Self-consistent field (SCF) methods

Modules: `pyscf.scf` [../pyscf_api_docs/pyscf.scf.html#module-pyscf.scf], `pyscf.pbc.scf` [../pyscf_api_docs/pyscf.pbc.scf.html#module-pyscf.pbc.scf], `pyscf.soscf` [../pyscf_api_docs/pyscf.soscf.html#module-pyscf.soscf]

Introduction

Self-consistent field (SCF) methods include both Hartree-Fock (HF) theory and Kohn-Sham (KS) density functional theory (DFT). Self-consistent field theories only depend on the electronic density matrices, and are the simplest level of quantum chemical models. Details that are specific to DFT can be found in Density functional theory (DFT) [dft.html#user-dft].

In both HF and KS-DFT, the ground-state wavefunction is expressed as a single Slater determinant Φ_0 of molecular orbitals (MOs) ψ, $\Phi_0 = \mathcal{A}|\psi_1(1)\psi_2(2)\dots\psi_N(N)|$. The total electronic energy $E = \langle\Psi_0|\hat{H}|\Psi_0\rangle$ is then minimized, subject to orbital orthogonality; this is equivalent to the description of the electrons as independent particles that only interact via each others' mean field.

It can be shown that the minimization of the total energy within a given basis set (see e.g. [1 [reference.html#id66]] or any standard textbook on quantum chemistry like [2 [reference.html#id61]]) leads to the equation

$$\mathbf{FC} = \mathbf{SCE}$$

where $\mathbf{C}$ is the matrix of molecular orbital coefficients, $\mathbf{E}$ is a diagonal matrix of the corresponding eigenenergies, and $\mathbf{S}$ is the atomic orbital overlap matrix. The Fock

matrix $\mathbf{F}$ is defined as

$$\mathbf{F} = \mathbf{T} + \mathbf{V} + \mathbf{J} + \mathbf{K}$$

where $\mathbf{T}$ is the kinetic energy matrix, $\mathbf{V}$ is the external potential, $\mathbf{J}$ is the Coulomb matrix, and $\mathbf{K}$ is the exchange matrix.

Initial guess

The Coulomb and exchange matrices depend on the occupied orbitals, meaning that the SCF equation $\mathbf{FC} = \mathbf{SCE}$ needs to be solved self-consistently by some sort of iterative procedure, which begins from some initial guess. The accuracy of several initial guesses for SCF calculations has recently been assessed in [3 [reference.html#id62]], to which we refer for detailed discussion on initial guesses.

Several of initial guess have been implemented in PySCF; the used variant is controlled by the `init_guess` [../pyscf_api_docs/pyscf.scf.html#pyscf.scf.hf.SCF.init_guess] attribute of the SCF solver. The following values are possible

- `'minao'` (default)

  > A superposition of atomic densities [4 [reference.html#id86], 5 [reference.html#id87]] technique, in which the guess is obtained by projecting the minimal basis of the first contracted functions in the cc-pVTZ or cc-pVTZ-PP basis set onto the orbital basis set, and then forming the density matrix. The guess orbitals are obtained by diagonalizing the Fock matrix that arises from the spin-restricted guess density.

- `'1e'`

  > The one-electron guess, also known as the core guess, obtains the guess orbitals from the diagonalization of the core Hamiltonian $\mathbf{H}_0 = \mathbf{T} + \mathbf{V}$, thereby ignoring all interelectronic interactions and the screening of nuclear charge. The 1e guess should only be used as a last resort, because it is so bad for molecular systems; see [3 [reference.html#id62]].

- `'atom'`

  > Superposition of atomic densities [4 [reference.html#id86], 5 [reference.html#id87]]. Employs spin-restricted atomic HF calculations that employ spherically averaged fractional occupations with ground states determined with fully numerical calculations at the complete basis set limit in [6 [reference.html#id65]].

- `'huckel'`

  > This is the parameter-free Hückel guess described in [3 [reference.html#id62]], which is based on on-the-fly atomic HF calculations that are performed analogously to `'atom'`. The spherically averaged atomic spin-restricted Hartree-Fock calculations yield a minimal basis of atomic orbitals and orbital energies, which are used to build a Hückel type matrix that is diagonalized to obtain guess orbitals.

- `'vsap'`

  > Superposition of atomic potentials as described in [3 [reference.html#id62]]. A sum of pretabulated, fully numerical atomic potentials determined with the approach of [6 [reference.html#id65]] is used to build a guess potential on a DFT quadrature grid; this potential is then used to obtain the orbitals. Note this option is only available for DFT calculations in PySCF.

- `'chk'`

  > Read in the orbitals from the checkpoint file and use them as the initial guess (see below for more details).

Alternatively, the user can also override the initial guess density matrix for an SCF calculation through the `dm0` argument. For example, the following script first computes the HF density matrix for the Cr^{6+} cation, and then uses it as an initial guess for a HF calculation of the Cr atom.

```
# First calculate the Cr6+ cation
mol = gto.Mole()
mol.build(
    symmetry = 'D2h',
    atom = [['Cr',(0, 0, 0)], ],
```

```
    basis = 'cc-pvdz',
    charge = 6,
    spin = 0,
)

mf = scf.RHF(mol)
mf.kernel()
dm1 = mf.make_rdm1()

# Now switch to the neutral atom in the septet state
mol.charge = 0
mol.spin = 6
mol.build(False,False)

mf = scf.RHF(mol)
mf.kernel(dm0=dm1)
```

More examples can be found in examples/scf/15-initial_guess.py [https://github.com/pyscf/pyscf/tree/master/examples/scf/15-initial_guess.py], and examples/scf/31-cr_atom_rohf_tune_init_guess.py [https://github.com/pyscf/pyscf/tree/master/examples/scf/31-cr_atom_rohf_tune_init_guess.py].

Restart from an old calculation

Although alike many other quantum chemistry codes, there is no *restart* mechanism available in PySCF package, calculations can still be "restarted" by reading in an earlier wave function as the initial guess for the wave function. The initial guess can be prepared in many ways. One is to read the `chkpoint` file which is generated in the previous or other calculations:

```
>>> from pyscf import scf
>>> mf = scf.RHF(mol)
>>> mf.chkfile = '/path/to/chkfile'
>>> mf.init_guess = 'chkfile'
>>> mf.kernel()
```

`/path/to/chkfile` can be found in the output in the calculation (if `mol.verbose >= 4`, the filename of the chkfile will be dumped in the output). If the results of the

calculation are needed at a later stage (e.g. for an eventual restart or use as an initial guess for a larger calculation), the **`chkfile`** attribute should be set explicitly as the chkfile might otherwise be deleted upon successful completion of the calculation, see comments in examples/scf/14-restart.py [https://github.com/pyscf/pyscf/tree/master/examples/scf/14-restart.py]. By setting **`chkfile`** and **`init_guess`**, the SCF module can read the molecular orbitals from the given **`chkfile`** and rotate them to representation of the required basis.

The initial guess can also be fed in directly to the calculation. For example, we can read in the density matrix from a checkpoint file, and pass it directly to the SCF solver with:

```
>>> from pyscf import scf
>>> mf = scf.RHF(mol)
>>> dm = scf.hf.from_chk(mol, '/path/to/chkfile')
>>> mf.kernel(dm)
```

This approach leads to the same result as setting **`init_guess`** to *chkfile*.

N.B. The `chkfile` initial guess is not limited to calculations on the same molecule or the same basis set. One can first do a cheaper SCF calculation with smaller basis sets, or run an SCF calculation on a model system (e.g. drop a few atoms or run the same system in an easier charge/spin state), then use **`scf.hf.from_chk()`** to project the results to the target basis sets.

Converging SCF iterations

Even with a very good initial guess, making the SCF procedure converge is sometimes challenging. PySCF implements two kinds of approaches for SCF, namely, direct inversion in the iterative subspace (DIIS) and second-order SCF (SOSCF).

- DIIS (default)

  > With DIIS, the Fock matrix at each iteration is extrapolated using Fock matrices from the previous iterations, by minimizing the norm of the

> commutator $[\mathbf{F}, \mathbf{PS}]$ where $\mathbf{P}$ is the density matrix [7 [reference.html#id68], 8 [reference.html#id69]]. Two variants of DIIS are implemented in PySCF, namely, EDIIS [9 [reference.html#id70]] and ADIIS [10 [reference.html#id71]]. Examples of selecting different DIIS schemes can be found in examples/scf/24-tune_diis.py [https://github.com/pyscf/pyscf/tree/master/examples/scf/24-tune_diis.py].

- SOSCF

> To achieve quadratic convergence in the orbital optimization, PySCF implements a general second-order solver called the co-iterative augmented hessian (CIAH) method [11 [reference.html#id72], 12 [reference.html#id73]]. This method can be invoked by decorating the SCF objects with the **`newton()`** [../pyscf_api_docs/pyscf.scf.html#pyscf.scf.newton] method:
>
> ```
> mf = scf.RHF(mol).newton()
> ```
>
> More examples can be found in examples/scf/22-newton.py [https://github.com/pyscf/pyscf/tree/master/examples/scf/22-newton.py].

- Damping

> The Fock matrix can be damped before DIIS acceleration kicks in. This is achieved by setting the attributes **`damp`** [../pyscf_api_docs/pyscf.scf.html#pyscf.scf.hf.SCF.damp] and **`diis_start_cycle`** [../pyscf_api_docs/pyscf.scf.html#pyscf.scf.hf.SCF.diis_start_cycle]. For example,
>
> ```
> mf.damp = 0.5
> mf.diis_start_cycle = 2
> ```
>
> means that DIIS will start at the second cycle, and that the Fock matrix is damped by 50% in the first cycle.

- Level shifting

> A level shift increases the gap between the occupied and virtual orbitals, thereby slowing down and stabilizing the orbital update. A level shift can help to converge SCF in the case of systems with small HOMO-LUMO gaps. Level shifting is invoked by setting the attribute `level_shift` [../pyscf_api_docs/pyscf.scf.html#pyscf.scf.hf.SCF.level_shift]. See examples in examples/scf/03-level_shift.py [https://github.com/pyscf/pyscf/tree/master/examples/scf/03-level_shift.py], and examples/scf/52-dynamically_control_level_shift.py [https://github.com/pyscf/pyscf/tree/master/examples/scf/52-dynamically_control_level_shift.py].

- ✓ Fractional occupations

> Fractional occupations can also be invoked to help the SCF converge for small gap systems. See the example in examples/scf/54-fractional_occupancy.py [https://github.com/pyscf/pyscf/tree/master/examples/scf/54-fractional_occupancy.py].

- ✓ Smearing

> Smearing sets fractional occupancies according to a temperature function. See the example examples/pbc/23-smearing.py [https://github.com/pyscf/pyscf/tree/master/examples/pbc/23-smearing.py].

Stability analysis

Even when the SCF converges, the wave function that is found may not correspond to a local minimum; calculations can sometimes also converge onto saddle points. Since saddle points are also extrema of the energy functional, the orbital gradient vanishes and the SCF equation $\mathbf{FC} = \mathbf{SCE}$ is satisfied [1 [reference.html#id66]]. However, in such cases the energy can be decreased by perturbing the orbitals away from the saddle point, which means that the wave function is unstable.

Instabilities in the wave function are conventionally classified as either internal or external [13 [reference.html#id74]]. External instabilities mean that the energy can be decreased by loosening some constraints on the wave function, such as allowing restricted Hartree-Fock orbitals to transform into unrestricted Hartree-Fock, whereas internal instabilities mean that the SCF has converged onto an excited state instead of the ground state. PySCF allows detecting both internal and external instabilities for a given SCF calculation; see the examples in examples/scf/17-stability.py [https://github.com/pyscf/pyscf/tree/master/examples/scf/17-stability.py].

Property calculations

Various properties can be computed by calling the corresponding functions, for example,

- dipole moment:

```
mf.dip_moment()
```

- Mulliken population:

```
mf.mulliken_pop()
```

- nuclear gradients:

```
g = mf.Gradients()
g.kernel()
```

Also several response properties are available in PySCF with the properties [https://github.com/pyscf/properties] extension, see the examples there [https://github.com/pyscf/properties/tree/master/examples].

Spin-restricted, spin-unrestricted, restricted open-shell, and generalized calculations

The general spin-orbital used in the HF or KS-DFT wave function can be written as

$$\psi_i(1) = \phi_{i\alpha}(r)\alpha + \phi_{i\beta}(r)\beta \ ,$$

Four variants of the ansatz $\psi(1)$ are commonly used in quantum chemistry; they are also all available in PySCF.

- Restricted (RHF/RKS)

  The spin-orbitals are either alpha (spin-up) or beta (spin-down), $\psi_i = \phi_i(r)\alpha$ or $\psi_i = \phi_i(r)\beta$, and the alpha and beta orbitals share the same spatial orbital $\phi_i(r)$. The closed-shell determinant is thus $\Phi = \mathcal{A}|\phi_1(r_1)\alpha\phi_1(r_2)\beta\ldots\phi_{N/2}(r_{N-1})\alpha\phi_{N/2}(r_N)\beta|$ and $S = 0$.

- Unrestricted (UHF/UKS)

  The orbitals can have either alpha or beta spin, but the alpha and beta orbitals may have different spatial components. The determinant is thus $\Phi = \mathcal{A}|\phi_1(r_1)\sigma_1\phi_2(r_2)\sigma_2\ldots\phi_N(r_N)\sigma_N|$ where $\sigma \in \{\alpha, \beta\}$. Spin contamination is introduced for states that don't have maximal S_z.

- Restricted open-shell (ROHF/ROKS)

  Equivalent to RHF/RKS for $N_\alpha = N_\beta$. For $N_\alpha > N_\beta$, the first N_β orbitals have the same spatial components for both α and β spin. The remaining $N_\alpha - N_\beta$ orbitals are of α spin. $\Phi = \mathcal{A}|\phi_1\alpha\phi_1\beta\ldots\phi_{N_\beta}\alpha\phi_{N_\beta}\beta\phi_{N_\beta+1}\alpha\ldots\phi_N\alpha|$ The final wavefunction is an eigenfunction of the $\hat{S}^2$ operator with $S_z = S$.

- Generalized (GHF/GKS)

  The general form of the spin-orbital ψ is used. GHF/GKS is useful when none of the previous methods provide stable solutions (see examples/scf/17-stability.py [https://github.com/pyscf/pyscf/tree/master/examples/scf/17-stability.py]), or when the Hamiltonian does not commute with $\hat{S}_z$ (e.g. in the presence of spin-orbit coupling, see examples/scf/44-soc_ecp.py [https://github.com/pyscf/pyscf/tree/master/examples/scf/44-soc_ecp.py]).

Calculations with these methods can be invoked by creating an instance of the corresponding class:

```
mf = scf.RHF(mol).run()
mf = scf.UHF(mol).run()
mf = scf.ROHF(mol).run()
mf = scf.GHF(mol).run()
mf = scf.RKS(mol).run()
mf = scf.UKS(mol).run()
mf = scf.ROKS(mol).run()
mf = scf.GKS(mol).run()
```

More examples can be found in examples/scf/00-simple_hf.py [https://github.com/pyscf/pyscf/tree/master/examples/scf/00-simple_hf.py], examples/scf/01-h2o.py [https://github.com/pyscf/pyscf/tree/master/examples/scf/01-h2o.py], examples/scf/02-rohf_uhf.py [https://github.com/pyscf/pyscf/tree/master/examples/scf/02-rohf_uhf.py], and examples/scf/02-ghf.py [https://github.com/pyscf/pyscf/tree/master/examples/scf/02-ghf.py].

Linear dependencies

Most quantum chemistry programs solve the self-consistent field equations

$$\mathbf{FC} = \mathbf{SCE}$$

in an orthonormal basis, which is formally obtained as

$$\mathbf{C} = \mathbf{X}\tilde{\mathbf{C}}$$

where the orthogonalizing matrix $\mathbf{X}$ is typically chosen as $\mathbf{X} = \mathbf{S}^{-1/2}$. By expressing the orbitals in terms of the half-inverse overlap matrix, the generalized eigenproblem $\mathbf{FC} = \mathbf{SCE}$ can be rewritten as a regular eigenproblem $\tilde{\mathbf{F}}\tilde{\mathbf{C}} = \tilde{\mathbf{C}}\mathbf{E}$ [1 [reference.html#id66]].

Moreover, as the half-inverse overlap matrix $\mathbf{S}^{-1/2}$ is typically formed by the canonical orthonormalization procedure [14 [reference.html#id67]] in which eigenvectors of the overlap matrix with small eigenvalues are thrown out, this procedure typically results in a better-conditioned basis since linearly dependent combinations of the basis functions are excluded by the procedure.

At variance, PySCF relies on SciPy's generalized eigenvalue solver by default, which may fail for poorly conditioned basis sets. One can, however, switch to the use of canonical orthonormalization by toggling e.g.:

```
mf = scf.RHF(mol).apply(scf.addons.remove_linear_dep_)
```

In the presence of truly pathological linear dependencies, such as those that occur in molecular calculations with multiply augmented basis sets, and at extreme molecular geometries where two nuclei are close to each other, also canonical orthonormalization fails. However, the addons module also implements the partial Cholesky orthonormalization technique [15 [reference.html#id63], 16 [reference.html#id64]], which has been shown to work reliably even in the presence of such truly pathological linear dependencies.

Scalar relativistic correction

Scalar relativistic effects can be applied on the one-body operators through spin-free eXact-2-component (SFX2C) Hamiltonian [17 [reference.html#id75]]. The SFX2C Hamiltonian can be invoked by decorating the SCF objects with the `x2c()` method, three other equivalent function names are also listed below:

```
mf = scf.RHF(mol).x2c()
mf = scf.RHF(mol).x2c1e()
mf = scf.RHF(mol).sfx2c()
mf = scf.RHF(mol).sfx2c1e()
```

Note that the SFX2C Hamiltonian only changes the one-body operators, and it only accounts for the mass-velocity effect, while picture change effect and spin-orbit

coupling are not included. Once the SCF object is decorated by `x2c()` method, the corresponding post-SCF objects will also automatically have the SFX2C Hamiltonian applied. To turn it off explicitly, one can do:

```
mf.with_x2c = False
```

More examples can be found in examples/scf/21-x2c.py [https://github.com/pyscf/pyscf/tree/master/examples/scf/21-x2c.py].