MOKIT Manual

version 1.0 (beta)

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
```

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) are 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(s) of MOKIT in your work, please at least cite the gitlab page of MOKIT, just like

[1] Molecular Orbital KIT (MOKIT), https://gitlab.com/jxzou/mokit

If any of the keyword ist=0,1,3 (see Section 4.4.4) is used, please also cite this paper

J. Chem. Theory Comput. 2019, 15, 141-153. DOI: 10.1021/acs.jctc.8b00854

2 Installation

The installation of MOKIT does not require the root privilege.

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.

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 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 ~ 4.4.15 for details). If you have any difficulty in installing software, please read corresponding manuals carefully. Or you can find answers from their official site, forums, or github/gitlab page.

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

The original GAMESS code can only deal with GVB up to 12 pairs.

2.2 How to compile

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

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

to uncompress it. Then simply run the following two command lines in Shell

```
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. 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.

2.3 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
```

Note that the last path is 'PYTHONPATH', not 'LD_LIBRARY_PATH'. A logout and re-login on your terminal is strongly recommended.

Besides, you need to modify the paths of various software packages in the file mokit/program.info. Please modify the paths to suit your situations. Since the PySCF is run by python, there is no corresponding path variable in file program.info.

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=16GB
%nprocshared=8
#p CASSCF/cc-pVQZ

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

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 symbol of 'mokit{}' in the Title Card line of a .gjf file.

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

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

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. Do not use unit MB, MW, or %nproc, %cpu. Since multireference computations often require large memory, it is recommended to use at least %mem=2GB 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 serial mode.

4.3 Supported methods in AutoMR

Currently, all supported methods in AutoMR are: GVB, CASCI, CASSCF, DMRGCI, DMRGSCF, NEVPT2, CASPT2. More multi-configurational and multi-reference methods will be supported in the near future. These terms should be written in the '#p …' line in the .gjf file. Here the terms DMRGCI and DMRGSCF actually means DMRG-CASCI and DMRG-CASSCF method.

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. The 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.

There must be a '/' symbol between the method and the basis set. 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 the user still needs to provide a basis set name, although it is not used in this case.

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.

4.4.5 LocalM

Specify the orbital localization method. Only the Boys localization and Pipek-Mezey localization method are supported. The corresponding keywords are 'LocalM=Boys' and 'LocalM=PM'. By default, the Boys 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 molecule containing only σ bonds. But if you are dealing with 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 CASCI or CASSCF. Geometry optimization is probably to be supported in the next version of

MOKIT. Note that this keyword do not have any attribute value, i.e. do not write ‘force=.True.’ but only ‘force’ in {}. Numerical gradient is not supported.

4.4.8 NoCart

Request no use of Cartesian functions. In another word: try to use spherical harmonic functions. These two types of basis functions are corresponding to ‘6D 10F’ (Cartesian functions) and ‘5D 7F’ (spherical harmonic functions) in Gaussian. Note that GAMESS only supports Cartesian functions, thus this keyword cannot be specified when using strategy 0, 1 or 3 (ist=0, 1 or 3), or setting CASCI_prog, CASSCF_prog as GAMESS, because GAMESS program is called in these strategies.

When using other strategies, it is possible to specify ‘NoCart’. For example, Gaussian and OpenMolcas support both Cartesian and spherical harmonic functions. For ORCA, it does not support Cartesian functions, thus you must specify ‘NoCart’ if you use ORCA as the CASCI solver.

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.

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) 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 replaced and destroyed during re-compilation.

Besides, the original GAMESS only supports 32 CPU cores, if your machine has more than that and 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, and Gaussian. If you want to use a CASCI program instead of PySCF, I recommend GAMESS and OpenMolcas.

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, and Gaussian. If you want to use a CASSCF program instead of PySCF, I recommend GAMESS and OpenMolcas.

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.

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.

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.

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.

4.4.16 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.

4.4.17 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.

4.4.18 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.

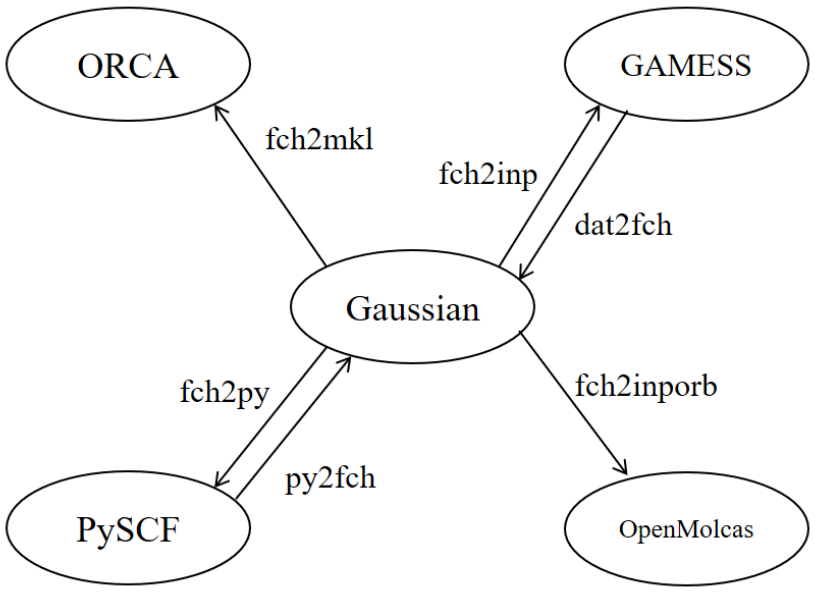
4.4.19 no10cycle

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.

4.5 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 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.

4.5.3 bas_gms2molcas

Generate a OpenMolcas or Molcas .input file from a GAMESS .inp or .dat file. The Cartesian coordinates and basis set data are written in the .input file.

4.5.3 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.

4.5.4 dat2fch

Transfer MOs from GAMESS (.inp or .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.

The best way is to firstly use Gaussian to generate a .fch file (with keywords ‘nosymm int=nobasistransform 6D 10F’), then generate the .inp file from .fch. After GAMESS computations finished, you can transfer MOs from .dat to .fch file. This procedure seems a bit of tedious, but it ensures an exact reproduce of energy in GAMESS.

To transfer MOs from Gaussian to GAMESS, see 4.5.6.

4.5.5 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.

4.5.6 fch2inp

Transfer MOs from Gaussian to GAMESS. A GAMESS .inp file will be generated, with MOs written in it. 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.

4.5.7 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.

Note: the default option is to transfer RHF MOs. If you want to transfer UHF MOs, use '`fch2inporb a.fch -ab`'. 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 only want to compile fch2inporb, you have to compile 'fch2inp' and 'bas_gms2molcas' additionally.

4.5.8 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 '-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.

4.5.9 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.

4.5.10 fch_u2r

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

4.5.11 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.

4.5.12 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.

4.5.13 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'.

4.5.14 fch2py

Transfer MOs from Gaussian to PySCF. By importing fch2py, PySCF Python script is able to read alpha and/or beta MOs from a specified .fch file.

4.5.15 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.

5 Examples

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

Appendix

A1 FAQ

Q1: Find errors like 'xxx: command not found'. What is the solution?
A1: This is simply because you haven't install 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: Since the MOKIT is open-source, you can compile the source code on your laptop/computer/node. The Mac OS pre-compiled executables will never be considered by the author. The Windows pre-compiled executables containing utilities of MOKIT are probably to be released in near future. But the AutoMR module will be not included in the Windows pre-compiled executables, since multi-reference calculations are usually performed on high-performance platforms, which usually contain Unix/Linux systems.

Q3: Why the utilities 'dat2fch' and 'py2fch' cannot generate a .fch file from scratch, but requiring 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 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. 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.

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 all modules 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 poor 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 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 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 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.

A3 Bug Report

If you find any bug frequently occurs, you can contact the author jxzou via 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.