MANUAL

FCBand 2017

by
Shaohong L. Li, and Donald G. Truhlar
Department of Chemistry, Chemical Theory Center, and Minnesota Supercomputing Institute, University of Minnesota, Minneapolis, MN 55455-0431, USA

Program version date: March 26, 2017
Documentation version date: March 27, 2017

Table of Contents

1. Introduction

FCBand is a Python package for simulating vibronic bands of electronic absorption spectroscopy using the Franck-Condon displaced harmonic oscillator (FC-DHO) models. These models are suitable for simulating the unresolved vibronic band shapes of electronic spectra that involve many electronic transitions and vibrational modes.

2. Citation

Citation for the code:
S. L. Li and D. G. Truhlar, *FCBand 2017;* http://comp.chem.umn.edu/fcband/

Citation for the method:
S. L. Li and D. G. Truhlar, Franck-Condon Models for Simulating the Band Shape of Electronic Absorption Spectra, to be published.

3. Installation

3.1. Dependencies

Before installing FCBand, you need to have the following programs/packages installed:

- Python 2.7 (https://www.python.org/) (tested with version 2.7.8)
- NumPy (http://www.numpy.org) (tested with version 1.11.2)
- SciPy (https://www.scipy.org) (tested with version 0.18.1)

See the websites of the packages for how to install them.

3.2. Installing FCBand

To install FCBand, simply unzip and untar the package in a directory (let's call it `your_path`). For example, in Linux and OSX you can use the following commands in the terminal:

```
cd your_path
tar -zxvf fcband2016.tgz
```

A directory `fcband` should be created, which contains the package files.

Next, add the path to the `PYTHONPATH` environment variable. For example, if you use a bash shell in Linux, add the following line to your `$HOME/.bashrc` file

```
export PYTHONPATH=$PYTHONPATH:your_path
```

and run the command

```
source $HOME/.bashrc
```

3.3. Validation of installation

Execute the `run_tests.sh` script to validate the installation, which runs all the tests. Installation is successful if no error message is printed. The user may compare the generated .out files to the provided .out.ref files in the `test/` directory.

4. Execution

4.1. Overview

FCBand is a Python package. To use it, the user should have basic knowledge of Python. However, if you are not familiar with Python, you may refer to the test cases we provide (described in Section 7) and learn how write a Python script to use FCBand. In any case we strongly recommend reading the test cases to understand how the package works.

FCBand defines a Python module, `fcband`, which has two classes, `AsymmBand` and `GaussSpectrum`, defined in submodules `asymmband` and `gaussspectrum` respectively. `AsymmBand` has methods to read quantities from input file(s), simulate a spectral band, and print the band. `GaussSpectrum` is a container of `AsymmBand` objects for simulating a spectrum consisting of multiple bands. The most important methods are listed below; for complete documentation, see Section 8.

Important `AsymmBand` methods:

- `from_input`: Read from an input file and create an `AsymmBand` object.
- `readstuff`: Read in a single piece of data from an input file.
- `printstuff`: Print a certain piece of data.
- `xe_q0`: Compute the excited-state equilibrium geometry by using the ground-state equilibrium geometry and the excited-state gradient at the ground-state equilibrium geometry. Compute the mass-weighted Cartesian displacement. Also compute other important energetic quantities.
- `calc_band_param`: Calculate parameters needed for simulating a band.
- `calc_band_point`: Compute one point of band at a certain photon energy.
- `plot`: Print a band or return an array containing the spectral data.

Important `GaussSpectrum` methods:

- `add_band`: Add an `AsymmBand` object to the spectrum.
- `gen_spectrum`: Generate a spectrum.
- `print_spectrum`: Print a spectrum.
- `read_exp_spectrum`: Read in an experimental spectrum from a file.
- `scale_sim_to_exp`: Scale the simulated spectrum to the experimental spectrum to minimize their mean squared difference.

4.2. Typical FCBand workflow

(See Section 4.1 and Section 8 for description of the methods.)

To simulate a single spectral band, follow these steps:

- Prepare an input file (see Section 5)
- Create an `AsymmBand` object by reading in the input file using the `from_input` method
- (Optional) Read in additional data from other input files by using the `readstuff` method
- Compute necessary quantities by calling the `xe_q0` and `calc_band_param` methods
- Simulate and print the band, or obtain the band as an array for further use, by calling the `plot` method

To simulate a spectrum consisting of one or more bands, follow these steps:

- Create `AsymmBand` objects, each one corresponding to one band, using the steps above
- Create an `GaussSpectrum` object and add the `AsymmBand` objects to the `GaussSpectrum` object by using the `add_band` method
- (Optional) Read in an experimental or reference spectrum by using the `read_exp_spectrum` method
- Simulate a spectrum by calling the `gen_spectrum` method
- (Optional) Scale the simulated spectrum to minimize its mean squared difference to the read-in experimental or reference spectrum by calling the `scale_sim_to_exp` method
- Print the spectrum by calling the `print_spectrum` method

5. Input File

5.1. Overview

The contents of an input file are organized in blocks. Each block is identified by an identifier line in the format “`$`*`block_name`*” where *`block_name`* is the identifier of the block. The input data starts in the next line to the identifier line.

Lines starting with a “`#`” (comment lines) will be ignored.

5.2. Input blocks

`$evert`: Vertical excitation energy. It accepts a single floating-point number. The energy unit is specified in the `$unit` block.

`$forcees`: (Optional) Excited-state forces (negative of the gradients with respect to nuclear coordinates) at the ground-state equilibrium geometry. This is needed only if `$geomes` is not given. Input format is determined by `gradformat` (see `$misc` block).

`$freqes`: Excited-state frequencies and normal modes. In the FC-DHO approximations, these are the same as the ground-state values. Input format is determined by `freqformat` (see `$misc` block).

`$geomes`: (Optional) Excited-state equilibrium geometry. If not provided, it will be computed from the data provided in the `$geomgs`, `$forcees`, and `$freqes` blocks. Input format is determined by `geomformat` (see `$misc` block).

`$geomgs`: Ground-state equilibrium geometry. Input format is determined by `geomformat` (see `$misc` block).

`$misc`: (Optional) Miscellaneous settings. It accepts inputs in "*`keyword=value`*" format.

`keyword`	meaning	valid `value`
`freqformat`	input format for frequencies and normal modes (for the `$freqes` block)	`=g09` (default): As printed in the "Frequencies" section of the output file of a Gaussian 09 job with the "freq=hpmodes" keyword. `=adf`: As printed in the "Vibrations and Normal Modes" section of an ADF 2014 frequencies job.
`geomformat`	input format for geometries (for the `$geomgs`, `$geomes` blocks)	`=g09` (default): As printed in the "Standard orientation" section of a Gaussian 09 output file. Each row corresponds to one atom and there are 6 columns per row: [dummy] [atomic number] [dummy] [x] [y] [z] where [dummy] is a placeholder whose value will not be read in. `=general`: General format. Each row corresponds to one atom and there are 4 columns per row: [atomic symbol] [x] [y] [z]
`gradformat`	input format for forces (negative of gradients; `$forcees` block)	`=g09` (default): As printed in the "Forces" section of a Gaussian 09 output file. Each row corresponds to one atom and there are 5 columns per row: [dummy] [dummy] [x] [y] [z] where [dummy] is a placeholder whose value will not be read in.

`$natom`: Number of atoms. It accepts a single integer.

`$oscstr`: (Optional) Oscillator strength. It accepts a single floating-point number. This is needed only when you want to scale the simulated band by the oscillator strength.

`$plot`: (Optional) Range of photon energy to print the band. It accepts three numbers separated by whitespace: start energy (floating-point), end energy (floating-point), number of points in the range (integer). The energy unit is specified in the `$unit` block.

`$unit`: (Optional) Unit for the input energies in the `$evert` and `$plot` blocks. It accepts a single string of either "`eV`" (default) or "`cm_1`".

6. Output

The user can choose what to print out by using the `printstuff` and `plot` methods of `AsymmBand` and the `print_spectrum` method of `GaussSpectrum`. See the documentation of those methods in Section 8 for details.

7. Test Cases

The test cases are in the `test/` directory. Change to that directory first before executing the commands shown below. Note that these test cases only serve to show how the package works. Their results should not be used for scientific purposes.

7.1. Naphthalene $S_0 \rightarrow S_2$ transition

This test case shows how to use the `AsymmBand` class.

Execute

```
python example_main.py example_naphthalene.in >
example_naphthalene.out
```

and compare `example_naphthalene.out` to the provided `example_naphthalene.out.ref`.

7.2. Permanganate 1 $A_1 \rightarrow 1\ T_2$ transition

This test case shows how to use the `AsymmBand` class.

Execute

```
python example_main.py example_permanganate.in >
example_permanganate.out
```

and compare `example_permanganate.out` to the provided `example_permanganate.out.ref`.

7.3. $Fe(CO)_5$ low-lying transitions

This test case shows how to use the `GaussSpetrum` class.

First change directory to `FeCO5/`. Execute

```
python simulate.py > example_FeCO5.out
```

and compare `example_FeCO5.out` to the provided `example_FeCO5.out.ref`.

8. Technical Documentation of Classes `AsymmBand`, `GaussSpectrum`

class **AsymmBand**(__builtin__.object)

```
Class for computing overall vibronic shape of a single band.
```

Methods defined here:

__init__(self)

calc_band_param(self)

```
Calculate parameters needed for simulating band.
```

calc_band_point(self, omega, order=0, norm='one')

```
Compute one point of band at photon energy omega.

omega: photon energy in unit of self.unit
order: band shape model type.
        0 = Gaussian FC-DHO
       -1 = third-order FC-DHO
norm: normalization/scale.
      ='one': no scale
      ='oscstr': scale by oscillator strength
      ='tdm': scale by transition dipole moment squared
              times excitation energy
```

plot(self, mode='print', norm='oscstr', start=None, end=None, npoints=None, order=0)

```
Plot spectra or return an array containing the spectral data.

mode: 'print'= print the data for plot;
      'array'= return the data array.
norm: band multiplied by osc. str ('oscstr') or
      transition dipole moment squared ('tdm')
      times excitation energy.
start, end: range of plot.
npoints: number of points in the range.
order: band shape model type.
        0 = Gaussian FC-DHO
       -1 = third-order FC-DHO
return: None if mode='print';
        numpy array if mode='array',
          with col1 = energy, col2 = abs. strength
```

printstuff(self, stuff)

```
Print something.

stuff: string of what to print.
    'geomgs' = ground-state equilibrium geometry (x0)
    'geomes' = excited-state equilibrium geometry (xe)
    'dE' = difference of ES energy at x0 and xe
    'Q' = Dimensionless normal coordinate displacement x0->xe
    'bandparam' = band parameters
    'freq' = harmonic frequencies
    'header' = header
```

readstuff(self, fin, keyw)

```
Read in a piece of data.

fin: input file name
keyw: keyword
```

xe_q0(self)
```
Extrapolate from x0 to xe if necessary.
Calculate mass-weighted Cart. displacement q0.
Calculate deltaE = E_vert - E_adiab.
```

Class methods defined here:

from_input(cls, fin) from __builtin__.type
```
Read all stuff from input.

fin: string of input file name.
```

class **GaussSpectrum**(__builtin__.object)
```
Class of a container of AsymmBand to form full spectrum of a system.
```

Methods defined here:

__init__(self)

add_band(self, band)
```
Add a band to the spectrum.

band: AsymmBand object
```

gen_spectrum(self, xs=None, use_exp_xs=False, norm='oscstr', order=0)
```
Generate spectrum.

xs: a list of energy points at which the spectrum is generated.
use_exp_xs: whether use energy points of the exp. band as xs
norm: spectrum multiplied by osc. str ('oscstr') or
      transition dipole moment squared ('tdm')
      times excitation energy.
order: band shape model type.
        0 = Gaussian FC-DHO
       -1 = third-order FC-DHO
```

print_spectrum(self, which_spectrum='sim', energy_unit='eV', xs=None, use_exp_xs=False, norm='oscstr', order=0)
```
which_spectrum: which to print; 'sim' or 'exp' or 'scaled_sim'
energy_unit: energy unit in the print; 'eV' or 'cm_1' or 'nm'
xs: a list of energy points at which the spectrum is generated.
use_exp_xs: whether use energy points of the exp. band as xs
norm: spectrum multiplied by osc. str ('oscstr') or
      transition dipole moment squared ('tdm')
      times excitation energy.
order: band shape model type.
        0 = Gaussian FC-DHO
       -1 = third-order FC-DHO
```

read_exp_spectrum(self, filename, energy_unit='cm_1', normalize_to=None)
```
Read in an experimental spectrum from a file.

The file should have two columns,
where col 1 is energy and col 2 is absorption strength

filename: input file name of experimental spectrum
energy_unit: energy unit of experimental spectrum;
                   'cm_1' or 'eV' or 'nm'
normalize_to: whether and how to normalize the exp. spectrum
              ='unit_area': normalize to unit area
              ='unit_maxstr': normalize to unit max abs
              =None: don't normalize
```

scale_sim_to_exp(self)

```
Scale simulated spectrum to experimental to minimize
the mean squared difference.

return: final mean squared difference
```

Static methods defined here:

calc_spectrum_area(spectrum)

normalize(spectrum, normalize_to)

```
Normalize spectrum to unit area (regardless of energy unit).

spectrum: npoints * 2 array; col1 is energy, col2 is strength
normalize_to: how to normalize
    'unit_area' = normalize to unit area
    'unit_maxstr' = normalize maximum strength to one
return: npoints * 2 array of normalized spectrum
```