京都大学大学院理学研究科附属天文台技報
Technical Reports from Astronomical Observatory
Graduate School of Science, Kyoto University

Vol.7-1

SPECTRA: A flexible non-LTE radiative transfer Python library, I. Solver for statistical equilibrium and line formation with slab model

Yuwei Huang[*1] and Kiyoshi Ichimoto[*1]

2023年 7月12日 first version

[*1]Astronomical Observatory, Kyoto University

SPECTRA:

A flexible non-LTE radiative transfer Python library, I. Solver for statistical equilibrium and line formation with slab model

Yuwei Huang *[1] and Kiyoshi Ichimoto †[1]

[1]Astronomical Observatory, Kyoto University, Kitashirakawa-oiwake-cho, Kyoto 606-8502, Japan

Abstract

Non-LTE radiative transfer numerical simulation is the key to understand the physics behind the non-linearly coupled system consisting of the solar radiation and the plasma in solar atmosphere. A highly flexible library (hereafter, *SPECTRA* for short) for Non-LTE radiative transfer numerical simulation was developed in Python programming language, for the purpose of solving problems scaling from statistical equilibrium, cloud model to non-LTE radiative transfer. This library is also focusing on interactively interpreting the observed spectroscopy data of the Sun and experimental plasma.

Keywords: non-LTE, radiative transfer, statistical equilibrium, solar spectrum

1 Introduction

Non-LTE radiative transfer is an essential ingredient for interpreting spectroscopic data of astronomical objects. Numerically it is described by a set of non-linear equations, where the radiation field and the physical quantities of local plasma are coupled with each other.

To date, Non-LTE radiative transfer codes are most developed in compiling language, such as C/C++, Fortran, for the sake of performance and memory management (e.g. MULTI,[1] PROM7,[2] RH[3]). In this case, code is packaged as an end-to-end program, where users are hardly allowed to probe the intermediate output of the simulation or to make some modification to the source code; compilation is needed whenever modification was made which is also known as "Ahead-

*kou@kusastro.kyoto-u.ac.jp
†ichimoto@kwasan.kyoto-u.ac.jp

of-Time"(AOT) compilation. Recently, as CPUs become more and more powerful and the needs of interaction between numerical simulation and real world data analysis, interpreter language including Python and Julia, etc, comes into play in the field of scientific computing due to a new compilation technique, "Just-in-Time"(JIT) compilation, making it possible for interpreter language to run as fast as compiling language.

SPECTRA is developed for the purpose of reproducing spectral data observed on the sun, on other stars and in laboratory plasma and visualizing internal atomic/radiative processes to understand the physics behind the formation of spectra. *SPECTRA* has the following characteristics:

1. **Easy installation and execution:** Python is a cross-platform language that the library are available in all Windows, MacOS and Linux without making any modification (subsection 3.1). Simulation also could be performed through browser using the cloud service without installation on local computer.

2. **Easy accessing to the observation data:** Python provides various packages for data exchanging among almost all kinds of scientific data formats, including standard *fits*, *hdf5*, *netCDF*, and IDL *.sav* formats.

3. **Flexibility of customizing a user-defined simulation:** with an understanding of the structure of *data class* and *high level function* (subsection 3.3), a user-defined atomic model and simulation could be easily constructed .

In the following sections, we describe the mathematical formulas and equations in section 2, the library architecture in section 3, a standard solar background radiation intensity profile in various wavelength range in section 4. Examples of customizing simulation with different kinds of physics model are shown in section 5. Future prospect and the extendability of this library are concluded in section 6.

2 Formulas and Equations

in *SPECTRA*, we adopt CGS system of unit, and the specific intensity $I(\mathbf{x}, t; \mathbf{n}, \omega)$ of radiation at position $\mathbf{x}$ $[cm]$, time t $[s]$, travaling in direction $\mathbf{n}$ through unit area $[cm^2]$, into a unit solid angle $[sr]$, having a wavelength in the unit range $[cm]$ is in an unit of $[erg/cm^2/sr/cm/s]$.

2.1 Radiative process

In case of bound-bound transition, considering the transition between lower level i to upper level j, given the ambient mean intensity J_λ, the radiative de-decitation and excitation rates[4] are described by

$$R_{ji} = A_{ji} + B_{ji} \int_0^\infty \psi_{ji}(\lambda) J_\lambda d\lambda \tag{1}$$

$$R_{ij} = B_{ij} \int_0^\infty \phi_{ji}(\lambda) J_\lambda d\lambda \tag{2}$$

where ψ_{ji} is the emission profile; ϕ_{ij} is the absorption profile. Currently only CRD (Complete frequency redistribution) is available, so we have

$$\psi_{ji}(\lambda) \equiv \phi_{ij}(\lambda) \tag{3}$$

For hydrogen atom, Einstein coefficient A_{ji} could be calculated by using theoretical formula.[5] Otherwise, Einstein coefficient A_{ji} is given by experiment results, and the corresponding Einstein coefficient B_{ji} and B_{ij} are calculated using the Einstein relation[4]

$$B_{ji} = A_{ji}/(2hc^2\ \lambda^5) \tag{4}$$

$$B_{ij} = B_{ji}\frac{g_j}{g_i} \tag{5}$$

where g_i and g_j are the statistical weight of the lower and upper levels, respectively.

In case of bound-free transition (photoionization), consider the transitions between lower level i and upper continuum level j, given the ambient mean intensity J_λ, electron temperature T, photoionization cross section $\alpha_{ij}(\lambda)$, the normalized LTE energy level population fraction n_i^{LTE} and n_j^{LTE}, the radiative transition rates for a single atom are described by

$$R_{ij} = 4\pi \int_0^{\lambda_0} \alpha_{ij}(\lambda)(hc/\lambda)^{-1} J_\lambda d\lambda \tag{6}$$

for radiative ioniozation,[4]

$$R_{ji}^{stim} = \frac{n_i^{LTE}}{n_j^{LTE}} 4\pi \int_0^{\lambda_0} \alpha_{ij}(\lambda)(hc/\lambda)^{-1} J_\lambda e^{-hc/kT\lambda} d\lambda \tag{7}$$

for stimulated radiative recombination,[4]

$$R_{ji}^{spon} = \frac{n_i^{LTE}}{n_j^{LTE}} 4\pi \int_0^{\lambda_0} \alpha_{ij}(\lambda)(hc/\lambda)^{-1} (2hc^5/\lambda^3) e^{-hc/kT\lambda} d\lambda \tag{8}$$

for spontaneous radiative recombination.[4]

The sources of Einstein coefficient A_{ji} and photoionization cross section α_{ij} for the atomic model are shown in Table 1.

2.2 Collisional process

Given the ambient electron temperature T, electron density N_e and the effective collisional strength Ω_{ij}, the collisional excitation rate[4] is described by

$$N_e C_{ij} = N_e \frac{8.6 \times 10^{-6}\ \Omega_{ij}}{g_i T^{0.5}} e^{-dE_{ij}/kT} \tag{9}$$

and the collisional ionization rate[4] is described by

$$N_e C_{ij} = N_e \Omega_{ij} T^{0.5} e^{-dE_{ij}/kT} \tag{10}$$

where dE_{ij} is the excitation/ionization energy for the transition between levels i and j.

In both cases, the downward collisional rate coefficient C_{ji} follows the relationship of detailed balance[4]

$$C_{ji} = C_{ij} \frac{n_i^{LTE}}{n_j^{LTE}} \tag{11}$$

The sources of effective collisional strength Ω_{ij} of the our atomic model are listed in Table 1.

Table 1: source of atomic data for atomic models. TC: theoretically calculated;[5] OP: Opacity Project;[6] NIST: NIST Atomic database;[7] RH: atomic data in RH code[3]

	A_{ji}	α_{ij}	Ω_{ij}
H	TC	TC	TC
He	NIST	RH	RH
Ca	NIST	OP+RH	RH

2.3 Statistical equilibrium

Given the electron density N_e, the radiative transition rates R_{ij}, R_{ji} and the collisional transition rates C_{ij}, C_{ji}, the energy level population n_i , which is normalized by the number density of the atom in interest, is calculated by solving a set of kinetic equilibrium equations[4]

$$n_i \sum_{j \neq i} (R_{ij} + C_{ij}) = \sum_{j \neq i} n_j (R_{ji} + C_{ji}) \tag{12}$$

Since the set of above system of equations is not linearly independent, the equation belonging to the highest level is replaced by

$$\sum_i n_i = 1 \tag{13}$$

3 About the library

3.1 Installing the library

The easiest way to install this library is using *git* to pull the latest version of *SPECTRA*[8] and *conda*[9] to manage the Python environment.

1. create a new environment using *conda* with command

   ```
   $ conda create -n spectra python=3.10
   ```

   replace the environment name *spectra* to whatever you like. The Python version is required to be larger than 3.9.

2. make the path of spectra directory visible to this Python environment

   ```
   $ echo /path/to/spectra/ > "/path/to/miniconda3/envs/spectra \
   ```

```
        /lib/python3.10/site-packages/module.pth"
```

3. activate environment

```
        $ conda activate spectra
```

4. install minimum third-party dependency into current environment

```
        $ pip install numba numpy scipy debtcollector ipykernel
```

5. make the python kernel in this environment visible to jupyter notebook

```
        $ ipython kernel install --user --name spectra
```

For detailed installation instrcutions in Unix and Windows Systems, refer to the *install.txt* located in github repository.[8]

3.2 library architecture

The *SPECTRA* library consists of four main folders (Listing 1).

- *data/*: contains input data and configuration files.
  - *atmos/*: contains the atomospheric model, e.g., FAL and user defined models.
  - *atom/*: contains the atomic data files for atoms, in a folder named as <*element*>_<*ionization-stages*> for each atom. See documentation[10] for details.
  - *intensity/atlas/QS/*: contains the binary file of the intensity profile and wavelength mesh of solar background radiation. See section 4 for details.
  - *conf/*: contains configuration files, which defines the paths to the atomic data files. Required as an input argument to initialize a specific atom object. See section 5 for details.
- *notebooks/*: jupyter notebook gallery of simulations and analysis.
- *spectra_src/*: contains several folders of source code. The details of the data structures and functions could be found in documentation.[10]

- *Atomics/*: module of basic atomic process, e.g., Boltzman distribution, electron collisional excitation coefficient, etc.
- *Math/*: module of mathematical formulas, e.g., special functions, numerical integral methods, etc.
- *RadiativeTransfer/*: module of numerical radiative transfer methods.
- *Struct/*: module of data structures, e.g., *Atom*, *Atmosphere*, etc.
- *Util/*: module of utilities, e.g., functions to import atomic data from RH code and NIST atomic database, etc.
- *Visual/*: module of visualization, e.g., transition rate heat map, Grotrian diagram, etc.
- *Experimental/*: temporary folder for experimental features and functions to maintain the backward compatibility.
- *Functions/*: high level functions, e.g., statistical equilibrium simulation, non-LTE slab model simulation (cloud model in case of homogeneous slab)

- *test/*: examples in python scripts, benchmarks of optimization with using *numba* library, and unittest.

3.3 Design pattern

The source code mainly consists of three components:

1. data class
2. low level function
3. high level function

Data class is a structured data type, such as an "Atom" class (Listing 2), which carries the array of level parameters, array of line/continuum transition parameters and etc. To perform a simulation, we need the following four kinds of data classes:

1. *Atom* (Listing 2) : atomic data (energy level and transition data, etc.).
2. *Wavelength_Mesh* (Listing 4) : wavelength mesh of line/continuum transitions.

3. *Atmosphere0D* (Listing 5) : plasma physical parameters. 0D dimension model as for a uniform slab; Infinite slab 1D dimension model where spatial variation is considered.

4. *Radiation* (Listing 3) : Solar spectrum Atlas (as background or ambient radiation) and its wavelength mesh

Listing 1: main directory architecture in tree view

```
|---data
    |---atmos
        |---FAL
    |---atom
        |---Ca_I-II-III
        |---Ca_II
        |---C_III
        |---H
        |---He
        |---O_V
        |---Si_III
        |---...
    |---conf
    |---intensity
        |---atlas
            |---QS
|---notebooks
|---spectra_src
    |---Atomic
    |---Experimental
    |---Function
        |---SEquil
        |---SlabModel
    |---Math
    |---RadiativeTransfer
    |---Struct
    |---Util
        |---AtomicDataUtils
            |---RH2Spectra
```

```
            |---NIST2Spectra
        |---AtomUtils
    |---Visual
|---test
    |---examples
    |---numba
    |---unittest
|---...
```

Listing 2: Atom class

```
@_dataclass(**STRUCT_KWGS)
class Atom:

    Z       : T_INT   # atomic number
    Mass    : T_FLOAT # atomic mass
    Abun    : T_FLOAT # solar abundance

    nLevel : T_INT   # number of levels
    nLine  : T_INT   # number of line transition
    nCont  : T_INT   # number of continuum transition
    nTran  : T_INT   # number of total transition

    # "Radiative" transitions is defined in data file *.RadiativeLine
    # where type of line shape, number of wavelength mesh, etc.
    # are customized.
    nRL    : T_INT   # number of "Radiative" transition

    Level  : T_ARRAY    # struct array of Level infomation
    Line   : T_ARRAY    # struct array of Line transition infomation
    Cont   : T_ARRAY    # struct array of Continuum transition information

    _has_continuum      : T_BOOL # whether has continuum

    _atomic_data_source : ATOMIC_DATA_SOURCE # experiment or theoretical
    _atom_type          : T_E_ATOM # hydrogen or normal atom
```

```
    # ctj : configuration, term, angular momentum J
    _ctj_table : CTJ_Table   # table of ctj of Levels/Transitions
    _idx_table : Index_Table # table of index <-> ctj

    # struct of Collisional excitation data
    CE         : Collisional_Transition
    # struct of Collisional ionization data
    CI         : Collisional_Transition
    PI         : Photo_Ionization # struct of photoionization data
    # struct of mesh information("Radiative" line)
    RL         : Radiative_Line
```

Listing 3: Radiation class

```
@_dataclass(**STRUCT_KWGS_UNFROZEN)
class Radiation:

    backRad : T_ARRAY  # 2d, (2, n_wavelength)

    PI_intensity : T_ARRAY  # 2d, (nCont, _N_CONT_MESH )
```

Listing 4: Wavelength_mesh class

```
@_dataclass(**STRUCT_KWGS_UNFROZEN)
class Wavelength_Mesh:

    ##: initialized given the Atom.Cont
    #   could be modified due to doppler shift
    #   currently, we assume that Cont_mesh will
    #   not be affected by doppler shift
    Cont_mesh       : T_ARRAY # 2d
    Cont_Coe        : T_ARRAY # struct

    ##: initialized given the atmosphere model
    # 1d, doppler width unit, without doppler shift
    Line_mesh       : T_ARRAY
```

```
    Line_mesh_idxs   : T_ARRAY # 2d,   (nLine, 2)
    # 1d, un-normalized. `/= dopWidth_cm` --> normalized
    Line_absorb_prof: T_ARRAY

    Line_Coe         : T_ARRAY # struct

    ##: we need Line_mesh_share to deal with non-uniform atmoshere
    Line_mesh_share      : T_ARRAY # 1d,
    Line_mesh_share_idxs : T_ARRAY # 2d,   (nLine, 2)
```

Listing 5: Atmosphere class

```
@_dataclass(**STRUCT_KWGS_UNFROZEN)
class Atmosphere0D:

    Nh : T_FLOAT   # hydrogen number density, [cm^{-3}]
    Ne : T_FLOAT   # electron number density, [cm^{-3}]
    Te : T_FLOAT   # electron temperature, [K]
    Vd : T_FLOAT   # doppler velocity, [cm/s]
    Vt : T_FLOAT   # micro turbulence velocity, [cm/s]
    Ti : T_FLOAT = -1.0    # ion temperature, [K]
    # if Ti smaller than 0, use Te instead

    ndim : T_INT     = 0   # dimension
    is_uniform : T_BOOL = True   # is spatial uniform
    Tr : T_FLOAT     = 6.E3      # radiation temperature, [K]
    # whether to use Tr to calculate background radiation
    use_Tr : T_BOOL  = False
    Pg : T_FLOAT     = 0.05   # [Ba] = 0.1 [Pa] gas pressure
    # whether to utilize doppler shift in continuum
    doppler_shift_continuum : T_BOOL = False
```

A low level function is a function that takes a native data type, such as integer, float, double array, etc., as input argument, while a high level function is a function that normally takes a data class as input argument and calls the corresponding low level functions to process the data contained in the data class. The distinction between low level and high level functions is nec-

essary, because the "JIT" optimization is currently only available to functions with native data type (including *numpy* array) as input arguments, since a data class is much more complicated than an array to be analyzed when being "compiled" into machine code in the runtime.

For example, a low level function *spectra_src.Atomic.SEsolver.solve_SE_* (Listing 6) to solve statistical equilibrium takes the radiative transition matrix R and collisional transition matrix C as input. While the corresponding high level function *spectra_src.Function.SEquil.SELib.cal_SE_* (Listing 7) takes the data classes as inputs, and then calls the corresponding low level function for detailed calculation (1. radiative and collisional transition rates in bound-bound and bound-free transitions; 2. solving linear equations for statistical equilibrium), and finally returns the data classes including the level population and transition rate as a result.

Listing 6: low level function for solving statistical equilibrium

```
def solve_SE_(Rmat : T_ARRAY, Cmat : T_ARRAY) -> T_ARRAY:


    nLevel = Rmat.shape[0]
    A = Cmat[:,:] + Rmat[:,:]
    b = _numpy.zeros(nLevel, dtype=DT_NB_FLOAT)


    #------------------------------------------------------------
    # diagnal components
    #------------------------------------------------------------
    for k in range(nLevel):
        A[k,k] = -A[:,k].sum()


    #------------------------------------------------------------
    # abundance definition equation
    #------------------------------------------------------------
    A[-1,:] = 1.
    b[-1] = 1.


    nArr = _numpy.linalg.solve(A, b)


    return nArr
```

Listing 7: high level function for solving statistical equilibrium

```
def cal_SE_(atom : _Atom.Atom, atmos : _Atmosphere.Atmosphere0D,
            wMesh : _WavelengthMesh.Wavelength_Mesh,
            radiation : _Radiation.Radiation,
            Nh_SE : T_UNION[T_ARRAY, None],
            ):
    #[the body of the function is omitted due to space limitation]

    # radiative bound-free transition
    Rik, Rki_stim, Rki_spon = _bf_R_rate_( Cont, Cont_mesh[:,:],
        Te, nj_by_ni_Cont[:], alpha_interp[:,:], PI_intensity[:,:],
       backRad[:,:], Tr, use_Tr, doppler_shift_continuum)
    # radiative bound-bound transition
    Bij_Jbar, Bji_Jbar, wave_mesh_cm_shifted_all, absorb_prof_cm_all, \
    Jbar_all  = _B_Jbar_(Line, Line_mesh_Coe, Line_mesh[:],
        Line_mesh_idxs[:,:],Te, Vt, Vd, Ne, Nh_I_ground, Mass,
        atom_type, backRad[:,:], Tr, use_Tr)
    # collisional bound-bound and bound-free transition
    Cij = _get_Cij_(Line, Cont, Te, atom_type,
        CE_Omega_table, CE_Te_table, CE_Coe, data_src_CE,
        CI_Omega_table, CI_Te_table, CI_Coe, data_src_CI)
    Cji = _Collision.Cij_to_Cji_(Cij[:], nj_by_ni[:])
    # pack array data of radiative bound-bound and bound-free transition
    Rij, Rji_stim, Rji_spon = _make_Rji_Rij_(Aji[:], Bji_Jbar[:],
        Bij_Jbar[:], Rki_spon[:], Rki_stim[:], Rik[:])
    # format collisional transition rate matrix
    Cmat = _numpy.zeros((nLevel,nLevel), dtype=DT_NB_FLOAT)
    _SEsolver.set_matrixC_(Cmat[:,:],Cji[:],Cij[:],idxI[:],idxJ,Ne)
    # format radiative transition rate matrix
    Rmat = _numpy.zeros((nLevel,nLevel), dtype=DT_NB_FLOAT)
    _SEsolver.set_matrixR_(Rmat[:,:], Rji_spon[:], Rji_stim[:],
        Rij[:], idxI[:], idxJ[:])
    # solve Ax=b linear equations for statistical equilibrium
    n_SE = _SEsolver.solve_SE_(Rmat, Cmat)
    # pack the result
```

```
    SE_con = _Container.SE_Container(
        n_SE = n_SE,
        n_LTE = n_LTE,
        nj_by_ni = nj_by_ni,
        wave_mesh_shifted_1d = wave_mesh_cm_shifted_all,
        absorb_prof_1d = absorb_prof_cm_all,
        Line_mesh_idxs = Line_mesh_idxs,
        Jbar = Jbar_all)
    tran_rate_con = _Container.TranRates_Container(
        Rji_spon = Rji_spon[:],
        Rji_stim=Rji_stim[:],
        Rij=Rij[:],
        Cji_Ne = Cji[:] * Ne,
        Cij_Ne = Cij[:] * Ne,
        Rmat = Rmat,
        Cmat = Cmat)
    return SE_con, tran_rate_con
```

4 Solar background radiation

We constructed a continuous high resolution Solar spectrum as the background radiation in the simulation. The detailed information of the data source of each wavelength range is described in Table 2.

As shown in Figure 1, the full constructed Solar spectrum extends from 1 [Å] to 0.1 $[cm]$. The intensity level alignment is performed in the following steps:

1. align intensity spectrum normalized by continuum in [3000,13400][Å] from BASS2000[11] to Allen's continuum intensity.[12]
2. the wide range Solar spectrum from ASTM E-490[13] covers from 1 [Å] to 0.1 $[cm]$ with a coarse resolution, therefore we align it to Allen's continuum intensity to set a reference for UV/EUV spectrum.
3. align spectrum from SORCE/XPS,[14] SDO/EVE/ELS,[15] SOHO/Sumer[16] and SORCE/SOLSTICE[14] to the adjusted ASTM E-490 spectrum.

Figure 2 illustrates how the optical range of the spectrum is aligned to Allen's continuum intensity. Figure 3 shows the UV/EUV range of the spectrum, with a bunch of emission spectral lines.

Note that absorption line suddenly disappeared beyond 11000[Å] due to the switch of spectrum data source.

Table 2: detailed information of the Solar spectrum. QS: Quiet Sun; FD: Full Disk

Wavelength range [Å]	Instrument(date of observation)	Region	Resolution [Å]
1-333	XPS(2018/04/11)	QS	1
333-670	EVE/ELS(2018/04/18)	QS	0.2
670-1609	SOHO/Sumer	QS	0.04
1609-3000	SOLSTICE(2018/04/11)	FD	1
3000-11000	Jungfraujoch	QS	0.002
11000-13400	Kitt Peak	QS	0.004
13400-0.1[cm]	ASTM E-490	FD	20

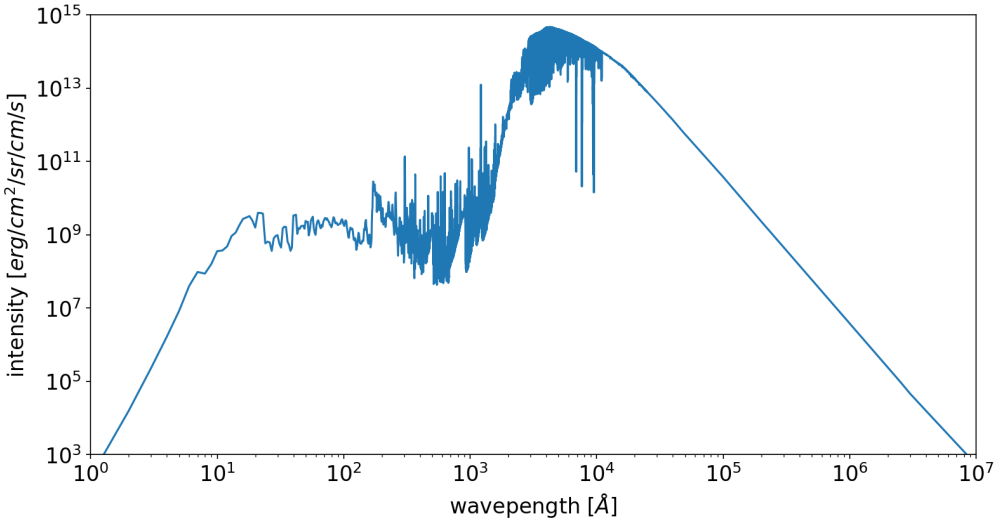

Figure 1: full range of the constructed Solar spectrum in log scale.

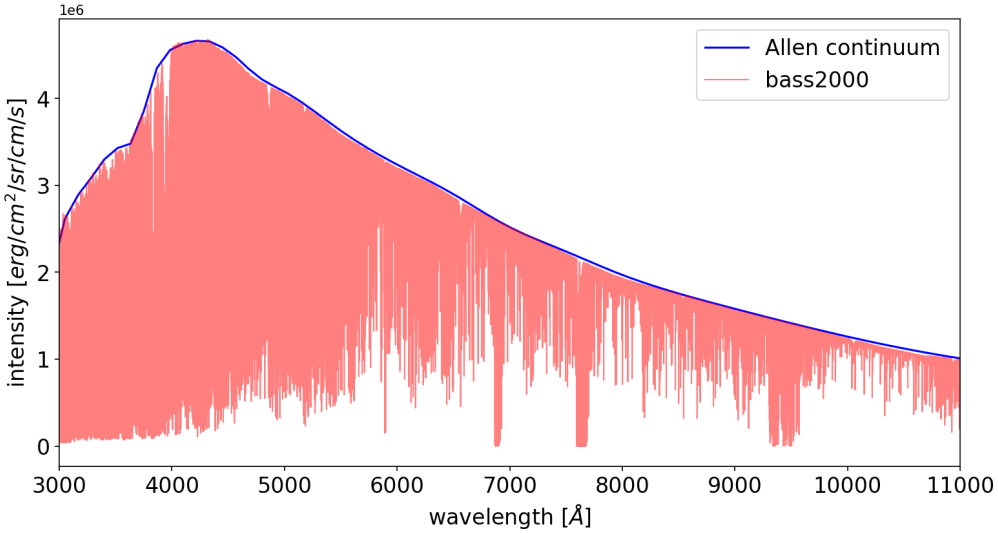

Figure 2: comparison between Allen's continuum profile and the scaled solar spectrum in wavelength range of [3000,11000] [Å] in linear scale

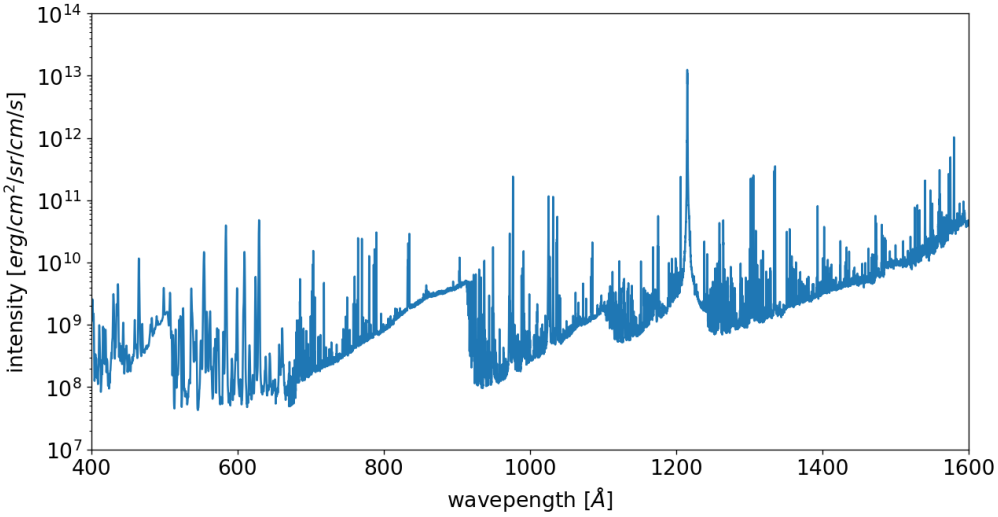

Figure 3: UV/EUV range of the constructed Solar spectrum.

5 Examples

In this section, we introduce the basic usage of *SPECTRA*. More comprehensive examples containing outputs and data visulaization are collected in folder *notebooks/*.

Simulation normally starts from loading the atomic data by specifying the path to the **.conf* file, which contains the path to the directory of the specific atomic model and the filenames of the provided atomic data. For example, the configuration of a Helium atom shown in Listing 8 contains

- *folder*: path to the directory where the data files are located.
- *Level*: atomic level data file.
- *Aji*: Einstein A coefficient data file.
- *CEe*: electron impact collisional excitation data file.
- *CIe*: electron impact collisional ionization data file.
- *PI*: photoionization cross section data file.
- *RadiativeLine*: radiative line transition data file.
- *Grotrian*: grotrian plot setup file.

A line transition with non-zero Einstein A_{ji} coefficient has 41 wavelength mesh points and Gaussian absorption profile in defualt. One is able to customize parameters like the type of absoption profile, number of mesh points, etc., in the *RadiativeLine* data file.

Listing 8: configuration for atomic data

```
folder              ../atom/He
Level               He.Level
Aji                 He.Aji
CEe                 He.CE.electron
CIe                 He.CI.electron
PI                  He.Alpha
RadiativeLine       He.RadiativeLine
Grotrian            He.Grotrian
```

5.1 Statistical equilibrium

given the *.conf* file, to perform a statistical equilibrium simulation,

1. construct the data class of *Atom* and *Wavelength_Mesh* using *Atom.init_Atom_*
2. construct the data class of *Atmosphere0D* using *Atmosphere.Atmosphere0D*
3. construct the data class of *Radiation* using *Radiation.init_Radiation_*
4. perform Statistical equilibrium using *SELib.cal_SE_with_Ne_Te_*. In this case, statistical equilibrium is calculated given the electron density N_e and electron temperature T_e.

Listing 9: statistical equilibrium simulation

```
import os
# import the necessary modules
from spectra_src.ImportAll import *
from spectra_src.Struct import Atom, Atmosphere, Radiation
from spectra_src.Function.SEquil import SELib
# spicify the path to the configuration of the atomic data files
conf_path = os.path.join( CFG._ROOT_DIR, "data","conf","He.conf" )
# load atomic data and create data class for atom and wavelength mesh
# is_hydrogen is set to False in case of non hydrogen atom
atom, wMesh, path_dict = Atom.init_Atom_(conf_path,is_hydrogen=False)
# create a 0 dimension atmosphere, or to say, optically thin.
# specify physical parameters
atmos = Atmosphere.Atmosphere0D(Nh=1.E11, Ne=5.E10, Te=7.E3, Vd=0., Vt=5.E5)
# load background radiation
radiation = Radiation.init_Radiation_(atmos, wMesh)
# solve statistical equilibrium given
# electron density and electron temperature
SE_con, Rate_con = SELib.cal_SE_with_Ne_Te_(atom,atmos,wMesh,radiation,None)
```

5.2 Cloud model

Starting from Listing 9, by specifying the *thickness* (which is called *depth* in the function call) of the slab and the output of statistical simulation, the emergent intensity from a cloud model could be calculated by adding a single line of code.

Listing 10: cloud model simulation

```
# [the body of the function is omitted due to space limitation]
# given the level population contained in "SE_con"
# calculate the emergent intensity from a cloud model
# with thickness of 1000 [km]
Cloud_con = SlabModel.SE_to_slab_0D_(atom,atmos,SE_con,depth=1.E3*1.E5)
```

5.3 Visualization functions

Various visualization functions has been developed to help user understanding the population distribution and the statistical equilibrium transition balancing in the atomic model.

Listing 11 and Figure 4 gives an example of generating a Grotrian plot with radiative net transition rates of a 9-level hydrogen atomic model. The full tutorial could be found in */notebooks/StatisticalEquilibrium/Hydrogen_atom.ipynb* in the github repository.[8]

Listing 11: Grotrian and net transition rate plot of a 9-level hydrogen atomic model

```
from spectra_src.Struct import Atom, Atmosphere, Radiation
from spectra_src.Function.SEquil import SELib
from spectra_src.Visual import Grotrian
import os

conf_path = os.path.join( CFG._ROOT_DIR, "data/conf/H.conf" )
atom, wMesh, path_dict = Atom.init_Atom_(conf_path , is_hydrogen=True)
atmos = Atmosphere.Atmosphere0D(Nh=1E12, Ne=1E11, Te=1E4,
    Vd=0E5, Vt=5E5, use_Tr=False)
radiation = Radiation.init_Radiation_(atmos, wMesh)

SE_con, Rate_con = SELib.cal_SE_with_Ne_Te_(atom, atmos,
    wMesh,radiation, None)
# extract the line/continuum transition indexing
ni_Line = SE_con.n_SE[ atom.Line["idxI"] ][:]
nj_Line = SE_con.n_SE[ atom.Line["idxJ"] ][:]
ni_Cont = SE_con.n_SE[ atom.Cont["idxI"] ][:]
nj_Cont = SE_con.n_SE[ atom.Cont["idxJ"] ][:]
ni = np.append( ni_Line, ni_Cont )
nj = np.append( nj_Line, nj_Cont )
# extract the level indexing
idxI = np.append( atom.Line["idxI"], atom.Cont["idxI"] )
idxJ = np.append( atom.Line["idxJ"], atom.Cont["idxJ"] )

# user difined function to scale Y-axis
# linear scale in default
```

```
scaleFunc     = lambda x: x**(7)
scaleFunc_inv = lambda x : x**(1/7)
# initilize the grotrian object
gro = Grotrian.Grotrian(atom, path_dict["Grotrian"],
    _scaleFunc=scaleFunc, _scaleFunc_inv=scaleFunc_inv, )
# make a figure with only energy levels plotted
gro.make_fig(_figsize=(10,6),_dpi=150, _f=50)
# define a color map and colorbar scale
cmap = plt.get_cmap('cool')
norm = LogNorm(1E-5, 1E-1, clip=True)
# calculate the radiative net transition rate
rate_rad = - Rate_con.Rij * ni  + \
        (Rate_con.Rji_stim + Rate_con.Rji_spon) * nj
# add arrows of transition to the plot
gro.plot_transition_rate(_idxI=idxI,
                         _idxJ=idxJ,
                         _rate=rate_rad,
                         _direction="j->i",
                         _cmap=cmap, _norm=norm, _abserr=1E-5)
# add colorbar to the plot
gro.add_colorbar(_cmap=cmap, _norm=norm)
```

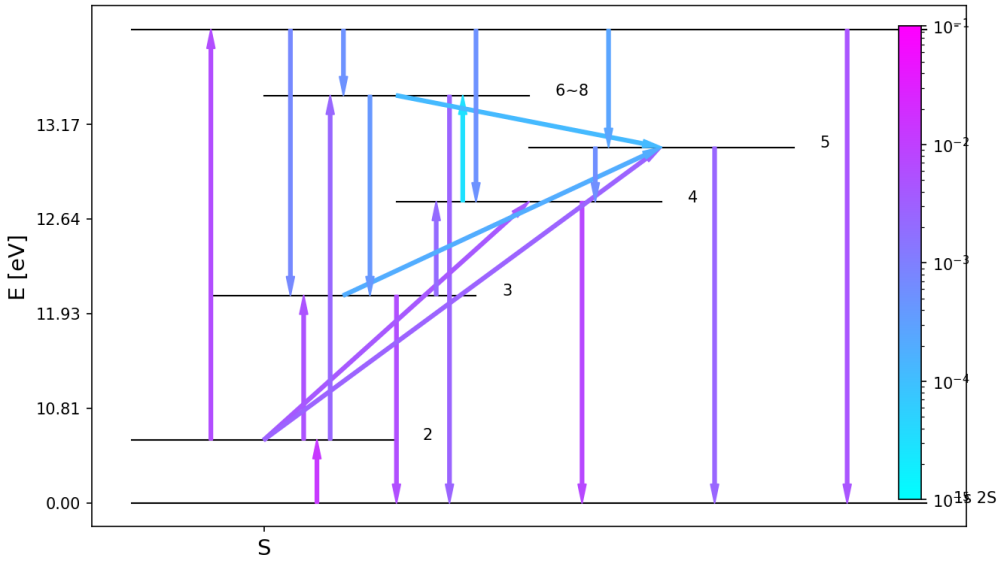

Figure 4: radiative net transition rate of a 9-level hydrogen atomic model. the number labeled close to each level is the quantum number n; vertical axis is the energy in $[eV]$ but scaled in an user defined scale; normally horizontal axis is order by quantum number L and multiplet, since sub-levels are degenerated in this hydrogen atomic model, the horizontal position of levels are adjusted for better display

Figure 5 shows the heat map of the Radiative/Collisional transition rate among 27-level helium atomic model (transitions among the first 21 levels are plotted). The upper left and the lower right of diagonal in each panel correspond to the upward and downward transitions, respectively. Statistical equilibrium is calculated given a hydrogen density of 1×10^{11} $[cm^{-3}]$ and an electron temperature of 8000 $[K]$. The full tutorial could be found in */notebooks/StatisticalEquilibrium/spectra_multi_atom.ipynb* in the github repository.[8]

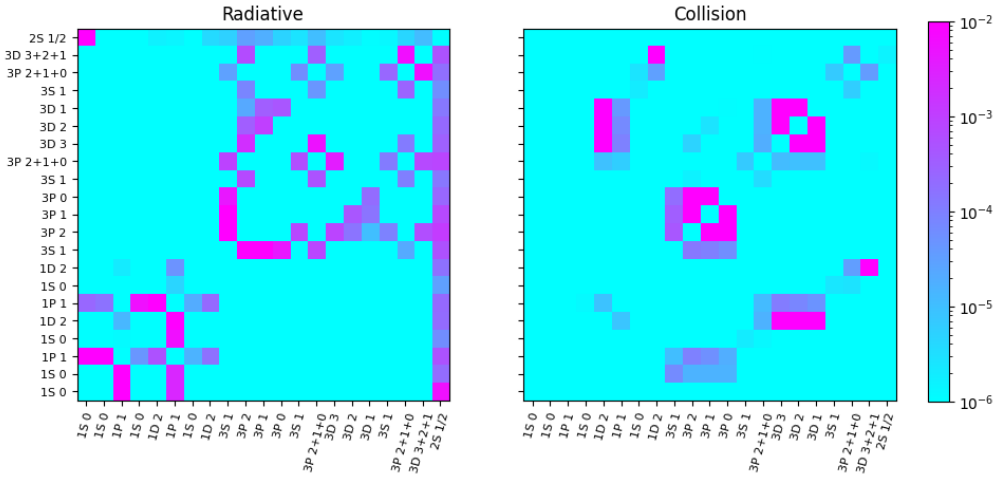

Figure 5: Heat map of the Radiative/Collisional transition rate among 27-levels of the helium atomic model (transitions among the first 21 levels are plotted). X and Y ticklabels are the *term* and *Angular Momumtum* J for upper and lower levels, respectively. Levels are sorted according to ionization stage and

type of *multiplet*.

Figure 6 shows how the fraction of specific ionization stage changes for a 9-level hydrogen atomic model, a 27-level helium atomic model and a 7-level calsium atomic model, given a range of hydrogen density and electron temperature. The full tutorial could be found in */notebooks/StatisticalEquilibrium/spectra_multi_atom.ipynb* in the github repository.[8]

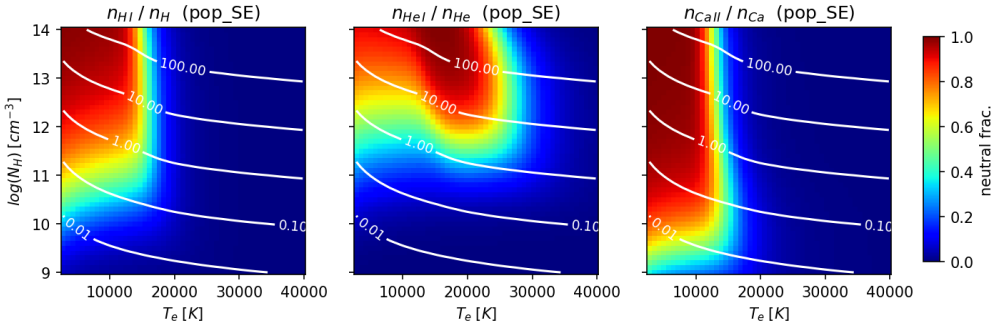

Figure 6: 2D map of the fraction of *H I*, *He I* and *Ca II* as a function of electron temperature and hydrogen density, which comes from the reusult of statistical equilibrium simulation of 9-level hydrogen atomic model, 27-level helium atomic model and 7-level calsium atomic model, respectively. White contour lines correspond to the gas pressure $[dyn/cm^2]$

6 Summary and Future work

This non-LTE radiative transfer library is developed to explain the multi-wavelength spectroscopic data in Solar observation. Currently the statistical equilibrium(CRD) module is com-

pleted, functionality related PRD (Partial redistribution) and radiative transfer are still under test. After combining the non-LTE statisticalequilibrium and radiative transfer, we are looking forward to extend the library to include polarization.

References

[1] Carlsson, M. *Uppsala Astronomical Observatory Reports* **1986**, *33*.

[2] Gouttebroze, P.; Heinzel, P.; Vial, J. C. **1993**, *99*, 513–543.

[3] Han Uitenbroek, radiative transfer code RH. https://www2.hao.ucar.edu/events/workshop/spectropolarimetry-2022/inversion-codes/rh-code, 2010.

[4] Hubeny, I.; Mihalas, D. *Theory of Stellar Atmospheres. An Introduction to Astrophysical Non-equilibrium Quantitative Spectroscopic Analysis*; 2015.

[5] Johnson, L. C. **1972**, *174*, 227.

[6] The Opacity Project Team, Opaity Project TOP database. https://cds.unistra.fr//topbase/topbase.html, 1995.

[7] Kramida, A., Ralchenko, Yu., Reader, J. and NIST ASD Team, NIST Atomic Spectra Database (version 5.10). https://www.nist.gov/pml/atomic-spectra-database, 1995; Affiliations: National Institute of Standards and Technology, Gaithersburg, MD.

[8] Y. Huang, github repository of SPECTRA. https://github.com/kouui/spectra, 2022.

[9] Continuum Analytics, Inc. (dba Anaconda, Inc.), Package, dependency and environment management for any language—Python, R, Ruby, Lua, Scala, Java, JavaScript, C/ C++, Fortran, and more. https://docs.conda.io/en/latest/index.html, 2017.

[10] Y. Huang, documentation of SPECTRA. https://kouui.github.io/spectra, 2023.

[11] Malherbe, J.-M.; Cornu, F.; Bualé, I. *arXiv e-prints* **2023**, arXiv:2305.14804.

[12] Allen, C. W. *Astrophysical Quantities*; 1976.

[13] Tobiska, W. K.; Bouwer, S. D. *Advances in Space Research* **2006**, *37*, 347–358.

[14] Ahmad, S.; Johnson, J.; Serafino, G. Solar spectral and total irradiance data from the SORCE mission. 34th COSPAR Scientific Assembly. 2002; p 2806.

[15] Hock, R. A.; Woods, T. N.; Eparvier, F. G.; Woodraska, D. L. Accessing and Using SDO EVE Data. Solar Heliospheric and INterplanetary Environment (SHINE 2010). 2010; p 127.

[16] Schuehle, U. H. Solar Ultraviolet Measurements of Ultraviolet Radiation (SUMER) instrument on SOHO: design, performance predictions, and calibration aspects. X-Ray and Ultraviolet Spectroscopy and Polarimetry. 1994; pp 47–52.