MOKIT Manual

version 1.2.1 2020-12-15

jxzou

1. Introduction

1.1 Introduction to MOKIT

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

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

```
  UHF(UNO) -> CASSCF -> CASPT2
  Gaussian     PySCF      OpenMolcas
or
  UHF(UNO) -> GVB     -> CASSCF -> NEVPT2
  Gaussian     GAMESS    PySCF      PySCF

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

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

Note that although MOKIT aims to make the multi-reference calculations block-box, the users are still required to have practical experiences of quantum chemistry computations (e.g. familiar with routine 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] 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

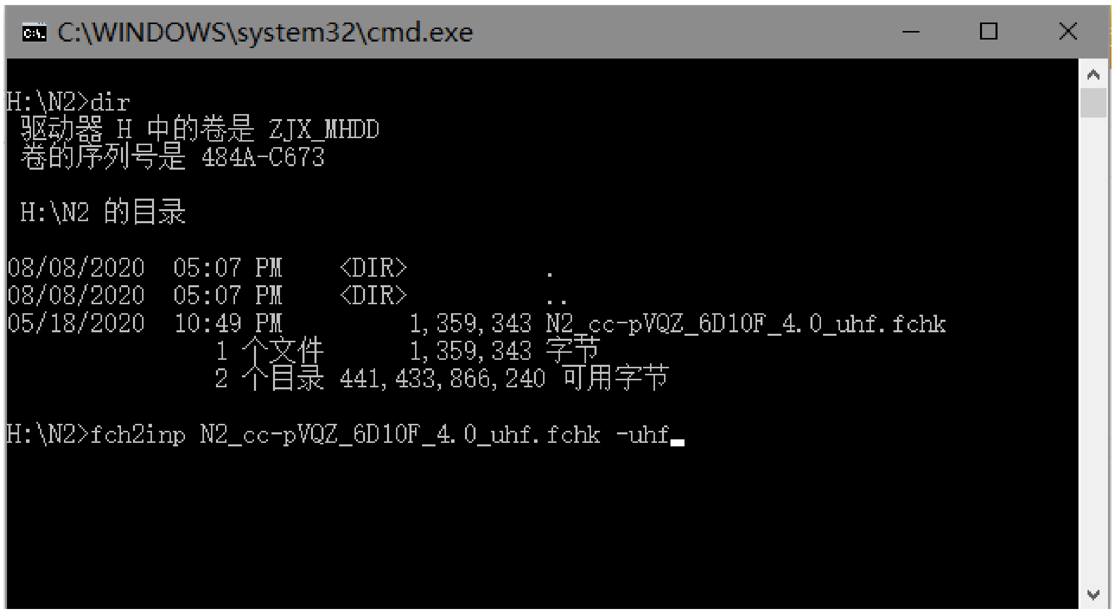
The installation of MOKIT does not require any root privilege.

2.1 Pre-compiled 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-compiled Windows executables.

Only 17 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 used in any directory. To set the environment variables, search 'environment' ("环境变量" in Chinese) on the Windows desktop, 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 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 (ifort and MKL library)
2. Anaconda Python3

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

If you want to use the Fortran compiler gfortran instead of ifort, 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. The recommended version of gfortran is >= 4.8.5 but < 8.0.

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, and ORCA.

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 ~ 4.4.17 for details). If you have any difficulty in installing software, please read their corresponding manuals carefully. Or you can find answers from their official site, forums, Github/Gitlab page, etc.

If you can read Chinese, the following installation tutorials of common software packages on the WeChat Official Accounts 'quantumchemistry' are strongly recommended:
1. 《Linux 下 Gaussian 16 安装教程》https://mp.weixin.qq.com/s/ffGo6eOEacfgqg3sYbrLJA
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
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

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.

2.2.2 How to compile

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

```
unzip mokit-master.zip
```

to uncompress it. 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 want to compile only 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:$PYTHONPATH
export ORCA=/home/$USER/software/orca-4.2.1/orca
export GMS=/home/$USER/software/games/rungms
```

Note that do not mistake 'PYTHONPATH' for 'LD_LIBRARY_PATH'. Please modify the above paths to suit your situations. Since the 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 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.

3 Quickstart for MOKIT

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

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

```
--------------------------------------------------------------------------------
%chk=4GB
%nprocshared=4
#p CASSCF/cc-pVQZ
```

```
mokit{readuhf='N2_cc-pVQZ_6D10F_4.0_uhf.fchk',ist=1,cart}
--------------------------------------------------------------------------------
```

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 (maximum) 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 the Molcas manual for details.

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, MRMP2, MRCISD**, and **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 for Gaussian). When 'readrhf', 'readuhf', or 'readno' is used in mokit{}, the basis set after '/' symbol is actually 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.

GVB

Generalized Valence Bond theory.

CASSCF

Complete Active Space Self-consistent Field.

CASCI

Complete Active Space Configuration Interaction.

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.

NEVPT2

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

CASPT2

Second order Perturbation Theory based on CASSCF reference.

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 also read Section 4.4.16 for related information.

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.

*****************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 (14,14), 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 (14,14). The only exception is that the users just want to perform a DMRG calculation with active space smaller than (14,14), then the DMRGCI or DMRGSCF must be specified.

For NEVPT2 and CASPT2 methods in the current version of AutoMR, the program will perform successively RHF/UHF (optional), GVB (optional), CASCI or CASSCF, and NEVPT2/CASPT2 computations. The CASSCF active space is automatically determined by AutoMR. Usually the 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 Å, the CAS(4,4) may be automatically determined by AutoMR, but for for *d*(N-N) = 4.0 Å, the active space turns into CAS(6,6). Thus, if you want to keep the size of active space to be CAS(6,6), you have to perform CASSCF(6,6) computations, rather than NEVPT2. Because in the CASSCF job, you can assign the number of active electrons and active orbitals (but you cannot do that in NEVPT2 job, this bug may be fixed in the next version of MOKIT). After CASSCF finished, you can use the obtained xxx_NO.fch file and the utility 'py2fch*.so' in $MOKIT_ROOT/lib/ to perform a single NEVPT2 job.

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. If you provide a UHF-type .fch file, only the Alpha MO section will be read.

4.4.2 readuhf

Read UHF orbitals from a specified .fch file. MOKIT will generate UHF natural orbitals (UNO) using the 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 may lead to improper GVB or CASSCF results.

The author 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 4.4.19 for no10cycle).

Note: 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 is 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.

4.4.4 ist

Request the use of the *i*-th *st*rategy. Default is 0. This means: (1) if the spin of the target molecule is singlet, MOKIT will call the Gaussian software to perform RHF and UHF computations, then determine whether to change 'ist' to 1 or 3. If the $E_{\mathrm{UHF}} = E_{\mathrm{RHF}}$, ist will be changed to 3. If $E_{\mathrm{UHF}} < E_{\mathrm{RHF}}$, 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 should do the UHF computation by yourself and use ist=1 later. 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: if RHF wfn 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 -> ...

4.4.5 LocalM

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

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

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 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 >=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), GAMESS, OpenMolcas, Gaussian, ORCA and Molpro.

4.4.11 CASSCF_prog

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

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. Also note that the MPI version used by Block may be contradicted with MPI used by ORCA, you have to comment one version of MPI environment variables at a time. See Section 2.1 for related information.

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, you have to comment one version of MPI environment variables at a time. See Section 2.1 for related information.

4.4.14 CASPT2_prog

Specify the program for performing CASPT2 calculation, e.g. CASPT2_prog=OpenMolcas. Currently only OpenMolcas is supported and it is the default program. All core orbitals are not frozen. Note that the default IP-EA shift is 0.25 a.u. and it cannot be modified. This value is also the default value in OpenMolcas. 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 only PySCF is supported and it is the default program. All core orbitals are not frozen.

The NEVPT2 method in PySCF is actually SC-NEVPT2, where ‘SC’ for strongly contracted.

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/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.

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.

To choose a contraction type, please read Section 4.4.18 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 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.

4.4.19 MaxM

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

4.4.20 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.

4.4.21 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_2 molecule is stretched to d(N-N) = 4.0 Å, this is a system which features strong correlation and requires a CAS(6,6) active space. The Davidson iterative diagonalization in determinant CASCI (using GAMESS) may not find the singlet state in the lowest 5 states. In this case, specifying ‘crazywfn’ will increase the NSTATE to 10, so that the singlet state can be found.

4.4.22 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.

4.4.23 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 all methods in AutoMR.

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

4.4.24 OtPDF

The choice of the on-top pair density functional. This keyword has identical meaning with the keyword KSDFT in OpenMolcas/Molcas software. Currently available functionals are tPBE, 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/Molcas. Old versions may not support some of the functionals.

4.4.25 DKH2

Request the 2nd order scalar relativistic Douglas–Kroll–Hess (DKH2) correction to the one-electron Hamiltonian. Note that: (1) the two keywords DKH2 and X2C are mutually exclusive; (2) you should use all-electron basis sets like ‘cc-pVTZ-DK’, ‘x2c-TZVPall’ or ‘ANO-RCC-VDZ’ for all-electron relativistic calculations.

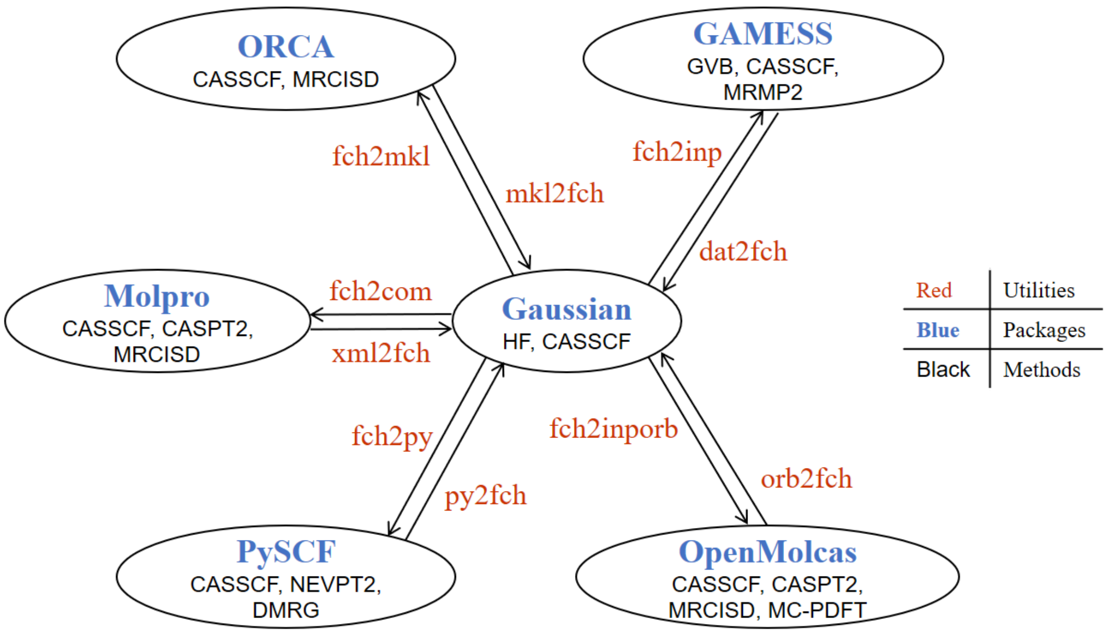
4.4.26 X2C

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.

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.

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      0.0      0.1
-4.0      0.0      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_gms2molcas

Generate an (Open)Molcas .input file from a GAMESS .inp or .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).

4.5.4 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).

4.5.5 bas_gms2py

Generate a PySCF .py file from a GAMESS .inp or .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).

4.5.6 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 4.5.9.

4.5.7 extract_noon2fch

Extract natural orbital occupation numbers (NOONs) from (1) .out file of PySCF or (2) .dat file of GAMESS, and write NOONs into a given .fch file. This is for the convenience of visualizing NOONs in GaussView.

4.5.8 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).

4.5.9 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 4.5.6.

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).

4.5.10 fch2inporb

Transfer MOs from Gaussian to OpenMolcas/Molcas. A .input file and a .INPORB file will be generated. The .input file is the input file of OpenMolcas/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).

4.5.11 fch2mkl

Transfer MOs from Gaussian to ORCA. A .inp file and a .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.

Note 1: you need to run ‘orca_2mkl a.mkl -gbw’ to generate the ORCA .gbw file since ORCA cannot read .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).

4.5.12 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.

4.5.13 fch_u2r

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

4.5.14 frag_guess_wfn

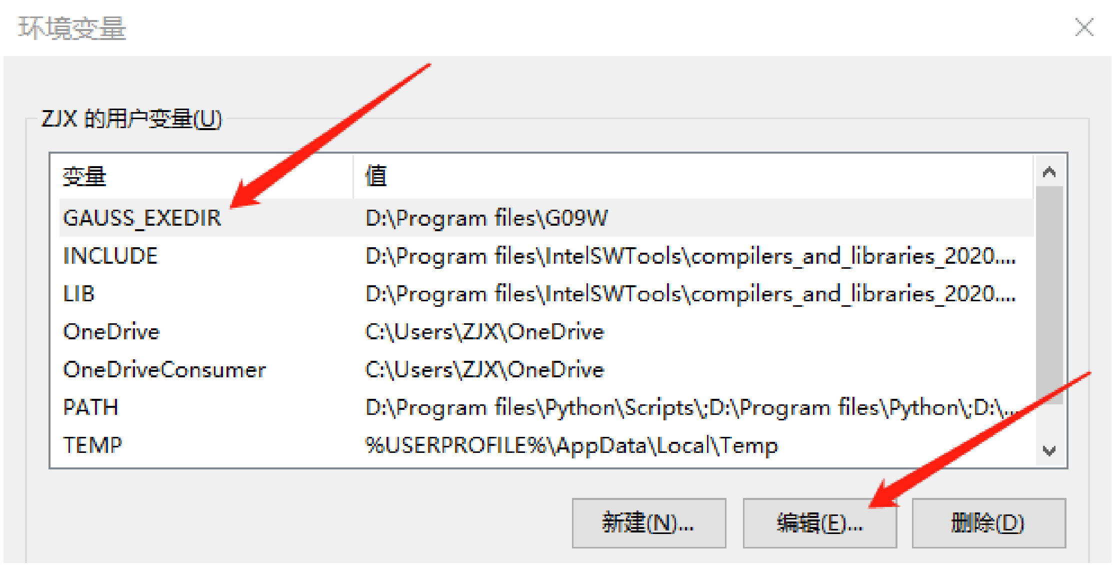
This is a utility designed to perform a SCF (i.e. HF or DFT) computation using initial guess constructed from molecular orbitals of fragments. The Gaussian software package will be called to perform SCF computation for each fragments (and only Gaussian is supported). The input file is exactly the Gaussian .gjf 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-compiled executable frag_guess_wfn is provided at Releases. Note that you should define the environment variable %GAUSS_EXEDIR% before using this utility, since Gaussian program would be called to perform SCF computations. For example, search "environment variable" on your Windows 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


环境变量

ZJX 的用户变量(U)

变量	值
GAUSS_EXEDIR	D:\Program files\G09W
INCLUDE	D:\Program files\IntelSWTools\compilers_and_libraries_2020....
LIB	D:\Program files\IntelSWTools\compilers_and_libraries_2020....
OneDrive	C:\Users\ZJX\OneDrive
OneDriveConsumer	C:\Users\ZJX\OneDrive
PATH	D:\Program files\Python\Scripts\;D:\Program files\Python\;D:\...
TEMP	%USERPROFILE%\AppData\Local\Temp

4.5.15 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.

4.5.16 mkl2fch

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

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

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.

4.5.17 orb2fch

Transfer MOs from OpenMolcas *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.

4.5.18 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.

4.5.19 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’.

4.5.20 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.

4.5.21 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.

4.5.22 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.

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: the error '-bash: f2py: command not found' means there is no f2py on your computer, see Section **2.1 Prerequisite** for details.
Example 2: 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 **2.3 environment variables** for details.

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

A2: The author offers 14 utilities of Windows OS 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 and orb2fch 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.

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: R02O02 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 R02O02; but for version >=20.10, this keyword become RELAtivistic = R02O02. 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 = R02O02’ to ‘R02O02’ in file src/automr.f90, and run ‘make automr’ in Shell to re-compile the program automr.

Q6: How does AutoMR read the 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 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.'
found in GAMESS .gms file (where xx is some integer)?

A7: Please read Section 4.4.9 carefully.

A2 Limitations and Suggestions

1. Basic knowledge of multi-reference methods

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

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.

3. The 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 assumed 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). 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 which feature block-box or automatic multi-reference calculations.

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.

5. Density in the .fch file

In the current version of MOKIT, the CASCI or CASSCF NOs are stored in the corresponding *_NO.fch files after CASCI or CASSCF job finished. However, the density in any *_NO.fch file (‘SCF Density’ Section) may be wrong, they are possibly not CASCI or CASSCF density!!! In MOKIT 2.0, this defect will be removed. The exception is that you use the Gaussian as the CASCI/CASSCF solver, so that the density in .fch file is right.

To conclude, if you need CASCI or CASSCF density (to perform population analysis, wave function analysis, etc), USE THE GAUSSIAN SOFTWARE PACKAGE AS THE CASCI OR CASSCF PROGRAM (i.e. casci_prog=Gaussian or casscf_prog=Gaussian).

A3 Bug Report

If you find any bug frequently occurs, you can contact the author jxzou via E-mail njumath[at]sina.cn, with your input file (.gjf, .fch) and output files attached. Or you can open an issue on gitlab page of MOKIT (https://gitlab.com/jxzou/mokit/). The author may not answer 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.