CHEM 740 Dec 17th, 2019

PYSCF HF/DFT CALCULATIONS AND COMPARISONS WITH ORCA, BUDAPEST MRCC

FINAL REPORT

Haobo Liu 20647040

CONTENTS

ABSTRACT

PySCF, or Python-based simulations of chemistry framework, is one of the major computational chemistry platforms used to develop and test advanced algorithms, known for its efficient in calculation. However, as a python code-based software package, there is no any graphical user interface and getting started on PySCF would require more efforts than some other popular computational chemistry programs like Gaussian, Orca or MRCC.

The CHEM 400/740 course in University of Waterloo, taught by professor Marcel Nooijen, offers students practical introductions to quantum chemical calculations. Throughout the course, students can select from Gaussian, ACES2, Orca and MOLPRO to complete their research project. Most of these programs have various functions and comprehensive input / output files, which are easy to learn and read. So, there is all kinds of difficulties to add PySCF into the CHEM 400/740 course. In order to create a user-friendly and a comprehensive multi-function highly efficient program, PySCFLab is created to fulfill the demand.

To see the overall performance for our PySCFLab program, it is necessary to compare the program with the existing quantum chemical calculation software which were used for the course. The comparison not only includes the interface, input / output files, performing standard Hartree Fock / Density Functional Theory calculations with small to large systems, but also provides a deep software analyze and the idea to make a better program.

This report also includes the basic documentation for the PySCFLab, and its source code for further research purpose. The program might not be perfect and finalized, and there might be updates for this program to add more features or instructions. Please follow the Github for more information.

BACKGROUND INFORMATION

The Python-based simulations of chemistry framework (PySCF), Orca and Budapest MRCC are three popular quantum chemistry programs. However, their functionalities and emphasises are different based on their unique designs. The following chapters will discuss and explain the main features and goals for each computational program. A neutral review would be given to each program based on their overall performance, user experience, calculation accuracy and so on.

PYTHON-BASED SIMULATIONS OF CHEMISTRY FRAMEWORK (PYSCF)

The Python-based simulations of chemistry framework (PySCF) is a collection of electronic structure programs powered by Python, based on the documentation of PySCF. (PySCF, 2019) According to Qingming Sun, the author of PySCF, this package would provide user a simple and highly efficient platform for both quantum chemistry calculations and the algorithm development. (Sun, Q, 2017)

DESIGN PRINCIPLE

The design principles for PySCF are:

- The package is completely modularized and could be easily installed, used, extended and embedded.
- The package will require minimal libraries and resources for computing.
- 90% of the package is written in python while the rest part is written in C which also is the computational hot spots.
- 90% of the package is written in functional programming while the rest part is written in objective oriented programming.

MAIN FEATURES

The features in PySCF are provided as interface to multiple computing packages, they are:

- Integral package Libcint (Sun, Q, 2015)
- DMRG CheMPS2 (Wouters, S., 2014)
- DMRG Block (G. K.-L. Chan, 2002)

- FCIQMC NECI (George Booth and Ali Alavi, 2013)
- XC functional library XCFun (Ulf Ekström, 2018)
- XC functional libriary Libxc (Susi Lehtola, 2018)
- Tensor contraction library TBLIS (Devin Matthews, 2017)
- Heat-bath Selected CI program Dice (A. A. Holmes, 2016)
- Geometry Optimizer PyBerny (Jan Hermann, 2019)

For details and documentations see package citations and references.

USER REVIEW

The PySCF package provides highly efficient calculations with minimal library and input requires, however this do sacrifices some of the performance and the user experience. (Sun, Q, 2017) It is not a beginner friendly package since it does not have an input / output file format. As a platform for testing and benchmarking, PySCF plays an important role in advanced algorithm design and efficiency testing among all other quantum chemistry packages. It is much easier to modify or embed the package to fit different kinds of needs in specific computing situations compared to others. On the other hand, extracting useful tracking information inside the PySCF package becomes difficult due to its design and purpose. When PySCF is used as a developing tool, it is always recommended to use some other complete quantum chemistry program to double check the accuracy and make sure everything is correct.

To sum up, PySCF can be a very useful development platform but not a complete and comprehensive quantum chemistry program. What's more, according to Qiming Sun, the PySCF will not be further updated and maintained, the reliability and stability could also be a problem to consider in the future. The PySCF needs to be modified if this program is going to be used in CHEM 400/740.

ORCA

ORCA is an ab initio, DFT and semi-empirical SCF-MO package developed by Frank Neese et al. at the Max Planck Institut für Kohlenforschung. (Neese, F., 2012) Unlike PySCF, the Orca program is designed and developed by the Max Planck Institute. Orca is mainly used for accurate and detailed calculations with various input size. It is a reliable program and has frequent updates. Orca run takes input files and then return the result with a detailed output files, which is very helpful.

PROGRAM COMPONENTS

Since Orca is an ab initio program, it includes almost every computing features and enable users to construct their calculation from their input files. In order to have a brief overview on what Orca could do, the following basic modules are included in the 4.2 version of Orca: (Neese, F., 2012)

- orca: Main input + driver program
- orca_anoint : Integral generation over ANOs
- orca_autoci : CI type program using the automated generation environment (ORCA-AGE)
- orca_ciprep : Preparation of data for MRCI calculations (frozen core matrices and the like)
- orca_cis : Excited states via CIS and TD-DFT
- orca_cipsi : Iterative Conguration Expansion Conguration Interaction (ICE-CI)
- orca_cpscf : Solution of the coupled-perturbed SCF equations
- orca_casscf : Main program for CASSCF driver
- orca_eprnmr : SCF approximation to EPR and NMR parameters
- orca_fci : Full-CI program
- orca_gtoint : Calculation of gaussian integrals
- orca_gstep : Relaxation of the geometry based on energies and gradients
- orca_loc : Calculation of localized molecular orbitals
- orca_md : Molecular dynamics program
- orca_mdci : Matrix driven correlation program: CI, CEPA, CPF, QCISD, CCSD(T)
- orca_mp2 : MP2 program (conventional, direct and RI)
- orca_mrci : MRCI and MRPT calculations (individually selecting)
- orca_ndoint : Calculates semiempirical integrals and gradients
- orca_numfreq : Numerical hessian computation

- orca_pc : Addition of point charge terms to the one-electron matrix
- orca_plot : Generation of orbital and density plots
- orca_pop : External program for population analysis on a given density
- orca_rel : (Quasi) Relativistic corrections
- orca_rocis : Excited states via the ROCIS method
- orca_scf : Self-consistent field program (conventional and direct)
- orca_scfgrad : Analytic derivatives of SCF energies (HF and DFT)
- orca_scfhess : Analytical hessian calculation for SCF
- orca_soc : Calculation of spin-orbit coupling matrices
- orca_vpot : Calculation of the electrostatic potential on a given molecular surface

The above functions are retrieved from The Orca documentation 4.0. For more details and documentations see package citations and references.

THE INPUT / OUTPUT FILE

Orca created an input / output file IO for running the program, which turns out to be a very efficient way to create a calculation with custom setting and fine control. The program would read the molecule geometry from the input file, as well as any important keyword like basis set or convergence type included in the input file.

For example, here is a standard input file from Orca documentation:

```
1.  ! HF def2-TZVP
2.  %scf
3.  convergence tight
4.  end
5.  * xyz 0 1
6.  C 0.0 0.0 0.0
7.  O 0.0 0.0 1.13
8.  *
```

In this case, the program would recognize that it needs to perform an Hartree Fock calculation with the CO molecule in basis set def2-TZVP with tight convergence. This is a simple calculation without many specific parameter settings and alternative selections. However, in the background the program will determine all the parameters that are needed for the calculation with default values to simplify the input files. This could help preventing unnecessary inputs and the program can choose the better algorithm automictically when running.

One more thing to considered is that the geometry of the molecule input is in xyz format, which is a very common input in many other quantum chemistry programs. Usually xyz files can be generated from Gaussian program, or the user can create their own xyz files based on experimental result.

Apart from input files, the output files from Orca is also very detailed and useful when doing calculations. Here is an example output from running an Hartree Fock calculation on ethylene (part of it):

```
1.  ============================================================================

2.                                       INPUT FILE
3.  ============================================================================

4.  NAME = C2.inp
5.  |  1> ! HF 6-31G
6.  |  2> * xyz 0 1
7.  |  3>  C                 0.00000000   -0.67759997    0.00000000
8.  |  4>  H                 0.92414473   -1.21655198    0.00000000
9.  |  5>  H                -0.92414473   -1.21655198    0.00000000
10. |  6>  C                 0.00000000    0.67759997    0.00000000
11. |  7>  H                -0.92414473    1.21655198    0.00000000
12. |  8>  H                 0.92414473    1.21655198    0.00000000
13. |  9> **                        ****END OF INPUT****
14. ============================================================================

15. --------------------------------
16. LOEWDIN REDUCED ORBITAL CHARGES
17. --------------------------------
18.   0 C s       :     3.001999  s :     3.001999
19.       pz      :     1.000000  p :     3.232974
20.       px      :     1.121699
21.       py      :     1.111274
```

```
  1 H s       :     0.882513  s :     0.882513
  2 H s       :     0.882513  s :     0.882513
  3 C s       :     3.001999  s :     3.001999
      pz      :     1.000000  p :     3.232974
      px      :     1.121699
      py      :     1.111274
  4 H s       :     0.882513  s :     0.882513
  5 H s       :     0.882513  s :     0.882513
               ****************************************************
               *                    SUCCESS                       *
               *           SCF CONVERGED AFTER   7 CYCLES         *
               ****************************************************


----------------
TOTAL SCF ENERGY
----------------

Total Energy       :          -78.00264277 Eh           -2122.55982 eV

Components:
Nuclear Repulsion  :           33.30863347 Eh             906.37400 eV
Electronic Energy  :         -111.31127625 Eh           -3028.93381 eV
One Electron Energy:         -169.85239564 Eh           -4621.91866 eV
Two Electron Energy:           58.54111939 Eh            1592.98484 eV

Virial components:
Potential Energy   :         -156.02887481 Eh           -4245.76153 eV
Kinetic Energy     :           78.02623203 Eh            2123.20172 eV
Virial Ratio       :            1.99969768
-------
TIMINGS
-------

Total SCF time: 0 days 0 hours 0 min 3 sec

Total time                  ....       3.861 sec
Sum of individual times     ....       3.120 sec  ( 80.8%)
```

```
Fock matrix formation      ....       2.283 sec  ( 59.1%)
Diagonalization            ....       0.003 sec  (  0.1%)
Density matrix formation   ....       0.000 sec  (  0.0%)
Population analysis        ....       0.009 sec  (  0.2%)
Initial guess              ....       0.718 sec  ( 18.6%)
Orbital Transformation     ....       0.000 sec  (  0.0%)
Orbital Orthonormalization ....       0.000 sec  (  0.0%)
DIIS solution              ....       0.014 sec  (  0.4%)
SOSCF solution             ....       0.023 sec  (  0.6%)

Maximum memory used throughout the entire calculation : 227.6 MB
```

The above is a small part of the actual output file that was created when running. Users are able to see which packages are used throughout the calculations as well as many property analyses during the run.

USER REVIEW

Orca is a complete program that has various functions and the most comprehensive and easy to read output files, also has the best error handling abilities. The constant update gives Orca the ability to gain a much better reliability and stability. The input / output file structure is a very efficient user interface to consider, and it works just fine with both the beginner students and experience researchers. However, it is very difficult to modify and use orca as a module for another program since Orca is a standalone program.

However, the input files can be very abstract for beginners as the documentation is very complex and takes time to read. And some of the features like EPR in Orca are still develop in progress, which is fixed later in version 4.2.1.

BUDAPEST MRCC

MRCC is a suite of ab initio and density functional quantum chemistry programs for high-accuracy electronic structure calculations developed and maintained by the quantum chemistry research group at the Department of Physical Chemistry and Materials Science, TU Budapest. (M. Kallay, 2013)

MAJOR FEATURES

The available features for MRCC are summarized in the below section:

- Single-point energy calculations

- Geometry optimizations and first-order properties
- Harmonic frequencies and second-order properties
- Higher-order properties
- Electronically excited states
- Relativistic calculations
- Reduced-scaling and local correlation calculations
- Optimization of basis sets

The above functions are retrieved from The MRCC documentation. For more details and documentations see package citations and references.

THE INPUT / OUTPUT FILE

The Budapest MRCC has a similar input / output files format just as the Orca. However, they do have some different in both the style and instructions. For example, here is an example input file for MRCC:

```
basis=6-31G
calc=HF
scfmaxit=1000
mem=8gb
dfbasis_scf=auto


unit=bohr
geom=xyz
34
 C                  0.00000000   -0.67759997    0.00000000
 H                  0.92414473   -1.21655198    0.00000000
 H                 -0.92414473   -1.21655198    0.00000000
 C                  0.00000000    0.67759997    0.00000000
 H                 -0.92414473    1.21655198    0.00000000
 C                  1.33030246    1.45341913    0.00000000
```

```
18.  H                  2.25444719    0.91446712    0.00000000
19.  C                  1.33030246    2.80861907    0.00000000
20.  H                  2.25444719    3.34757108    0.00000000
21.  C                  0.00000000    3.58443823    0.00000000
22.  H                 -0.92414473    3.04548622    0.00000000
23.  C                  0.00000000    4.93963817    0.00000000
24.  H                 -0.92414473    5.47859018    0.00000000
25.  C                  1.33030246    5.71545733    0.00000000
26.  H                  2.25444719    5.17650532    0.00000000
27.  C                  1.33030246    7.07065727    0.00000000
28.  H                  2.25444719    7.60960928    0.00000000
29.  C                  0.00000000    7.84647643    0.00000000
30.  H                 -0.92414473    7.30752442    0.00000000
31.  C                  0.00000000    9.20167637    0.00000000
32.  H                 -0.92414473    9.74062838    0.00000000
33.  C                  1.33030246    9.97749553    0.00000000
34.  H                  2.25444719    9.43854352    0.00000000
35.  C                  1.33030246   11.33269547    0.00000000
36.  H                  2.25444719   11.87164748    0.00000000
37.  C                  0.00000000   12.10851463    0.00000000
38.  H                 -0.92414473   11.56956262    0.00000000
39.  C                  0.00000000   13.46371457    0.00000000
40.  H                 -0.92414473   14.00266658    0.00000000
41.  C                  1.33030246   14.23953373    0.00000000
42.  H                  2.25444719   13.70058172    0.00000000
43.  C                  1.33030246   15.59473367    0.00000000
44.  H                  2.25444719   16.13368568   -0.00000000
45.  H                  0.40615773   16.13368568   -0.00000000
```

In the above input, the MRCC will take keyword and setting as variables. All the variables are pre-defined in the MRCC program, so the user needs to read the manual before running calculations.

The output files for MRCC is not as beautiful as Orca but they have similar outputs styles and showing intermediate results during the calculations. Some of the example output could be:

```
1.  Calculation of overlap integrals...
2.   CPU time [min]:     0.009                  Wall time [min]:     0.031
3.
4.   Calculation of the square root of the overlap matrix...
```

```
1 AOs are eliminated for irrep 1.
1 AOs are eliminated for irrep 4.
Minimum eigenvalue of the overlap matrix: 0.693656E-08
Warning! Near linear dependence in the AO basis set.
This may result in convergence problems.
CPU time [min]:     0.009                   Wall time [min]:     0.032

Calculation of kinetic energy integrals...
CPU time [min]:     0.010                   Wall time [min]:     0.033
Calculation of nuclear attraction integrals...
CPU time [min]:     0.012                   Wall time [min]:     0.035

Calculation of prescreening integrals...
CPU time [min]:     0.030                   Wall time [min]:     0.052

Calculation of two-electron integrals...
  1% done.
 14% done.
 29% done.
 40% done.
 54% done.
 71% done.
 91% done.
100% done.
CPU time [min]:     4.432                   Wall time [min]:     4.460

********************** 2019-12-09 04:20:12 **********************
Executing scf...

Allocation of 4096.0 Mbytes of memory...
=======================================================================
ITERATION STEP   1
CPU time [min]:     0.003                   Wall time [min]:     0.004

ALPHA OCC:  25   4   4  24
BETA  OCC:  25   4   4  24
***HARTREE-FOCK ENERGY IN STEP   1 IS    -576.8429509940397111 [AU]
=======================================================================
ITERATION STEP   2
```

```
CPU time [min]:     0.021                   Wall time [min]:     0.022

ALPHA OCC:  25   4   4  24
BETA  OCC:  25   4   4  24
***HARTREE-FOCK ENERGY IN STEP   2 IS    -580.4019489969591632 [AU]
================================================================================
ITERATION STEP   3
CPU time [min]:     0.036                   Wall time [min]:     0.038

ALPHA OCC:  25   4   4  24
BETA  OCC:  25   4   4  24
***HARTREE-FOCK ENERGY IN STEP   3 IS    -580.5709732964151044 [AU]
================================================================================
ITERATION STEP   4
CPU time [min]:     0.051                   Wall time [min]:     0.053

ALPHA OCC:  25   4   4  24
BETA  OCC:  25   4   4  24
***HARTREE-FOCK ENERGY IN STEP   4 IS    -580.5930690644313472 [AU]
```

MRCC shows every step in the self-consistent field method, which gives user a clear view of their running towards convergence. At the same time, the occupation number and orbital energy are printed out to see how the energy convergences as well.

USER REVIEW

The Budapest MRCC is well known for its accuracy and good tracing tools when running calculations. It is also a complete program just as Orca. Although they are very similar, the MRCC has a very poor style of input files which can be very messy if the number of required input parameters gets bigger. What's more, the output files are not managed properly to show full descriptions for calculations which is very important for beginners to learn from the program.

Unlike the PySCF, the Budapest MRCC takes larger memory but produce less effective performance. The program tends to choose better accuracy then a good efficient or a good result in a reasonable time. It is not recommended to run large systems on the MRCC. However, there is an option to turn on the density fitting feature during the calculation which will help with the efficiency. More detailed result would be given in the comparing chapter.

THE DESIGN RECIPE FOR PYSCFLAB

In order to provide the students in CHEM 400/740 with a highly efficient and easy to use program, it is necessary to develop a helper function or a new wrapper program that takes full advantages from the PySCF package. However, there are many aspects of consideration in designing the wrapper program and the user interface. Some of them are quite challenging and needs more resources to solve the problem.

The wrapper program is called the PySCFLab, and the following are its design recipe:

EASY TO USE

A good interface means user don't have to learn a lot before do anything. It is a good idea for PySCFLab to use the same style of input files as Orca, but with more documentation and explanations.

```
1.  ### This is a standard input file for PySCFLab package
2.
3.  Please select your functions: 2
4.  ### Functions: 1. Energy
5.  ###            2. Optimization
6.  ###            3. Frequency
7.  ###            4. Opt + Freq
8.  ------------------------------------------------
9.  Please select your Basis set: 3
10. ### Basis: 1. STO-3G
11. ###        2. 3-21G
12. ###        3. 6-31G
13. ###        4. 6-311G
14. ###        5. cc-pVDZ
15. ###        6. cc-pVTZ
16. ###        7. cc-pvQz
17. ###        8. LanL2DZ
18. ###        9. LanL2TZ
19. ------------------------------------------------
20. Please select your method: 1
21. ### Method: 1. Hartree-Fock
22. ###         2. Kohn-Sham
23. ###         3. Density functional theory
24. ###         4. Non-relativistic Hartree-Fock analytical nuclear gradients
25. ###         5. Non-relativistic ROHF analytical nuclear gradients
```

```
###         6. Non-relativistic unrestricted Hartree-
Fock analytical nuclear gradients
###         7. Non-relativistic restricted Kohn-Sham
###         8. Non-relativistic ROKS analytical nuclear gradients
###         9. Non-relativistic Unrestricted Kohn-Sham
###         10. Non-relativistic Unrestricted Density functional theory
```

The frustration-free documentation will help beginners to understand the program, and for experienced researchers' common selections are within-reach. The major functionalities are easy to use.

ADVANCED SETTINGS AND PARAMETERS

Advanced settings / parameters should be existing but hidden from the user just like Orca.

```
------------------------------------------------
You could modify the following settings: (now they are in default value.)

charge = 0
spin = 0
unit = 'Angstrom'
conv_tol = 1e-12 #converge threshold
conv_tol_grad = 1e-8 #gradients converge threshold
direct_scf_tol = 1e-13 #direct SCF cutoff threshold
init_guess = 'minao' #initial guess method
level_shift = 0 #Level shift (in AU) for virtual space
max_cycle = 100 #max number of iterations
max_memory = 8000 #The maximum size of cache to use (in MB)
xc = None #XC function type
atomic_radii = 'BRAGG' #1D array
becke_scheme = 'BECKE' #weight partition function
prune = 'NWCHEM' #scheme to reduce number of grids
radi_method = 'TREUTLER_AHLRICHS' #scheme for radial grids
radii_adjust = 'TREUTLER' #Function to adjust atomic radii
```

All parameters from PySCF documentation are ready to be override from input files. Input file can be blank without advanced settings, but output file will tell user which setting they are using for their calculation.

ERROR HANDLING

A good program should provide useful information if the calculation goes wrong. The PySCFLab is built on many layers of error handling so the program will check the input before running the calculation.

```
1.   # set KS attributes
2.  if DFT:
3.
4.      mf.xc = xc
5.      if mf.xc is None:
6.          print( "The input XC is None!")
7.      mf.nlc = nlc
8.
9.      if isinstance(xc_grid, int):
10.         mf.grids.level = xc_grid
11.     elif isinstance(xc_grid, tuple) or isinstance(xc_grid, dict):
12.         mf.grids.atom_grid = xc_grid
13.     else:
14.         print( "The xc_grid is None!")
15.
16.     if isinstance(nlc_grid, int):
17.         mf.nlcgrids.level = nlc_grid
18.     elif isinstance(nlc_grid, tuple) or isinstance(nlc_grid, dict):
19.         mf.nlcgrids.atom_grid = nlc_grid
20.     else:
21.         print( "The Nlc_grid is None!")
22.
23.     if atomic_radii == 'BRAGG':
24.         mf.grids.atomic_radii = dft.radi.BRAGG_RADII
25.         mf.nlcgrids.atomic_radii = dft.radi.BRAGG_RADII
26.     elif atomic_radii == 'COVALENT':
27.         mf.grids.atomic_radii = dft.radi.COVALENT_RADII
28.         mf.nlcgrids.atomic_radii = dft.radi.COVALENT_RADII
29.     else:
30.         print( "Atomic_radii not found!")
31.
32.     if becke_scheme == 'BECKE':
33.         mf.grids.becke_scheme = dft.gen_grid.original_becke
34.         mf.nlcgrids.becke_scheme = dft.gen_grid.original_becke
35.     elif becke_scheme == 'STRATMANN':
36.         mf.grids.becke_scheme = dft.gen_grid.stratmann
```

```
        mf.nlcgrids.becke_scheme = dft.gen_grid.stratmann
    else:
        print( "The Becke_scheme not Found!")
```

CALCULATION TRACKING

The output files should provide useful tracking information when running the calculation.

```
 # properties
    if prop:
        output_file.write("----------------------------------------
\nProperty Analysis\n")
        # dipole moment
        mf.dip_moment()

        # S^2
        if scf_type == 'U':
            (ssq, mult) = mf.spin_square()
            print( '(S^2, 2S+1):', (ssq, mult))
        elif scf_type in ['R', 'RO']:
            S = float(mol.spin)/2.
            (ssq, mult) = (S*(S+1.), 2.*S+1.)
            print( '(S^2, 2S+1):', (ssq, mult))
        else:
            print( "Property analysis failed!")

        str_buffer = '\n(S^2, 2S+1):'+  "(" + str(ssq) + ", " +  str(mult)  + ")
\n"
        output_file.write(str_buffer)

        # population analysis
        print( "Mulliken population analysis")
        output_file.write("\nMulliken population analysis\n")
        for ia in range(mol.natm):
            symb = mol.atom_symbol(ia)
            print( symb, ':', mf.mulliken_meta(verbose=verbose)[1][ia])
            output_file.write(str(symb) + ":" + str(mf.mulliken_meta(verbose=ver
bose)[1][ia]) + "\n")

```

```
        print( "Mulliken population analysis, based on meta-Lowdin AOs")
        output_file.write("\nMulliken population analysis, based on meta-Lowdin AOs\n")
        for ia in range(mol.natm):
            symb = mol.atom_symbol(ia)
            print( symb, ':', mf.mulliken_pop(verbose=verbose)[1][ia])
            output_file.write(str(symb) + ":" + str(mf.mulliken_pop(verbose=verbose)[1][ia]) + "\n")
```

In the above functions, the PySCFLab would provide molecule properties and population analysis (Mulliken) when running energy calculations.

```
# stability analysis (optional)
if stable:
    output_file.write("-------------------------------------\nStability Analysis\n")
    str_buffer = ""
    print( 'Initial SCF energy: ', mf.e_tot)
    print( 'Performing Stability Analysis (up to ', stable_cyc, 'iterations)')
    str_buffer = 'Initial SCF energy: ' + str(mf.e_tot) + "\n" + 'Performing Stability Analysis (up to ' + str(stable_cyc) + 'iterations)\n'
    output_file.write(str_buffer)
    str_buffer = ""
    for i in range(stable_cyc):
        new_mo_coeff = mf.stability(internal=True, external=False)[0]
        if numpy.linalg.norm(numpy.array(new_mo_coeff) - numpy.array(mf.mo_coeff)) < 10**-14:
            print( "Stable!")
            output_file.write("Stable!\n")
            break
        else:
            print( "Unstable!")
            output_file.write("Unstable!\n")
            if scf_type == 'U':
                n_alpha = numpy.count_nonzero(mf.mo_occ[0])
                n_beta = numpy.count_nonzero(mf.mo_occ[1])
                P_alpha = numpy.dot(new_mo_coeff[0][:, :n_alpha], new_mo_coeff[0].T[:n_alpha])
```

```
                P_beta = numpy.dot(new_mo_coeff[1][:, :n_beta], new_mo_coeff[1].T[:n_beta])
                mf.kernel(dm0=(P_alpha, P_beta))
            elif scf_type in ['R', 'RO']:
                n_alpha = numpy.count_nonzero(mf.mo_occ)
                P_alpha = 2*numpy.dot(new_mo_coeff[:, :n_alpha], new_mo_coeff.T[:n_alpha])
                mf.kernel(dm0=(P_alpha))
            else:
                print( "Stability analysis failed!")
            print( 'Updated SCF energy: ', mf.e_tot)
            output_file.write("Updated SCF energy: " + str(mf.e_tot) + "\n")
```

In the above functions, the PySCFLab would provide stability analysis when running energy calculations and produce warning when stability analysis failed.

GOOD DEFAULT VALUES

It is important to have good default values in order to reduce the input and increase the efficient. For PySCFLab, most default values are retrieved from PySCF libraries, and some of the parameters are modified for CHEM 400/740 specifically. The user can also change them using the input file.

HIGHLY EFFICIENCY AND GOOD PRECISION

High efficiency achieved by better algorithm, better multicore performance and better output information.

High precision can be obtained within reasonable time.

OTHER FEATURES

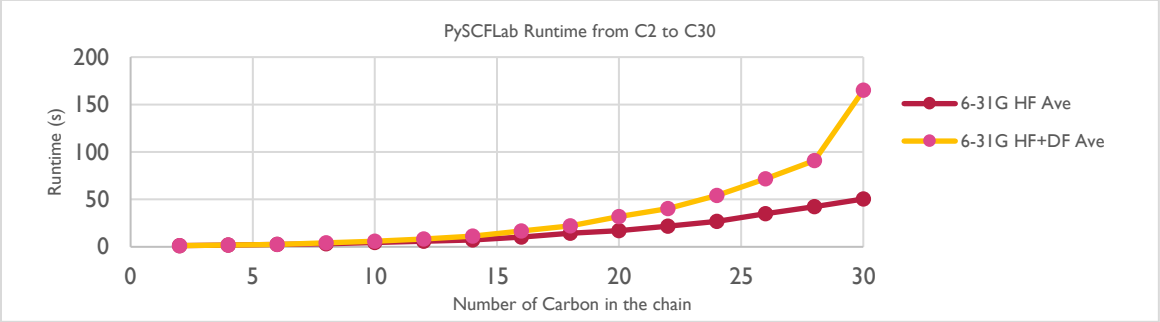
The PySCFLab can also read from files and save into files. The geometry can be provided from standard XYZ files, and the geometry optimization can be saved into XYZ files as well.

Run_batch function enables the user to run multiple files at once and store them into a pandas database structure for further analysis.

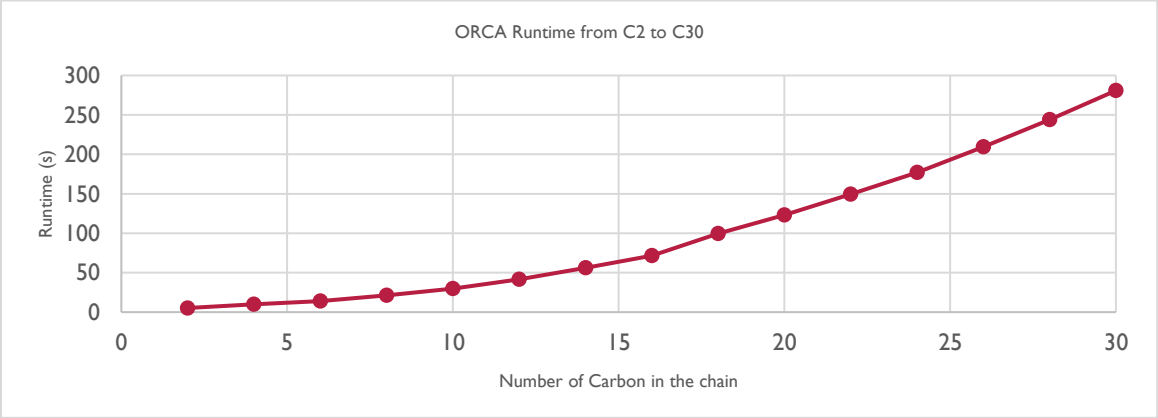
THE COMPARISION

In order to compare the efficiency and the accuracy between the PySCFLab, Orca and Budapest MRCC, a simple polyethylene model is used to determine the result. Two of the major calculation types are used.

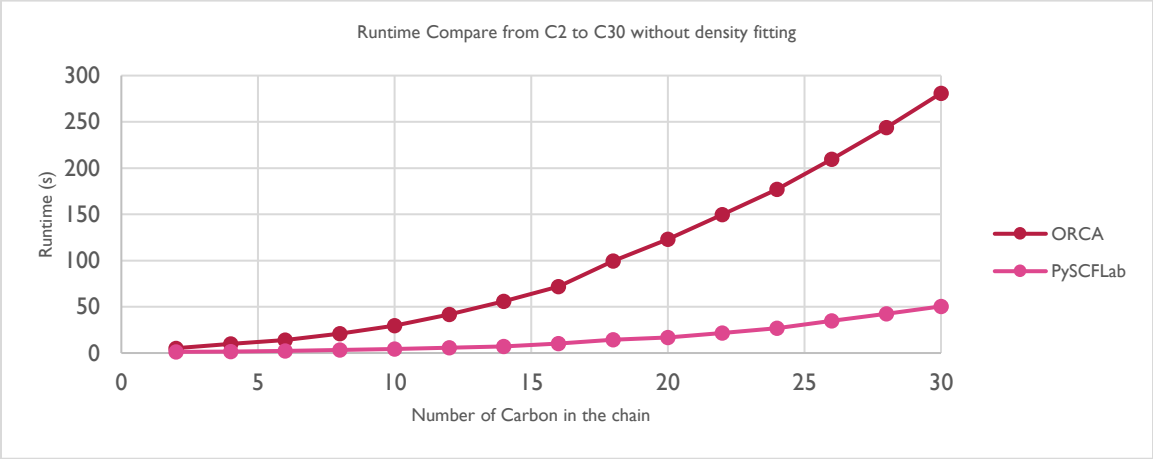
HARTREE FOCK CALCULATIONS



In the figure above, the Hartree Fock calculation for polyethylene are very efficient in PySCFLab, the runtime for a large system like C30 only takes around a minute. However, an interesting finding is that it took the PySCFLab longer time to compute if the density fitting is enabled. Normally, enabling density fitting would reduce the runtime with little sacrifices in accuracy. In this case, the PySCF DFT module is written in C and during each cycle in the SCF iteration, the data need to be transferred between Python and C many times. And that cause the Runtime penalty.

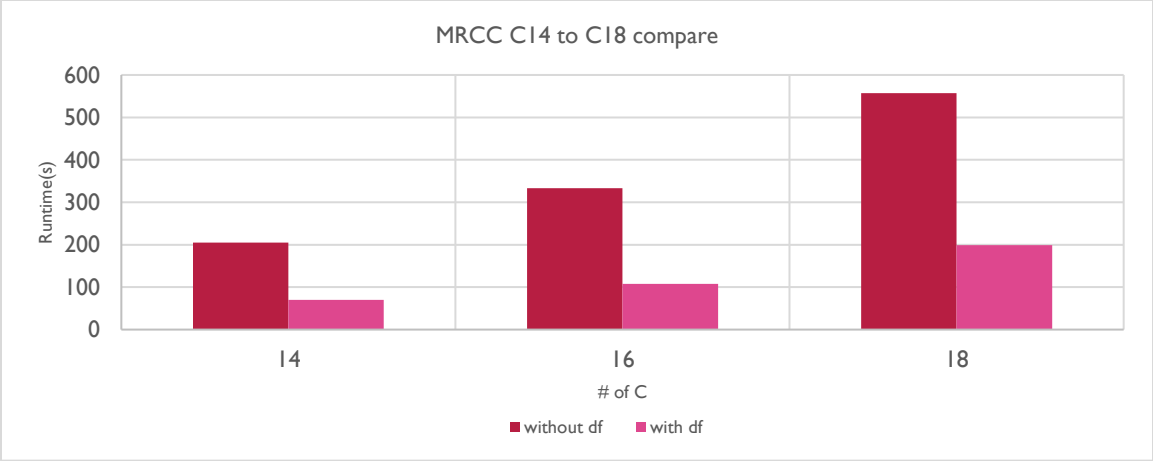


In the figure above, the Orca program produces a very system-size dependent result. However, Orca is very slow compared to PySCFLab.



The result can be very obvious that PySCFLab can be as much as three times faster than the Orca program while having the same accuracy as 10^-7 hartree.

CALCULATIONS WITH DENSITY FITTINGS



Since MRCC performs the most accuracy result, the density fitting will have the most impact on it. The above figure shows that by enabling the density fitting in calculations, the runtime can be greatly reduced. What's more, the density fitting would greatly reduce the iteration steps in the self-consistent field method. For C14, 69 steps reduced to 17 steps; C16, 73 steps to 19 steps; C18, 163 steps to 17 steps.

CONCLUSION AND FUTURE WORK

The PySCFLab could be a very useful and efficient program for future CHEM 400 course.

The compare between different programs are not that useful, however the compare between different method / algorithm can provide many helpful information.

Density fitting can reduce the runtime greatly without having huge errors.

FUTURE WORK

- Complete the PySCFLab Documentation.
- Improve with more graphic and nicer interface.
- Come up with more tests to do more benchmarking and compare.

CODE

THE PYSCFLAB

The source file is too big to show here, so only the name for major functions will be displayed below:

- Read input
- Main function run
- Run_batch function
- Parse_kwargs function
- Read_molecule function
- XYZtoMF function
- Check function selection function
- Check input error function
- Stability analysis function
- Properties analysis function
- Geometry optimization function
- Thermochemistry analysis function
- Density fitting module

THERMOCHEMISTRY ANALYSIS PACKAGE

- Harmonic_analysis function
- Rotation_const function
- Thermo function
- Rotational_symmetry_number function

- Dump_thermo function
- Dump_normal_mode function

BERNY SOLVER MODULE

- To_berny_gemom function
- Geom_to_atom function
- To_berny_log function
- Kernel function
- Optimize function
- GeometryOptimizer datatype

THE PYSCFLAB INPUT FILE EXAMPLE

```
### This is a standard input file for PySCFLab package

Please select your functions: 2
### Functions: 1. Energy
###            2. Optimization
###            3. Frequency
###            4. Opt + Freq
-----------------------------------------------
Please select your Basis set: 3
### Basis: 1. STO-3G
###        2. 3-21G
###        3. 6-31G
###        4. 6-311G
###        5. cc-pVDZ
###        6. cc-pVTZ
###        7. cc-pvQz
###        8. LanL2DZ
###        9. LanL2TZ
-----------------------------------------------
Please select your method: 1
### Method: 1. Hartree-Fock
```

```
###          2. Kohn-Sham
###          3. Density functional theory
###          4. Non-relativistic Hartree-Fock analytical nuclear gradients
###          5. Non-relativistic ROHF analytical nuclear gradients
###          6. Non-relativistic unrestricted Hartree-
Fock analytical nuclear gradients
###          7. Non-relativistic restricted Kohn-Sham
###          8. Non-relativistic ROKS analytical nuclear gradients
###          9. Non-relativistic Unrestricted Kohn-Sham
###          10. Non-relativistic Unrestricted Density functional theory
-----------------------------------------------
Please provide the gerometry for the molecule:
### xyz format here

-----------------------------------------------
You could modify the following settings: (now they are in default value.)

charge = 0
spin = 0
unit = 'Angstrom'
conv_tol = 1e-12 #converge threshold
conv_tol_grad = 1e-8 #gradients converge threshold
direct_scf_tol = 1e-13 #direct SCF cutoff threshold
init_guess = 'minao' #initial guess method
level_shift = 0 #Level shift (in AU) for virtual space
max_cycle = 100 #max number of iterations
max_memory = 8000 #The maximum size of cache to use (in MB)
xc = None #XC function type
atomic_radii = 'BRAGG' #1D array
becke_scheme = 'BECKE' #weight partition function
prune = 'NWCHEM' #scheme to reduce number of grids
radi_method = 'TREUTLER_AHLRICHS' #scheme for radial grids
radii_adjust = 'TREUTLER' #Function to adjust atomic radii
```

THE PYSCFLAB OUTPUT FILE EXAMPLE

```
Welcome to PySCFLab!
This is your output file, your input file is : pyinput.inp
---------------------------------------
```

```
Function selected: Energy
Method used: HF
Basis sets: 6-311G
Input geometry:
  C                 -3.06962441   -2.50255464    0.03676511
  H                 -2.93892716   -3.57547251    0.09430092
  N                 -4.25004605   -1.92264577    0.24832970
  C                 -3.96116944   -0.58897139    0.07801872
  C                 -4.71339842    0.59449589    0.14251559
  N                 -6.03447842    0.60180134    0.46428333
  H                 -6.55114546    1.45596179    0.33331944
  H                 -6.53000166   -0.27439489    0.48934631
  N                 -4.10824095    1.77551565   -0.09926335
  C                 -2.80216977    1.78392688   -0.38472796
  H                 -2.36665360    2.76196554   -0.56870010
  N                 -1.97654078    0.73788815   -0.47147517
  C                 -2.60132224   -0.44735494   -0.23272677
  N                 -2.01798464   -1.67962011   -0.25980776
  Na                -0.00882900    0.01078800   -0.07043700
  C                  2.56454629    2.26823575    0.11956519
  H                  2.20850081    3.28980844    0.15370558
  N                  3.85600180    1.97497660   -0.02393254
  C                  3.85158673    0.59982085   -0.01561363
  C                  4.84499418   -0.38579922   -0.12651301
  N                  6.16478471   -0.08046408   -0.24382222
  H                  6.80456631   -0.82004548   -0.48337344
  H                  6.42464139    0.87454497   -0.42909009
  N                  4.49078183   -1.68692563   -0.09356980
  C                  3.19939432   -1.99976212    0.05418858
  H                  2.97162463   -3.06170971    0.07587318
  N                  2.16376655   -1.16540135    0.17543944
  C                  2.53423594    0.14345846    0.13311592
  N                  1.69675244    1.21588763    0.22262309
Charge = -1 Spin = 0
---------------------------------------
Running log:
---------------------------------------
Stability Analysis
Initial SCF energy: -1089.588742367645
```

```
Performing Stability Analysis (up to 3iterations)
Stable!
---------------------------------------
Property Analysis

(S^2, 2S+1):(0.0, 1.0)

Mulliken population analysis
C:0.03632961141934299
H:0.12686039468206156
N:-0.4021040813755423
C:-0.07426487302379492
C:0.27718736501042596
N:-0.4497022985591048
H:0.2386803318735331
H:0.24791949303070826
N:-0.4114529320189817
C:0.10430543329057418
H:0.1358723771313668
N:-0.39703878147014215
C:0.13410791375089826
N:-0.4345372296925518
Na:0.7240999733075117
C:0.04261655806375231
H:0.12846402627460485
N:-0.39489328637099064
C:-0.07181123025498337
C:0.27478131147200546
N:-0.4505612241296282
H:0.2392587470575599
H:0.24736344900371454
N:-0.4085269883761722
C:0.09886734558312504
H:0.13765205485571474
N:-0.36991590209403036
C:0.1371849879528284
N:-0.46674254639381907

Mulliken population analysis, based on meta-Lowdin AOs
```

```
C:0.17492087498329667
H:0.16211552717733446
N:-0.5615100152348402
C:-0.01373695469175118
C:0.7777788809173591
N:-0.9597250288907242
H:0.33159429499743387
H:0.35509949725188905
N:-0.6154304505311252
C:0.24105005974271343
H:0.16788217278267936
N:-0.7319110631609673
C:0.40573680517787825
N:-0.6408569867179246
Na:0.8106047611179505
C:0.20237466930376247
H:0.1685117177132054
N:-0.5629296984376069
C:-0.006548887296866646
C:0.7776853855794945
N:-0.9607571949401956
H:0.3324030882274983
H:0.3547055171519281
N:-0.6083886523348792
C:0.2113408870489284
H:0.16760481709395036
N:-0.6085136374430036
C:0.4178280762547768
N:-0.788928462842204

Total Run Time: 378.39197879999983
```

ACKNOWLEDGEMENT

I would like to express my very great appreciation to professor Marcel Nooijen for being my supervisor. His wisdom and knowledge make the PySCFLab possible and could be a very useful package for both beginners and advanced developers.

I would like to thank the PhD student Mike Lecours, he helped me a lot with the use of pyscf and numpy packages and the hessian functions. Special thanks for him enduring my keyboard noise when I am creating bugs or debugging besides him.

I would like to thank the PhD student Dmitri Iouchtchenko for providing me private cpu nodes for calculation benchmark.

I would like to thank the PhD student Neil George Raymond for kindly providing help with github.

I would like to offer my thanks to the PhD student Songhao Bao, he helped me with understand many tough and complicated theories, which are extremely useful and helpful for me. He also helped me with the use of Gaussian program.

Finally, I wish to thank my parents and Erqian Gao for their support and encouragement throughout this term.

REFERENCES

A. A. Holmes, N. M. Tubman, C. J. Umrigar, "Heat-Bath Configuration Interaction: An Efficient Selected Configuration Interaction Algorithm Inspired by Heat-Bath Sampling." Journal of Chemical Theory and Computations, 2016, 12, 3674.

D. Ghosh, J. Hachmann, T. Yanai, and G. K.-L. Chan, J. Chem. Phys., 128, 144117 (2008),

Devin Matthews, tblis, 2015 – 2017. Retrieved from https://github.com/devinamatthews/tblis. Retrieved in Dec 2019.

George Booth and Ali Alavi, FCIQMC, 2013. Retrieved from https://github.com/ghb24/NECI_STABLE, retrieved in Dec 2019.

G. K.-L. Chan and M. Head-Gordon, J. Chem. Phys. 116, 4462 (2002),

G. K.-L. Chan, J. Chem. Phys. 120, 3172 (2004),

Jan Hermann, PyBerny, 2017 – 2019. Retrieved from https://jan.hermann.name/pyberny/. Retrieved in Dec 2019.

Marcel Nooijen, CHEM 400/740: Introduction to Computational Quantum Chemistry Course description, retrieved from http://scienide2.uwaterloo.ca/~nooijen/website_new_20_10_2011/440W.html, retrieved in Dec, 2019.

Mrcc, a quantum chemical program suite written by M. Kallay, Z. Rolik, J. Csontos, I. Ladjanszki, L. Szegedy, B. Ladoczki, and G. Samu. See also Z. Rolik, L. Szegedy, I. Ladjanszki, B. Ladoczki, and M. Kallay, J. Chem. Phys. 139, 094105 (2013), as well as: www.mrcc.hu.

Neese, F. (2012) The ORCA program system, Wiley Interdiscip. Rev.: Comput. Mol. Sci., 2, 73 - 78.

PySCF 1.7 documentation, retrieved from https://sunqm.github.io/pyscf/, retrieved in Dec, 2019.

S. Guo, M. A. Watson, W. Hu, Q. Sun, G. K.-L. Chan, J. Chem. Theory Comput. 12, 1583 (2016)

Susi Lehtola, Conrad Steigemann, Micael J.T. Oliveira, and Miguel A.L. Marques, Recent developments in Libxc - A comprehensive library of functionals for density functional theory, Software X 7, 1 (2018)

Sun, Q. (2015). Libcint: An efficient general integral library for Gaussian basis functions. Journal of Computational Chemistry, 36(22). Retrieved from http://search.proquest.com/docview/1697792921/

Sun, Q., Berkelbach, T., Blunt, N., Booth, G., Guo, S., Li, Z., ⋯ Chan, G. (2018). PySCF: the Python-based simulations of chemistry framework. Wiley Interdisciplinary Reviews: Computational Molecular Science, 8(1), n/a–n/a. https://doi.org/10.1002/wcms.1340

S. Sharma and G. K-.L. Chan, J. Chem. Phys. 136, 124121 (2012).

S. Sharma, A. A. Holmes, G. Jeanmairet, A. Alavi, C. J. Umrigar, "Semistochastic Heat-bath Configuration Interaction method: selected configuration interaction with semistochastic perturbation theory." Journal of Chemical Theory and Computations, 2017, 13, 1595.

Ulf Ekström, XCFun, 2009 – 2018, retrieved from https://github.com/dftlibs/xcfun, retrieved in Dec 2019.

Wouters, S., Poelmans, W., Ayers, P., & Van Neck, D. (2014). CheMPS2: A free open-source spin-adapted implementation of the density matrix renormalization group for ab initio quantum chemistry. Computer Physics Communications, 185(6), 1501–1514. https://doi.org/10.1016/j.cpc.2014.01.019