The HFSCF user manual

Alexander F Borro

September 7, 2021

Contents

1 HFSCF - Perform UHF/RHF calculations and beyond on small polyatomic molecules

Inpired by the crawdad exercises, hfcxx, McMurchie-Davidson, and a 30 year hiatus from physical chemistry and spectroscopy, a small *ab initio* package was born, written in C++. The original Crawdad exercises hfcxx and McMurchie-Davidson may be found here and here. With the exception of the DL algorithm you will find solutions to all the Crawdad exercises, and more.

1.1 Supported features

- RHF.
- UHF.
- SOSCF.
- DIIS.
- SAD and CORE initial guess methods.
- MP2, MP3 RHF/UHF.
- CIS excitation energies and oscillator strengths.
- RPA energies
- CCSD CCSD(T) in spin and closed shell basis (RHF)
- Analytic gradients, SCF and MP2 (RHF).
- Analytic gradients, SCF (UHF).
- Numeric gradients, MP2 (UHF)
- Conjugate gradient or RFO geometry optimization at the SCF and MP2 level in redundant internal coordinates using the IRC library.
- Shell based Cartesian Gaussian integrals up to 2nd derivative (Obara-Saika).
- Support for pure angular momentum and cartesian basis sets in Psi4 compatible format.
- Symmetry and SALC determination (RHF and UHF only) using libmsym.
- Analytic Hessian SCF and MP2 (RHF).
- Numeric Hessian SCF and MP2 (UHF).
- Vibrational Harmonic frequencies.
- IR Intensities.
- Normal modes.
- Thermochemistry analysis.
- Lowdin and Mulliken population analysis.
- Dipole moments, Quadrupole moments and static dipole polarizabilities (using direct or iterative CPHF).

1.2 Building the software

To build the software you can use either cmake or meson running on any regular Linux distribution, or the WSL subsystem for Windows. The following packages must be installed (if not present already).

- Optional: git (if you you want to clone the repository)
- The make or ninja build system.
- gcc++ 9 or later. (versions must support the C++17 standard)
- Optional: OpenMP support, but highly recommended for increased performance; gcc/g++ has OpenMP support out of the box.
- meson or cmake
- Library; Eigen3 (3.3.6 or later).
- pkgconfig. required to detect tclap.
- Library: tclap.
- Library: C++ boost libraries (1.58 or later).
- libmsym: if using meson only. A cmake build includes it.

Note: while clang++ works it is unsupported at this time.

1.3 Downloading the software

Either download as a zip file from here, or use git and at the command prompt issue the command

```
git clone https://github.com/AlexB67/hfscf.git
```

1.4 Building with cmake

Assuming the project is at the top level home directory and your current path is the home directory, it will look like

```
/home/*your_user_name*/hfscf
```

issue the following commands

```
cd hfscf
mkdir cm_build/
cd cm_build/
ccmake .. # or you can use cmake-gui
```

Change the Release build flags for C and C++ to O3, configure and generate.

```
cmake ..
make -j # or whatever build system you have elected to use, for example, for ninja it
     would be "ninja".
```

You are now done building the software.

1.5 Building with meson

NB: If you have already build the software with cmake you may skip this step. Install libmsym as a static library. Generally it is not available in distribution repositories, with the exception of AUR Arch Linux (which may well be outdated). 0.2.4 is required. It can be obtained from here. Follow the instructions within. Assuming the project is at the top level home directory and your current path is the home directory, it will look like

```
/home/*your_user_name*/hfscf
```

issue the following commands

```
cd hfscf
mkdir m_releasebuild/
meson m_releasebuild/ --buildtype=release
ninja -C m_releasebuild/
```

You are now done building the software. Note: Do not mix meson and cmake build directories. I have given the build directories unique names as described above. You can use whatever build directory names you like.

1.6 Using the software

For more information how to use the software, see the examples directory. You can run the program from the build directory. Installation is not necessary and unsupported. Running the program simply involves

```
cd *to_build_directory*
src/hfscf -i ../examples/inputfile
```

For example

```
src/hfscf -i ../examples/h2o_1.dat
```

or for more verbose output

```
src/hfscf -i ../examples/h2o_1.dat -v 3
```

or should you wish to write to an ouput file, you can simply redirect the ouput as follows

```
src/hfscf -i ../examples/h2o_1.dat -v 3 > /path_to_output_file/h2o_1.out
```

For additional arguments type

```
src/hfscf --help
```

1.7 Preparing an input file

Input files are in text format. The first line is comment line and is mandatory, any other comment lines are optional and start with the # symbol. Keywords are always in the form

```
keyword = value
```

and often case sensitive. For a list of keywords see the source file

```
hfscf/src/molecule/hfscf_keywords.hpp
```

Keywords will also be discussed in the next section. Coordinates are placed in curly braces, no keywords can be used in this section. Both Cartesian and Z matrix formats are supported, but may not be mixed. Variable names are not allowed for bond lengths, angles and dihedrals.
Important: *The default unit for length is in atomic units, if not specified, contrary to most packages using Angstroms as the default unit for length.* A typical input Z matrix input file will look like

```
# ethene SCF frequencies
scf_type = RHF
basis_set = 6-31+G**
gradient_type = SCF
eri_screen = false
symmetrize_geom = true
geom_opt = SCF
geom_opt_tol = HIGH
frequencies = SCF
units = angstrom
{
  C
  C    1  1.3358
  H    1  1.0855   2 121.051
  H    1  1.0855   2 121.051   3 180.000
  H    2  1.0855   1 121.051   3   0.000
  H    2  1.0855   1 121.051   3 180.000
}
-
```

dummy atoms are denoted by the symbol `X`. A Cartesian input will look like

```
# Benzene HF/6-311G no dummy atoms
initial_guess = SAD

# DIIS extrapolation steps
diis_range = 8

# Basis set
basis_set = 6-311G

# turn on symmetry
use_symmetry = true

# force point group
point_group_override = D2h

# Units used. Cartesian input allows atomic numbers, or atomic symbols
# A Z matrix allows atomic symbols only.

units = angstrom
{
  6  0.000  1.396  0.000
  6  1.209  0.698  0.000
  6  1.209 -0.698  0.000
  6  0.000 -1.396  0.000
  6 -1.209 -0.698  0.000
  6 -1.209  0.698  0.000
  1  0.000  2.479  0.000
  1  2.147  1.240  0.000
  1  2.147 -1.240  0.000
  1  0.000 -2.479  0.000
  1 -2.147 -1.240  0.000
  1 -2.147  1.240  0.000
}
```

Keywords often assume a default value if not entered. If no basis set is specified, `STO-3G` will be used. The hfscf program will validate your input file. In most cases an error message will be displayed followed by a graceful exit if an error is detected, with a suggestion how to fix it.

1.8 Basis sets

Basis sets are `psi4` format compatible only, a number are already included. New basis sets can be obtained from the basis set exchange website. hfscf maximum supported angular momentum is $L = 5$. Density fitted basis sets are not supported. Make sure that the `psi4` format option is selected when downloading. You can place a new basis in the basis set directory found at `hfscf/basis` Note: You must remove the middle part of the file name; Typically the names are *somebasis.0.gbs* or *somebasis.1.gbs* Remove the 0 or 1, so the new file name becomes *somebasis.gbs*.

2 Keywords

2.1 Keywords summary

The following will give a listing of all supported HFSCF keywords and a brief explanation of each; Default values will be specified where they exist.

- `basis_set`
  - `STRING: STO-3G (default)`

  The basis set to use for computations. Only one per input file.

- `ccsd_diis_range`
  - `INTEGER: 6 (default) RANGE: 2 - 8`

  The number of DIIS steps to use during a post SCF RHF CCSD computation. A value of 0 disables DIIS.

- `ccsd_energy_tol`
  - `DOUBLE: 1E-07 (default)`

  The energy tolerance for CCSD (RHF) convergence.

- `ccsd_rms_tol`
  - `DOUBLE: 1E-07 (default)`

  The RMS tolerance for CCSD (RHF) convergence.

- `charge`
  - `INTEGER: 0 (default)`

  The charge of the molecule.

- `ci_type`
  - `STRING: CIS`
  - `STRING: RPA`

  Specify the type of configuration interaction computation to perform (RHF only).

- `density_damp`
  - `DOUBLE: 1.0 (default) RANGE: 0.2 - 0.9`

  The amount of damping to apply to changes in the density matrix (RHF) or density matrices (UHF) during SCF cycles. A value of 1.0 disables damping. The smaller the value, the greater the damping at the expense of an increase in SCF cycles (rarely useful but in rare cases may tame SCF oscillations).

- `diis_range`
  - `INTEGER: 6 (default) RANGE: 2 - 8`

  The number of DIIS steps to use during a SCF (RHF and UHF) computation. A value of 0 disables DIIS.

- `energy_tol`
  - `DOUBLE: 1E-07 (default)`

  The energy tolerance for SCF (RHF and UHF) convergence.

- `eri_screen`
  - `BOOL: false`
  - `BOOL: true (default)`

  Whether to apply screening to the 2 electron ERI integrals. Particularly useful for larger molecules, reduces computation cost of ERI integrals.

- `electronic_properties`
  - `BOOL: true`
  - `BOOL: false (default)`

  Whether to compute various electronic properties. Current properties include dipole moment, Quadrupole moment, static dipole polarizabilities, Mulliken and Löwdin population analysis.

- `feeze_core`
  - `BOOL: false`
  - `BOOL: true (default)`

  Whether to freeze core (non valence) electrons in post SCF calculations. Used in CCSD and MPn type calculations.

- `frequencies`
  - `STRING: SCF`
  - `STRING: MP2`

  Compute vibrational frequencies at the SCF or MP2 level. Note that computing MP2 frequencies in particular is a very expensive calculation, dwarfing any Hartree Fock SCF computation by orders of magnitude.

- `initial_guess`
  - `STRING: CORE`
  - `STRING: SAD (default)`

  The initial guess to use for RHF and UHF computations. SAD uses a super position of atomic densities to build a molecular density guess matrix. CORE uses the Core Hamiltonian eigenvectors/eigenvalues. SAD is by far the superior method.

- `geom_opt`
  - `STRING: SCF`
  - `STRING: MP2`

  Whether to perform a geometry optimization at the SCF or MP2 level. You may wish to disable symmetry altogether, or lower the symmetry for geometry optimization computations. Currently optimization with symmetry is not well implemented.

- `geom_opt_algorithm`
  - `STRING: CGSD`
  - `STRING: RFO (default)`

  The algorithm to use for geometry optimization. `RFO` uses rotational function optimization. `CGSD` is the simpler conjugate gradient (steepest decent))method, only used when gradients or the hessian are determined numerically, `CGSD` is for UHF only and may be removed in a future version. `RFO` converges much faster and is the superior method.

- `geom_opt_constrain_angles`
  - `BOOL: true`
  - `BOOL: false (default)`

  Whether to constrain angles during a geometry optimization. depending on the structure this flag may be ignored. `RFO` only.

- `geom_opt_constrain_dihedrals`.
  - `BOOL: true`
  - `BOOL: false (default)`

  Whether to constrain dihedral angles during a geometry optimization. depending on the structure this flag may be ignored. `RFO` only.

- `geom_opt_constrain_oop`.
  - `BOOL: true`
  - `BOOL: false (default)`

  Whether to constrain out of plane angles during a geometry optimization. depending on the structure this flag may be ignored. `RFO` only.

- `geom_opt_stepsize`
  - `DOUBLE: 0.5 (default) RANGE: 0.05 - 1.0`

  For the `CGSD` algorithm only; The step size to use. Note that the step size is automatic for `RFO`, so this value will be ignored.

- `geom_opt_tol`
  - `STRING: LOW`
  - `STRING: MEDIUM (default)`
  - `STRING: HIGH`
  - `STRING: VERYHIGH`

  The tolerance to use for geometry optimization convergence. `LOW` is loose, `VERYHIGH` is tightest. `CGSD` will only allow a maximum of `MEDIUM` due to numeric limitations. For determining vibrational frequencies `MEDIUM` is typically sufficient.

- `geom_opt_write_xyz`
  - `BOOL: true`
  - `BOOL: false (default)`

  Whether to write a trajectory file during geometry optimization. The resultant file may be viewed in Avogrado.

- `gadient_type`
  - `STRING: SCF`
  - `STRING: MP2`

  Whether to compute SCF or MP2 gradients.

- `grad_step_from_energy`
  - `DOUBLE: 1E-06 (default) RANGE: 1E-04 - 1E-06`

  UHF MP2 only; The step size to use (by varying coordinates) for a numeric gradient computation. Uses the 3 point formula to compute numeric derivatives in Cartesian coordinates.

- `hessian_step_from_energy`
  - `DOUBLE: 1E-04 (default) RANGE: 1E-02 - 1E-04`

  UHF MP2 only; The step size to use (by varying coordinates) for a numeric Hessian computation. Uses the 3 point formula to compute numeric derivatives in Cartesian coordinates.

- `integral_tol`
  - `DOUBLE: 1E-14 (default)`

  The tolerance for detecting small terms during ERI computations and treat them as zero. Also used during ERI screening where it effectively serves the same role as the Cauchy-Schwarz cutoff.

- `max_ccsd_iter`
  - `INTEGER: 30 (default)`

  The maximum number of iterations allowed for convergence of a CCSD computation.

- `max_geomopt_iter`
  - `INTEGER: 15 (default)`

  The maximum number of iterations allowed for convergence of a geometry optimization computation. `CGSD` usually requires a higher number due to slower convergence. For reasonable structures, i.e. structures reasonably close to equilibrium `RFO` typically converges in 5 - 10 cycles for small molecules (around 5 - 15 atoms).

- `max_sad_iter`
  - `INTEGER: 25 (default)`

  The maximum number of iterations allowed for convergence of an atomic SAD SCF computation. Typically one should converge well before this limit using DIIS (the default value is 6 for SAD and currently hard coded).

- `max_scf_iter`
  - `INTEGER: 50 (default)`

  The maximum number of iterations allowed for convergence of a SCF (RHF or UHF) computation. Typically one should converge well before this limit using DIIS or SOSCF. If not, there is probably an issue with the molecule or structure, or a single Slater determinant Hartree Fock type computation is simply not sufficient to describe the molecule.

- `max_soscf_iter`
  - `INTEGER: 4 (default) RANGE: 3 - 6`

  The maximum number of micro iterations allowed for convergence of SOSCF (RHF/UHF).

- `mol_props_cphf_iter`
  - `BOOL: true`
  - `BOOL: false (default)`

  The CPHF method used to compute static dipole polarizabilities, iterative CPHF, or direct CPHF via matrix inversion.

- `multiplicity`
  - `INTEGER: 1 (default)`

  The multiplicity of the molecule, given by $2S + 1$.

- `point_group_override`
  - `STRING`

  Specify a subgroup to use. The subgroup must be a valid group of the molecular point group. Use this setting if you wish to run a computation in lower symmetry. Point groups are specified as per libmsym convention, for example `Cs` or `C2v` etc. If you specify a group using this keyword, there is no need to set the `use_symmetry` keyword, symmetry will be enabled automatically.

- `point_group_threshold`
  - `STRING: RELAXED`
  - `STRING: DEFAULT (default)`
  - `STRING: TIGHT`

  The tolerance to use to determine the molecular point group. `RELAXED` is useful if libmsym fails to assign a point group. If the `RELAXED` still fails with this setting no point group will be assigned, which is effectively equivalent to the C_1 point group. `TIGHT` is useful for a more strict assignment. Consider for example benzene with 6 bond lengths almost equal with D_{2h} symmetry, but very close to D_{6h} symmetry. libmsym will assign D_{6h} with the `DEFAULT` tolerance setting, however the `TIGHT` tolerance setting will assign D_{2h}. Having a point group with tighter tolerance, may also, in some cases help with SCF convergence if symmetry is enabled, see the `use_symmetry` keyword.

- `rms_tol`
  - `DOUBLE: 1E-07 (default)`

  The RMS tolerance for SCF (RHF and UHF) convergence.

- `sad_energy_tol`
  - `DOUBLE: 1E-5 (default)`

  The energy tolerance for SCF SAD convergence, does not need to be very precise given the crudeness of the guess density.

- `sad_rms_tol`
  - `DOUBLE: 1E-6 (default)`

  The RMS tolerance for SCF SAD convergence.

- `scf_direct`
  - `BOOL: true`
  - `BOOL: false (default)`

  Compute ERI integrals on the fly during SCF cycles, thus avoiding the storage of ERI integrals in memory. Currently disabled.

- `soscf`
  - `BOOL: true`
  - `BOOL: false (default)`

  Whether to perform the second order accelerated SCF scheme (RHF and UHF), which can help with potential convergence issues, at the expense of increased computation time, thus, only use if necessary.

- `soscf_rms_tol`
  - `DOUBLE: 5E-3 (default)`

  The RMS tolerance for `SOSCF` micro iterations to convergence.

- `symmetrize_geom`
  - `BOOL: true`
  - `BOOL: false (default)`

  Whether to update the molecular geometry to correct for any deviations from the actual point group. This flag is also useful for geometry optimization. If symmetry is enabled and a geometry optimization is performed, this setting will be enabled automatically.

- `thermo_chem_pressure`
  - `DOUBLE: 101325.0 (default)`

  Thermochemistry; The pressure to use in `Pa`. The defaults for temperature and pressure are at SATP.

- `thermo_chem_temperature`
  - `DOUBLE: 298.15 (default)`

  Thermochemistry; The temperature to use in `K`. The defaults for temperature and pressure are at SATP.

- `use_symmetry`
  - `BOOL: true`
  - `BOOL: false (default)`

  Whether to use point group symmetry during a SCF (RHF and UHF) computation.

- `use_puream`
  - `BOOL: true`
  - `BOOL: false`

  Whether to treat basis functions as pure $5d$, $7f$ etc. i.e $2l+1$ functions per shell, or Cartesian $6d$, $10f$ etc. $(l+1)(l+2)/2$ function per shell. If not set, `use_puream` will be determined by the basis set. Note that symmetry support only works when `use_puream` is enabled.

- `uhf_guess_mix`
  - `BOOL: true`
  - `BOOL: false (default)`

  Whether to perform mixing of the HOMO and LUMO guess orbitals. Symmetry must be disabled.

- `units`
  - `STRING: angstrom`
  - `STRING: bohr (default)`

  The unit used for length, which is used to specify the molecular geometry in the input file.

A few keywords have not been mentioned but may be added in future, they are currently for development purposes only.

3 Some final comments

While HFSCF does not have a test suite currently, all results in the examples directory have been verified against the latest version of Psi4 on numerous occasions during development.

3.1 Performing a geometry optimization

`RFO` is preferred over `CGSD`. Depending on the molecule and structure, numeric gradients can show significant rounding errors near equilibrium, this is only an issue for numeric UHF/MP2 gradients. SCF UHF and SCF/MP2 RHF gradients are analytic. Using symmetry is currently not very robust for optimising geometries. The geometry optimization routines do not symmetrize coordinates, only the SCF computation uses symmetry. You may wish to disable symmetry altogether or use a lower symmetry should a geometry optimization blow up, nevertheless, small molecules or well behaved structures can work okay. See the examples directory. Only minimum structure searches are supported, but not transition state searches.

3.2 Frequency calculations

As for numeric gradients, numeric UHF Hessian calculations can suffer from error build up, for example, when two degenerate vibrational frequencies result from a Hessian, they will not be quite equal, this is normal, adjusting the step size may help to reduce this problem.

The Hessian projection routine will always remove rotation and translation type modes, whether the structure is at a minimum or not, as such then, the projected frequencies only make sense when gradients are close to zero, i.e. this happens when a geometry optimization has been performed prior to calculating the Hessian, or the supplied structure is already at a minimum for the given basis set.

Finally Hessian calculations can take a long time and use a lot of memory, in particular the MP2 variant, you will not be warned by the program, there are no checks build in currently what the memory consumption will be for a given computation. Play around with examples to get a feel. All the examples will run comfortably on machine with 8GB of memory.

3.3 Parallel computing and Threads

By default HFSCF will use all cores available when executed without the `-n` *num_ threads* argument, unless you have OMP environment variables set up on your system to change that. In particular for hyper-threaded CPUs with short runs (of the order of seconds) on small molecules reducing the core count can in fact speed up a calculation slightly. In any case, you can manually set the numbers of cores/threads with the `-n` argument, for example

```
src/hfscf -i ../examples/h2o_5.dat -v 3 -n 6 # Set 6 threads with -n #
```

3.4 Bugs and other things

Feel free to raise a bug if needed. Have fun playing around. I sure had fun writing HFSCF.