THE SCINE SPARROW DEVELOPERS:

FRANCESCO BOSIA, TAMARA HUSCH, CHARLOTTE H. MÜLLER, SEVERIN POLONIUS, JAN-GRIMO SOBEZ, MIGUEL STEINER, JAN P. UNSLEBER, ALAIN C. VAUCHER, THOMAS WEYMUTH, AND MARKUS REIHER

# USER MANUAL

## SCINE SPARROW 3.0.0

ETH ZÜRICH

# *Contents*

# *Introduction*

The availability of fast electronic energies and gradients is essential for the SCINE project. The SCINE Sparrow module contains electronic structure models which were designed to yield electronic energies, energy gradients with respect to the nuclear coordinates, and Hessians rapidly. The SCINE Sparrow module can be driven from SCINE Interactive, SCINE ReaDuct, and SCINE Chemoton. However, as with all SCINE modules it is also a stand-alone program which can be applied on its own or easily interfaced to other programs.

SCINE Sparrow is a command-line tool that implements many popular semiempirical models. SCINE Sparrow 3.0.0 provides the `MNDO`, `AM1`, `RM1`, `PM3`, `PM6`, non-SCC DFTB (`DFTB0`), `DFTB2`, and `DFTB3` methods (open- and closed-shell formalisms are implemented). The application of semiempirical models usually allows for rapid calculation of electronic energies and energy gradients for a small molecular structure with a given charge and spin state.

In this manual, we describe the installation of the software, an example calculation as a hands-on introduction to the program, and the most import functions and options.[1] A prospect on features in future releases and references for further reading are added at the end of this manual.

[1] Throughout this manual, the most import information is displayed in the main text, whereas useful additional information is given as a side note like this one.

# *Obtaining the Software*

SPARROW is distributed as open source software in the framework of the SCINE project (www.scine.ethz.ch). Visit our website (www.scine.ethz.ch/download/sparrow) to obtain the software.

## *System Requirements*

SPARROW itself has only modest requirements regarding the hardware performance. However, the underlying quantum-chemical calculations might become resource intensive if extremely large systems are studied.

# *Installation*

Sparrow is distributed as an open source code. In order to compile Sparrow from this source code, you need

- A C++ compiler supporting the C++14 standard,
- cmake (at least version 3.9),
- the Boost library (at least version 1.65.0), and
- the Eigen3 library (at least version 3.3.2).

In order to compile the software, either directly clone the repository with git or extract the downloaded tarball, change to the source directory and execute the following steps:

```
git submodule init
git submodule update
mkdir build install
cd build
cmake -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=../install ..
make
make test
make install
export PATH=$PATH:<source code directory>/install/bin
```

This will configure everything, compile your software, run the tests, and install the software into the folder "install". Finally, it will add the Sparrow binary to your `PATH`, such that you can use it without having to specify its full location. In this last command, you have to replace `<source code directory>` with the full path where you stored the source code of Sparrow.

In case you need support with the setup of Sparrow, please contact us by writing to scine@phys.chem.ethz.ch.

# *Example Calculation*

Sparrow is a command-line-only binary; there is no graphical user interface. Therefore, you always work with the Sparrow binary on a command line such as the Gnome Terminal or KDE Konsole. All functionality is accessed via command line arguments. All possible command line options can be listed with the following command:

```
sparrow --help
```

In order to provide a practical demonstration of the Sparrow program, we present here a step-by-step example calculation that guides you through the complete process of calculating the total electronic energy as well as the nuclear gradient and Hessian of a molecular structure with Sparrow. We start with the following Cartesian coordinates for water:

```
3

O      -0.27939703       0.83823215       0.00973345
H      -0.52040310       1.77677325       0.21391146
H       0.54473632       0.90669722      -0.53501306
```

Store these coordinates in a file named "h2o.xyz". Then, call Sparrow with the following command:

```
sparrow  --structure h2o.xyz --molecular_charge 0 --spin_multiplicity 1 --method PM6
```

This will calculate the electronic energy for the neutral water molecule with PM6. You can also use the short options

```
sparrow -x h2o.xyz -c 0 -s 1 -M PM6
```

to achieve the same result. If you also want to calculate the nuclear gradient, simply add the option `--gradient` or `-G`. For the Hessian, specify `--hessian` or `-H`. For the atomic second derivatives (which can be used to approximate the Hessian as a block-diagonal matrix of second derivatives. Both coordinates with respect to which the

energy is derived refer to the same atom), specify `--atomic_hessians` or `-A`.

# *Detailed Documentation*

## *Command Line Arguments*

In this section, the full functionality of SPARROW is documented, *i.e.*, all possible command line arguments are listed and explained.

- `--structure`, `-x`: This argument specifies the structure which should be calculated. It must be given as a path to an XYZ file.
- `--molecular_charge`, `-c`: This is used to specify the overall charge of the system to be calculated, `e.g.`, `--molecular_charge 0` or `-c -1`. The default charge is zero. Only charges between $-20$ and $+20$ are supported.
- `--spin_multiplicity`, `-s`: This is used to specify the spin multiplicity of the system to be calculated, `e.g.`, `-s 1` for a singlet state. The default multiplicity is one. The maximal spin multiplicity is 10.
- `--gradients`, `-G`: If given, the nuclear gradients will be calculated.
- `--hessian`, `-H`: If given, the Hessian and the nuclear gradients will be calculated.
- `--atomic_hessians`, `-A`: If given, a list of the atomic second derivatives will be calculated.
- `--thermochemistry`, `-C`: If given, the heat capacities at constant volume or pressure, the enthalpy, the entropy, the Gibbs free enthalpy as well as the zero point vibrational energy (ZPVE) are calculated from the total molecular partition function. The calculation of the thermochemical properties requires the Hessian matrix: its calculation is therefore implied by setting this option.
- `--symmetry_number`, `-S`: Specifies the point group-dependent molecular symmetry number $\sigma$ for the calculation of thermochemical properties. It is not adapted automatically for molecules with

more than two atoms. The default value is one.

- `--temperature`, `-T`: This is used to specify the temperature in Kelvin at which the thermochemical properties are calculated. The default temperature is 298.15 K. The maximum allowed temperature is 10'000 K.

- `--suppress_normal_modes`, `-N`: If given, the full Hessian will be printed instead of the normal modes and the vibrational frequencies. This option has no effect if the Hessian is not calculated (see above).

- `--bond_orders`, `-B`: If given, the bond order matrix will be calculated.

- `--method`, `-M`: With this option, the desired calculation method can be set. Options in SPARROW 3.0.0 are `MNDO`, `AM1`, `RM1`, `PM3`, `PM6`, `DFTB0`, `DFTB2`, and `DFTB3`. By default, `PM6` is selected.

- `--output_to_file`, `-o`: If this option is given, the output will not only be printed to the screen, but also to files. By default, the energy is stored in a file named "energy.dat", the nuclear gradients in a file named "gradients.dat", the Hessian in a file named "hessian.dat", the atomic Hessians in a file named "atomic_hessians.dat", and the bond order matrix in a file named "bond_orders.dat". If a description is given with the option `-d` (see below), this description will also be used in the file name.

- `--description`, `-D`: This can be used to add a (short) description of the calculation. This description will appear in the output. This allows to quickly find a certain calculation output at a later point. The description should be enclosed by quotation marks if it is composed by more than one word, *e.g.*, `-d "This is an example"`.

- `--unrestricted_calculation`, `-u`: If this option is given, an unrestricted (UHF) calculation will be performed. By default, a restricted calculation will be done.

- `--wavefunction`, `-W`: If this option is given, a Molden input file is generated after the calculation to allow for the visualization of the molecular orbitals.The basis functions used for the generation of the Molden input are defined in the files `<source code directory>/Sparrow/Sparrow/Resources/<method>/<method>-STO-6G.basis`. All `NDDO` methods have their own STO-6G expansion which are fine tuned based on the method's parameters. For the `DFTB` methods, the same basis set as `PM6` is used.

- `--excited_states`, `-E`: If this option is given, an electronic excited

states calculation will be performed. Vertical transition energies and transition properties result from single electron substitutions for DFTB0, TD-DFTB formalism for DFTB2 and DFTB3 and NDDO-CIS for all NDDO methods.

- `--number_eigenstates`, `-r`: Specifies the desired number of excited states. Note that this option is only considered when the option `-E` (see above) is also given. By default, one excited state is calculated.
- `--initial_subspace_dimension`, `-n`: Specifies the desired number of initial guess vectors for the calculation of the excited states with TD-DFTB or NDDO-CIS. Multiples of the desired number of excited states may have a beneficial impact on the computational time. Note that this option is only considered when the option `-E` (see above) is also given. The default value is 0.
- `--spin_block`, `-b`: Specifies the desired excited-state symmetry to compute. Possible values are `-b singlet` for singlet excited states, `-b triplet` for triplet excited states, and `-b both` if excited states of both spin symmetries are desired. This option has no effect if an unrestricted calculation is performed. Note that this option is only considered when the option `-E` (see above) is also given. By default, the singlet states are computed.
- `--number_orbital_mixes -O`: Sets the number of orbital steers to carry out, if 0, no steering occurs. The default is 0, so no steering. The command `--number_orbital_mixes` 3 causes the program to first do a reference calculation, then three rounds of random mixing of the orbitals, followed by an acceptance of the lowest energy.
- `--number_orbitals_to_mix`: Specifies how many occupied–virtual orbital pairs are sampled without replacement and mixed together in each mixing round of the orbital steering. The default is 15 orbital pairs.
- `--number_orbitals_to_consider`: Specifies the number of orbitals around the Fermi level that will form the set from which the orbital steering samples the occupied–virtual orbital pairs that can be mixed. The default is 0, which causes the program to consider all the orbitals
- `--maximal_mixing_angle`: Specifies the maximal angle in degrees for the mixing of the occupied–virtual orbital pairs in the orbital steering. The default value is 90 degrees, i.e. the angle for which the two orbitals are swapped.
- `--minimal_mixing_angle`: Specifies the minimal angle in degrees

for the mixing of the occupied–virtual orbital pairs in the orbital steering. The default value is 0 degrees, i.e. the angle for which nothing happens. The new, mixed orbitals are obtained according to:

$$\phi_{\text{occ,new}} = \cos\alpha \cdot \phi_{\text{occ,old}} + \sin\alpha \cdot \phi_{\text{vir,old}}$$
$$\phi_{\text{vir,new}} = \cos\alpha \cdot \phi_{\text{vir,old}} - \sin\alpha \cdot \phi_{\text{occ,old}} \,.$$

The following options are usually not needed:

- `--help`, `-h`: This prints a short help message listing and explaining all possible command line arguments
- `--scf_mixer`, `-m`: With this option, the method used to accelerate the convergence of the self- consistent-field (SCF) calculations can be set. Possible options are 'no_mixer' (no convergence acceleration), 'diis' (direct inversion of the iterative subspace, DIIS), 'ediis' (energy DIIS), and 'ediis_diis'. The default is 'diis'.
- `--max_scf_iterations`, `-I`: This is used to specify the maximum number of SCF iterations. Default is 100.
- `--self_consistence_criterion`, `-t`: This specifies the convergence threshold for the electronic energy. This value is given in hartree. By default, it is '1e-7'.
- `--density_rmsd_criterion`: This specifies the convergence threshold for the density difference between iterations (density matrix RMSD). By default, it is '1e-5'.
- `--method_parameters`, `-p`: This option can be used to specify the path of the parameter file to be used. This option is usually not needed, since Sparrow provides default parameter for all its methods.
- `--excited_parameterfile`, `-j`: This option can be used to specify the path of the parameter file for the excited-state calculation with NDDO-CIS to be used. This option is usually not needed, since Sparrow provides default parameter for the excited-state calculation with all its methods. Note that this option is only considered when the option `-E` (see above) is also given.
- `--log`, `-l`: This sets the log level for warning messages and errors. Supported levels are `debug`, `output`, `warning`, `error`, and `none`. By default, the level is set to `warning`, *i.e.*, all warnings are printed to STDOUT and errors are printed to STDERR. If you set this option to `error`, only errors are printed. If you set this to `none`, neither

warnings nor errors are printed. If you set this option to a string other than the ones indicated above, an exception is thrown.

- `--log_filename`, `-f`: This sets the name of the file where the logging shall be printed. By default, logging to file is disabled.
- `--distance_threshold`, `-d`: specifies the desired distance cutoff for the 2-electron interactions in nddo-cis excited-state calculations. This feature can lead to calculations not converging. Default is no screening.
- `--prune_basis`, `-y`: if this option is given, the TD-DFTB calculation is performed in a pruned determinant basis. The pruning is specified by the options `--energy_threshold` and `--pt_threshold`.
- `--energy_threshold`, `-e`: specifies the desired energy cutoff in a.u. for the basis functions spanning the excited stated to be included. Every basis function characterized by a single orbital substitution between orbitals with an energy difference smaller than this threshold is included to the pruned basis.
- `--pt_threshold`: specifies the desired perturbation theory cutoff in a.u. (for TDA calculations) or in $\text{a.u.}^2$ (for full TD-DFTB calculations) for the basis functions spanning the excited stated to be included. Every basis function whose cumulative interaction with basis functions included with the `--energy_threshold` criterion is larger than this threshold is included to the pruned basis.
- `--max_memory`, `-g`: specifies the maximum memory the excited-state calculation can take. This is estimated from the size of the excited-state basis dimension and the number of desired roots.
- `--scf_mixer_steering`: With this option, the method used to accelerate the convergence of the self- consistent-field (SCF) calculations after an orbital steering mix can be set. Possible options are ‘no_mixer‘ (no convergence acceleration), ‘diis‘ (direct inversion of the iterative subspace, DIIS), ‘ediis‘ (energy DIIS), and ‘ediis_diis‘. The default is ‘diis‘.
- `--max_iterations_in_steering`: This is used to specify the maximum number of SCF iterations after an orbital steering mix. Default is 100.

## *Running* Sparrow *in Parallel*

By default, Sparrow will be compiled with OpenMP support and hence, it can be run in parallel. In order to use multiple CPU cores,

simply specify

```
export OMP_NUM_THREADS=n
```

where `n` is the number of CPUs you want to use. Note that by default, SPARROW uses all available cores, *i.e.*, it will also run in parallel if you do not specify the above environment variable.

# *Using the Python Bindings*

Sparrow provides Python bindings such that all functionality of Sparrow can be accessed also via the Python programming language. In order to build the Python bindings, you need to specify `-DSCINE_BUILD_PYTHON_BINDINGS=ON` when running cmake (see also chapter Installation). Some options, for instance, the convergence of the excited-state subspace solver, are only present in the Python bindings. Please note that the setting types are rigorously checked in Python. It is important not to confuse integer and floating point types! For instance, there is a difference between 1 (an integer type) and 1.0 (a floating point type).

In order to use the Python bindings, you need to specify the path to the Python library in the environment variable `PYTHONPATH`, *e.g.*, you have to run the command

```
export PYTHONPATH=$PYTHONPATH:<source code directory>/install/lib/python<version>/site-packages
```

where `<version>` is the Python version you are using (*e.g.*, 3.6). Now, you can simply import the library and use it as any other Python library. For example, in order to calculate the total electronic energy of $H_2$ with the `AM1` method as well as the gradient and the atomic second derivatives (where the latter can be used to approximate the Hessian as a block-diagonal matrix of second derivatives), use the following Python statements:

```
import scine_utilities as su
import scine_sparrow

# Generate Structure
structure = su.AtomCollection()
structure.elements = [su.ElementType.H, su.ElementType.H]
structure.positions = [[-0.7, 0, 0], [0.7, 0, 0]]

# Get calculator
manager = su.core.ModuleManager()
```

```
calculator = manager.get('calculator', 'AM1')

# Configure Calculator
calculator.structure = structure
calculator.set_required_properties([su.Property.Energy,
                                    su.Property.Gradients,
                                    su.Property.AtomicHessians])

# Calculate
results = calculator.calculate()
print(results.energy)
print(results.gradients)
# Print the atomic second derivatives as a list of 3x3 matrices.
# To use this as an approximation of the total Hessian, this output
# needs to be rearranged in a block-diagonal matrix.
print(results.atomic_hessian.get_atomic_hessians())
```

The output of Sparrow is always in Hartree atomic units, also the input is expected in Hartree atomic units.

A detailed list of all the functions provided by the SCINE Python libraries can be found by running

```
import scine_utilities

help(scine_utilities)
```

The import of the SCINE Sparrow module mainly makes the included methods available. A shorter version of the above code could be written using an XYZ file input.

```
import scine_utilities as su
import scine_sparrow

# Load xyz into calculator
calculator = su.core.get('calculator', 'AM1')
calculator.structure = su.io.read('h2.xyz')[0]

# Configure Calculator
calculator.set_required_properties([su.Property.Energy,su.Property.Gradients])

# calculate
results = calculator.calculate()
print(results.energy)
print(results.gradients)
```

In the latter example, the XYZ file is expected in Å, all results will

still be in Hartree atomic units.

Excited-state calculations can also be calculated with the SCINE Sparrow module:

```
import scine_utilities as su
import scine_sparrow

# Generate structure
# This can also be done with
# structure = su.io.read('structure.xyz')
# (will automatically be converted to bohr)
structure = su.AtomCollection(6)
structure.elements = [su.ElementType.C, su.ElementType.C, su.ElementType.H,
                      su.ElementType.H, su.ElementType.H, su.ElementType.H]
# in angstrom
structure.positions = [[0.94815,    0.05810,    0.05008],
                       [2.28393,    0.05810,    0.05008],
                       [0.38826,   -0.56287,    0.74233],
                       [0.38826,    0.67907,   -0.64217],
                       [2.84382,   -0.56287,    0.74233],
                       [2.84382,    0.67907,   -0.64217]]
structure.positions = structure.positions * su.BOHR_PER_ANGSTROM

# Get excited-state calculator
manager = su.core.ModuleManager()
excited_states_calculator = manager.get('calculator_with_reference', 'TD-DFTB')
# Generate a suitable ground-state reference calculator
excited_states_calculator.reference_calculator = manager.get('calculator', 'DFTB2')

# Configure the calculation
excited_states_calculator.reference_calculator.settings['self_consistence_criterion']=1e-9
excited_states_calculator.settings['number_eigenstates'] = 3
excited_states_calculator.settings['initial_subspace_dimension'] = 3

# Carry out the ground-state calculation
excited_states_calculator.reference_calculator.structure = structure
excited_states_calculator.reference_calculation()

# Carry out the excited-state calculation
result = excited_states_calculator.calculate()

# Print the vertical transition energies and the oscillator strengths
print(result.excited_states.singlet.eigenstates.eigenvalues * su.EV_PER_HARTREE)
su.transition_dipole_to_oscillator_strength(result.excited_states.singlet.transition_dipoles,
```

```
                                                result.excited_states.singlet.eigenstates.eigenvalues)

# Print the labels of the transition density matrix elements
# and their coefficients in the first excited state
print(result.excited_states.mo_labels)
print(result.excited_states.singlet.eigenstates.eigenvectors[:,0])
```

Orbital steering calculations can also be calculated with the SCINE Sparrow module:

```
import scine_utilities as su
import scine_sparrow

# Generate structure
# This can also be done with
# structure = su.io.read('structure.xyz')
# (will automatically be converted to bohr)
structure = su.AtomCollection(6)
structure.elements = [su.ElementType.C, su.ElementType.C, su.ElementType.H,
                      su.ElementType.H, su.ElementType.H, su.ElementType.H]
# in angstrom
structure.positions = [[0.94815,    0.05810,    0.05008],
                       [2.28393,    0.05810,    0.05008],
                       [0.38826,   -0.56287,    0.74233],
                       [0.38826,    0.67907,   -0.64217],
                       [2.84382,   -0.56287,    0.74233],
                       [2.84382,    0.67907,   -0.64217]]
structure.positions = structure.positions * su.BOHR_PER_ANGSTROM

# Get orbital steering calculator
manager = su.core.ModuleManager()
orbital_steering_calculator = manager.get('calculator_with_reference',
'orbital_steering')
# Generate a suitable reference calculator
calc = manager.get('calculator', 'PM6')
calc.structure = structure
calc.settings['self_consistence_criterion'] = 1e-8
calc.settings['spin_mode'] = 'unrestricted'
orbital_steering_calculator.reference_calculator = calc

# Configure the calculation

# How often a steering of the orbitals is carried out (1 = every call to
"calculate()", 2 = every second call to "calculate()", ...)
# Often  steering the orbitals after each single point is not needed, and
```

```
# the additional cost is not justified. In those cases, one can carry out
# a steering calculation, e.g., every 5th single-point calculations. Since the
# density guess is taken to be the solution of the previous calculation,
# a steered guess will provide a suitable solution for subsequent calculations.
orbital_steering_calculator.settings['mixing_frequency'] = 5
# How many occupied-virtual orbital pairs are created to mix
orbital_steering_calculator.settings['number_orbitals_to_mix'] = 15
# Note: not 0 and 90, as those would be integers.
orbital_steering_calculator.settings['minimal_mixing_angle'] = 0.0
orbital_steering_calculator.settings['maximal_mixing_angle'] = 90.0
# You can set other convergence criteria in the orbital steering calculations.
orbital_steering_calculator.settings['max_scf_iterations'] = 200
orbital_steering_calculator.settings['scf_mixer'] = 'ediis_diis'

# Carry out ground-state calculation without steering (not needed, but you can
# do it). This will not increment the counter used to determine when a steering is
# needed.
orbital_steering_calculator.reference_calculation()

# Carry out the orbital-steering calculation
result = orbital_steering_calculator.calculate()
```

# *Extensions Planned in Future Releases*

- Availability of OMx models
- Availability of the CISE approach
- Implementation of multireference semiempirical approach
- Implementation of periodic boundary conditions

# *References*

Please consult the following references for more details on Sparrow. We kindly ask you to cite the appropriate references in any publication of results obtained with Sparrow.


- Primary reference for Sparrow 3.0.0: F. Bosia, T. Husch, C. H. Müller, S. Polonius, J.-G. Sobez, M. Steiner, J. P. Unsleber, A. C. Vaucher, T. Weymuth, M. Reiher, "qcscine/sparrow: Release 3.0.0 (Version 3.0.0)", Zenodo, 2021.
- Presentation of the formalism of MNDO-type and OMx models:

  T. Husch, A. C. Vaucher, M. Reiher "Semiempirical Molecular Orbital Models Based on the Neglect of Diatomic Differential Overlap Approximation", *Int. J. Quantum Chem.*, **2018**, *118*, e25799.
- Presentation of DFTB approaches:

  M. Elstner, G. Seifert, "Density functional tight binding", *Phil. Trans. R. Soc. A*, **2014**, *371*, 20120483.
- Presentation of CISE:

  T. Husch, M. Reiher "Comprehensive Analysis of the Neglect of Diatomic Differential Overlap Approximation", *J. Chem. Theory Comput.*, **2018**, *14*, 5169.
- Presentation of the Orbital Steering approach:

  A. C. Vaucher, M. Reiher "Steering Orbital Optimization out of Local Minima and Saddle Points Toward Lower Energy", *J. Chem. Theory Comput.*, **2017**, *13*, 1219.