

# MOKIT Manual

version 1.2.2 Feb 12, 2021

jxzou

<b>1 Introduction.....</b>	<b>4</b>
1.1 Introduction to MOKIT.....	4
1.2 Citation.....	4
<b>2 Installation .....</b>	<b>5</b>
2.1 Pre-built Windows executables .....	5
2.2 Installation under Linux .....	6
2.2.1 Prerequisite.....	6
2.2.2 How to compile .....	8
2.2.3 Environment variables .....	8
2.2.4 Test .....	9
2.3 Installation on Cluster .....	9
<b>3 Quickstart for MOKIT.....</b>	<b>10</b>
3.1 Which method should I use? .....	11
3.2 Which basis set should I use?.....	11
<b>4 User's Guide .....</b>	<b>12</b>
4.1 Syntax Rules of AutoMR .....	12
4.2 Memory and Parallel Settings .....	13
4.3 Keywords of supported methods in AutoMR.....	13
GVB .....	14
CASSCF .....	14
CASCI.....	14
DMRGSCF.....	14
DMRGCI.....	14
NEVPT2.....	15
CASPT2 .....	15
CASPT3 .....	15
SDSPT2.....	15
MRMP2.....	15
MRCISD .....	16
MCPDFT.....	16
4.4 List of AutoMR Keywords.....	16
4.4.1 readrhf .....	16
4.4.2 readuhf .....	17
4.4.3 readno.....	17

4.4.4 ist.....	17
4.4.5 LocalM.....	18
4.4.6 COnly.....	18
4.4.7 Force.....	19
4.4.8 Cart.....	19
4.4.9 GVB_prog.....	19
4.4.10 CASCI_prog.....	20
4.4.11 CASSCF_prog.....	20
4.4.12 DMRGCI_prog.....	20
4.4.13 DMRGSCF_prog.....	20
4.4.14 CASPT2_prog.....	21
4.4.15 NEVPT2_prog.....	21
4.4.16 MRCISD_prog.....	21
4.4.17 MRMP2_prog.....	22
4.4.18 MCPDFT_prog.....	22
4.4.19 CtrType.....	22
4.4.20 MaxM.....	23
4.4.21 hardwfn.....	23
4.4.22 crazywfn.....	23
4.4.23 noI0cycle.....	23
4.4.24 charge.....	24
4.4.25 OtPDF.....	24
4.4.26 DKH2.....	24
4.4.27 X2C.....	24
4.4.28 RI.....	25
4.4.29 RIJK_bas.....	25
4.4.30 F12.....	25
4.4.31 F12_cabs.....	25
4.4.32 DLPNO.....	26
4.4.33 FIC.....	26
4.5 List of Utilities in MOKIT.....	26
4.5.1 add_bgcharge_to_inp.....	27
4.5.2 bas_fch2py.....	27
4.5.3 bas_gau2molcas.....	27
4.5.4 bas_gms2bdf.....	28
4.5.5 bas_gms2molcas.....	28
4.5.6 bas_gms2molpro.....	28
4.5.7 bas_gms2psi.....	29
4.5.8 bas_gms2py.....	29
4.5.9 bdf2fch.....	29
4.5.10 bdf2mkl.....	30
4.5.11 dat2fch.....	30
4.5.12 extract_noon2fch.....	31
4.5.13 fch2bdf.....	31

4.5.14 fch2com.....	32
4.5.15 fch2inp .....	33
4.5.16 fch2inporb .....	33
4.5.17 fch2psi .....	34
4.5.18 fch2mkl .....	35
4.5.19 fch_mo_copy.....	35
4.5.20 fch_u2r .....	35
4.5.21 frag_guess_wfn .....	35
4.5.22 gvb_sort_pairs .....	36
4.5.23 mkl2fch .....	36
4.5.24 mkl2gjf.....	37
4.5.25 orb2fch .....	37
4.5.26 mo_svd .....	38
4.5.27 solve_ON_matrix .....	38
4.5.28 fch2py.....	39
4.5.29 py2fch.....	39
4.5.30 xml2fch .....	39
4.6 APIs in MOKIT.....	40
4.6.1 load_mol_from_fch.....	40
4.6.2 read_intle_from_gau_log .....	40
4.6.3 read_mo_from_fch.....	41
4.6.4 read_density_from_gau_log.....	41
4.6.5 read_density_from_fch.....	41
4.6.6 gen_ao_tdm.....	42
4.6.7 export_mat_into_txt .....	42
<b>5 Examples .....</b>	<b>43</b>
<b>Appendix.....</b>	<b>43</b>
A1 Frequently Asked Questions (FAQ) .....	43
A2 Limitations and Suggestions .....	47
A2.1 Basic knowledge of multi-reference methods .....	47
A2.2 Symmetry .....	47
A2.3 Validity of MOs obtained by AutoMR for excited state calculations .....	47
A2.4 Possible multiple solutions of UHF .....	48
A2.5 Implicit Solvent Model.....	48
A3 Bug Report .....	48
A4 Acknowledgement.....	48

# 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  $< 10^{-6}$  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 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.

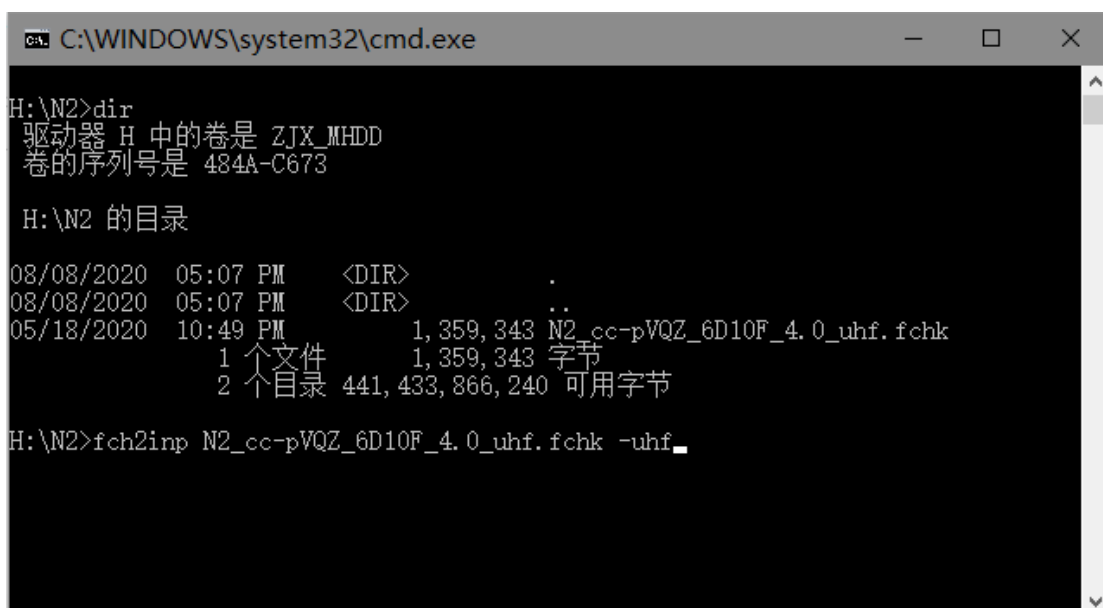
### 2.1 Pre-built Windows executables

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 around 20 out of all utilities in MOKIT are provided. These executables are compressed into a .zip file and can be downloaded at 'Project overview' -> 'Releases' on <https://gitlab.com/jxzou/mokit/>. Download, uncompress it, and set the environment variables, then these utilities can be 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



```
C:\WINDOWS\system32\cmd.exe

H:\N2>dir
驱动器 H 中的卷是 ZJX_MHDD
卷的序列号是 484A-C673

H:\N2 的目录

08/08/2020  05:07 PM    <DIR>          .
08/08/2020  05:07 PM    <DIR>          ..
05/18/2020  10:49 PM             1,359,343  N2_cc-pVQZ_6D10F_4.0_uhf.fchk
               1 个文件             1,359,343  字节
               2 个目录  441,433,866,240  可用字节

H:\N2>fch2inp N2_cc-pVQZ_6D10F_4.0_uhf.fchk -uhf
```

## 2.2 Installation under Linux

### 2.2.1 Prerequisite

1. Intel compiler (for ifort and MKL library)
2. [Anaconda Python3](#) (for python, f2py, numpy, 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 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 mokit/src/Makefile for gfortran settings. And of course, use ‘#’ to comment lines of ifort in that Makefile. The recommended version of gfortran is  $\geq 4.8.5$  but  $< 8.0$ . 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** wrapper is valid. For example, run ‘which f2py’ in Shell to see whether f2py exists. The filenames of \*.so files compiled by f2py from Python3 are shorter than those from Anaconda Python3, but they are OK to be used.

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](#), and [PSI4](#). Some recommended versions are shown below

Gaussian: G09 or G16

PySCF >=1.7.4

GAMESS >=2017

ORCA >= 4.2

OpenMolcas >= v20.10

BDF >= 0.9.6

PSI4 >= 1.3.2

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 ~ 4.4.18 for details). Usually three software packages (Gaussian, PySCF and GAMESS) are used extensively 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>

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 Gaussian 16 安装教程》 <https://mp.weixin.qq.com/s/ffGo6eOEacfggg3sYbrLJA>
2. 《ORCA 软件安装教程》 <https://mp.weixin.qq.com/s/9j3U5v0wBaAabRk-uo4EhQ>
3. 《GAMESS 编译教程》 <https://mp.weixin.qq.com/s/w689PJc4c2H8sWj03zsUBA>
4. 《离线安装 PySCF 程序（1.5 及更高版本）》  
<https://mp.weixin.qq.com/s/QZPG5jRFZ5s1it2OaBGJqw>
5. 《OpenMolcas 与 QCMAquis 的安装》 [https://mp.weixin.qq.com/s/ldhxz3I3txmlDxjk\\_3a5bw](https://mp.weixin.qq.com/s/ldhxz3I3txmlDxjk_3a5bw)
6. 《Block-1.5 的编译和安装》 <https://mp.weixin.qq.com/s/EUZKLYSqbuIUL9-zlySfbQ>
7. 《Boost.MPI 的编译》 <https://mp.weixin.qq.com/s/AMYUTB5pTNLFZ8NEtFIG-Q>
8. 《安装基于 openmpi 的 mpi4py》 <https://mp.weixin.qq.com/s/f5bqgJYG5uAK1Zubngg65g>

## 9. 《自动做多参考态计算的程序 MOKIT》

<https://mp.weixin.qq.com/s/bM244EiyhsYKwW5i8wq0TQ>

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 4.4.9 carefully. After re-compiling GAMESS (followed by instructions in Section 4.4.9), 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`, and Molpro is run by `molpro`, there is no extra variable to be exported here. The environment variables of ORCA and BDF are optional, do not write them if you do not want to use ORCA or BDF. 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 encounter any errors in output, please search your problem in Section A1 FAQ of Appendix firstly. Some common questions have been 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 avail          # 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 2<sup>nd</sup> and 3<sup>rd</sup> lines load the corresponding software. And the 4<sup>th</sup> 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.gif &
```

However, usually you are not allowed to submit any job in the main node or login node. And there exists a queue system or 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
```

Then you can run

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

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 or FIC-MRCISD. 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, def2-TZVP, def2-TZVPP or def-TZVP. 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 methyl by a hydrogen atom, 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 4.4.28) in CASSCF and CASSCF-NEVPT2 computations.

- (2) Pople-type basis set like 6-311G(d,p) is less recommended.

- (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  $\geq$  the 4<sup>th</sup> 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 the usage of 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.  $\text{MnO}_4^-$ , it is strongly recommended to use basis set with diffuse functions, such as aug-cc-pVTZ, def2-TZVPD. 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.

- (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.

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.

## 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**. 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 4.4.28) approximation is turned on, the auxiliary basis set will be automatically determined (by AutoMR) according to the basis set. And if F12 (see Section 4.4.30) is turned on, the F12-CABS will be

automatically determined (by AutoMR) according to the basis set, too.

For the methods listed below, 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](https://doi.org/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.**

## 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 4.4.9 carefully.

## 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 4.4.11.

## 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, except 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 4.4.10.

## DMRGSCF

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

## DMRGCI

Here the term DMRGCI actually means the DMRG-CASCI method. Please also read Section 4.4.12 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.

## NEVPT2

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

Please read related keyword NEVPT2\_prog in Section 4.4.15.

## CASPT2

Second order Perturbation Theory based on CASSCF reference.

Please read related keyword CASPT2\_prog in Section 4.4.14.

## CASPT3

Third order Perturbation Theory based on CASSCF reference.

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

## 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.

## 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.

## MRCISD

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

Please read related keyword MRCISD\_prog in Section 4.4.16.

## 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 4.4.18. Also note that in GAMESS, the MC-PDFT is only supported for version  $\geq 2019(R2)$ .

\*\*\*\*\*YOU MUST ALSO READ\*\*\*\*\*

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. 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  $d(N-N) = 1.15 \text{ \AA}$ , the CAS(4,4) may be automatically determined by AutoMR, but for  $d(N-N) = 4.0 \text{ \AA}$ , 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.

## 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 4.4.23 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 strategy. 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  $E_{\text{UHF}} = E_{\text{RHF}}$ , `ist` will be changed to 3. If  $E_{\text{UHF}} < E_{\text{RHF}}$ , `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  $\sigma$  and  $\pi$  orbitals, while the PM method tends to keep them separated. These two methods make no difference when the target molecule contains only  $\sigma$  bonds (and possibly a few isolated  $\pi$  bonds). But if you are dealing with multiple  $\pi$  bonds or conjugated  $\pi$  systems like oligoacene(benzene, naphthalene, etc), or if you want the active space to contain only  $\pi$  orbitals, please 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' should also be taken into considerations, 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 GVB\_prog

Specify the GVB program. Default is GAMESS, and this is the only program supported currently. The second-order SCF (SOSCF) algorithm in GAMESS is used to converge the GVB wave function. The GAMESS version  $\geq 2017$  is strongly recommended. Older versions of GAMESS may work or may not, since they are not tested by the author jxzou.

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, the original GAMESS only supports 32 CPU cores. If your machine has more cores (and if you want to use  $>32$  cores), you need to modify the source code. See a simple guide in file

src/modify\_GMS\_beyond32CPU.txt.

#### 4.4.10 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 and BDF.

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.

#### 4.4.11 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, and BDF. 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.

#### 4.4.12 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.

#### 4.4.13 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.

#### 4.4.14 CASPT2\_prog

Specify the program for performing CASPT2 calculation, e.g. CASPT2\_prog=OpenMolcas. Currently only OpenMolcas(default) and Molpro 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.

#### 4.4.15 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 4.4.33. 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.

#### 4.4.16 MRCISD\_prog

Specify the program for performing MRCISD calculation. By default, MRCISD\_prog=OpenMolcas. Currently, AutoMR supports the interfaces of 3 variants of the MRCISD method: (1) uncontracted MRCISD; (2) internally contracted MRCISD (ic-MRCISD); (3) fully internally contracted MRCISD (FIC-MRCISD). And AutoMR is able to call OpenMolcas/Molpro/ORCA/Gaussian programs to perform MRCISD. 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.

If MRCISD\_prog=Gaussian, only (1) is supported and no Davidson correction is given.

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, please choose the same type of ic-MRCISD, i.e. both using Molpro or both using OpenMolcas.

The ic- and FIC-MRCISD are both approximations of uncontracted MRCISD. If the Davidson correction energy is added, then it should be denoted as MRCISD+Q. It is recommended to use ic- or FIC-MRCISD+Q in practical calculations since the uncontracted MRCISD is very expensive.

You must also specify a contraction type, please read Section 4.4.19 carefully.

#### 4.4.17 MRMP2\_prog

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

#### 4.4.18 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  $\geq 2019(R2)$ , and currently it can only be run in serial.

#### 4.4.19 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 functions, then you cannot use FIC-MRCISD in ORCA. In such case, you should choose ic-MRCISD in OpenMolcas instead.

#### 4.4.20 MaxM

Specify the bond dimension MaxM in DMRG-related calculations. The default value 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.

#### 4.4.21 hardwfn

This option can only be applied to CASCI/CASSCF calculations using PySCF or GAMESS. 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.

#### 4.4.22 crazywfn

This option can only be applied to CASCI/CASSCF calculations using PySCF or GAMESS. 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<sub>2</sub> molecule is stretched to  $d(\text{N-N}) = 4.0 \text{ \AA}$ , 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.

#### 4.4.23 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.

#### 4.4.24 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).

#### 4.4.25 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  $\geq 2019(R2)$ .

#### 4.4.26 DKH2

Request the 2<sup>nd</sup> 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.

#### 4.4.27 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.

#### 4.4.28 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 something like 'mokit{RI=True}', '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/ORCA programs. To learn more about the auxiliary basis set used in RI-JK approximation, see the following Section 4.2.29.

#### 4.4.29 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/.

#### 4.4.30 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 4.2.28) 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

CASSCF\_prog=ORCA,NEVPT2\_prog=ORCA,F12

in mokit{ }. To learn more about the near-complete auxiliary basis set used in F12 technique, see the following Section 4.2.31.

#### 4.4.31 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.

### 4.4.32 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 4.2.28) and FIC (see 4.2.33) 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

CASSCF\_prog=ORCA,NEVPT2\_prog=ORCA,DLPNO

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

CASSCF\_prog=ORCA,NEVPT2\_prog=ORCA,DLPNO,F12

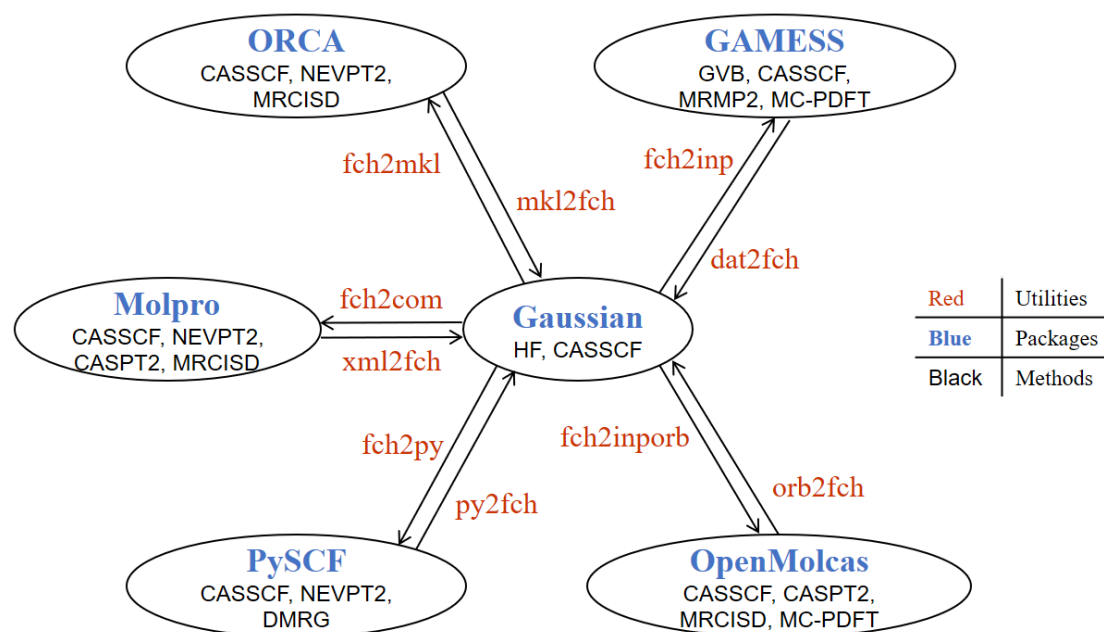
in mokit{}.

### 4.4.33 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.

## 4.5 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`, 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`, 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 4.2.28) 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.

Two examples are shown and explained below

(1) `bas_gms2bdf a.inp`

Generate a \_bdf.inp and A.BAS files, for RHF or ROHF type wave function.

(2) `bas_gms2bdf a.inp -uhf`

Generate a \_bdf.inp and A.BAS files, for UHF type wave function.

#### 4.5.5 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).

#### 4.5.6 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).

### 4.5.7 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).

### 4.5.8 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).

### 4.5.9 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. Three examples are shown and explained below

(1) `bdf2fch a.scforb a.fch`

This is used for transferring RHF, ROHF or CASSCF orbitals.

(2) `bdf2fch a.scforb a.fch -uhf`

This is used for transferring UHF orbitals.

(3) `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 4.5.13.

### 4.5.10 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. Three examples are shown and explained below

(1) `bdf2mkl a.scforb a.fch`

This is used for transferring RHF, ROHF or CASSCF orbitals.

(2) `bdf2mkl a.scforb a.fch -uhf`

This is used for transferring UHF orbitals.

(3) `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.

### 4.5.11 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. Five examples are shown and explained below

(1) `dat2fch a.dat a.fch`

This is used for transferring RHF, ROHF or CASSCF orbitals.

(2) `dat2fch a.dat a.fch -uhf`

This is used for transferring UHF orbitals.

(3) `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.

(4) `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.

(5) `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 4.5.15.

#### 4.5.12 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

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.

#### 4.5.13 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. Three examples are shown and explained below

(1) `fch2bdf a.fch`

This is used for transferring RHF/ROHF orbitals. Three files will be generated: a\_bdf.inp, A.BAS and a.scforb. The data of 'Alpha Orbital Energies' section in .fch file (assumed orbital energies/levels) 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.

(3) `fch2bdf a.fch -uhf`

This is used for transferring UHF orbitals. Three files will be generated: a\_bdf.inp, A.BAS and a.scforb. Note the data of 'Alpha Orbital Energies' and 'Beta Orbital Energies' sections in .fch file (assumed alpha/beta orbital energies) will be read and printed into the a.scforb file.

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 4.5.9.

## 4.5.14 fch2com

Generate a Molpro .com file from a Gaussian .fch(k) file, with alpha MOs written in a .a file. Two examples are shown and explained below

(1) `fch2com a.fch`

This is used for transferring RHF, ROHF or CASSCF orbitals.

(2) `fch2com a.fch -uhf`

This is used for transferring UHF orbitals. Besides, a new file with suffix '.b' will be generated, with beta MOs written in.

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 4.5.30.

### 4.5.15 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 (RHF, UHF, GVB, and CASSCF orbitals), the fch2inp offers different options. Four examples are shown and explained below

(1) `fch2inp a.fch`

This is used for transferring RHF, ROHF or CASSCF orbitals.

(2) `fch2inp a.fch -uhf`

This is used for transferring UHF orbitals.

(3) `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.

(4) `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 4.5.11.

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).

### 4.5.16 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. Three examples are shown and explained below

(1) `fch2inporb a.fch`

This is used for transferring RHF, ROHF or CASSCF orbitals.

(2) `fch2inporb a.fch -uhf`

This is used for transferring UHF orbitals.

(3) `fch2inporb a.fch -no`

This is used for transferring NOs.

Note: the default option is to transfer RHF MOs. If you want to transfer UHF MOs, use '`fch2inporb a.fch -uhf`'. There is also another option '`-no`', which 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 4.5.25.

### 4.5.17 fch2psi

Transfer MOs from Gaussian to PSI4. One `_psi.inp` file and one .A file will be generated. If '`-uhf`' argument is specified, a .B file will be generated, too. The `_psi.inp` file of PSI4 holds the geometry, basis set data and keywords. The .A (and .B) file contains the Alpha (Beta) MOs. Similarly, you need to add an argument '`-uhf`' for transferring UHF MOs. Two examples are shown and explained below

(1) `fch2psi a.fch`

This is used for transferring RHF, ROHF or CASSCF orbitals.

(2) `fch2psi a.fch -uhf`

This is used for transferring UHF orbitals.

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

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

## 4.5.18 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. Similarly, add an argument '-uhf' for transferring UHF MOs.

To transfer MOs from ORCA back to Gaussian, see Section 4.5.23.

**Note 1:** Assuming your ORCA input file is a.inp, you need to run 'orca\_2mkl a -gbw' to generate the ORCA a.gbw file since ORCA cannot read a.mkl file directly.

**Note2:** 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).

## 4.5.19 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.

## 4.5.20 fch\_u2r

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

## 4.5.21 frag\_guess\_wfn

This is a utility designed to perform a SCF (i.e. HF or DFT) computation using initial guess constructed from molecular orbitals of fragments. The Gaussian software package will be called to perform SCF computation for each fragments (and only Gaussian is supported). The input file is exactly the Gaussian .gif file, in which the atoms, charge and spin multiplicities of each fragment are properly defined.

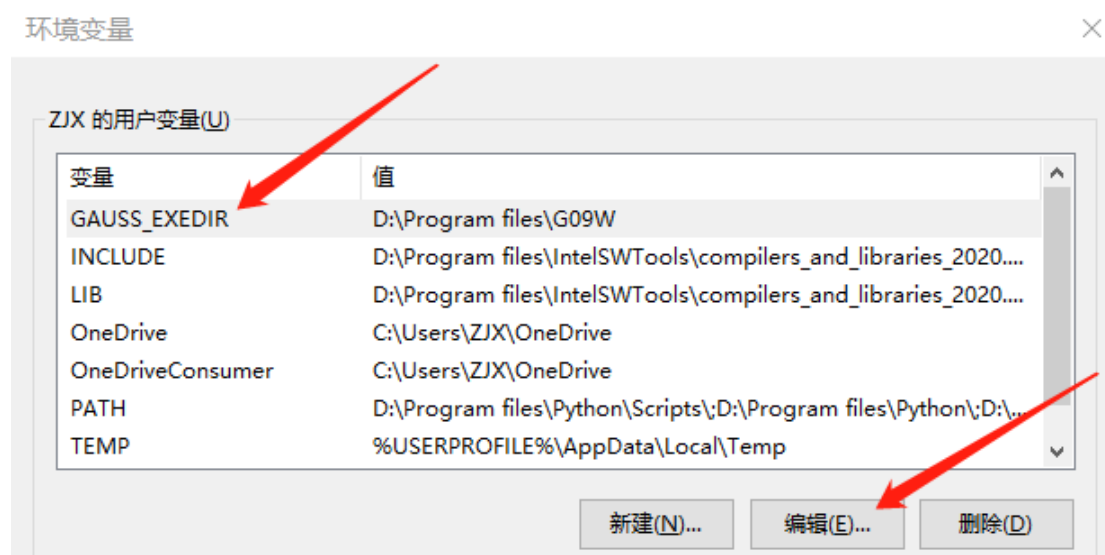
The SCF computations of radicals and transition-metal-containing molecules often suffers from multiple UHF/UKS solutions. Using a proper fragment guess (i.e., the 'guess(fragment=*N*)'

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.

The Windows pre-built executable frag\_guess\_wfn is provided at [Releases](#). 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



#### 4.5.22 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.

#### 4.5.23 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 analysis, 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.

Three examples are shown and explained below

(1) `mkl2fch a.mkl a.fch`

This is used for transferring R(O)HF orbitals.

(2) `mkl2fch a.mkl a.fch -uhf`

This is used for transferring UHF orbitals.

(3) `mkl2fch a.mkl a.fch -no`

This is used for transferring NOs.

To transfer MOs from Gaussian to ORCA, see Section 4.5.18.

### 4.5.24 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).

### 4.5.25 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 -uhf`

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 4.5.16.

### 4.5.26 `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.

### 4.5.27 `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.

The utilities below are compiled by `f2py`, which is a Fortran to Python interface generator.

These utilities are not executable files, but dynamic libraries \*.so in \$MOKIT\_ROOT/lib. They can only be called in a Python script. And this is the reason that why one of the environment variables of MOKIT is 'PYTHONPATH', not 'LD\_LIBRARY\_PATH'.

#### 4.5.28 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.

#### 4.5.29 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 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.

If you want to read MOs from a Gaussian .fch file, see Section 4.5.28.

#### 4.5.30 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 orbitals.

(2) `xml2fch a.xml a.fch -uhf`

This is used for transferring UHF orbitals.

(3) `xml2fch a.xml a.fch -no`

This is used for transferring NOs.

To transfer MOs from Gaussian to Molpro, see Section 4.5.14.

## 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)
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)
gen_ao_tdm(logname, nbf, nif, mo, istate)
export_mat_into_txt(txtname, n, mat, lower, label)
```

Meanings of frequently appeared arguments are

fchname: file name of .fch(k) file

logname: file name 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 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.

### 4.6.3 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.

### 4.6.4 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.

### 4.6.5 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)

### 4.6.6 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='transition 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.

### 4.6.7 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_intle_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)

## 5 Examples

See examples in \$MOKIT\_ROOT/examples/. More details 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. Two 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.

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

**A2:** The author offers 16 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.

#### **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 4.4.9 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.

**Q14: Is it possible that AutoMR takes more (computational) time than that by manually doing multi-reference computations (by chemical intuition, manual inspection, etc) ?**

**A14:** 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.

## 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/dlxt/?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.

### 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 in 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.

## 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) post/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

Thanks Mr. Shirong Wang for constructive suggestions and bugs report.