Run Extended Broken Symmetry geometrical optimizations with Orca

Francesco Cappelluti

- Create, in the directory where the optimization is to be run, three subdirectories called `geo_opt`, `spin_ladder` and `wfs_opt`.
- Put `heisenberg.inp` in the main directory along with `ebs.py`.
- Create subdirectory `init` in `spin_ladder` and subdirectories `init` and `1` in `wfs_opt`.
- Create, both in `wfs_opt/init` and `wfs_opt/1`, N subdirectories with names going from 0 to N-1 where N is the number of broken symmetry states of the system we want to optimize. Please note that high spin state is considered to be a particular broken symmetry state, so $N = \frac{N_{sc}(N_{sc}-1)}{2}$ where N_{sc} is the number of spin centers.
- Put in `wfs_opt/init/?` the Orca input files for energy plus gradient calculation (engrad) for each of the N broken symmetry states and run these calculation.
- Once the calculations are finished, run `init_parsing.py` in `wfs_opt/init` in order to obtain `engrad.dat`, `Energies.dat`, `M_values.dat` and `S2_tot.dat`.
  - `engrad.dat` contains, as first line, the number of broken symmetry states and of gradient components. Each one of the following lines is the gradient of the corresponding broken symmetry state.
  - `Energies.dat` contains the energies of the broken symmetry states.
  - `M_values.dat` contains as first line the number of broken symmetry states and of spin centers. Each of the following lines contains the spin moments of all the spin centers (whose chemical name is specified in `init_parsing.py` and in `ebs.py`). In order to have the optimization working properly (so in order to have, for example, the components of each broken symmetry gradient in `engrad.dat` to correspond), the cartesian coordinates file has to be the same for each single point calculation. A very simple way of doing so (that will be mandatory in the following steps) is to force Orca to read system coordinates from an external file through the use of `xyzfile` keyword.

- `S2_tot.dat` contains the $\langle \hat{S}^2 \rangle$ values for each broken symmetry state, calculated using Mulliken method.

- Copy the previous `.dat` files in `spin_ladder/init`.
- Run `spin_ladder.x` in `spin_ladder/init`, redirecting `heisenberg.inp` as input:

  ```
  spin_ladder.x < ../../heisenberg.inp
  ```

  Remember that `spin_ladder.x` need the BLAS and LAPACK libraries to be dynamically linked.
- Copy the produced file `GS.extcomp.out` in `geo_opt` renaming it as `inputname.extcomp.out`.
- Put `inputname.inp` (an Orca input file containing only the keywords for performing an optimization using external optimizer feature), cartesian coordinates file `inputname_last_step.xyz` (the same used for previous single point calculations) in `geo_opt`. Create also a file called `iter_step` containing only the number 1 (it will work as a counter for the optimization process).
- Copy Orca input files from `wfs_opt/init/?` to `wfs_opt/1/?`, along with wavefunction `.gbw` file obtained, renamed as `inputname_last_step.gbw`.
- Edit previously copied Orca input files deleting (or commenting) `flipspin` and `finalms` keywords in `%scf` block, adding `guess moread` and `moinp "inputname_last_step.gbw"` lines in the same block and enforcing coordinates reading from external file `inputname_last_step.xyz` (using `xyzfile` keyword).
- Copy starting coordinates file from `geo_opt` to `wfs_opt/1/?`.
- Assure that `ebs.py` is correctly edited (check for the name of the main directory, path of Orca executable file, path of `spin_ladder.x`, chemical name of spin centers, etc.).
- Impose that `ebs.py` is executed when Orca calls `orca_External` (the default executable Orca looks for when the use of external optimizer is required).

  What I do is to put in my `bin` directory (whose path is therefore exported) a BASH executable file called `orca_External`, containing an expression of this kind:

  ```
  if [ $PWD == absolutepathtogeo_opt ]; then
      python absolutepathtoebs.py/ebs.py;
  fi
  ```

Doing so, it is also possible to run multiple EBS optimizations at the same time, using the construct `elif`.

- Submit the job, remembering that the number of nodes and/or processes required has to be consistent with what is written in `ebs.py` and with the number of MPI processes required in Orca single point input scripts with the construct `%pal nprocs N end`. It is fundamental that, when the job starts, the names of the nodes on which it is running get written in a file called `machinefile` that is put in the main directory. `ebs.py` script will use this file for connecting (through ssh) to the nodes and submit single point calculations. For such a reason, ssh connection to calculation nodes in non-interactive mode must be enabled (we couldn't, for example, succeed in doing that when authetication to nodes is handled by Kerberos system).

  When SLURM job scheduler is employed, `machinefile` can be obtained adding the subsequent line to submit script (prior than Orca invocation, of course):

  ```
  scontrol show hostnames $SLURM_JOB_NODELIST > machinefile
  ```

BS states single point calculations can be performed with CP2K too. It is sufficient to set to `"cp2k"` the `ES_PROGRAM` constant in `ebs.py` and `init_parsing.py` files but this approach has not been well tested such as the Orca one. Moreover, the geometrical optimization calculation (running in `geo_opt` directory) has to be mandatory performed through Orca thanks to its `extopt` feature.

A complete working example for the EBS optimization of a [4Fe-4S] cluster can be found in `example/` directory.