Scientific Programming

Practical exercises on Python - II

Julien Bloino Marco Mendolicchio

May 25, 2020

Introduction

The exercises in this session are intended to be done in Jupyter Notebooks.

This session uses several features of **Matplotlib** and **NumPy**. Information on the functions can be obtained either on Internet (respectively, `https://matplotlib.org/3.2.1/contents.html` and `https://docs.scipy.org/doc/numpy/user/index.html`) or within the notebook, from the autocompletion or the magic commands to invoke the documentation.

Exercise 1

This exercise extends the work done in **exercise 5** of the "basic exercises".

As a reminder, the Joint Committee on Atomic and Molecular Physical Data exchange (JCAMP-DX, `http://www.jcamp-dx.org`) format is a file format specification proposed by the International Union of Pure and Applied Chemistry (IUPAC) committee to facilitate the exchange of spectroscopic data, primarily related to Nuclear Magnetic Resonance (NMR). This format has been adopted by various organizations, for instance the American National Institute of Standards and Technology (NIST, `http://nist.gov`), which uses it as the export format for the spectra stored in their database (`http://webbook.nist.gov/chemistry/`). This format is not readable directly by most plotting programs, and must be converted to a more common format. Such a format is provided by the comma-separated (alternatively, character-separated) values (CSV) file format, where data are stored as columns separated by a specific type of characters (plotters may differ on this part). Here, a graphical plotter will be built.

Note: the energies and intensities are stored in a multi-column format, with the following layout below "`XYDATA`":

$$\begin{array}{lllllll} x_i & & y_i & y_{i+1} & y_{i+2} & \cdots & y_{i+m} \\ x_{i+m+1} & & y_{i+m+1} & y_{i+m+2} & & \cdots & y_{i+2m} \end{array}$$

Note: m may vary between JDX files. *All y values correspond to the **same** spectrum.*

Reference file: `naphthalene-IR.jdx`

Exercise 1.1:

Write a logic offering a "developer" and "user" modes. In the developer mode, the runtime parameters need to be set in the code. In the user mode, options should be provided to set up the input filename and any option mentioned in the following sub-exercises, using `argparse`.

Note: Remember that `argparse` does not need to process an actual commandline.

Exercise 1.2:

Plot the spectrum contained in the JDX file (note that you will need to know the step between x points to generate the full spectrum).

Exercise 1.3:

Give the possibility to convert the x axis values between the following units: μm, eV and cm^{-1}.

In this version, only the unit provided in the reference file needs to be supported (`XUNITS`).

The conversion from cm^{-1} to the supported units are:

- 1 cm^{-1} = 10^{-4} μm^{-1}
- 1 cm^{-1} = 8065.5 eV

Remember to check that the conversion between cm^{-1} and μm is valid!

Exercise 1.4:

Experimental infrared spectra are commonly registered in transmittance, but the molecular properties are related to the absorbance.

Assuming that the absorbance A and the transmittance T are related as $T = e^{-A}$, give the possibility to convert the y values to absorbance if the default "`YUNIT`" is "transmittance".

For absorbance spectra, give the possibility to normalize the spectrum, setting the highest value of y to 1.

Exercise 1.5:

Add the possibility to define a range of the x axis (lower and upper bounds) to be plotted, checking that the user or internally set values are within the range covered in the JDX file.

Exercise 1.6:

Add the possibility to interpolate the spectrum, the user providing their own step between points of the discretized spectrum.

Note: the spline interpolation of order 1 is simply linear.

Exercise 2

Electronic spectra are usually measured by exposing a sample to a radiative source with a wavelength within the visible and/or ultraviolet (UV) range. Depending on the sample and the incident energy, multiple electronic transitions can occur. Computationally, it is possible to simulate some of those spectra by evaluating the transition moments of the relevant properties between the ground and each electronic state within the energy range of interest. Here, we will be interested in two spectra, called for the sake of simplicity OP and CD. It will be assumed that the intensity of those spectra (y) at a given point of the x axis is given as,

$$\text{OP: } y = \frac{10\pi\mathcal{N}_A}{3\ln(10)\epsilon_0\hbar c}\sum_{s=1}^{N_s} E_s|\boldsymbol{\mu}_s|^2 f(E_s - x)$$

$$\text{CD: } y = \frac{40\pi\mathcal{N}_A}{3\ln(10)\epsilon_0\hbar c^2}\sum_{s=1}^{N_s} E_s\Im\{\boldsymbol{\mu}_s\boldsymbol{m}_s\} f(E_s - x)$$

with,

$\mathcal{N}_A$ Avogadro constant (1/mol)

ϵ_0 Vacuum permittivity ($\epsilon_0 = 1/4 * 10^{-7}\pi c^2$, C^2/J/m)

$\hbar$ Reduced Planck constant ($\hbar = h/2\pi$)

c Speed of light (m/s)

E_s Transition energy from the ground to electronic state s

$\boldsymbol{\mu}_s$ Transition moment of the electric dipole between the ground and electronic state s (C m)

$\boldsymbol{m}_s$ Transition moment of the magnetic dipole between the ground and electronic state s (J/T)

f Broadening function

In this exercise, the data will be extracted from the formatted checkpoint file, a file exchange format used by the suite of quantum chemical programs Gaussian to facilitate migration of data between major revisions or with different programs. The data are expected in atomic units and can be mostly sorted in 2 categories:

scalars with the form: "*title* *type* *value*"

arrays with the form: "*title* *type* `N=` *length*"

where

title is the field name

type is the data type (`I` for integer, `R` for real, `C` for characters...)

value actual value (for scalars)

length number of elements of the vectors or arrays

Vectors or arrays are stored in Fortran order, using 5 columns for floating point numbers (Fortran format, "`E16.8`"), and 6 column for integers (Fortran format, "`I12`").

In the following, the sections of interest have the keyword "`ETran`" in the *title* name, plus "SCF ENERGY" for the ground-state energy (to compute E_s):

`'SCF Energy'` Ground-state energy E_0

`'ETran scalars'` List of integers, with first element corresponding to N_s

`'ETran state values'` List of at least ($N_s \times 16$) real numbers, starting with the following data for each electronic state s:

1. Excited-state energy, $E'_s = E_0 + E_s$
2. x component of the electronic transition moment of the electric dipole (length gauge), $\boldsymbol{\mu}_s^L$
3. y component of the electronic transition moment of the electric dipole (length gauge), $\boldsymbol{\mu}_s^L$
4. z component of the electronic transition moment of the electric dipole (length gauge), $\boldsymbol{\mu}_s^L$
5. x component of the electronic transition moment of the electric dipole (velocity gauge), $\boldsymbol{\mu}_s^V$
6. y component of the electronic transition moment of the electric dipole (velocity gauge), $\boldsymbol{\mu}_s^V$
7. z component of the electronic transition moment of the electric dipole (velocity gauge), $\boldsymbol{\mu}_s^V$
8. x component of the electronic transition moment of the magnetic dipole, $\boldsymbol{m}'$]
9. y component of the electronic transition moment of the magnetic dipole, $\boldsymbol{m}'$]
10. z component of the electronic transition moment of the magnetic dipole, $\boldsymbol{m}'$]
11. xx component of the electronic transition moment of the quadrupole tensor, $\boldsymbol{Q}'$]
12. yy component of the electronic transition moment of the quadrupole tensor, $\boldsymbol{Q}'$]
13. zz component of the electronic transition moment of the quadrupole tensor, $\boldsymbol{Q}'$]
14. xy component of the electronic transition moment of the quadrupole tensor, $\boldsymbol{Q}'$]
15. xz component of the electronic transition moment of the quadrupole tensor, $\boldsymbol{Q}'$]
16. yz component of the electronic transition moment of the quadrupole tensor, $\boldsymbol{Q}'$]

For the spectra, we will use $\boldsymbol{\mu}_s^L = \boldsymbol{\mu}_s$ and $\boldsymbol{m}'_s = 2i\boldsymbol{m}_s$ (note that $\boldsymbol{m}_s$ is purely imaginary).
Reference files: `RR-dimethyloxirane_S3_frq.fchk`, `gxx_units.py`, `plotter.py`

Exercise 2.1:

Implement developer and user modes (as **Exercise 1.1**).

Exercise 2.2:

Add the possibility to extract all transition-related properties (beware of the definition of E_s).

Exercise 2.3:

Add the support for "OP" and "CD" spectroscopy. `PhyCon` can be loaded from `gxx_units.py`

Note that $\boldsymbol{\mu}_s$ and $\boldsymbol{m}_s$ must be converted to SI units,

- For $\boldsymbol{\mu}_s$, the conversion factor is "`e2C*PhyCon(1)*1.0e-10`" with "`e2c = PhyCon(3)/PhyCon(9)*10.0`"
- For $\boldsymbol{m}_s$, the conversion factor is "`hbar*e2C/PhyCon(12)`", with "`hbar = PhyCon(4)/(2.0*pi)`"

Exercise 2.4:

Add the possibility to choose a range of electronic states to include (lowest and highest ones), and check that the user-defined bounds are valid (highest electronic state lower or equal to N_s, lowest greater or equal to 1).

Exercise 2.5:

Add the possibility to choose the unit for the x axis between cm^{-1}, nm and eV (check the conversion from **Exercise 1.3**).

Note: The conversion from Hartree to cm^{-1} is "`PhyCon(8)/(PhyCon(4)*PhyCon(9))`".

Exercise 2.6:

Implement the support for Lorentzian and Gaussian distribution functions. The half-width at half-maximum (HWHM) should be set depending on the unit of x and modifiable by users.

Note: The magic command `%load` can be used to directly load the function from `plotter.py`

Exercise 2.7:

Add support for user-defined lower and upper energy bounds to display the spectrum, and the step for the discretization of the spectrum.

By default, the lower and upper bounds should be calculated based on the range of electronic states to include, considering that most of the area of a Gaussian function is recovered within a range of $2 \times 3.5 \times$ HWHM, and $2 \times 13 \times$ HWHM for a Lorentzian function.

Exercise 3

This exercise deals with data processing related to vibrationally-resolved electronic spectroscopy in Gaussian. Due to the sensitivity of the theoretical model used to simulate such spectra, intermediate data are produced and can be relevant to assess the reliability of the results.

Vibrationally resolved electronic spectra are computed here as an ensemble of transitions between vibrational states of 2 distinct electronic states. A different basis set is used for each electronic state, here named $\{\boldsymbol{Q}'\}$ and $\{\boldsymbol{Q}''\}$, which are related through an affine transformation called Duschinsky transformation,

$$\boldsymbol{Q}' = \boldsymbol{J}\boldsymbol{Q}'' + \boldsymbol{K},$$

to compute the integral $\int \psi(\boldsymbol{Q}')\boldsymbol{\mu}_s\psi(\boldsymbol{Q}'')^* d\boldsymbol{Q}' d\boldsymbol{Q}''$. $\boldsymbol{J}$ is called the Duschinsky matrix and $\boldsymbol{K}$ the shift vector.

The validity of the transformation and the quality of the overall calculation are strongly related to the shift vector (which should be small) and the similarity of the Duschinsky matrix with the identity matrix (the more similar it is, the better).

Finally, the convergence of the calculations is given by the "spectrum progression".

In this exercise, a Gaussian output file will be parsed and processed to extract, analyze and display the properties selected by the user.

Reference file: `vibronic.log`

Exercise 3.1:

Implement developer and user modes (as **Exercise 1.1**) to have the possibility to choose a Gaussian output file.

Exercise 3.2:

Extract the full version (major and minor) of GAUSSIAN used to run the calculation and print it in output. The version can be found in a block of the form,

```
*****************************************
Gaussian 16:  ES64L-G16RevB.01 20-Dec-2017
               30-May-2019
*****************************************
```

where the second line contains all relevant information.

Exercise 3.3:

Extract and plot the final spectrum, displayed after

```
    ------------------------------------------------
                     Final Spectrum
    ------------------------------------------------
```

Add the possibility to extract the assignment information, contained in the block after,

```
    ------------------------------------------------
                Information on Transitions
    ------------------------------------------------
```

The assignment information have the form,

```
Energy =  -3102.2876 cm^-1: |0> -> |30^1>
  -> Intensity = -.2174     (RotStr = -.1938E-03)
```

where "`Energy`" is the transition energy relative to the reference (0-0) transition, whose energy is given after: "`Energy of the 0-0 transition:`".

The energies must be absolute to match the final spectrum. The other relevant information is after "`Intensity =`"

Display the assignment information as sticks on the final spectrum using the "`vlines`" function of **Matplotlib**.

Exercise 3.4:

Add the possibility to print the progression of the calculations instead, given after the keywords "`Spectrum progression:`". The progression is printed as a cumulative value (progression at step $i+1$ contains the progression up to step i). Print a warning message if the final value is below 60%.

Display the progression as a bar chart ("`bar`" function of **Matplotlib**), with the step-specific (non-cumulative) progression printed as a curve.

Exercise 3.5:

Add the possibility to display instead the shift vector, printed after,

```
Shift Vector
------------
```

as horizontal bars ("`barh`" function of **Matplotlib**).

Since the most important information is the magnitude of the shift, the absolute value of the displacements should be printed.

Extra: Use different colors to display negative and positive shifts in the final spectrum (hint: look for the `set_color` method).

Exercise 3.6:

Add the possibility to display instead the Duschinsky matrix, printed after,

```
Duschinsky matrix
-----------------
```

Note: the block is always terminated by an empty line (or containing at most 1 space).

The Duschinsky matrix is orthogonal (reminder: $\boldsymbol{J}^T = \boldsymbol{J}^{-1}$). Implement 2 distinct tests to check that the matrix is orthogonal (remember that the displayed precision is limited).

An interesting property is that the sum of the squared elements along a row or a column is 1. Print the matrix of squared elements of $\boldsymbol{J}$, using a heat map (for instance, the "`matshow`" function of **Matplotlib**).

Add the possibility instead to display the distribution of squared terms (between 0 and 100%) along a given list of rows or columns (the two options should be mutually exclusive). If such an option is given by the user, histograms displaying the distribution of squared terms should be printed instead of the matrix (1 histogram per row/column, "`hist`" function of **Matplotlib**). The user should have the possibility to set the size of the bins, with a default value of 10%.

Exercise 4

The least-squares (LS) method is a procedure to find the approximated solution of an over-determined system, in which the number of equations (m) is larger than that of unknowns (n):

$$\boldsymbol{C}\boldsymbol{x} = \boldsymbol{d}$$

where $\boldsymbol{C} \in \mathbb{R}^{m\times n}$ is the matrix of coefficients, $\boldsymbol{x} \in \mathbb{R}^n$ is the vector of unknowns and $\boldsymbol{d} \in \mathbb{R}^m$ is the vector of constant terms.

The minimization of the residual function $R(\boldsymbol{x})$

$$R(\boldsymbol{x}) = |\boldsymbol{d} - \boldsymbol{C}\boldsymbol{x}|^2$$

with respect to all unknowns ($x_i,\ i = 1, \ldots, n$) yields the following expression,

$$\boldsymbol{C}^T\boldsymbol{C}\boldsymbol{x} = \boldsymbol{C}^T\boldsymbol{d}$$

Upon inversion of the matrix product $\boldsymbol{C}^T\boldsymbol{C}$, the vector $\boldsymbol{x}$ is obtained as,

$$\boldsymbol{x} = (\boldsymbol{C}^T\boldsymbol{C})^{-1}\boldsymbol{C}^T\boldsymbol{d} = \boldsymbol{C}^\dagger\boldsymbol{d}$$

where $\boldsymbol{C}^\dagger = (\boldsymbol{C}^{\mathrm{T}}\boldsymbol{C})^{-1}\boldsymbol{C}^{\mathrm{T}}$ is the pseudo-inverse of the matrix $\boldsymbol{C}$.

In this exercise, the LS method is applied to the study of the kinetics of a chemical reaction, where a species A decomposes to molecules B and C as follows,

$$2\mathrm{A} \longrightarrow 2\mathrm{B} + \mathrm{C}$$

The concentration (quantity of species per unit of volume) $C_{\mathrm{A}}(t)$ of the species A at each instant t can be expressed through the following equation:

$$\frac{1}{C_{\mathrm{A}}(t)} = \frac{1}{C^0_{\mathrm{A}}} + kt$$

where $C^0_{\mathrm{A}} = C_{\mathrm{A}}(t = 0)$ is the initial concentration of A (known), and k is the so-called rate constant, which can be determined by a LS calculation starting from a set of experimental data, which are the concentrations of A at different selected instants.

Reference file: `kinetics.dat`

Exercise 4.1:

Implement a developer and user modes as done previously to set the name of the file containing the values of the concentrations of A. If the file does not exist, print an error message. Extract the data from the file and build two arrays containing the times and the corresponding concentrations, respectively. At least two concentrations (and the corresponding times) are supposed to be given, otherwise print an error message. Finally check that the data set contains the concentration at $t = 0$ (C^0_{A}).

Plot $C_{\mathrm{A}}(t)$ with respect to t.

Exercise 4.2:

Add the calculation of the rate constant k through the LS method, following the steps reported below:

- Compute the matrix **C**.
  Note: in this case the first column should be equal to 1 ($C_{i1} = 1$ with $i = 1, \ldots, m$), while the second one is given by the times corresponding to the different points ($C_{i2} = t_i$ with $i = 1, \ldots, m$ and t_i being the time corresponding to the i^{th} experimental data).
- Compute explicitly the pseudo-inverse $\mathbf{C}^{\dagger}$.
- Compute the vector of unknowns **x**. One of the elements is supposed to be equal to $1/C_A^{(0)}$, while the other one is k. Print the rate constant with at least eight digits of precision.

Exercise 4.3:

The **NumPy** module provides user-friendly alternative methods to solve the LS problem instead of computing explicitly the pseudo-inverse.

Solve the LS problem and compute the rate constant k by employing:

- `numpy.linalg.lstsq`: performs the resolution of the LS problem starting from the matrix **C** and the vector **d**.
- `numpy.polyfit`: this method fits a generic set of data with a polynomial of any degree.

Check that the results agree with each other, and with the value of k obtained in **Exercise 4.2**.