Pyflosic: FLO-SIC powered by PySCF

Lenz Fiedler / Sebastian Schwalbe

August 10, 2018

Contents

Introduction

Pyflosic is a PySCF [1] based implementation of the FLO-SIC method [2–9]. The FLO-SIC method has been an useful improvement to the DFT methodology and can be used to obtain a correctly localized density, corrected total energy values and corrected energy eigenvalues. It thus allows for an excellent computational treatment of electronic structure problems. Pyflosic aims to extend the scope of FLO-SIC set in comparison to its reference implementation in NRLMOL [10–17]. This is done by drawing on PySCF's many features, including multiple basis sets, a wide variety of exchange-correlation functionals and periodic boundary conditions. The following document serves as an user reference for Pyflosic. It is divided into four parts. The first part provides information on how Pyflosic is installed. The second part deals with FLO-SIC calculations based upon fixed FOD geometries. This part shows how to perform basic as well as advanced FLO-SIC calculations and guides the reader through all necessary steps. It is based upon Ref. [18], where the implementation process as well as all necessary equations are explained in detail. Figures and examples provided within this document stem from this publication as well. The third part of this document explains how FOD geometry optimizations can be done with Pyflosic. After such an optimization has been done, one may then go back to the second part of this documentation to properly analyze the results. The fourth part explains how Fermi-orbital guesses can be generated automatically. It should be noted that this documentation does not imply ownership over software produced at the Institute of Theoretical Physics.

Chapter 1

Setting up Pyflosic

1.1 Pyflosic files

Before you start the installation, please make sure the following files are correctly included in your `pyflosic` directory:

1. The Pyflosic manual (this document) for any issue surrounding installation or usage of this software.
2. README: Contains all the important information needed to start Pyflosic.
3. INSTALL: Installation guide.
4. AUTHORS: The authors of Pyflosic along with contact information.
5. VERSION: The version of Pyflosic you have acquired.
6. src/: Contains the source files for Pyflosic.
7. examples/: Contains useful examples to get started with Pyflosic. Also shows more advanced applications of Pyflosic.
8. utils/: Utilities that are useful for Pyflosic.

If all of these files and directories are correctly in place, you can commence the installation process.

1.2 Installation

The installation process is twofold: First you need to install PySCF on your computer, afterwards you can proceed to install Pyflosic. If you already have a working PySCF installation on your computer, please refer directly to Sec. 1.2.2.
Requirements:

- Python 2.7
- ASE 3.13.0
- NumPy / SciPy
- Matplotlib (optional)

1.2.1 PySCF

This section gives an overview on how to install PySCF as a requirement for Pyflosic. It is based upon the official PySCF documentation (`https://sunqm.github.io/pyscf/install.html`). For any problems during the installlation of PySCF, please refer to the official PySCF documentation. The overview in this documentation only deals with Linux systems (Debian, Ubuntu, OpenSUSE) for support for Mac OS and / or Windows please refer to PySCF's official documentation.
Debian / Ubuntu

1. Install the libcint library.

```
git clone https://github.com/sunqm/libcint.git
cd libcint
git checkout origin/cint3
cd .. && tar czf libcint.tar.gz libcint
tar xvzf libcint.tar.gz
cd libcint
mkdir build && cd build
cmake -DWITH_F12=1 -DWITH_RANGE_COULOMB=1 -DWITH_COULOMB_ERF=1
    \
    -DCMAKE_INSTALL_PREFIX:PATH=/path/to/your/library/folder -
       DCMAKE_INSTALL_LIBDIR:PATH=lib ..
make && make install
```

2. Install the xcfun library.

```
tar xvzf libxc-4.0.4.tar.gz
cd libxc-4.0.4
mkdir build && cd build
../configure --prefix=/path/to/your/library/folder  --libdir=/
   path/to/your/library/folder/lib --enable-shared --disable-
   fortran LIBS=-lm
make && make install
```

3. Install the libxc library (IMPORTANT: Version 4.0.0 or higher, or else the SCAN functional will not work).

```
git clone https://github.com/sunqm/xcfun.git
cd xcfun
git checkout origin/stable-1.x # If you are experiencing
   trouble with loading stable-1.x this way, clone https://
   github.com/dftlibs/xcfun.git/tree/stable-1.x instead. In
   this case, no further checkout is needed.
cd .. && tar czf xcfun.tar.gz xcfun
tar xvzf xcfun.tar.gz
cd xcfun
mkdir build && cd build
cmake -DCMAKE_BUILD_TYPE=RELEASE -DBUILD_SHARED_LIBS=1 -
   DXC_MAX_ORDER=3 -DXCFUN_ENABLE_TESTS=0 \
    -DCMAKE_INSTALL_PREFIX:PATH=/path/to/your/library/folder
       -DCMAKE_INSTALL_LIBDIR:PATH=lib ..
make && make install
```

4. Install PySCF.

```
cd pyscf/pyscf/lib
mkdir build && cd build
cmake -DBUILD_LIBCINT=0 -DBUILD_LIBXC=0 -DBUILD_XCFUN=0 -
   DCMAKE_INSTALL_PREFIX:PATH=/path/to/your/library/folder/ ..
make
```

OpenSUSE

1. Install all necessary libraries using e.g. the YaST Softwaremanager (libcint, xcfun, libxc and the required Python packages mentioned above)

2. Install PySCF.

```
git clone https://github.com/sunqm/pyscf.git
cd pyscf/lib
  mkdir build; cd build
  cmake ..
  make
```

1.2.2 Pyflosic

1. Copy the content of pyflosic/src to an arbitrary directory dir.

2. Update the PYTHONPATH variable on your system with export `PYTHONPATH=/path/to/dir/:$PYTHONPATH`.

Chapter 2

Fixed FOD geometry

2.1 Important Routines

There are three main routines one can access to start FLO-SIC calculations. They are listed below together with their parameters and specifications.

`calculate_flosic`: This routine in `flosic_os.py` performs a DFT calculation and then applies FLO-SIC to the results of this DFT calculation (Full one-shot FLO-SIC).

Parameter	Purpose	Default value
`ase_atoms` and `fname`	Specifies the system that is calculated, either by giving the filename `fname` of a `fname.xyz` file or by giving an ASE `Atoms` object.	**Required Parameter.**
`spin`	Spin polarization of the system.	`None`
`charge`	Charge of the system.	0
`xc`	Exchange-correlation functional that is used.	`'LDA,PW'`
`basis`	Basis set used during the calculation.	`'6-311++Gss'`
`verbose`	Output level of PySCF and Pyflosic; 0 is the lowest, 4 is the highest.	0
`max_cycle`	Maximum number of iterations done in the self-consistency cycle of the DFT calculation.	300
`conv_tol`	Convergence tolerance of the self-consistency cycle of the DFT calculation.	10^{-5}
`grid`	Numerical grid level (Integer in the range of 1 to 9).	3
`debug`	Enables full debugging output if set to `True`.	`False`
`ghost`	If set to `True`, *Ghost* atoms (nuclei without charge) are set to the FOD positions. Helpful for e.g. debugging purposes.	`False`

Example:

```
b = '6-311++Gss' # Basis set.
verbose = 4 # Amount of output.
max_cycle = 300 # Number of SCF iterations.
conv_tol = 1e-7 # Accuracy of the SCF cycle.
grids_level = 3 # Level of the numerical grid.
xc = 'LDA,PW' # Exchange-correlation functional.
sysname = 'H2' # Filename specifier.
flosic_values = calculate_flosic(spin=0,fname=sysname,basis=b,
   verbose=verbose,max_cycle=max_cycle,conv_tol=conv_tol,grid=
   grids_level,xc=xc) # Get the FLO-SIC values.
```

flosic: This routine in `flosic_os.py` performs FLO-SIC on a DFT calculation that is given as a parameter in the form of an `UKS` class object. This is helpful when performing FLO-SIC with highly customized DFT calculations (Post-processing one-shot FLO-SIC).

Parameter	Purpose	Default value
`mol`	Molecular geometry.	**Required Parameter.**
`mf`	DFT calculator object from the preceding DFT calculation.	**Required Parameter.**
`fod1` / `fod2`	FOD geometry for both spin channels. The FOD geometry has to be given as an ASE `Atoms` object.	**Required Parameter.**
`datatype`	Data type used for the FLOs. Determines whether or not complex orbitals are used, although the usage of real orbitals is **highly** advised.	`numpy.float64`
`print_dm_one` / `print_dm_all`	Determines whether or not the density matrices created in the process of calculating the SIC are printed on the screen. This can help debugging.	`False`
`debug`	Enables full debugging output if set to `True`.	`False`

Example:

```
from flosic import ase2pyscf # Routine that parses ASE data to
    PySCF format.
mol = gto.M(atom=ase2pyscf(nuclei), basis={'default':'6-311++
   Gss'},spin=0)  # Build the mol object.
dft_object = dft.UKS(mol) # Build the DFT object.
dft_object.kernel() # Perform a DFT calculation.
flosic_values = flosic(mol,dft_object,fod1,fod2) # Correct the
    DFT results with FLO-SIC.
```

FLOSIC: This function from `flosic_scf.py` creates an instance of the `SIC` class when called with `sic_object = FLOSIC(parameters)`. This instance can then be used for self-consistent FLO-SIC calculations.

Parameter	Purpose	Default value
`mol`	Molecular geometry.	**Required Parameter.**
`xc`	Exchange-correlation functional that is used.	**Required Parameter.**
`fod1` / `fod2`	FOD geometry for both spin channels. The FOD geometry has to be given as an ASE `Atoms` object.	**Required Parameter.**
`grid_level`	Numerical grid level (Integer in the range of 1 to 9). Note that for regular `UKS` classes this can be specified at any point *after* the initialization. Due to the nature of the `SIC`class the grid level has to be specified during the initialization.	3
`calc_forces`	If set to `True`, the forces needed for the FOD geometry optimization are calculated at every step of the self-consistency cycle. This is computationally very demanding.	`False`
`ldax`	If set to `True`, LDA exchange is used to calculate the FLO-SIC energy, no matter which functional is specified in `xc`.	`False`
DFT parameters	`SIC` class objects can be customized with the same attributes regular `UKS` objects can. The only exception is the numerical grid level, as detailed above. Other parameters modifying the calculation include the number of self-consistency iterations, self-consistency tolerance, etc. See the PySCF documentation for the `UKS` class for a full listing.	various

Example:

```
from flosic import ase2pyscf # Routine that parses ASE data to
    PySCF format.
mol = gto.M(atom=ase2pyscf(nuclei), basis={'default':'6-311++
   Gss'},spin=0)  # Build the mol object.
sic_object = SIC(mol,xc=xc,fod1=fod1,fod2=fod2,grid_level=
   grid_level) # Build the SIC object.
total_energy_sic = sic_object.kernel() # Perform a FLO-SIC SCF
    calculation.
```

2.2 Input

The principal input for all FLO-SIC calculations is a `.xyz` file that contains the molecular geometry as well as the FOD geometry. Note that FODs for the first spin channel have to be given with the chemical symbol `X` and FODs for the second spin channel have to be given with the chemical symbol `He`. Depending on how you access the FLO-SIC calculation you might have to process the contents of the `.xyz` file before starting the calculation. Furthermore, there is a variety of optional parameters to customize FLO-SIC calculations, with the most notable modification option being the choice of the exchange-correlation functional. **Example (`CH3Cl.xyz`)**:

```
31
FOD positions (X/He) together with structure atoms.
C +0.00000000 +0.00000000 -1.12139000
Cl +0.00000000 +0.00000000 +0.65595000
H +0.00000000 +1.02932000 -1.47428000
H +0.89141000 -0.51466000 -1.47428000
H -0.89141000 -0.51466000 -1.47428000
X -0.00126248 +0.00267783 -1.13480091
X +0.00008686 +0.00016228 +0.65620656
X -0.00002433 +0.95944028 -1.22054612
X +0.81825324 -0.48704110 -1.22580810
X -0.82097015 -0.48837555 -1.23848469
X -0.19280216 +0.09961802 +0.63556257
X +0.66741048 -0.39590329 +0.88505143
X +0.03951951 +0.73583271 +0.91160610
X +0.00652497 -0.18955993 +0.54446966
X -0.70623325 -0.30550904 +0.87376701
X +0.16618766 +0.11684703 +0.57168764
X +0.02247387 -0.02871654 +0.87186834
X -0.01012818 -0.00154272 -0.31748419
He -0.00411959 -0.00719297 -0.33651762
He -0.00016998 +0.00328755 -1.13957428
He -0.00007050 +0.00004904 +0.65567884
He +0.00379557 +0.95320413 -1.21715926
He +0.81838622 -0.48138761 -1.23481921
He -0.81405863 -0.48286657 -1.21807084
He +0.13224695 +0.17417227 +0.66173718
He +0.77597373 +0.06863124 +0.89543735
He -0.37581106 +0.62823535 +0.87865330
He +0.11533385 -0.15550066 +0.55392524
He -0.40669220 -0.66235194 +0.88360771
He -0.06029611 -0.02532107 +0.86576773
He -0.18804304 +0.00585714 +0.54411519
```

2.3 Starting a FLO-SIC calculation

- One-Shot mode with standard DFT call:
  1. Import `calculate_flosic` from `flosic_os.py`.
  2. Provide either a `fname.xyz` file or an ASE `Atoms` object.
  3. Call `calculate_flosic` with `fname` or the `Atoms` object.
     - Provide additional parameters if wished.
- One-Shot mode with customized DFT call (useful for binding energy curves, multiple DFT calculations, etc.):
  1. Import `flosic` from `flosic_os.py`.
  2. Create `nuclei`, `fod1` and `fod2` with `xyz_to_nuclei_fod` from `flosic_os.py`.
  3. Customized DFT call, e.g.
     - Import `gto, dft` from `pyscf`.
     - Specify `spin` and `charge` of the system.
     - Choose a basis set.
     - Create a `mol` object with `ase2pyscf(nuclei)` from `flosic_os.py` and `mol = gto.M(atom=ase2pyscf(nuclei), basis={'default':b},spin=spin,charge=charge)`.
     - Create an `UKS` class object with `dft_object = dft.UKS(mol)`.
     - Customize and run your DFT calculation (see PySCF documentation for further information).
  4. Call `flosic` with `flosic_values = flosic(mol,dft_object,fod1,fod2)`.
- Self-consistent mode:
  1. Import `FLOSIC` from `flosic_scf.py`.
  2. Import `gto` from `pyscf`.
  3. Create `nuclei`, `fod1` and `fod2` with `xyz_to_nuclei_fod` from `flosic_os.py`.
  4. Specify `spin` and `charge` of the system.
  5. Choose a basis set.
  6. Choose an exchange-correlation functional.
  7. Create a `mol` object using `ase2pyscf(nuclei)` from `flosic_os.py` and `mol = gto.M(atom=ase2pyscf(nuclei), basis={'default':b},spin=spin,charge=charge)`.
  8. Create a `SIC` class object with `sic_object = SIC(mol,xc='LDA,PW',fod1=fod1,fod2=fod2)`
     - Specify self-consistency parameters if wished.
  9. Start the calculation with `total_energy = sic_object.kernel()`.

2.4 Output

- In the one-shot mode, the output is realized as a Python dictionary. It includes:
  - `etot_dft`: The total energy of the underlying DFT calculation.
  - `etot_sic`: The FLO-SIC corrected total energy.
  - `homo_dft`: The HOMO energy eigenvalue of the underlying DFT calculation.
  - `homo_sic`: The HOMO energy eigenvalue corrected by FLO-SIC.
  - `hamil`: The SIC Hamiltonian.
  - `fforces`: The forces for the FOD geometry optimization.
  - `flo`: The Fermi-Löwdin orbitals in the AO formalism.
- In the self-consistent mode, the total energy is the direct return value of `.kernel()`, with further output options realized through class attributes:
  - `.homo_flosic`: The HOMO energy eigenvalue corrected by FLO-SIC.
  - `.flo`: The Fermi-Löwdin orbitals in the AO formalism.
  - `.fforces`: The forces for the FOD geometry optimization.
  - `.esic`: FLO-SIC total energy correction.

2.5 Examples

In order to enable an easy and effective use Pyflosic comes with a number of examples that can be found in `examples/`. The examples can be executed by e.g. `bbash run.sh 01_dft.py`. Note: You need to adjust the PYTHONPATH in the run.sh file according to your pyscf installation.

File	Purpose
`basic_calculations/01_dft.py`	Illustrates the principal way of performing a DFT calculation in PySCF.
`basic_calculations/02_flosic_os.py`	Illustrates how FLO-SIC can be accessed and customized in the one-shot mode.
`basic_calculations/03_flosic_scf.py`	Illustrates how FLO-SIC can be used in the self-consistent mode.
`advanced_applications/01_variablespin.py`	Illustrates how calculations with variable spin can be done (Note: Currently only tested with H_2).
`advanced_applications/02_plot_flo.py`	Illustrates how FLOs can be saved in the `.cube` format.
`advanced_applications/03_plot_density.py`	Illustrates how the density can be visualized on the numerical grid. Please note that you have to have visual output enabled if you use Pyflosic from a remote server.
`advanced_applications/04_efield.py`	Illustrates how an electric field can be applied to DFT and FLO-SIC calculations.
`advanced_applications/05_energy_contribution.py`	Illustrates how the different contributions to the exchange-correlation energy of a FLO-SIC calculation can be made visible.
`advanced_applications/06_coupling_constant.py`	Illustrates how magnetic coupling constants can be calculated with FLO-SIC.
`advanced_applications/07_newton.py`	Illustrates how a second order self-consistency solver can be used in PySCF. Doing so increases stability and speed of the calculation but restricts the choice of exchange-correlation functionals to LDA and GGA functionals.

2.6 Known Issues

Although thoroughly tested, some issues as well as the possibility of undiscovered bugs remain. The following list serves as an overview of documented issues that can easily be solved.

Issue	**Solution / Explanation**
`UnboundLocalError: local variable 'M' referenced before assignment`	The spin polarization is taken from a database that draws on the ASE code if not directly specified by the user. This error message indicates that no spin polarization has been given to Pyflosic although a system is being calculated for which Pyflosic has no spin polarization in its database. Please specify a spin polarization.
Self-consistency could not be achieved.	Should a FLO-SIC calculation fail to achieve self-consistency, this may be due to a complicated energy landscape and/or incorrect spin configuration. Try to enable a second-order self-consistency solver or variable spin configuration (see the advanced examples above).
The ground state total energy is obviously wrong.	This error is most likely caused by an incorrect spin configuration, either in the spin variable itself or in the provided FOD geometry. Check if the FODs for the first spin channel are given with `X` and the FODs for the second spin channel with `He` as chemical symbols. If the symbols are switched, Pyflosic will assume there are no FODs available for the first spin channel, which naturally causes the energy to be drastically wrong.

Chapter 3

FOD geometry optimization

3.1 Optimization strategies

PySCF delivers energies and gradients (Note: F = - grad U), you need external packages to perform geometry optimization. You can use the PySCF preferred `pyberny` package or a framework based on the atomic simulation environment (ase), where we offering an ase calculator- as well as an ase optimizer- class.

3.1.1 pyberny

For a proper pyberny installation you need only to follow the next steps, while keeping in mind you not realy need to install something!

- `git clone https://github.com/azag0/pyberny /path/to/pyberny`
- `export PYTHONPATH=path/to/pyberny:$PYTHONPATH`

3.1.2 ase - calculator and optimizer

The ase calculator as well as the ase optimizer can be found in the source directory of PyFLOSIC `pyflosic/src`.

File	Purpose
`ase_pyflosic_calculator.py`	ASE calculator for PyFLOSIC. Can perform DFT, one-shot or scf FLO-SIC calculations.
`ase_pyflosic_optimizer.py`	ASE Calculator can perform DFT and FOD optimization using different optimzers (CG, BFGS, LBFGS, FIRE etc.)

3.2 Single-point and geometry optimization examples

In order to enable an easy and effective use Pyflosic comes with a number of examples for the ase-pyflosic-calculator in the `examples/ase_pyflosic_calculator` directory and for the ase-pyflosic-optimizer in the `examples/ase_pyflosic_optimizer` directory . The

examples can be executed by e.g. `bash run.sh 01_dft.py`. Note: You need to adjust the PYTHONPATH in the run.sh file according to your pyscf installation.

File	**Purpose**
`ase_pyflosic_calculator/01_dft.py`	DFT calculation (single-point) using the ase-pyflosic-calculator.
`ae_pyflosic_calculator/02_flosic_os.py`	FLO-SIC one-shot (OS) calculation (single-point) using the ase-pyflosic-calculator.
`ase_pyflosic_calculator/03_flosic_scf.py`	FLO-SIC SCF calculation (single-point) using the ase-pyflosic-calculator.
`ase_pyflosic_optimizer/01_dft_pure_ase_optimizer.py`	DFT nuclei geometry optimization using pyscf and ase.
`ase_pyflosic_optimizer/02_dft_pylfosic_optimizer.py`	DFT nuclei geometry optimization using ase-pyflosic-optimizer.
`ase_pyflosic_optimizer/03_fod_pyflosic_os_optimizer.py`	FLOSIC OS FOD geometry optimization using ase-pyflosic-optimizer.
`ase_pyflosic_optimizer/04_fod_pyflosic_scf_optimizer.py`	FLOSIC SCF FOD geometry optimization using ase-pyflosic-optimizer.
`ase_pyflosic_optimizer/05_pyberny_nuclei_optimization.py`	DFT nuclei geometry optimization using pyscf and pyberny.

Chapter 4

Automatic generated FOD geometries

One of the authors (SS) implemented a automatic Fermi-orbital-descriptor guess generator (see `src/automatic_guess.py`). In the first step a single-point DFT calculation is performed. The KS-orbitals are localized using a user specified localization method (e.g. Foster-Boys (FB), Edminster-Ruedenberg (ER) etc.). Note that each spin channel is localized separately and only occupied orbitals are localized. All localized orbitals are saved as cube files. Given these cube files the centroid (or may called also center-of-mass or center-of-gravity) of $2|\phi_{\text{loc}}|$ is calculated and all centroids are saved in a xyz file, which is now your starting guess for the Pyflosic calculation. The method works for all-electron as well as for pseudo-potential basis sets.

File	**Purpose**
`automatic_guess.py`	Generates FOD starting guess from a xyz file containing only nuclei information.

4.1 Automatic guess generation examples

In order to enable an easy and effective use Pyflosic comes with a number of examples that can be found in `examples/automatic_guessing`. The examples can be executed by e.g. `bash run.sh 01_CH4.py`. Note: You need to adjust the PYTHONPATH in the run.sh file according to your pyscf installation.

File	**Purpose**
`automatic_guessing/01_CH4.py`	Automatic guess generation for the CH_4 molecule.
`automatic_guessing/02_O3.py`	Automatic guess generation for the O_3 molecule.
`automatic_guessing/03_TCNE.py`	Automatic guess generation for the TCNE molecule.
`automatic_guessing/further`	C_{60} example should be executed on a bigger node with may be 24 cores.

Bibliography

[1] Q. Sun, T. C. Berkelbach, N. S. Blunt, G. H. Booth, S. Guo, Z. Li, J. Liu, J. D. McClain, E. R. Sayfutyarova, S. Sharma, *et al.*, "Pyscf: the python-based simulations of chemistry framework," *Wiley Interdisciplinary Reviews: Computational Molecular Science*, vol. 8, no. 1, 2018.

[2] M. R. Pederson, R. A. Heaton, and C. C. Lin, "Local-density hartree–fock theory of electronic states of molecules with self-interaction correction," *The Journal of chemical physics*, vol. 80, no. 5, pp. 1972–1975, 1984.

[3] M. R. Pederson, R. A. Heaton, and C. C. Lin, "Density-functional theory with self-interaction correction: Application to the lithium moleculea," *The Journal of chemical physics*, vol. 82, no. 6, pp. 2688–2699, 1985.

[4] T. Hahn, S. Liebing, J. Kortus, and M. R. Pederson, "Fermi orbital self-interaction corrected electronic structure of molecules beyond local density approximation," *The Journal of chemical physics*, vol. 143, no. 22, p. 224104, 2015.

[5] M. R. Pederson, "Fermi orbital derivatives in self-interaction corrected density functional theory: Applications to closed shell atoms," *The Journal of chemical physics*, vol. 142, no. 6, p. 064112, 2015.

[6] M. R. Pederson and T. Baruah, "Self-interaction corrections within the fermi-orbital-based formalism," in *Advances in Atomic, Molecular, and Optical Physics*, vol. 64, pp. 153–180, Elsevier, 2015.

[7] M. R. Pederson, T. Baruah, D.-y. Kao, and L. Basurto, "Self-interaction corrections applied to mg-porphyrin, c60, and pentacene molecules," *The Journal of Chemical Physics*, vol. 144, no. 16, p. 164117, 2016.

[8] M. R. Pederson, R. A. Heaton, and J. G. Harrison, "Metallic state of the free-electron gas within the self-interaction-corrected local-spin-density approximation," *Physical Review B*, vol. 39, no. 3, p. 1581, 1989.

[9] M. R. Pederson and C. C. Lin, "Localized and canonical atomic orbitals in self-interaction corrected local density functional approximation," *The Journal of chemical physics*, vol. 88, no. 3, pp. 1807–1817, 1988.

[10] A. Briley, M. R. Pederson, K. A. Jackson, D. C. Patton, and D. V. Porezag, "Vibrational frequencies and intensities of small molecules: All-electron, pseudopotential, and mixed-potential methodologies," *Physical Review B*, vol. 58, no. 4, p. 1786, 1998.

[11] K. Jackson and M. R. Pederson, "Accurate forces in a local-orbital approach to the local-density approximation," *Physical Review B*, vol. 42, no. 6, p. 3276, 1990.

[12] M. R. Pederson and K. A. Jackson, "Variational mesh for quantum-mechanical simulations," *Physical Review B*, vol. 41, no. 11, p. 7453, 1990.

[13] D. Porezag and M. R. Pederson, "Infrared intensities and Raman-scattering activities within density-functional theory," *Physical Review B*, vol. 54, no. 11, p. 7830, 1996.

[14] D. D. Porezag, *Development of Ab-Initio and Approximate Density Functional Methods and their Application to Complex Fullerene Systems*. PhD thesis, 1997.

[15] A. A. Quong, M. R. Pederson, and J. L. Feldman, "First principles determination of the interatomic force-constant tensor of the fullerene molecule," *Solid state communications*, vol. 87, no. 6, pp. 535–539, 1993.

[16] M. R. Pederson and K. A. Jackson, "Pseudoenergies for simulations on metallic systems," *Physical Review B*, vol. 43, no. 9, p. 7312, 1991.

[17] D. Porezag and M. R. Pederson, "Optimization of Gaussian basis sets for density-functional calculations," *Physical Review A*, vol. 60, pp. 2840–2847, Oct 1999.

[18] L. Fiedler, "Implementation and reassessment of the Fermi-Löwdin orbital self-interaction correction for LDA, GGA and mGGA functionals." If you wish access, please contact the author (fiedler.lenz@gmail.com), 2018.