNIVERSAL-JKFIT (in (Open)Molcas format).

4.5.4 bas_gms2bdf

Generate two BDF files (_bdf.inp and .BAS) from a GAMESS .inp/.dat file. Cartesian coordinates are written in the _bdf.inp file, while the basis set data is held in .BAS file. Note that BDF does not support Cartesian-type basis functions, so only spherical harmonic functions will be used. The 'ISPHER' keyword in .inp/.dat file (if any) will be ignored. One example is shown below

bas_gms2bdf a.inp
Generate a_bdf.inp and A.BAS files, for R(O)HF or UHF type wave function.

4.5.5 bas_gms2dal

Generate Dalton .dal and .mol files from a GAMESS .inp/.dat file. The Cartesian coordinates and basis set data are written in the .mol file.

Two examples are shown and explained below

(1) bas_gms2dal a.inp

Generate .mol and .dal file, in which the keywords 'Cartesian' for all atom are written (in order to use pure Cartesian type of basis functions)

(2) bas_gms2dal a.inp -sph
Generate .,mol and .dal file, in which the keywords 'Cartesian' of each atom are not written (in order to use pure spherical harmonic type of basis functions).

4.5.6 bas_gms2molcas

Generate an (Open)Molcas .input file from a GAMESS .inp/.dat file. The Cartesian coordinates and basis set data are written in the .input file.
Two examples are shown and explained below

(1) bas_gms2molcas a.inp
Generate .inp file, in which the keywords 'Cartesian' of each atom are written (in order to use pure Cartesian type of basis functions)

(2) bas_gms2molcas a.inp -sph
Generate .inp file, in which the keywords 'Cartesian' of each atom are not written (in order to use pure spherical harmonic type of basis functions).

4.5.7 bas_gms2molpro

Generate a Molpro .com file from a GAMESS .inp or .dat file. The Cartesian coordinates and basis set data are written in the .com file.
Two examples are shown and explained below

(1) bas_gms2molpro a.inp
Generate .inp file, in which the keyword 'Cartesian' is written (in order to use pure Cartesian type of basis functions)

(2) bas_gms2molpro a.inp -sph
Generate .inp file, in which the keyword 'Cartesian' is not written (in order to use pure spherical harmonic type of basis functions).

4.5.8 bas_gms2psi

Generate a PSI4 _psi.inp file from a GAMESS .inp/.dat file. The Cartesian coordinates and basis set data are written in the _psi.inp file.
Two examples are shown and explained below

(1) bas_gms2psi a.inp

Generate a_psi.inp file, in which the keyword 'cartesian' is written (in order to use pure Cartesian type of basis functions)

(2) bas_gms2psi a.inp -sph
Generate a_psi.inp file, in which the keyword 'spherical' is written (in order to use pure spherical harmonic type of basis functions).

4.5.9 bas_gms2py

Generate a PySCF .py file from a GAMESS .inp/.dat file. The Cartesian coordinates and basis set data are written in the .py file.
Two examples are shown and explained below

(1) bas_gms2py a.inp
Generate .inp file, in which the keyword 'mol.cart = True' is written (in order to use pure Cartesian type of basis functions)

(2) bas_gms2py a.inp -sph
Generate .inp file, in which the keyword 'mol.cart = True' is not activated (in order to use pure spherical harmonic type of basis functions).

4.5.10 bdf2fch

Transfer MOs from BDF (.scforb/.inporb/.casorb, etc) to Gaussian .fch file. Note that bdf2fch can not generate a .fch file from scratch. The user must provide a .fch(k) file, and MOs in that file will be replaced. Two examples are shown and explained below

(1) bdf2fch a.scforb a.fch
This is used for transferring R(O)HF, UHF or CASSCF orbitals.

(2) bdf2fch a.casorb a.fch -no
This is used for transferring CASCI or CASSCF NOs and NOONs.

NOTE: the recommended way is to firstly use Gaussian to generate a .fch file (with keywords 'nosymm int=nobasistransform'), then generate the _bdf.inp file from .fch file. After BDF computations finished, you can transfer MOs from .scforb/.casorb back to .fch file. This procedure seems a little bit tedious, but it ensures an exact reproduce of energy in BDF.
This utility supports only spherical harmonic functions. To transfer MOs from Gaussian to BDF, see Section 4.5.15 fch2bdf.

4.5.11 bdf2mkl

Transfer MOs from BDF (.scforb/.inporb/.casorb, etc) to ORCA. The ORCA .inp and .mkl file will be generated. This utility is actually a wrapper of two utilities bdf2fch and fch2mkl. Thus bdf2mkl can not generate a .fch file from scratch, either. The user must provide a .fch(k) file, and MOs in that file will be replaced. Two examples are shown and explained below

(1) bdf2mkl a.scforb a.fch
This is used for transferring RHF, ROHF, UHF or CASSCF orbitals.

(2) bdf2mkl a.casorb a.fch -no
This is used for transferring CASCI or CASSCF NOs and NOONs.

NOTE: the recommended way is to firstly use Gaussian to generate a .fch file (with keywords 'nosymm int=nobasistransform'), then generate the _bdf.inp file from .fch file. After BDF computations finished, you can transfer MOs from .scforb/.casorb back to .fch file. This procedure seems a little bit tedious, but it ensures an exact reproduce of energy in BDF.

4.5.12 dal2fch

Transfer MOs from Dalton (i.e. DALTON.MOPUN file) to Gaussian .fch(k) file. Note that dal2fch can not generate a .fch file from scratch. The user must provide a .fch(k) file, and MOs in that file will be replaced. Two examples are shown and explained below

(1) dal2fch a.dat a.fch
This is used for transferring R(O)HF or CASSCF orbitals.

(2) dal2fch a.dat a.fch -no
This is used for transferring CASCI/CASSCF natural orbitals and corresponding natural orbital occupation numbers. To transfer MOs from Gaussian to Dalton, see Section 4.5.17 fch2dal.

4.5.13 dat2fch

Transfer MOs from GAMESS (.inp/.dat file) to Gaussian .fch file. Note that dat2fch can not generate a .fch file from scratch. The user must provide a .fch(k) file, and MOs in that file will be replaced. Four examples are shown and explained below

(1) dat2fch a.dat a.fch
This is used for transferring R(O)HF, UHF or CASSCF orbitals.

(2) dat2fch a.dat a.fch -gvb 5
This is used for transferring GVB orbitals for spin singlet molecule. The order of GVB orbitals is

different between Gaussian and GAMESS. Thus you must specify '-gvb [npair]' to tell the utility the number of GVB pairs, so that dat2fch can adjust the order of MOs.

(3) dat2fch a.dat a.fch -gvb 5 -open 1
This is used for transferring GVB orbitals for non-singlet molecule. In this way, you tell the utility the number of GVB pairs and singly-occupied orbitals, so that dat2fch can adjust the order of MOs.

(4) dat2fch a.dat a.fch -no 5 10
This is used for transferring natural orbitals (NOs) of CASCI/CASSCF. In this way, you tell the utility the beginning index and final index of NOs in .fch file, so that dat2fch can work correctly.

NOTE: the recommended way is to firstly use Gaussian to generate a .fch file (with keywords 'nosymm int=nobasistransform'), then generate the .inp file from .fch file. After GAMESS computations finished, you can transfer MOs from .dat back to .fch file. This procedure seems a little bit tedious, but it ensures an exact reproduce of energy in GAMESS.

This utility supports two types of basis functions: (1) pure spherical harmonic functions; (2) pure Cartesian functions. To transfer MOs from Gaussian to GAMESS, see Section 4.5.18 fch2inp.

4.5.14 extract_noon2fch

Extract natural orbital occupation numbers (NOONs) from the following types of files
(1) .out file of PySCF
(2) .dat file of GAMESS
(3) .gms file of GAMESS
(4) .out file of ORCA
(5) .out file of PSI4
and write NOONs (as the 'Alpha Orbital Energies' section) into a given .fch file. This is for the convenience of visualizing orbitals using GaussView or Multiwfn+VMD.

4.5.15 fch2bdf

Generate three BDF files (_bdf.inp, .BAS, .scforb/.inporb) from a Gaussian .fch(k) file. Cartesian coordinates are written in the _bdf.inp file, while the basis set data is held in .BAS file. The molecular orbitals are written in the .scforb/.inporb file. Note that BDF does not support Cartesian-type basis functions, so only spherical harmonic functions will be used. If there exists any Cartesian-type basis function in .fch(k) file, this utility will signal errors. Two examples are shown and explained below

(1) fch2bdf a.fch
This is used for transferring RHF/ROHF/UHF orbitals. Three files will be generated: a_bdf.inp, A.BAS and a.scforb. The data in 'Alpha Orbital Energies' and 'Beta Orbital Energies' sections in .fch file will be read and printed into the a.scforb file.

(2) `fch2bdf a.fch -no`

This is used for transferring NOs. Three files will be generated: a_bdf.inp, A.BAS and a.inporb. Note the data of 'Alpha Orbital Energies' section in .fch file (assumed occupation numbers) will be read and printed into the a.inporb file, but the occupation numbers do not affect subsequent computations.

This utility will call another two utilities 'fch2inp' and 'bas_gms2bdf'. So if you want to compile fch2bdf, you have to compile 'fch2inp' and 'bas_gms2bdf' additionally.

Note that to transfer HF orbitals, the data in 'Alpha Orbital Energies' and 'Beta Orbital Energies' section should be genuine orbital energies, since BDF program will use these values. Random values (like zero) will affect SCF computations and thus it cannot converge in 1 cycle (or even fails to converge). This is totally different with other quantum chemistry software packages where only orbitals are useful and orbital energies are useless. When transferring NOs, the occupation numbers in 'Alpha Orbital Energies' section do not affect subsequent computations.

To transfer MOs from BDF back to Gaussian, see Section 4.5.10 bdf2fch.

4.5.16 fch2com

Generate a Molpro .com file from a Gaussian .fch(k) file, with alpha MOs written in a .a file. One example is shown below

`fch2com a.fch`

This is used for transferring R(O)HF, UHF or CASSCF orbitals.

This utility will call another two utilities 'fch2inp' and 'bas_gms2molpro'. So if you want to compile fch2com, you have to compile 'fch2inp' and 'bas_gms2molpro' additionally.

Note that in Windows* OS, any file with .com suffix/extension may be automatically associated with system, in which case double click of the mouse to open this file does not work. You have to right click on the .com file and choose 'open with'. You can modify the suffix/extension to .inp if you do not like that.

Note that if you use background charges in your studied system, the background charges are not recorded in the .fch(k) file. So there are no background charges in the generated .com file, either. To add background charges, you need to use the utility add_bgcharge_to_inp (see Section 4.5.1).

To transfer MOs from Molpro back to Gaussian, see Section 4.5.33 xml2fch.

4.5.17 fch2dal

Generate Dalton .dal and .mol files from a Gaussian .fch(k) file, where MOs are written in .dal file, and the Cartesian coordinates as well as basis set data are written in .mol file. One example is shown below

`fch2dal a.fch`

This is used for transferring R(O)HF or CASSCF orbitals. Note that there is no UHF method (or

any methods based on UHF), thus you cannot use a .fch file containing UHF orbitals to transfer orbitals. To transfer MOs from Dalton back to Gaussian, see Section 4.5.12 dal2fch.

4.5.18 fch2inp

Generate a GAMESS .inp file from a Gaussian .fch(k) file, with MOs written in the .inp file. The keywords in .inp file is already suitable for common simple calculations, but do check or modify it if you have additional requirements.

Note that due to the different types of MOs (R(O)HF, UHF, GVB, and CASSCF orbitals), the fch2inp offers different options. Three examples are shown and explained below

(1) fch2inp a.fch
This is used for transferring R(O)HF, UHF or CASSCF orbitals.

(2) fch2inp a.fch -gvb 5
This is used for transferring GVB orbitals for spin singlet molecule. The order of GVB orbitals is different between Gaussian and GAMESS. Thus you must specify '-gvb [npair]' to tell the utility fch2inp the number of GVB pairs, so that fch2inp can adjust the order of MOs.

(3) fch2inp a.fch -gvb 5 -open 1
This is used for transferring GVB orbitals for non-singlet molecule. In this way, you tell the utility fch2inp the number of GVB pairs and singly-occupied orbitals, so that fch2inp can adjust the order of MOs.

This utility supports two types of basis functions: (1) pure spherical harmonic functions; (2) pure Cartesian functions. To transfer MOs from GAMESS back to Gaussian, see Section 4.5.13 dat2fch.
Note that if you use background charges in your studied system, the background charges are not recorded in the .fch(k) file. So there are no background charges in the generated .inp file, either. To add background charges, you need to use the utility add_bgcharge_to_inp (see Section 4.5.1).

4.5.19 fch2inporb

Transfer MOs from Gaussian to (Open)Molcas. A .input file and a .INPORB file will be generated. The .input file is the input file of (Open)Molcas, and it contains the geometry, basis set data and keywords. The .INPORB file contains the MOs. Two examples are shown and explained below

(1) fch2inporb a.fch
This is used for transferring R(O)HF, UHF or CASSCF orbitals.

(2) fch2inporb a.fch -no

This is used for transferring NOs.

The option '-no' means reading NOONs and NOs from .fch(k) file and printing them into the .INPORB file. The NOONs written in the .INPORB file does not affect the calculations in OpenMolcas at all. It just make the file appear more readable.

This utility will call another two utilities 'fch2inp' and 'bas_gms2molcas'. So if you want to compile fch2inporb, you have to compile 'fch2inp' and 'bas_gms2molcas' additionally.

Note that if you use background charges in your studied system, the background charges are not recorded in the .fch(k) file. So there are no background charges in the generated .input file, either. To add background charges, you need to use the utility add_bgcharge_to_inp (see Section 4.5.1).

To transfer MOs from (Open)Molcas back to Gaussian, see Section 4.5.29 orb2fch.

4.5.20 fch2mkl

Transfer MOs from Gaussian to ORCA. One .inp file and one .mkl file will be generated. The .input file of ORCA holds the geometry, basis set data and keywords. The .mkl file contains the MOs. One example is shown below

fch2mkl a.fch

This is used for transferring R(O)HF, UHF or CASSCF orbitals. Two files will be generated: a_o.inp and a_o.mkl. The '_o' characters are added to avoid file overwritten in case that there is already a xxx.inp file in the current directory (for example, generated by utility fch2inp).

To transfer MOs from ORCA back to Gaussian, see Section 4.5.27 and 4.5.28.
Note 1: Assuming your ORCA input file is a.inp, you need to run 'orca_2mkl a_o -gbw' to generate the ORCA a.gbw file since ORCA cannot read a.mkl file directly.
Note 2: The default keywords in .inp file does not contain any RI approximations, and settings of VeryTightSCF are written (the author does this to ensure an exact reproduce of energy in ORCA). You can modify keywords as you wish, but note that an exact reproduce of energy may not be assured.

Note that if you use background charges in your studied system, the background charges are not recorded in the .fch(k) file. So there are no background charges in the generated .inp or .mkl file, either. To add background charges, you need to use the utility add_bgcharge_to_inp (see Section 4.5.1).

4.5.21 fch2psi

Transfer MOs from Gaussian to PSI4. Two or three files will be generated: *_psi.inp, *.A, and *.B (if Beta MOs exist). The _psi.inp file of PSI4 holds the geometry, basis set data and keywords. The .A (and .B) file contains the Alpha (Beta) MOs. One example is shown below

fch2psi a.fch

This is used for transferring R(O)HF, UHF or CASSCF orbitals.

Note that this utility will call another two utilities – fch2inp and bas_gms2psi, remember to compile them additionally.

Note that if you use background charges in your studied system, the background charges are not recorded in the .fch(k) file. So there are no background charges in the generated .com file, either. To add background charges, you need to use the utility add_bgcharge_to_inp (see Section 4.5.1).

PSI4 can generate a .fch(k) file after calculation finished, which is equivalent to transferring MOs from PSI4 back to Gaussian.

4.5.22 fch2qm4d

Transfer MOs from Gaussian to QM4D. Input files for QM4D (.xyz, .inp, .xml and basis files for each element) will be generated. One example is shown below

fch2qm4d a.fch

This is used for transferring RHF or UHF orbitals.

Note 1: Currently QM4D supports only Cartesian-type basis sets. Thus this utility will signal error if you provide a .fch(k) file which has spherical harmonic functions (i.e. 5D 7F).

Note 2: QM4D supports ECP, but currently it seems to use spherical harmonic functions in ECP, so even if you specify '6D 10F' in Gaussian, the electronic energies of Gaussian *v.s.* QM4D will not be equal to each other. This tiny defect does not affect transferring MOs, and SCF can be converged in several cycles with almost no energy change.

Note that if you use background charges in your studied system, the background charges are not recorded in the .fch(k) file. So there are no background charges in the generated .inp file, either. To add background charges, you need to use the utility add_bgcharge_to_inp (see Section 4.5.1).

4.5.23 fch_mo_copy

Copy MOs from one .fch file into another .fch file. An example is shown below

./fch_mo_copy a.fch b.fch

The default is to copy Alpha MOs in a.fch to Alpha MOs in b.fch. There are 4 optional parameters '-aa', '-ab', '-ba', '-bb'. For example, '-ab' means copying Alpha MOs from a.fch to Beta MOs in b.fch.

4.5.24 fch_u2r

Transform a UHF-type .fch file into a RHF-type one. Only alpha MOs are retained.

4.5.25 frag_guess_wfn

This utility has two functions: (1) perform a SCF (i.e. HF or DFT) computation using initial guess constructed from molecular orbitals of fragments. (2) generate GAMESS .inp file for GKS-EDA or Morokuma-EDA calculation.

The Gaussian software package will be called to perform SCF computation for each fragment. The input file is exactly the Gaussian .gjf file, in which the atoms, charge and spin multiplicities of each fragment are properly defined. See examples in Section 5.3.

Explanations of function (1):
Note: implementation of this function is not yet finished.
The SCF computations of radicals and transition-metal-containing molecules often suffers from multiple UHF/UKS solutions. Using a proper fragment guess (i.e., the 'guess(fragment=N)' keyword in Gaussian), one can then obtain a desired SCF solution or a lower energy solution after SCF converged and a check of wavefunction stability of the whole system. However, the wavefunction stability of each fragment in the initial guess cannot be assured in Gaussian, thus not efficient for complicated molecules. This utility – frag_guess_wfn – can ensure the wavefunction stability of each fragment and allow the usage of RHF/UHF (or RKS/UKS) for different fragments, which would improve the quality of initial guess and save time for subsequent SCF computation of the whole system. Examples of the fragment guess can be found at http://gaussian.com/afc. If you can read Chinese, a nice introduction to this topic from Sobereva's blog
《谈谈片段组合波函数与自旋极化单重态》 http://sobereva.com/82
is recommended.

Explanations of function (2):
The Morokuma-EDA calculation is supported by GAMESS-US program, while GKS-EDA calculation is supported by the XEDA program (a modified GAMESS-US program from Prof. Peifeng Su of Xiamen University, supi@xmu.edu.cn). These methods are very useful but often suffer from SCF convergence failure due to the "just so-so" SCF convergence techniques in GAMESS. Even if all SCF converge, their wave function stabilities cannot be checked in GAMESS.
Fortunately, these two methods can read wave function of each fragment (and of total system in GKS-EDA), if we can provide such wave function as initial guess. Therefore, the utility frag_guess_wfn will call Gaussian to perform SCF calculations for each fragment (and for total system in GKS-EDA), then write all MOs into the .inp file, so that all SCF steps during an EDA calculation will converge immediately.
If UHF/UDFT method is specified, wave function stability of each fragment (and of total system in GKS-EDA) will be checked. This is very important for biradical and transition-metal-containing systems. If any fragment is singlet and UHF/UDFT method is specified, broken

symmetry initial guess (i.e., guess=mix) will be automatically applied.

The Windows* pre-built executable frag_guess_wfn will be provided at Releases in the near future. Note that you should define the environment variable %GAUSS_EXEDIR% (in Windows* OS) before using this utility, since Gaussian program would be called to perform SCF computations. For example, search "environment variable" in your Windows search bar and click Edit to add a new environment variable, and set it as the correct G03W, G09W or G16W directory. A screenshot is shown below
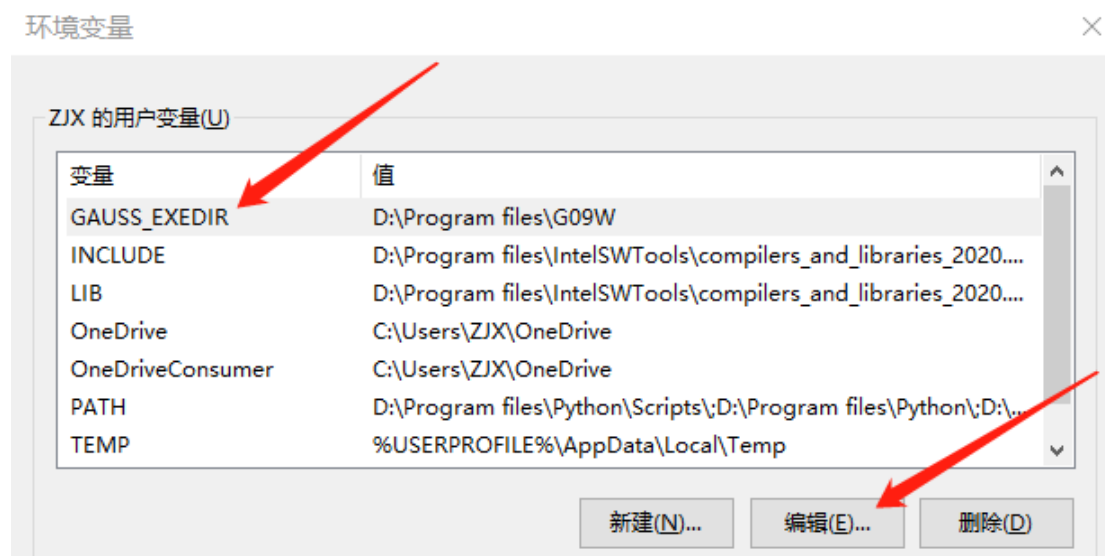
4.5.26 gvb_sort_pairs

Sort (part of) MOs in descending order of the pair coefficients of the 1st natural orbital in each pair. This utility is designed only for the GAMESS .dat file. A new .dat file will be generated, in which the sorted MOs and pair coefficients are held.

4.5.27 mkl2fch

Transfer MOs from ORCA .mkl file into Gaussian .fch(k) file. A .fch(k) file must be provided by the user, in which the geometry and basis set data are valid (i.e. specifying 'noysmm int=nobasistransform 5D 7F' in Gaussian).

Note that the number of digits in .mkl file is only 7, and the scientific notation is not used. Therefore, the transferred MOs will not be very accurate. This should be sufficient for visualizing orbitals and common wavefunction amalysis, but may cause up to 10^{-5} a.u. error on electronic energy in further computations. This can be viewed as a defect of the orca_2mkl utility in ORCA program. You are recommended to bring up an issue or request in the ORCA forum (https://orcaforum.kofo.mpg.de), to suggest ORCA developers to fix this. If the scientific notation is not used, then 10 digits of MO coefficients should be sufficient for further computations.

Two examples are shown and explained below

(1) mkl2fch a.mkl a.fch
This is used for transferring R(O)HF or UHF orbitals.

(2) mkl2fch a.mkl a.fch -no
This is used for transferring NOs.

To transfer MOs from Gaussian to ORCA, see Section 4.5.20 fch2mkl.

4.5.28 mkl2gjf

Generate the Gaussian .gjf file from the ORCA .mkl file. The Cartesian coordinates, basis set data and MOs (optional) will be printed into the .gjf file. Two examples are shown and explained below

(1) mkl2gjf a.mkl
Generate the Gaussian .gjf file from the ORCA .mkl file. The Cartesian coordinates, basis set data will be printed into the .gjf file. There would be no MOs in it.

(2) mkl2gjf a.mkl -mo
Generate the Gaussian .gjf file from the ORCA .mkl file. The Cartesian coordinates, basis set data plus MOs will be printed into the .gjf file.

Note that the ORCA .mkl file has a defect: it does not contain ECP/PP information. Therefore, if you did not use ECP in your ORCA calculations, there would be no problem. But in case you used ECP, there would be no ECP data in the generated .gjf file (you must add them manually).

4.5.29 orb2fch

Transfer MOs from (Ope)nMolcas *Orb file (e.g. .ScfOrb, .RasOrb.1, .UnaOrb, .UhfOrb, etc) to Gaussian .fch(k) file. A .fch(k) file must be provided by the user, in which the geometry and basis set data are available. Five examples are shown and explained below

(1) orb2fch a.ScfOrb a.fch
This is used for transferring RHF orbitals.

(2) orb2fch a.RasOrb a.fch
This is used for transferring CASCI/CASSCF or RASCI/RASSCF (pseudo)canonical orbitals.

(3) orb2fch a.UnaOrb a.fch -no
This is used for transferring UNOs.

(4) orb2fch a.UhfOrb a.fch

This is used for transferring UHF alpha and beta MOs.

(5) orb2fch a.RasOrb.1 a.fch -no

This is used for transferring CASCI/CASSCF or RASCI/RASSCF NOs.

To transfer MOs from Gaussian to (Open)Molcas, see Section 4.5.19 fch2inporb.

4.5.30 mo_svd

Output the singular values of the overlap matrix between two sets of MOs. This utility will first read the atomic overlap matrix from a given file, then calculate the overlap matrix between two sets of MOs. Finally, perform singular value decomposition (SVD) on the molecular orbital overlap matrix and print information about singular values. The first two command line arguments can be both Gaussian .fch files, or both OpenMolcas orbital files (.INBORB, .RasOrb, etc). The third argument is a Gaussian .log file or a OpenMolcas output file, in which the atomic overlap matrix is written.

The singular values can be used to measure the overlap or similarity of two sets of MOs. If all singular values are close to 1, the compared two sets of MOs are very similar. Any singular value being close to 0 means there exists at least 1 distinct orbital.

4.5.31 solve_ON_matrix

Compute the occupation number matrix (a non-diagonal matrix) of a set of MOs. Assuming you have a .fch(k) file which holds some kind of MOs, and another .fch(k) file which holds some kind of NOs and corresponding NOONs, then this utility can compute the occupation numbers of this set of MOs (transformed from NOs and NOONs). Of course, in this case the occupation number matrix of MOs is not a diagonal matrix, only the diagonal elements are written into the 'Alpha Orbital Energies' section of the file which holds MOs.

4.5.32 replace_xyz_in_inp

Replace the Cartesian coordinates in an input file, by coordinates from a given .xyz file or an output file. For the input/output file, currently only (Open)Molcas and Molpro formats are supported. Other formats will be supported in the near future. Four examples are shown below

(1) replace_xyz_in_inp a.xyz b.input -molcas

Replace the Cartesian coordinates in b.input file, by coordinates from the a.xyz file. A file named b_new.input will be generated.

(2) replace_xyz_in_inp a.out b.input -molcas

Replace the Cartesian coordinates in b.input file, by coordinates from an (Open)Molcas output file.

A file named b_new.input will be generated.

(3) replace_xyz_in_inp a.xyz b.com -molpro
Replace the Cartesian coordinates in b.com file, by coordinates from the a.xyz file. A file named b_new.com will be generated.

(4) replace_xyz_in_inp a.out b.com -molpro
Replace the Cartesian coordinates in b.com file, by coordinates from a Molpro output file. A file named b_new.com will be generated.

4.5.33 xml2fch

Transfer MOs from Molpro .xml file to Gaussian .fch(k) file. Note that xml2fch cannot generate a .fch file from scratch. The user must provide one .fch file, in which the 'Alpha Orbital Energies' and 'Alpha MOs' sections (and possibly Beta sections) will be replaced by occupation numbers and MOs from Molpro .xml file. Three examples are shown and explained below

(1) xml2fch a.xml a.fch
This is used for transferring R(O)HF or UHF orbitals.

(2) xml2fch a.xml a.fch -no
This is used for transferring NOs.

To transfer MOs from Gaussian to Molpro, see Section 4.5.16 fch2com.

The utilities below are compiled by **f2py**, which is a Fortran to Python interface generator. These utilities are not binary executable files, but dynamic libraries *.so in $MOKIT_ROOT/lib. They can only be imported in a Python script. And this is the reason that why one of the environment variables of MOKIT is 'PYTHONPATH', not 'LD_LIBRARY_PATH'. These utilities can also be viewed as APIs (Application Programming Interface), but they are described in this section not in Section 4.6, just for better comparison with other utilities of transferring MOs.

4.5.34 fch2py

Transfer MOs from Gaussian to PySCF. By importing fch2py, PySCF Python script is able to read alpha and/or beta MOs from a provided .fch file. All attributes are shown below

fch2py(fchname, nbf, nif, ab, mf.mo_coeff)

There are 4 parameters required by fch2py: (1) filename of .fch(k) file; (2) the number of basis functions; (3) the number of molecular orbitals; and (4) a character 'a'/'b' for reading alpha/beta orbitals. You should write these contents into a Python script, and run that script using the python executable. If you want a more detailed example, just run any AutoMR job and see all *.py files generated.

4.5.35 py2fch

Export MOs from PySCF to a Gaussian .fch file. Note that py2fch cannot generate a .fch file from scratch. The user must provide one .fch file, in which the 'Alpha Orbital Energies' and 'Alpha MOs' sections (or beta orbital counterpart) will be replaced by occupation numbers and MOs from PySCF.

The provided .fch file must contain exactly the same geometry and basis set data as those data of PySCF. To obtain a valid .fch file, it is strongly recommended to use the bas_fch2py utility to generate a .py file from a .fch file first, and then import py2fch in this generated .py file. These descriptions seem a bit of tedious. But a rigorous transferring of MOs between two programs or two files is ensured. All attributes of py2fch are shown below

py2fch('a.fch', nbf, nif, mf.mo_coeff, ab, ev, gen_density)

The first few parameters are identical to those in Section 4.5.31 fch2py. The parameter ev means eigenvalues, it is supposed to contain orbital energies or orbital occupation numbers. The last parameter gen_density is a bool variable, where True/False meaning whether or not generating Total SCF Density (as well as writing density into .fch file) using mf.mo_coeff and ev. If gen_density is set to be True, the parameter ev must contain orbital occupation numbers, not orbital energies. Because density would be computed using MOs and occupation numbers. If gen_density is set to be False, you can assign proper values to ev as your wish.

Obviously this utility/module is supposed to used usually along with other three utilities: bas_fch2py (Section 4.5.2), fch2py (Section 4.5.34) and load_mol_from_fch (Section 4.6.1).

4.6 APIs in MOKIT

Currently only Python APIs are provided. C/C++ APIs may be provided in the future. Some commonly used Python APIs are shown below

load_mol_from_fch(fchname)
loc(fchname, method, idx)
read_int1e_from_gau_log(logname, itype, nbf)
read_mo_from_fch(fchname, nbf, nif, ab)
read_density_from_gau_log(logname, itype, nbf)
read_density_from_fch(fchname, itype, nbf)

write_pyscf_dm_into_fch(fchname, nbf, dm, itype, force)
gen_ao_tdm(logname, nbf, nif, mo, istate)
export_mat_into_txt(txtname, n, mat, lower, label)
read_natom_from_pdb(pdbname, natom)
read_nframe_from_pdb(pdbname, nframe)
read_iframe_from_pdb(pdbname, iframe, natom, cell, elem, resname, coor)
write_frame_into_pdb(pdbname, iframe, natom, cell, elem, resname, coor, append)

Meanings of frequently appeared arguments are
fchname: filename of .fch(k) file
logname: filename of .log/.out file
nbf: the number of basis functions
nif: the number of (linear-independent) MOs

Meanings of other arguments are shown in below subsections.

4.6.1 load_mol_from_fch

Load a PySCF mol object from a Gaussian .fch(k) file. For example, run python in Shell

```
$python
>>>from pyscf import scf
>>>from gaussian import load_mol_from_fch
>>>mol = load_mol_from_fch(fchname='00-h2o_cc-pVDZ_0.96_rhf.fchk')
>>>mf = scf.RHF(mol).run()
```

where the module gaussian is actually the file $MOKIT_ROOT/lib/gaussian.py.

4.6.2 loc

Perform orbital localization for a given .fch(k) file. Two localization methods are supported: Foster-Boys or Pipek-Mezey. For example, run python in Shell

```
$python
>>>from gaussian import loc
>>>loc(fchname='benzene_5D7F_rhf.fchk', method='pm', idx=range(6,21))
```

Only 'boys' or 'pm' is allowed for the 2nd argument method. You should specify the orbital indices or range in the 3rd argument. Note that the starting integer index is 0 in Python convention, and the final index cannot be reached. So range(6,21) means orbitals 7-21, which are valence occupied orbitals for benzene. The localized orbitals will be exported/written into a new file with suffix *_LMO.fch (in this case, it is benzene_5D7F_rhf_LMO.fch).

For system containing only σ bonds, these two methods make little difference. While for

system containing σ and π bonds, the Boys localization method tends to mix σ and π bonds, which leads to "banana" bonds. The PM localization method tends to keep σ and π separated. If you want to analyze localized π bonds after localization, you should choose method='pm'.

4.6.3 read_int1e_from_gau_log

Read various one-electron integral matrices from a Gaussian output file. For example, read AO-basis overlap

```
$python
>>>import rwwfn
>>>S = rwwfn.read_int1e_from_gau_log(logname='00-h2o_cc-pVDZ_1.5.log',itype=1,nbf=24)
```

Attribute itype:
The type of one-electron integral matrices. Allowed values are 1,2,3,4 for Overlap, Kinetic Energy, Potential Energy, and Core Hamiltonian, respectively.

4.6.4 read_mo_from_fch

Read MOs from a Gaussian .fch(k) file. For example

```
$python
>>>import rwwfn
>>>mo = rwwfn.read_mo_from_fch(fchname='00-h2o_cc-pVDZ_1.5.fchk',nbf=24,nif=24,ab='a')
```

Attribute ab:
character with length=1, 'a'/'b' for reading alpha/beta orbitals.

4.6.5 read_density_from_gau_log

Read various types of density matrix from a Gaussian output file. For example, read the Alpha Density Matrix from a .log file

```
$python
>>>import rwwfn
>>>den = rwwfn.read_density_from_gau_log(logname='00-h2o_cc-pVDZ_1.5.log',
itype=2,nbf=24)
```

Attribute itype:
1/2/3 for Total/Alpha/Beta Density Matrix.

4.6.6 read_density_from_fch

Read various types of density matrix from a Gaussian .fch(k) file. For example, read the Total SCF Density from a .fchk file

```
$python
>>>import rwwfn
>>>den = rwwfn.read_density_from_fch(fchname='00-h2o_cc-pVDZ_1.5.fchk',
itype=1,nbf=24)
```

Attribute itype:

itype	type of density	itype	type of density
1	Total SCF Density	6	Spin MP2 Density
2	Spin SCF Density	7	Total CC Density
3	Total CI Density	8	Spin CC Density
4	Spin CI Density	9	Total CI Rho(1)
5	Total MP2 Density	10	Spin CI Rho(1)

4.6.7 write_pyscf_dm_into_fch

Write a PySCF density matrix into a given Gaussian .fch(k) file. This module does two things: (1) deal with the order of basis functions and their normalization factors, (2) then export density matrix into a given .fch(k) file. All attributes of this module are shown below

write_pyscf_dm_into_fch(fchname, nbf, dm, itype, force)

where dm is the PySCF density matrix with dimension (nbf,nbf). itype has the same meaning with that in Section 4.6.5. force is a bool variable, and its meaning is:
(1) If the string corresponding to itype (e.g. 'Total SCF Density') can be found/located in the given .fch file, this parameter will not be used, i.e. setting True/False does not matter.
(2) If the string corresponding to itype cannot be found, setting force=True will enforce writing density matrix into the given file; setting force=False will stop/abort the program and signal errors immediately (this can be used to check whether desired strings exists in the specified file).

Note that this module cannot generate a .fch(k) file from scratch, the user must provide one such file. The recommended approach is firstly using the module bas_fch2py to generate PySCF input file or using load_mol_from_fch to generate proper PySCF object, then do computations in PySCF. Finally using the module write_pyscf_dm_into_fch to export desired density matrix.

4.6.8 gen_ao_tdm

Generate AO-basis ground->excited state Transition Density Matrix for CIS/TDHF/TDDFT from a Gaussian output file. Currently only closed shell is taken into consideration. For example, read the S0->S1 Transition Density Matrix from a Gaussian .log file (with iop(9/40=5) specified in .gjf file)

```
$python
>>>import rwwfn, excited
>>>mo = rwwfn.read_mo_from_fch(fchname='00-h2o_cc-pVDZ_0.96_rhf.fchk',nbf=24,
nif=24,ab='a')
>>>tdm = excited.gen_ao_tdm(logname='00-h2o_cc-pVDZ_0.96_rhf.log',nbf=24,nif=24,
mo=mo,istate=1)
>>>rwwfn.export_mat_into_txt(txtname='h2o_tdm.txt',n=24,mat=tdm,lower=False,label='transiti
on density matrix')
```

Attribute istate:
The i-th excited state. For example, i=1 for the first excited state.

The last python statement means exporting the transition density matrix into a plain text file, see the API below.

4.6.9 export_mat_into_txt

Export a square matrix into a plain text file. The example of exporting transition density matrix is shown in Section 4.6.6. Here I offer one more example – export the lower triangle of a symmetric AO-basis overlap matrix

```
$python
>>>import rwwfn
>>>S = rwwfn.read_int1e_from_gau_log(logname='00-h2o_cc-pVDZ_1.5.log',itype=1,nbf=24)
>>>rwwfn.export_mat_into_txt(txtname='ovlp.txt',n=24,mat=S,lower=True,label='Overlap')
```

Attribute	Explanation
txtname	file name
mat	the square matrix with dimension (n,n)
lower	True/False for whether or not printing only the lower triangle part of a matrix
label	a string, the meaning of exported matrix (provided by yourself)

4.6.10 read_natom_from_pdb

Read the number of atoms from a given pdb file. If there exists more than one frame in the pdb file, only the 1st frame will be detected.

4.6.11 read_nframe_from_pdb

Read the number of frames from a given pdb file.

4.6.12 read_iframe_from_pdb

Read the i-th frame from a given pdb file.

4.6.13 write_frame_into_pdb

Write a frame into the given pdb file. A python script is shown below, to illustrate how to use these pdb-related APIs:

```
import rwgeom as rg
natom = rg.read_natom_from_pdb(pdbname='test.pdb')
cell, elem, resname, coor = rg.read_iframe_from_pdb('test.pdb', 10, natom)
for i in range(0,natom):
    s1 = elem[i][0].decode('utf-8')
    s2 = elem[i][1].decode('utf-8')
    print(s1, s2)
rg.write_frame_into_pdb('a.pdb', 10, natom, cell, elem, resname, coor, False)
```

5 Examples

5.1 Examples of AutoMR

See examples in $MOKIT_ROOT/examples/automr. More details to be added.

5.2 Examples of Transferring MOs

See examples in $MOKIT_ROOT/examples/utilities. More details to be added.

5.3 Examples of using utility frag_guess_wfn

Introduction of the utility frag_guess_wfn has been provided in Section 4.5.25. Here several examples will be shown to illustrate how this utility is used and how it works.

5.3.1 Construct initial guess wave function from fragments

More details to be added.

5.3.2 Morokuma-EDA

The input file of H_2O-NH_3 complex is shown below

```
%chk=nh3_h2o_m.chk
%mem=16GB
%nprocshared=16
#p RHF/cc-pVDZ guess(fragment=2)

{morokuma}

0 1 0 1 0 1
N(fragment=1)    -1.67861672    -0.49374924    -0.00513612
H(fragment=1)    -1.39223602    -1.00824506     0.80312136
H(fragment=1)    -1.24821305     0.40883590     0.00449440
H(fragment=1)    -1.40093406    -0.98667606    -0.82970600
O(fragment=2)     0.68224018     0.74345713     0.00303850
H(fragment=2)     1.37946396     0.10226230    -0.15300059
H(fragment=2)     0.95548613     1.60139824    -0.32993853
```

This is clearly just a Gaussian .gjf file, and it looks very similar to two other types of calculations: 1) fragment guess calculation in Section 5.3.1; 2) BSSE calculation. The order of fragments is the same as that required in GAMESS. That is to say, you must define atoms in the order of monomer 1, monomer 2, monomer 3, etc. To conveniently add '(fragment=N)' in each row, you may need some advanced text editor like Sublime Text 3, UltraEdit, VSCode, etc.

In Title Card line, {morokuma} must be written to tell the utility frag_guess_wfn you want the input file of Morokuma-EDA. Currently Morokuma-EDA in GAMESS has several restrictions:

(1) Only the RHF method can be used. ROHF, UHF or any DFT methods cannot be used. Both the total system and every fragment should be treated with RHF. This means broken bonds cannot be dealt with.

(2) Spherical harmonic functions cannot be used. The utility frag_guess_wfn will automatically switch to Cartesian fucntions (i.e., 6D 10F) when calling Gaussian to perform SCF computations.

Assuming the filename is nh3_h2o_m.gjf, you simply need to run

frag_guess_wfn nh3_h2o_m.gjf

Since the molecule is small, you can obtain a file named nh3_h2o_m.inp in seconds. MOs of two fragments will be written into this file. You can submit the file nh3_h2o_m.inp to GAMESS for Morokuma-EDA computations. All SCF of fragments will converge in 1-2 cycles. The initial guess of orbitals of the total system (i.e. supermolecule) is not written in .inp file, but constructed from fragments within GAMESS, and it is supposed to be converged soon.

Simply replacing the {morokuma} by {gks} means you want the input file of GKS-EDA. UHF and UDFT methods can be applied in GKS-EDA. Thus it is powerful, especially when dealing with biradical system, where the symmetry broken UDFT calculations are necessary. See the following section for more examples.

5.3.3 GKS-EDA

To perform GKS-EDA calculations, you need the XEDA program (request via email to Prof. Peifeng Su of Xiamen University, supi@xmu.edu.cn). Usually you will receive a binary package, which can be directly used after corresponding environment variables are written. There is no need for installation or compilation.

Here I offer two examples to illustrate how to use the utility frag_wfn_guess to generate the input file of GKS-EDA. The first example: $[Ti(H_2O)_6]^{3+}$ cation, whose input file is shown below

%chk=Ti.chk
%mem=32GB
%nprocshared=16
#p UPBE1PBE/gen guess(fragment=2)

{gks}

3 2 3 2 0 1

Ti(fragment=1)	0.00000000	0.00000000	0.00000000
H(fragment=2)	0.00000000	2.68428109	-0.78751709
H(fragment=2)	2.68128062	0.78751744	0.00000000
H(fragment=2)	0.00000000	-2.68428109	0.78751709
H(fragment=2)	0.78751709	0.00000000	-2.68228109
H(fragment=2)	-2.68128062	0.78751744	0.00000000
H(fragment=2)	-0.78751709	0.00000000	2.68228109
O(fragment=2)	0.00000000	2.10100000	0.00000000
H(fragment=2)	0.00000000	2.68428109	0.78751709
O(fragment=2)	0.00000000	0.00000000	2.09900000
H(fragment=2)	0.78751709	0.00000000	2.68228109
O(fragment=2)	0.00000000	0.00000000	-2.09900000

H(fragment=2)	-0.78751709	0.00000000	-2.68228109
O(fragment=2)	2.09800000	0.00000000	0.00000000
H(fragment=2)	2.68128062	-0.78751744	0.00000000
O(fragment=2)	-2.09800000	0.00000000	0.00000000
H(fragment=2)	-2.68128062	-0.78751744	0.00000000
O(fragment=2)	0.00000000	-2.10100000	0.00000000
H(fragment=2)	0.00000000	-2.68428109	-0.78751709

```
Ti 0
cc-pVTZ
****
H O 0
cc-pVDZ
****
```

Mixed basis set, user-defined basis set or ECP/PP are all supported. DO REMEMBER to type at least 2 blank lines after the basis set data. The only exception of basis set is that definition for each atom tag is unsupported. For example, the following definition

```
1 0
cc-pVTZ
****
2 0
cc-pVDZ
```

is currently NOT supported for utility frag_guess_wfn. In this example the UHF method is applied only for illustration, you can use DFT methods if you like.

Assuming the filename is Ti.gjf, you simply need to run

```
frag_guess_wfn Ti.gjf &
```

in bash. Electronic energies of fragments may be directly printed onto the screen. You can run instead

```
frag_guess_wfn Ti.gjf >Ti.out 2>&1 &
```

if you do not like things printed on screen. This example may take about 20 min to generate the desired .inp file. Five SCF jobs will be performed in succession:
(1) monomer 1 in its own basis;
(2) monomer 2 in its own basis;
(3) monomer 1 in all basis;
(4) monomer 2 in all basis;
(5) supermolecule in all basis.

Utility frag_guess_wfn will automatically construct the initial guess for job (5) using MOs from job (1) and (2). This will save some time. After all 5 jobs normally terminated, the utility fch2inp will be automatically called to transfer 5 sets of MOs from Gaussian .fch files into one GAMESS .inp file. Proper keywords (including DFT name in DFTTYP, solvent name and radii of solute atoms in $PCM, etc) have been written in the generated .inp file. You can, of course, open this file and modify keywords according to your needs (e.g. dispersion correction).

For HF methods, all 5 SCF jobs during GKS-EDA will converge in 1-2 cycles. While for UDFT, they may take about 10 cycles since the DFT integral grids (and functional implementation details possibly) in GAMESS are not identical to those in Gaussian. Note that when using DFT methods, grid settings '$DFT NRAD=99 NLEB=590 $END' will be automatically added into the .inp file, to mimic the 'int=ultrafine' in Gaussian. If implicit solvent model is also applied, each SCF will take more cycles, since implementation details in GAMESS have differences with those in Gaussian (but the energy usually steadily decreases).

You can write an alias into your ~/.bashrc for convenience of running XEDA, for example,

alias xeda='/home/jxzou/software/xeda/rungms'

Once the calculation of frag_guess_wfn is done (and you've modified keywords according to your needs), you can simply run in bash

xeda Ti.inp 00 16 >& Ti.gms &

to perform the GKS-EDA calculation, where the 16 is the number of cores for parallel computation.

Then we come to the second example: the ethane molecule. Assuming we want to see the interaction of two ▪CH3 radicals, such fragmentation will involve **breaking a covalent bond**, which leads to an alpha unpaired electron in one ▪CH3 radical and a beta unpaired electron in another radical. In this case we need to specify **negative spin** in the input file, which represents the beta spin:

```
%mem=4GB
%nprocshared=4
#p UHF/cc-pVTZ guess(fragment=2)

{gks}

0 1 0 2 0 -2
C(fragment=1)    -3.42837266     0.18776078     0.00000000
H(fragment=1)    -3.07171823    -0.82104923     0.00000000
H(fragment=1)    -3.07169982     0.69215897    -0.87365150
H(fragment=1)    -4.49837266     0.18777396     0.00000000
C(fragment=2)    -2.91503044     0.91371705     1.25740497
H(fragment=2)    -1.84503225     0.91200998     1.25838332
```

| H(fragment=2) | -3.27008700 | 1.92309006 | 1.25642758 |
| H(fragment=2) | -3.27329940 | 0.41044905 | 2.13105517 |

It is equivalent to interchange 2/-2. The coupling of an alpha unpaired electron with a beta unpaired electron leads to the total spin singlet.

The GKS-EDA supports up to 20 fragments. For convenience only examples containing two fragments are shown above.

5.3.4 SAPT

The utility frag_guess_wfn is also able to generate the input file of SAPT computation in PSI4. Currently only SAPT0 (and sSAPT0) is supported. More advanced methods like SAPT2+ and SAPT2+(3)δMP2 will be supported in the near future. Similarly, frag_guess_wfn will first call the Gaussian to perform HF/DFT computations and then generate the input file of SAPT job for PSI4. Be aware that SAPT cannot deal with strong interactions come from two fragments by breaking covalent bonds (in that case you should use the GKS-EDA method).

Here is a SAPT example:

%mem=16GB
%nprocshared=8
#p HF/jun-cc-pVDZ guess(fragment=2)

{sapt,bronze}

0 1 0 1 0 1
O(fragment=1)	-1.90937854	1.06610853	-0.05598828
H(fragment=1)	-2.23867796	0.17728177	0.09615925
H(fragment=1)	-0.94953286	1.05932020	-0.04017097
N(fragment=2)	-2.02042680	-1.64298230	0.34416157
H(fragment=2)	-2.28323594	-1.98171649	-0.55927105
H(fragment=2)	-2.67940800	-1.96304329	1.02482651
H(fragment=2)	-1.10944265	-1.98408759	0.57601295

Both of SAPT0 and sSAPT0 results can be found in output. The SAPT paper (JCP 140, 094106 (2014)) recommends three levels of theory to be used: (1) bronze: sSAPT0/jun-cc-pVDZ; (2) silver: SAPT2+/aug-cc-pVDZ; (3) gold: SAPT2+(3)δMP2/aug-cc-pVTZ. So far the open shell calculations have not been implemented for silver and gold level in PSI4 (it is OK for sSAPT0). Currently only the bronze level is supported in utility frag_guess_wfn. {sapt} or {sapt,bronze} in the Title Card line will be recognized as the bronze level. You are always recommended to use jun-cc-pVDZ unless there is some element which is out of the range of jun-cc-pVDZ.

5.3.5 Tricks for accelerations in frag_guess_wfn

For some simple fragments (water molecules, organic ligands, etc), if you are sure that they have closed-shell wave function, you can append a character 'r' after the Cartesian coordinates of any atom of the fragment. Again, taking the $[Ti(H_2O)_6]^{3+}$ cation as the example,

```
%chk=Ti.chk
%mem=32GB
%nprocshared=16
#p UPBE1PBE/gen guess(fragment=2)

{gks}

3 2 3 2 0 1
```

Ti(fragment=1)	0.00000000	0.00000000	0.00000000
H(fragment=2)	0.00000000	2.68428109	-0.78751709 r
H(fragment=2)	2.68128062	0.78751744	0.00000000

… (not shown)

Note that there must be a blank between the z-component coordinate and the character 'r'. This will force the use of R(O)HF calculation for this fragment, thus saving computation time. After R(O)HF computation of this fragment is done, beta MOs will be directly copied from alpha MOs, so the net result is that it appears to perform a UHF computation, but with only R(O)HF cost.

If the 'r' is not written, UHF and wave function stability test will be performed for this fragment. In this special case we know that UHF is unnecessary for this fragment.

This trick accelerates computations only when UHF/UDFT is required for the supermolecule so that we can force some fragment(s) to use closed-shell methods. If restricted HF or DFT methods are required for the supermolecule, this trick will not reduce computational time.

5.4 Manually making consistent active space

The active space automatically determined by AutoMR of MOKIT may not be the same size for cases involving chemical reactions, different local minima, or different spin states of the same geometry. In such cases, if we want to obtain an accurate relative energy, we should pay attention to the following two points:

(1) Different species are required to have the same size of active space.
(2) Active orbitals of different species are supposed to contain the same main components.

Point (2) is merely recommended by the author jxzou, not a rule which you must obey. Taking the singlet-triplet gap of a molecule as an example: if MOKIT automatically determined/recommends the active space of singlet to be CAS(2,2), but triplet to be CAS(4,4), then it would be unfair to directly compare the energies of CASSCF(2,2) *v.s*. CASSCF(4,4). Of course,

it would be unfair to simply compare NEVPT2 energies (based on corresponding CASSCF), either. In such cases, the author recommends two approaches:

(i) If we find two orbitals in CAS(4,4) of triplet to be not active, i.e. occupation numbers close to 2 and 0, respectively, we can read CAS(4,4) orbitals, move these two orbitals out of the active space, and perform a CAS(2,2) computation. Then active space size of two spin states are the same.
(ii) If there are similar active orbitals but they are too active to be moved out of the active space, or if we cannot find any similar active orbital between two active space. We should consider to enlarge the size of CAS(2,2) for singlet. The extension target can be CAS(4,4), or can be even larger, depending on the similar extent of two active space.

Obviously the approaches above correspond to intersection set and union set in mathematics. Currently these two approaches have not been implemented in MOKIT, since it requires a lot of techniques to make them become automatic and black-box. Below I will provide two examples to illustrate how to manually make two active space consistent. For comparison of active space from more than two species, the spirit is identical.

5.4.1 The Diels-Alder reaction of 1,3-butadiene and ethane

To be added.

5.4.2 Singlet-Triplet gap of an iron-containing molecule

To be added.

Appendix

A1 Frequently Asked Questions (FAQ)

Q1: Find errors like 'xxx: command not found'. What is the solution?

A1: This is simply because you haven't installed the corresponding software, or you didn't write environment variables correctly. Four examples are shown below:

Example 1: error 'ifort: command not found' means there is no ifort compiler on your computer, see 2.2.1 Prerequisite for details.
Example 2: error '-bash: f2py: command not found' means there is no f2py on your computer, see Section 2.2.1 Prerequisite for details.
Example 3: assuming you've compiled MOKIT successfully, but got the '-bash: automr: command not found' error. Then you should check the MOKIT paths in your ~/.bashrc. See Section 2.2.3 Environment variables for details.

Example 4: errors like '/usr/bin/ld: cannot find -lmkl_rt' or 'ld: cannot find -lmkl_intel_lp64' have two possible reasons: (1) there is no MKL installed on your computer; (2) there is MKL on your node but you did not correctly (or you forgot to) write environment variables of MKL.

Q2: Is there any Windows/Mac OS pre-compiled or pre-built version of MOKIT?

A2: The author offers more than 20 utilities of Windows* OS pre-built executables, which are released as a .zip file. See instructions for download and setting environment variables in Section 2.1.

The AutoMR module is not included in pre-compiled executables, since multi-reference calculations are usually performed on high-performance computers/platforms, which usually contain Unix/Linux systems. However, if you really need all modules or utilities under Windows, you can compile the source code on your laptop/computer/node by yourself. The Mac OS pre-compiled executables will never be considered by the author.

Q3: Why the utilities dat2fch, py2fch, mkl2fch, orb2fch xml2fch, bdf2fch and bdf2mkl cannot generate a .fch file from scratch, but require the user to provide one?

A3: Strictly speaking, to correctly transfer MOs between different programs requires 'int=nobasistransform nosymm' (and also possibly '5D 7F' or '6D 10F') keywords in Gaussian, and equivalent keywords in other quantum chemistry programs. This rule also applies to other programs/software which claim they can transfer MOs or support file transformation.

However, the utility/subroutine (which transfers MOs) cannot detect whether the users have truly specified 'int=nobasistransform nosymm' (and also possibly '5D 7F' or '6D 10F') keywords, or equivalent keywords in other programs. Then transferring MOs in such case is very dangerous, i.e. correctness cannot be assured. For example, using the built-in basis sets in other programs instead of basis sets generated by MOKIT is dangerous, since the built-in basis sets in other programs are generally not exactly identical to those generated by MOKIT.

Therefore, the author jxzou has to require the users to provide a .fch file, to remind the users that proper keywords must be used in Gaussian. And the most recommended approach is to use utilities in MOKIT (e.g. fch2inp, bas_fch2py, etc) to generate input files of other quantum chemistry programs. The users use these generated input files to perform their desired computations. After jobs finished, use dat2fch and py2fch to transfer MOs back to the .fch file. Such an approach is already applied in AutoMR.

In the future version of MOKIT, the utility xml2fch will not require the user to provide a .fch file.

Q4: Can MOKIT support more types of files? Transferring MOs between other programs like QChem, NWChem is possible?

A4: The author jxzou wishes to make the MOKIT recognize all kinds of MO files of quantum chemistry programs, but he is not likely to be familiar with all programs. Therefore, if you are very familiar with any program (other than supported ones in MOKIT), and you happens to need a transferring of MOs, please contact jxzou and tell him the information in the MO file of that program. With the help from experienced users, the development will be much easier and quicker.

Q5: Why there is a keyword error in OpenMolcas output when using DKH2 Hamiltonian? Two possible errors are given: (1) ERROR: RELATIVISTIC is not a keyword!, Error in keyword. (2) ERROR: R02O is not a keyword!, Error in keyword.

A5: This is due to a recent update of OpenMolcas. If version <=18.09, the keyword for DKH2 is simply R02O; but for version >=20.10, this keyword become RELAtivistic = R02O. For version >18.09 and <20.10, the author jxzou has no manual and thus it is not tested (but you can test if you like).

 If you encounter any of the two errors, you have two choices: (1) update your OpenMolcas to a newer version, e.g. >=20.10; (2) modify the source code of MOKIT and re-compile it. If you choose (2), you will need to modify the words 'RELAtivistic = R02O' to 'R02O' in file src/automr.f90, and run 'make automr' in Shell to re-compile the program automr.

Q6: How does AutoMR read the executable paths of Gaussian, OpenMolcas, PySCF, ORCA, and GAMESS?

A6: For Gaussian, the paths of executable files are read from the environment variables $GAUSS_EXEDIR. For PySCF and OpenMolcas, the 'python' and 'pymolcas' executable files are used directly, assuming the user had installed the corresponding programs correctly. For ORCA and GAMESS, the user must define the $ORCA and $GMS environment variables in his/her ~/.bashrc file, such that the automr program can find corresponding paths.

Q7: What are the possible reasons and solutions of the following errors
(1) '*** ERROR **** DIMENSIONS EXCEEDED *****'**
(2) 'PAIR=   xx MAX=   12'
(3) 'DDI Error: Could not initialize xx shared memory segments.'
(4) 'DDI was compiled to support 32 shared memory segments.'
(5) 'The GAMESS executable gamess.01.x or else the DDIKICK executable ddikick.x could not be found in directory…'

in GAMESS .gms file (where xx is an integer >12)?

A7: Please read Section 4.4.10 carefully.

Q8: Errors like 'semget errno=ENOSPC -- check system limit for sysv semaphores' found in the .gms file. Why? How to solve the problem?

A8: Please search the error on this page [FAQ of GAMESS](#).

Q9: I found the error 'DDI Process 0: trapped a floating point error (SIGFPE).' How to solve it?

A9: See this page https://github.com/gms-bbg/gamess-issues/issues/40.

Q10: I found the following error "AttributeError: 'CASSCF' object has no attribute 'mo_occ'" in PySCF output (e.g. .out file). Why? How to solve the problem?

A10: This is because you were using PySCF version < 1.7.4, which has no mo_occ attribute in CASSCF object. Please update to PySCF >=1.7.4.

Q11: I found the following error "FileNotFoundError: [Errno 2] No such file or directory: '/path/to/Block/block.spin_adapted'" in PySCF output (e.g. .out file). Why? How to solve the problem?

A11: This is because you forgot to modify pyscf/dmrgscf/settings.py. You should copy file settings.py.example and rename it as settings.py. Then set the correct path (within the file) for the DMRG solver.

Q12: I found the following error 'HDF5-DIAG: Error detected in HDF5 (1.12.0) thread 0:' '#000: H5D.c line 435 in H5Dget_type(): invalid dataset identifier' 'major: Invalid arguments to routine' 'minor: Inappropriate type' occurs many times in OpenMolcas output, Why? How to solve the problem?

A12: This does not affect the computations. It is just a tiny bug of old versions of OpenMolcas if the QCMaquis support is compiled but not used. You can find this problem [here](#). It is recommended to update your OpenMolcas to the latest version.

Q13: Is 'Total SCF Density' in *_NO.fch file correct? Can it be used to perform

wave function analysis?

A13: Yes. The CASCI/CASSCF total densities are correct in generate *_NO.fch files. Besides, the GVB density is in *_s.fch file, and the UHF density are both in *_uhf.fch and *_uno.fch files. Other types of *.fch files (*_asrot.fch, etc) do not have well-defined density. Density and natural orbitals are not calculated for post-CASSCF methods (NEVPT2, CASPT2, MRCISD, etc).

Q14: I found the following error
'sh: 1: Syntax error: Bad fd number'
on the terminal/screen. Why? How to solve the problem?

A14: This often occurs on Ubuntu* OS, where the default sh is /bin/dash, not /usr/bin/bash. Please read Section 2.2.4 for details and examples.

Q15: I found the following warnings on terminal/screen, or output of AutoMR
'OMP: Warning #181: OMP_STACKSIZE: ignored because KMP_STACKSIZE has been defined'
Why? How to make it disappear?

A15: This is simply because both OMP_STACKSIZE and KMP_STACKSIZE environment variables are set. You can comment/delete one of them. Usually you can find these two environment variables in your ~/.bashrc file, or in file bdf-pkg/sbin/run.sh (if you use the BDF program).

Q16: Is it possible that AutoMR takes more (computational) time than that by manually doing multi-reference computations (by chemical intuition, manual inspection, etc) ?

A16: Yes, possible. Note that for some small and well-studied molecules, manually doing multi-reference computations (by chemical intuition, manual inspection, etc) may take less (computational) time than AutoMR. But for general molecules, especially with dozens/hundreds of possible active orbitals, manual inspection is tedious and sometimes chemical intuition is unreliable. AutoMR aims at tackling general and complicated cases.

Besides, to better compare the cost (of time) between the human and automatic approach, the cost of time of human operations must be taken into consideration, since the time of manual inspection and permuting orbitals (when >10 orbitals) is not negligible.

Q17: Why does my computation proceed slowly? It took a long time in HF and GVB calculations. Is there any acceleration techniques/tricks?

A17: Firstly, please read Section 3.1 and 3.2 for suggestions on choosing appropriate methods and basis sets. Secondly, assuming your method and basis set is already reasonable, there is no trick to

be used currently. The HF step is performed by Gaussian and it does not support RI or density fitting techniques. In the future version of MOKIT, users can specify PSI4 as the HF_prog, where RI is supported. The author jxzou will write a RI-GVB program in a year, then the GVB computation can be greatly accelerated.

A2 Limitations and Suggestions

A2.1 Basic knowledge of multi-reference methods

Although MOKIT is a useful tool for black-box multi-reference computations, the users are assumed to know basic knowledge of multi-reference methods. If he/she does not even know the meaning of CAS(m,n), then the results of MOKIT are meaningless to he/she, and even wrong explanations may be made from the results. Therefore, if you know little about the multi-reference methods, I recommend you the following materials:
(1) the ORCA CASSCF-tutorial (https://orcaforum.kofo.mpg.de/app.php/dlext/?cat=4), which is a quickstart guide (but also detailed) for CASSCF computations. To download this .pdf file, you may need to (register and) login to the forum.
(2) to be added.

If, unfortunately, you are forced by your supervisor/advisor to learn how to perform multireference calculations, but meanwhile you are a totally newbie and you don't even know how to perform routine DFT calculations, it is recommended that you change a task/project as soon as possible (ASAP). Or more radically, you are encouraged to switch to a new supervisor ASAP, since he/she does not know how to teach/train a student properly. This is not your fault, but your supervisor's.

A2.2 Symmetry

Unfortunately, molecular point group symmetry cannot be taken into consideration in any module of MOKIT. This is due to: (1) use of symmetry may change the orientation of the target molecule, and the MO coefficients will be changed accordingly; (2) localized orbitals are used in almost all modules of AutoMR, this usually contradicts with symmetry.

A2.3 Validity of MOs obtained by AutoMR for excited state calculations

When you use AutoMR to perform a ground state CASSCF calculation, the obtained CASSCF MOs (whether pseudo-canonical MOs or NOs) is supposed to be excellent for the ground state electronic structure of the target molecule. And you can use this set of MOs (held in a .fch file) to further conduct excited state calculations like state-averaged CASSCF (SA-CASSCF). And moreover, NEVPT2, CASPT2 or MRCISD, if you wish.

However, the resulting excitation energies and excited state MOs (e.g. state-averaged NOs) are not necessarily excellent. Here 'not necessarily excellent' means for some molecules you may get good results while may be unsatisfactory for some other molecules. The reason is simple: the current algorithms in AutoMR focus on the multi-reference characters in the ground state of the molecule. Thus AutoMR 'finds' excellent active orbitals of the ground state. But the active orbitals of excited states are not necessarily the same as those of the ground state. And the orbital optimization in SA-CASSCF does not guarantee leading to good MOs for excited states. This problem actually exists in almost all methods/programs which feature block-box or automatic multi-reference calculations.

There is a solution (although not perfect or elegant) to this problem: (visually) inspect the doubly occupied orbitals, pick up important orbitals (usually the lone-pair orbitals) and add them into the active space. For example, if you obtain a CAS(6e,6o) active space from MOKIT, and assuming you find 4 lone pair orbitals among doubly occupied orbitals, then you can combine them (by interchanging or permuting orbitals) to be a CAS(14e,10o) active space. Because 6+2*4=14 active electrons, and 6+4=10 active orbitals. You can do a SA-CASSCF(14e,10o) computation next. This usually converges in several cycles. Finally you can perform a NEVPT2/CASPT2/MRCISD computation based on SA-CASSCF(14e,10o) orbitals to get more accurate excitation energies.

A2.4 Possible multiple solutions of UHF

There may exist multiple UHF solutions when a covalent bond cleavages homolytically, or in a transition-metal-containing molecule. In these special cases, if you use ist=0, the UHF calculated by AutoMR may be not the lowest UHF solution (but it is stable). You may need to perform several UHF computations (by yourself) using various initial guesses. After you identify the lowest UHF solution, you can use keywords ist=1 and 'readuhf' to read in the desired UHF .fch file.

Alternatively, you can simply write the fragment guess information into the .gjf file, exactly as the syntax of Gaussian. See an example in file examples/automr/05-N2_cc-pVTZ_4.0.gjf.

Peter Pulay *et al* suggests that for cases involving multiple UHF solutions, one should use the averaged density (of all possible solutions) to generate UNOs. This approach is not supported in MOKIT currently.

A2.5 Implicit Solvent Model

Currently implicit solvent effect cannot be taken into consideration. But you can use the converged CASSCF wave function *_NO.fch file as an initial guess to your further calculations which takes implicit solvent effect into consideration.

A3 Bug Report

If you find any bug frequently occurs, please go to GitLab (https://gitlab.com/jxzou/mokit) to download the latest version of MOKIT and check whether the bug still exists. If it still exists, you can
(1) contact the author jxzou via E-mail njumath[at]sina.cn, with your input file (.gjf, .fch) and output

files attached;

(2) open an issue on GitLab page of MOKIT (https://gitlab.com/jxzou/mokit/-/issues);

(3) (if you can communicate in Chinese) you can join the Tencent QQ group (group number 470745084).

The author may not respond or update code frequently since he is being postponed due to his PhD program.

A4 Acknowledgement