

---

# **pyequib Documentation**

***Release 0.4.2***

**Ashkbiz Danekar**

**Mar 05, 2021**



# USER DOCUMENTATION

|          |                                       |           |
|----------|---------------------------------------|-----------|
| <b>1</b> | <b>Introduction</b>                   | <b>3</b>  |
| 1.1      | Collisional Excitation Unit . . . . . | 3         |
| 1.2      | Recombination Unit . . . . .          | 3         |
| 1.3      | Reddening Unit . . . . .              | 3         |
| <b>2</b> | <b>Installation</b>                   | <b>5</b>  |
| <b>3</b> | <b>Usage</b>                          | <b>7</b>  |
| 3.1      | Collisional Excitation Unit . . . . . | 7         |
| 3.2      | Recombination Unit . . . . .          | 12        |
| 3.3      | Reddening Unit . . . . .              | 20        |
| <b>4</b> | <b>References</b>                     | <b>23</b> |
| <b>5</b> | <b>pyequib.main package</b>           | <b>25</b> |
| 5.1      | pyequib main module . . . . .         | 25        |
| <b>6</b> | <b>Indices and tables</b>             | <b>71</b> |
|          | <b>Python Module Index</b>            | <b>73</b> |



- *User Documentation*
- *API Reference*



## INTRODUCTION

**pyEQUIB** package is a collection of [Python](#) programs developed to perform plasma diagnostics and abundance analysis using emission line fluxes measured in ionized nebulae. It uses the [AtomNeb Python Package](#) to read collision strengths and transition probabilities for collisionally excited lines (CEL), and recombination coefficients for recombination lines (RL). This Python package can be used to determine interstellar extinctions, electron temperatures, electron densities, and ionic abundances from the measured fluxes of emission lines. It mainly contains the following API functions written purely in Python:

### 1.1 Collisional Excitation Unit

**API functions for collisionally excited lines (CEL)** have been developed based on the algorithm of the FORTRAN program [EQUIB](#) written in FORTRAN by [Howarth & Adams \(1981\)](#). The program [EQUIB](#) calculates atomic level populations and line emissivities in statistical equilibrium in multi-level atoms for different physical conditions of the stratification layers where the chemical elements are ionized. Using the Python implementation of the program [EQUIB](#), electron temperatures, electron densities, and ionic abundances are determined from the measured fluxes of collisionally excited lines.

### 1.2 Recombination Unit

**API functions for recombination lines (RL)** have been developed based on the algorithm of the recombination scripts by X. W. Liu and Y. Zhang included in the FORTRAN program [MOCASSIN](#). These API functions are used to determine ionic abundances from recombination lines for some heavy element ions.

### 1.3 Reddening Unit

**API functions for reddening and extinctions** have been developed according to the methods of the reddening law functions from [STSDAS IRAF Package](#), which are used to obtain interstellar extinctions and deredden measured fluxes based on different reddening laws.





## INSTALLATION

To install the last version, all you should need to do is:

```
python setup.py install
```

To install the stable version, you can use the preferred installer program (pip):

```
pip install pyequib
```

or you can install it from the cross-platform package manager *conda*:

```
conda install -c conda-forge pyequib
```

To get this package with the AtomNeb FITS files, you can simply use `git` command as follows:

```
git clone --recursive https://github.com/equib/pyEQUIB
```

This package requires the following packages:

- NumPy
- SciPy
- AtomNeb



The Documentation of the Python functions provides in detail in the *API Documentation* ([equib.github.io/pyEQUIB/doc](http://equib.github.io/pyEQUIB/doc)). There are three main object units:

### 3.1 Collisional Excitation Unit

**Collision Unit** which have the API functions for plasma diagnostics and abundance analysis of collisionally excited lines. Here are some examples of using *Collision Unit*.

- *Temperature:*

```
import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_dir = os.path.join('atomic-data', 'chianti70')
atom_elj_file = os.path.join(base_dir, data_dir, 'AtomElj.fits')
atom_omij_file = os.path.join(base_dir, data_dir, 'AtomOmij.fits')
atom_aij_file = os.path.join(base_dir, data_dir, 'AtomAij.fits')

atom = 's'
ion = 'ii'
s_ii_elj = atomneb.read_elj(atom_elj_file, atom, ion, level_num=5)
s_ii_omij = atomneb.read_omij(atom_omij_file, atom, ion)
s_ii_aij = atomneb.read_aij(atom_aij_file, atom, ion)

upper_levels='1,2,1,3/'
lower_levels='1,5/'
density = np.float64(2550)
line_flux_ratio=np.float64(10.753)
temperature = pyequib.calc_temperature(line_flux_ratio=line_flux_ratio,
    ↪density=density,
    upper_levels=upper_levels, lower_levels=lower_levels,
    elj_data=s_ii_elj, omij_data=s_ii_omij, aij_data=s_ii_aij)
print("Electron Temperature:", temperature)
```

which gives:

```
Electron Temperature:      7920.2865
```

- *Density:*

```

import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_dir = os.path.join('atomic-data', 'chianti70')
atom_elj_file = os.path.join(base_dir, data_dir, 'AtomElj.fits')
atom_omij_file = os.path.join(base_dir, data_dir, 'AtomOmij.fits')
atom_aij_file = os.path.join(base_dir, data_dir, 'AtomAij.fits')

atom = 's'
ion = 'ii'
s_ii_elj = atomneb.read_elj(atom_elj_file, atom, ion, level_num=5)
s_ii_omij = atomneb.read_omij(atom_omij_file, atom, ion)
s_ii_aij = atomneb.read_aij(atom_aij_file, atom, ion)

upper_levels='1,2/'
lower_levels='1,3/'
temperature=np.float64(7000.0) #
line_flux_ratio=np.float64(1.506) #
density = pyequib.calc_density(line_flux_ratio=line_flux_ratio,
    ↳temperature=temperature,
                                upper_levels=upper_levels, lower_levels=lower_
    ↳levels,
                                elj_data=s_ii_elj, omij_data=s_ii_omij, aij_data=s_
    ↳ii_aij)
print("Electron Density:", density)

```

which gives:

```
Electron Density:      2312.6395
```

- *Ionic Abundance:*

```

import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_dir = os.path.join('atomic-data', 'chianti70')
data_rc_dir = os.path.join('atomic-data-rc')
atom_elj_file = os.path.join(base_dir, data_dir, 'AtomElj.fits')
atom_omij_file = os.path.join(base_dir, data_dir, 'AtomOmij.fits')
atom_aij_file = os.path.join(base_dir, data_dir, 'AtomAij.fits')
atom_rc_sh95_file = os.path.join(base_dir, data_rc_dir, 'rc_SH95.fits')

atom = 'h'
ion = 'ii' # H I Rec
hi_rc_data = atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)

atom = 'o'
ion = 'iii' # [O III]
o_iii_elj = atomneb.read_elj(atom_elj_file, atom, ion, level_num=5) # read Energy_
    ↳Levels (Ej)
o_iii_omij = atomneb.read_omij(atom_omij_file, atom, ion) # read Collision_
    ↳Strengths (Omegaij)
o_iii_aij = atomneb.read_aij(atom_aij_file, atom, ion) # read Transition_
    ↳Probabilities (Aij)

```

(continues on next page)

(continued from previous page)

```

levels5007='3,4/'
temperature=np.float64(10000.0)
density=np.float64(5000.0)
iobs5007=np.float64(1200.0)
abb5007 = pyequib.calc_abundance(temperature=temperature, density=density,
                                line_flux=iobs5007, atomic_levels=levels5007,
                                elj_data=o_iii_elj, omij_data=o_iii_omij, aij_
↪data=o_iii_aij,
                                h_i_aeff_data=hi_rc_data['aeff'][0])
print('N(O^2+)/N(H+):', abb5007)

```

which gives:

```
N(O^2+)/N(H+) :    0.00041256231
```

- *Emissivity:*

```

import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_dir = os.path.join('atomic-data', 'chianti70')
data_rc_dir = os.path.join('atomic-data-rc')
atom_elj_file = os.path.join(base_dir, data_dir, 'AtomElj.fits')
atom_omij_file = os.path.join(base_dir, data_dir, 'AtomOmij.fits')
atom_aij_file = os.path.join(base_dir, data_dir, 'AtomAij.fits')
atom_rc_sh95_file = os.path.join(base_dir, data_rc_dir, 'rc_SH95.fits')

atom = 'h'
ion = 'ii' # H I Rec
hi_rc_data = atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)

atom = 'o'
ion = 'iii' # [O III]
o_iii_elj = atomneb.read_elj(atom_elj_file, atom, ion, level_num=5) # read Energy_
↪Levels (Ej)
o_iii_omij = atomneb.read_omij(atom_omij_file, atom, ion) # read Collision_
↪Strengths (Omegaij)
o_iii_aij = atomneb.read_aij(atom_aij_file, atom, ion) # read Transition_
↪Probabilities (Aij)

levels5007='3,4/'
temperature=np.float64(10000.0)
density=np.float64(5000.0)
iobs5007=np.float64(1200.0)
emis = pyequib.calc_emissivity(temperature=temperature, density=density, atomic_
↪levels=levels5007,
                                elj_data=o_iii_elj, omij_data=o_iii_omij, aij_
↪data=o_iii_aij)
print('Emissivity(O III 5007):', emis)

```

which gives:

```
Emissivity(O III 5007) :    3.6041012e-21
```

- *Atomic Level Population:*

```

import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_dir = os.path.join('atomic-data', 'chianti70')
atom_elj_file = os.path.join(base_dir, data_dir, 'AtomElj.fits')
atom_omij_file = os.path.join(base_dir, data_dir, 'AtomOmiJ.fits')
atom_aij_file = os.path.join(base_dir, data_dir, 'AtomAij.fits')

atom = 's'
ion = 'ii'
s_ii_elj = atomneb.read_elj(atom_elj_file, atom, ion, level_num=5)
s_ii_omij = atomneb.read_omij(atom_omij_file, atom, ion)
s_ii_aij = atomneb.read_aij(atom_aij_file, atom, ion)

density = np.float64(1000)
temperature=np.float64(10000.0) #
nlj = pyequib.calc_populations(temperature=temperature, density=density,
                               elj_data=s_ii_elj, omij_data=s_ii_omij, aij_data=s_
↪ii_aij)
print('Populations:', nlj)

```

which prints:

```
Populations: 0.96992832 0.0070036315 0.023062261 2.6593671e-06 3.1277019e-06
```

- *Critical Density:*

```

import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_dir = os.path.join('atomic-data', 'chianti70')
atom_elj_file = os.path.join(base_dir, data_dir, 'AtomElj.fits')
atom_omij_file = os.path.join(base_dir, data_dir, 'AtomOmiJ.fits')
atom_aij_file = os.path.join(base_dir, data_dir, 'AtomAij.fits')

atom = 's'
ion = 'ii'
s_ii_elj = atomneb.read_elj(atom_elj_file, atom, ion, level_num=5)
s_ii_omij = atomneb.read_omij(atom_omij_file, atom, ion)
s_ii_aij = atomneb.read_aij(atom_aij_file, atom, ion)

temperature=np.float64(10000.0)
n_crit = pyequib.calc_crit_density(temperature=temperature,
                                    elj_data=s_ii_elj, omij_data=s_ii_omij, aij_
↪data=s_ii_aij)
print('Critical Densities:', n_crit)

```

which gives:

```
Critical Densities: 0.0000000 5007.8396 1732.8414 1072685.0 2220758.1
```

- *All Ionic Level Information:*

```

import pyequib
import atomneb

```

(continues on next page)

(continued from previous page)

```

import os
base_dir = 'externals/atomneb'
data_dir = os.path.join('atomic-data', 'chianti70')
data_rc_dir = os.path.join('atomic-data-rc')
atom_elj_file = os.path.join(base_dir, data_dir, 'AtomElj.fits')
atom_omij_file = os.path.join(base_dir, data_dir, 'AtomOmiJ.fits')
atom_aij_file = os.path.join(base_dir, data_dir, 'AtomAij.fits')
atom_rc_sh95_file = os.path.join(base_dir, data_dir, 'rc_SH95.fits')

atom = 'h'
ion = 'ii' # H I Rec
hi_rc_data = atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)

atom = 'o'
ion = 'iii' # [O III]
o_iii_elj = atomneb.read_elj(atom_elj_file, atom, ion, level_num=5) # read Energy_
↳ Levels (Ej)
o_iii_omij = atomneb.read_omij(atom_omij_file, atom, ion) # read Collision_
↳ Strengths (Omegaij)
o_iii_aij = atomneb.read_aij(atom_aij_file, atom, ion) # read Transition_
↳ Probabilities (Aij)

temperature=np.float64(10000.0)
density=np.float64(5000.0)
pyequib.print_ionic(temperature=temperature, density=density,
                    elj_data=o_iii_elj, omij_data=o_iii_omij, aij_data=o_iii_aij,
                    h_i_aeff_data=hi_rc_data['aeff'][0])

```

which gives:

```

Temperature = 10000.0 K
Density = 1000.0 cm-3

Level      Populations      Critical Densities
Level 1:    3.063E-01      0.000E+00
Level 2:    4.896E-01      4.908E+02
Level 3:    2.041E-01      3.419E+03
Level 4:    4.427E-05      6.853E+05
Level 5:    2.985E-09      2.547E+07

2.597E-05
88.34um
(2-->1)
2.859E-22

0.000E+00  9.632E-05
32.66um    51.81um
(3-->1)    (3-->2)
0.000E+00  7.536E-22

2.322E-06  6.791E-03  2.046E-02
4932.60A   4960.29A   5008.24A
(4-->1)    (4-->2)    (4-->3)
4.140E-25  1.204E-21  3.593E-21

0.000E+00  2.255E-01  6.998E-04  1.685E+00
2315.58A   2321.67A   2332.12A   4364.45A

```

(continues on next page)

(continued from previous page)

```

      (5-->1)      (5-->2)      (5-->3)      (5-->4)
0.000E+00  5.759E-24  1.779E-26  2.289E-23

H-beta emissivity: 1.237E-25 N(H+) Ne [erg/s]

```

## 3.2 Recombination Unit

**Recombination Unit** which have the API functions for plasma diagnostics and abundance analysis of recombination lines. Here are some examples of using *Recombination Unit*.

- *He+ Ionic Abundance:*

```

import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_rc_dir = os.path.join('atomic-data-rc')
atom_rc_he_i_file = os.path.join(base_dir,data_rc_dir, 'rc_he_ii_PFS12.fits')
atom_rc_sh95_file = os.path.join(base_dir,data_rc_dir, 'rc_SH95.fits')

atom = 'h'
ion = 'ii' # H I
h_i_rc_data = atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)

atom = 'he'
ion = 'ii' # He I
he_i_rc_data = atomneb.read_aeff_he_i_pfs12(atom_rc_he_i_file, atom, ion)

temperature=np.float64(10000.0)
density=np.float64(5000.0)
he_i_4471_flux= 2.104
linenum=10# 4471.50
abund_he_i = pyequib.calc_abund_he_i_rl(temperature=temperature, density=density,
                                       linenum=linenum, line_flux=he_i_4471_flux,
                                       he_i_aeff_data=he_i_aeff_data, h_i_aeff_data=h_i_
→aeff_data)
print('N(He^+)/N(H^+):', abund_he_i)

```

which gives:

```

N(He^+)/N(H^+) :      0.040848393

```

- *He++ Ionic Abundance:*

```

import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_rc_dir = os.path.join('atomic-data-rc')
atom_rc_sh95_file = os.path.join(base_dir,data_rc_dir, 'rc_SH95.fits')

atom = 'h'
ion = 'ii' # H I
h_i_rc_data = atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)

```

(continues on next page)



(continued from previous page)

```

atom = 'he'
ion = 'iii' # He II
he_ii_rc_data = atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)

temperature=np.float64(10000.0)
density=np.float64(5000.0)
he_ii_4686_flux = 135.833
abund_he_ii = pyequib.calc_abund_he_ii_rl(temperature=temperature,
↪density=density,
                                line_flux=he_ii_4686_flux,
                                he_ii_aeff_data=he_ii_aeff_data, h_i_aeff_
↪data=h_i_aeff_data)
print('N(He^2+)/N(H^+):', abund_he_ii)

```

which gives:

```
N(He^2+)/N(H^+) :      0.11228817
```

- *C++ Ionic Abundance:*

```

import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_rc_dir = os.path.join('atomic-data-rc')
atom_rc_all_file = os.path.join(base_dir,data_rc_dir, 'rc_collection.fits')
atom_rc_sh95_file = os.path.join(base_dir,data_rc_dir, 'rc_SH95.fits')

atom = 'c'
ion = 'iii' # C II
c_ii_rc_data = atomneb.read_aeff_collection(atom_rc_all_file, atom, ion)

atom = 'h'
ion = 'ii' # H I
h_i_rc_data = atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)

temperature=np.float64(10000.0)
density=np.float64(5000.0)
wavelength=6151.43
c_ii_6151_flux = 0.028
abund_c_ii = pyequib.calc_abund_c_ii_rl(temperature=temperature, density=density,
                                wavelength=wavelength, line_flux=c_ii_6151_flux,
                                c_ii_rc_data=c_ii_rc_data, h_i_aeff_data=h_i_aeff_
↪data)
print('N(C^2+)/N(H+):', abund_c_ii)

```

which gives:

```
N(C^2+)/N(H+) :      0.00063404650
```

- *C3+ Ionic Abundance:*

```

import pyequib
import atomneb
import os

```

(continues on next page)

(continued from previous page)

```

base_dir = 'externals/atomneb'
data_rc_dir = os.path.join('atomic-data-rc')
atom_rc_ppb91_file = os.path.join(base_dir,data_rc_dir, 'rc_PPb91.fits')
atom_rc_sh95_file = os.path.join(base_dir,data_rc_dir, 'rc_SH95.fits')

atom = 'c'
ion = 'iv' # C III
c_iii_rc_data = atomneb.read_aeff_ppb91(atom_rc_ppb91_file, atom, ion)

atom = 'h'
ion = 'ii' # H I
h_i_rc_data = atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)

temperature=np.float64(10000.0)
density=np.float64(5000.0)
wavelength=4647.42
c_iii_4647_flux = 0.107
abund_c_iii = pyequib.calc_abund_c_iii_rl(temperature=temperature,
↪density=density,
                                wavelength=wavelength,
                                line_flux=c_iii_4647_flux, c_iii_rc_data=c_iii_
↪rc_data,
                                h_i_aeff_data=h_i_aeff_data)
print('N(C^3+)/N(H+):', abund_c_iii)

```

which gives:

```
N(C^3+)/N(H+) : 0.00017502840
```

- *N++ Ionic Abundance:*

```

import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_rc_dir = os.path.join('atomic-data-rc')
atom_rc_all_file = os.path.join(base_dir,data_rc_dir, 'rc_collection.fits')
atom_rc_sh95_file = os.path.join(base_dir,data_rc_dir, 'rc_SH95.fits')

atom = 'n'
ion = 'iii' # N II
n_ii_rc_data = atomneb.read_aeff_collection(atom_rc_all_file, atom, ion)
n_ii_rc_data_br = atomneb.read_aeff_collection(atom_rc_all_file, atom, ion,
↪br=True)

atom = 'h'
ion = 'ii' # H I
h_i_rc_data = atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)

wavelength=4442.02
n_ii_4442_flux = 0.017
abund_n_ii = pyequib.calc_abund_n_ii_rl(temperature=temperature, density=density,
                                wavelength=wavelength, line_flux=n_ii_4442_flux,
                                n_ii_rc_br=n_ii_rc_data_br, n_ii_rc_data=n_ii_rc_
↪data,
                                h_i_aeff_data=h_i_aeff_data)
print('N(N^2+)/N(H+):', abund_n_ii)

```

which gives:

```
N(N2+)/N(H+) : 0.00069297541
```

- *N3+ Ionic Abundance:*

```
import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_rc_dir = os.path.join('atomic-data-rc')
atom_rc_ppb91_file = os.path.join(base_dir,data_rc_dir, 'rc_PPb91.fits')
atom_rc_sh95_file = os.path.join(base_dir,data_rc_dir, 'rc_SH95.fits')

atom = 'n'
ion = 'iv' # N III
n_iii_rc_data = atomneb.read_aeff_ppb91(atom_rc_ppb91_file, atom, ion)

atom = 'h'
ion = 'ii' # H I
h_i_rc_data = atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)

wavelength=4640.64
n_iii_4641_flux = 0.245
abund_n_iii = pyequib.calc_abund_n_iii_rl(temperature=temperature,
↪density=density,
                                wavelength=wavelength, line_flux=n_iii_4641_
↪flux,
                                n_iii_rc_data=n_iii_rc_data, h_i_aeff_data=h_i_
↪aeff_data)
print('N(N3+)/N(H+) : ', abund_n_iii)
```

which gives:

```
N(N3+)/N(H+) : 6.3366175e-05
```

- *O++ Ionic Abundance:*

```
import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_rc_dir = os.path.join('atomic-data-rc')
atom_rc_all_file = os.path.join(base_dir,data_rc_dir, 'rc_collection.fits')
atom_rc_sh95_file = os.path.join(base_dir,data_rc_dir, 'rc_SH95.fits')

atom = 'o'
ion = 'iii' # O II
o_ii_rc_data = atomneb.read_aeff_collection(atom_rc_all_file, atom, ion)
o_ii_rc_data_br = atomneb.read_aeff_collection(atom_rc_all_file, atom, ion,
↪br=True)

atom = 'h'
ion = 'ii' # H I
h_i_rc_data = atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)

wavelength=4613.68
o_ii_4614_flux = 0.009
```

(continues on next page)

(continued from previous page)

```

abund_o_ii = pyequib.calc_abund_o_ii_rl(temperature=temperature, density=density,
                                         wavelength=wavelength, line_flux=o_ii_4614_flux,
                                         o_ii_rc_br=o_ii_rc_data_br,
                                         o_ii_rc_data=o_ii_rc_data,
                                         h_i_aeff_data=h_i_aeff_data)
print('N(O^2+)/N(H+):', abund_o_ii)

```

which gives:

```
N(O^2+)/N(H+) :    0.0018886330
```

- *Ne++ Ionic Abundance:*

```

import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_rc_dir = os.path.join('atomic-data-rc')
atom_rc_all_file = os.path.join(base_dir, data_rc_dir, 'rc_collection.fits')
atom_rc_sh95_file = os.path.join(base_dir, data_rc_dir, 'rc_SH95.fits')

atom = 'ne'
ion = 'iii' # Ne II
ne_ii_rc_data = atomneb.read_aeff_collection(atom_rc_all_file, atom, ion)

atom = 'h'
ion = 'ii' # H I
h_i_rc_data = atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)

wavelength=3777.14
ne_ii_3777_flux = 0.056
abund_ne_ii = pyequib.calc_abund_ne_ii_rl(temperature=temperature,
↪ density=density,
                                         wavelength=wavelength, line_flux=ne_ii_3777_
↪ flux,
                                         ne_ii_rc_data=ne_ii_rc_data, h_i_aeff_data=h_i_
↪ aeff_data)
print('N(Ne^2+)/N(H+):', Abund_ne_ii)

```

which gives:

```
N(Ne^2+)/N(H+) :    0.00043376850
```

- *He I Emissivity:*

```

import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_rc_dir = os.path.join('atomic-data-rc')
atom_rc_he_i_file = os.path.join(base_dir, data_rc_dir, 'rc_he_ii_PFS12.fits')

atom = 'he'
ion = 'ii' # He I
he_i_rc_data = atomneb.read_aeff_he_i_pfs12(atom_rc_he_i_file, atom, ion)

```

(continues on next page)

(continued from previous page)

```

temperature=np.float64(10000.0)
density=np.float64(5000.0)
linenum=10# 4471.50
emiss_he_i = pyequib.calc_emiss_he_i_rl(temperature=temperature, density=density,
                                         linenum=linenum, he_i_aeff_data=he_i_aeff_data)
print('He I Emissivity:', emiss_he_i)

```

which gives:

```
He I Emissivity: 6.3822830e-26
```

- *He II Emissivity:*

```

import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_rc_dir = os.path.join('atomic-data-rc')
atom_rc_sh95_file = os.path.join(base_dir,data_rc_dir, 'rc_SH95.fits')

atom = 'he'
ion = 'iii' # He II
he_ii_rc_data = atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)

temperature=np.float64(10000.0)
density=np.float64(5000.0)
emiss_he_ii = pyequib.calc_emiss_he_ii_rl(temperature=temperature,
↪density=density,
                                         he_ii_aeff_data=he_ii_aeff_data)
print('He II Emissivity:', emiss_he_ii)

```

which gives:

```
He II Emissivity: 1.4989134e-24
```

- *C II Emissivity:*

```

import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_rc_dir = os.path.join('atomic-data-rc')
atom_rc_all_file = os.path.join(base_dir,data_rc_dir, 'rc_collection.fits')

atom = 'c'
ion = 'iii' # C II
c_ii_rc_data = atomneb.read_aeff_collection(atom_rc_all_file, atom, ion)

temperature=np.float64(10000.0)
density=np.float64(5000.0)
wavelength=6151.43
emiss_c_ii = pyequib.calc_emiss_c_ii_rl(temperature=temperature, density=density,
                                         wavelength=wavelength, c_ii_rc_data=c_ii_rc_data)
print('C II Emissivity:', emiss_c_ii)

```

which gives:

```
C II Emissivity: 5.4719511e-26
```

- *C III Emissivity:*

```
import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_rc_dir = os.path.join('atomic-data-rc')
atom_rc_ppb91_file = os.path.join(base_dir, data_rc_dir, 'rc_PPb91.fits')

atom = 'C'
ion = 'iv' # C III
c_iii_rc_data = atomneb.read_aeff_ppb91(atom_rc_ppb91_file, atom, ion)

temperature=np.float64(10000.0)
density=np.float64(5000.0)
wavelength=4647.42
emiss_c_iii = pyequib.calc_emiss_c_iii_rl(temperature=temperature,
↪density=density,
                                     wavelength=wavelength,
                                     c_iii_rc_data=c_iii_rc_data)
print('C III Emissivity:', emiss_c_iii)
```

which gives:

```
C III Emissivity: 7.5749632e-25
```

- *N II Emissivity:*

```
import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_rc_dir = os.path.join('atomic-data-rc')
atom_rc_all_file = os.path.join(base_dir, data_rc_dir, 'rc_collection.fits')

atom = 'N'
ion = 'iii' # N II
n_ii_rc_data = atomneb.read_aeff_collection(atom_rc_all_file, atom, ion)
n_ii_rc_data_br = atomneb.read_aeff_collection(atom_rc_all_file, atom, ion,
↪br=True)

wavelength=4442.02
emiss_n_ii = pyequib.calc_emiss_n_ii_rl(temperature=temperature, density=density,
                                     wavelength=wavelength,
                                     n_ii_rc_br=n_ii_rc_data_br, n_ii_rc_data=n_ii_rc_
↪data)
print('N II Emissivity:', emiss_n_ii)
```

which gives:

```
N II Emissivity: 3.0397397e-26
```

- *N III Emissivity:*

```

import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_rc_dir = os.path.join('atomic-data-rc')
atom_rc_ppb91_file = os.path.join(base_dir,data_rc_dir, 'rc_PPb91.fits')

atom = 'n'
ion = 'iv' # N III
n_iii_rc_data = atomneb.read_aeff_ppb91(atom_rc_ppb91_file, atom, ion)

wavelength=4640.64
emiss_n_iii = pyequib.calc_emiss_n_iii_rl(temperature=temperature,
↪density=density,
                                wavelength=wavelength, n_iii_rc_data=n_iii_rc_
↪data)
print('N III Emissivity:', emiss_n_iii)

```

which gives:

```
N III Emissivity: 4.7908644e-24
```

- *O II Emissivity:*

```

import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_rc_dir = os.path.join('atomic-data-rc')
atom_rc_all_file = os.path.join(base_dir,data_rc_dir, 'rc_collection.fits')

atom = 'o'
ion = 'iii' # O II
o_ii_rc_data = atomneb.read_aeff_collection(atom_rc_all_file, atom, ion)
o_ii_rc_data_br = atomneb.read_aeff_collection(atom_rc_all_file, atom, ion,
↪br=True)

wavelength=4613.68
emiss_o_ii = pyequib.calc_emiss_o_ii_rl(temperature=temperature, density=density,
                                wavelength=wavelength,
                                o_ii_rc_br=o_ii_rc_data_br, o_ii_rc_data=o_ii_rc_
↪data)
print('O II Emissivity:', emiss_o_ii)

```

which gives:

```
O II Emissivity: 5.9047319e-27
```

- *Ne II Emissivity:*

```

import pyequib
import atomneb
import os
base_dir = 'externals/atomneb'
data_rc_dir = os.path.join('atomic-data-rc')
atom_rc_all_file = os.path.join(base_dir,data_rc_dir, 'rc_collection.fits')

```

(continues on next page)

(continued from previous page)

```

atom = 'ne'
ion = 'iii' # Ne II
ne_ii_rc_data = atomneb.read_aeff_collection(atom_rc_all_file, atom, ion)

wavelength=3777.14
emiss_ne_ii = pyequib.calc_emiss_ne_ii_rl(temperature=temperature,
↪density=density,
                                wavelength=wavelength, ne_ii_rc_data=ne_ii_rc_
↪data)
print('Ne II Emissivity:', emiss_ne_ii

```

which gives:

```
Ne II Emissivity: 1.5996881e-25
```

### 3.3 Reddening Unit

**Reddening Unit** which have the API functions for estimating logarithmic extinctions at H-beta and dereddening observed fluxes based on reddening laws and extinctions. Here are some examples of using *Reddening Unit*.

- *Reddening Law Function:*

```

import pyequib
wavelength=6563.0
r_v=3.1
fl=pyequib.redlaw(wavelength, rv=r_v, ext_law='GAL')
print('fl(6563):', fl)

```

which gives:

```
fl(6563): -0.32013816
```

- *Galactic Reddening Law Function based on Seaton (1979), Howarth (1983), & CCM (1983):*

```

import pyequib
wavelength=6563.0
r_v=3.1
fl=pyequib.redlaw_gal(wavelength, rv=r_v)
print('fl(6563):', fl)

```

which gives:

```
fl(6563): -0.32013816
```

- *Galactic Reddening Law Function based on Savage & Mathis (1979):*

```

import pyequib
wavelength=6563.0
fl=pyequib.redlaw_gal2(wavelength)
print('fl(6563):', fl)

```

which gives:

```
fl(6563): -0.30925984
```



- *Reddening Law Function based on Cardelli, Clayton & Mathis (1989):*

```
import pyequib
wavelength=6563.0
r_v=3.1
fl=pyequib.redlaw_ccm(wavelength, rv=r_v)
prin('fl(6563):', fl)
```

which gives:

```
fl(6563):      -0.29756615
```

- *Galactic Reddening Law Function based on Whitford (1958), Seaton (1977), & Kaler(1976):*

```
import pyequib
wavelength=6563.0
fl=pyequib.redlaw_jbk(wavelength)
print('fl(6563):', fl)
```

which gives:

```
fl(6563):      -0.33113684
```

- *Reddening Law Function based on Fitzpatrick & Massa (1990), Fitzpatrick (1999), Misselt (1999):*

```
import pyequib
wavelength=6563.0
r_v=3.1
fmlaw='AVGLMC'
fl=pyequib.redlaw_fm(wavelength, fmlaw=fmlaw, rv=r_v)
print('fl(6563):', fl)
```

which gives:

```
fl(6563):      -0.35053032
```

- *Reddening Law Function for the Small Magellanic Cloud:*

```
import pyequib
wavelength=6563.0
fl=pyequib.redlaw_smc(wavelength)
print('fl(6563):', fl)
```

which gives:

```
fl(6563):      -0.22659261
```

- *Reddening Law Function for the Large Magellanic Cloud:*

```
import pyequib
wavelength=6563.0
fl=pyequib.redlaw_lmc(wavelength)
print('fl(6563):', fl)
```

which gives:

```
fl(6563):      -0.30871187
```

- *Dereddening Absolute Flux:*

```
import pyequib
wavelength=6563.0
m_ext=1.0
flux=1.0
ext_law='GAL'
r_v=3.1
flux_deredden=pyequib.deredden_reflux(wavelength, flux, m_ext, ext_law=ext_law,
↪rv=r_v)
print('dereddened flux(6563)', flux_deredden)
```

which gives:

```
dereddened flux(6563)      4.7847785
```

- *Dereddening Relative Flux:*

```
import pyequib
wavelength=6563.0
m_ext=1.0
flux=1.0
ext_law='GAL'
r_v=3.1
flux_deredden=pyequib.deredden_flux(wavelength, flux, m_ext, ext_law=ext_law,
↪rv=r_v)
print('dereddened flux(6563)', flux_deredden)
```

which gives:

```
dereddened flux(6563)      0.47847785
```

**REFERENCES**

- Danehkar, A. (2020). pyEQUIB Python Package, an addendum to proEQUIB: IDL Library for Plasma Diagnostics and Abundance Analysis. *J. Open Source Softw.*, **5**, 2798. doi:[10.21105/joss.02798](https://doi.org/10.21105/joss.02798) ads:[2020JOSS...5.2798D](https://arxiv.org/abs/2020JOSS...5.2798D).
- Danehkar, A. (2018). proEQUIB: IDL Library for Plasma Diagnostics and Abundance Analysis. *J. Open Source Softw.*, **3**, 899. doi:[10.21105/joss.00899](https://doi.org/10.21105/joss.00899) ads:[2018JOSS...3..899D](https://arxiv.org/abs/2018JOSS...3..899D).



## PYEQUIB.MAIN PACKAGE

### 5.1 pyequib main module

This module contains functions for Plasma Diagnostics and Abundance Analysis

```
pyequib.calc_abund_c_ii_rl(temperature=None, density=None, wavelength=None,  
                           line_flux=None, c_ii_rc_data=None, h_i_aeff_data=None)
```

This function determines the ionic abundance from the observed flux intensity for the given wavelength of C II recombination line by using the recombination coefficients from from Davey et al. (2000) 2000A&AS..142...85D.

**Returns** type=double. This function returns the ionic abundanc.

**Keywords**

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**wavelength** [in, required, type=float] Line Wavelength in Angstrom

**line\_flux** [in, required, type=float] line flux intensity

**c\_ii\_rc\_data** [in, required, type=array/object] C II recombination coefficients

**h\_i\_aeff\_data** [in, required, type=array/object] H I recombination coefficients

**Examples** For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_all_file= os.path.join(base_dir,data_rc_dir, 'rc_
↳collection.fits')
>>> atom_rc_sh95_file= os.path.join(base_dir,data_rc_dir, 'rc_SH95.
↳fits')
>>> atom='h'
>>> ion='ii' # H I
>>> h_i_rc_data=atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)
>>> h_i_aeff_data=h_i_rc_data['aeff'][0]
>>> atom='c'
>>> ion='iii' # C II
>>> c_ii_rc_data=atomneb.read_aeff_collection(atom_rc_all_file, atom,
↳ion)
```

(continues on next page)

(continued from previous page)

```
>>> temperature=np.float64(10000.0)
>>> density=np.float64(5000.0)
>>> c_ii_6151_flux = 0.028
>>> wavelength=6151.43
>>> abund_c_ii=pyequib.calc_abund_c_ii_rl(temperature=temperature,
↪ density=density,
>>>                                     wavelength=wavelength, line_flux=c_
↪ ii_6151_flux,
>>>                                     c_ii_rc_data=c_ii_rc_data, h_i_aeff_
↪ data=h_i_aeff_data)
>>> print('N(C^2+)/N(H+):', abund_c_ii)
      N(C^2+)/N(H+):      0.00063404650
```

**Categories** Abundance Analysis, Recombination Lines

**Dirs**

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on recombination coefficients for C II lines from Davey et al. 2000A&AS..142...85D.

Adopted from MOCASSIN, Ercolano et al. 2005MNRAS.362.1038E.

02/2003, Yong Zhang, added to MOCASSIN.

10/05/2013, A. Danehkar, Translated to IDL code.

15/04/2017, A. Danehkar, Integration with AtomNeb.

**10/07/2019, A. Danehkar, Made a new function calc\_emiss\_c\_ii\_rl()** for calculating line emissivities and separated it from calc\_abund\_c\_ii\_rl().

03/10/2020, A. Danehkar, Transferred from IDL to Python.

```
pyequib.calc_abund_c_iii_rl(temperature=None,      density=None,      wavelength=None,
↪ line_flux=None, c_iii_rc_data=None, h_i_aeff_data=None)
```

This function determines the ionic abundance from the observed flux intensity for the given wavelength of C III recombination line by using the recombination coefficients from Pequignot et al. 1991A&A...251..680P.

**Returns** type=double. This function returns the ionic abundanc.

**Keywords**

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**wavelength** [in, required, type=float] Line Wavelength in Angstrom

**line\_flux** [in, required, type=float] line flux intensity

**c\_iii\_rc\_data** [in, required, type=array/object] C III recombination coefficients

**h\_i\_aeff\_data** [in, required, type=array/object] H I recombination coefficients

## Examples

For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_ppb91_file=os.path.join(base_dir,data_rc_dir, 'rc_
↳PPB91.fits')
>>> atom_rc_sh95_file= os.path.join(base_dir,data_rc_dir, 'rc_
↳SH95.fits')
>>> atom='h'
>>> ion='ii' # H I
>>> h_i_rc_data=atomneb.read_aeff_sh95(atom_rc_sh95_file, atom,
↳ion)
>>> h_i_aeff_data=h_i_rc_data['aeff'][0]
>>> atom='c'
>>> ion='iv' # C III
>>> c_iii_rc_data=atomneb.read_aeff_ppb91(atom_rc_ppb91_file,
↳atom, ion)
>>> temperature=np.float64(10000.0)
>>> density=np.float64(5000.0)
>>> c_iii_4647_flux = 0.107
>>> wavelength=4647.42
>>> abund_c_iii=pyequib.calc_abund_c_iii_
↳rl(temperature=temperature, density=density,
>>>                                wavelength=wavelength, line_
↳flux=c_iii_4647_flux,
>>>                                c_iii_rc_data=c_iii_rc_data,
↳ h_i_aeff_data=h_i_aeff_data)
>>> print('N(C^3+)/N(H+):', abund_c_iii)
N(C^3+)/N(H+) :    0.00017502840
```

**Categories** Abundance Analysis, Recombination Lines

### Dirs

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on effective radiative recombination coefficients for C III lines from Pequignot, Petitjean, Boisson, C. 1991A&A...251..680P.

18/05/2013, A. Danehkar, Translated to IDL code.

06/04/2017, A. Danehkar, Integration with AtomNeb.

**10/07/2019, A. Danehkar, Made a new function calc\_emiss\_c\_iii\_rl()** for calculating line emissivities and separated it from calc\_abund\_c\_iii\_rl().

03/10/2020, A. Danehkar, Transferred from IDL to Python.

```
pyequib.calc_abund_he_i_rl(temperature=None, density=None, linenum=None, line_flux=None,
he_i_aeff_data=None, h_i_aeff_data=None)
```

This function determines the ionic abundance from the observed flux intensity for the given wavelength of He I recombination line by using the recombination coefficients from Porter et al. 2012MNRAS.425L..28P.

**Returns** type=double. This function returns the ionic abundanc.

**Keywords**

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**linenum** [in, required, type=int] Line Number for Wavelength

Wavelength=4120.84:linenum=7,

Wavelength=4387.93: linenum=8,

Wavelength=4437.55: linenum=9,

Wavelength=4471.50: linenum=10,

Wavelength=4921.93: linenum=12,

Wavelength=5015.68: linenum=13,

Wavelength=5047.74: linenum=14,

Wavelength=5875.66: linenum=15,

Wavelength=6678.16: linenum=16,

Wavelength=7065.25: linenum=17,

Wavelength=7281.35: linenum=18.

**line\_flux** [in, required, type=float] line flux intensity

**he\_i\_aeff\_data** [in, required, type=array/object] He I recombination coefficients

**h\_i\_aeff\_data** [in, required, type=array/object] H I recombination coefficients

**Examples**

For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_he_i_file= os.path.join(base_dir,data_rc_dir, 'rc_
↳he_ii_PFSd12.fits')
>>> atom_rc_sh95_file= os.path.join(base_dir,data_rc_dir, 'rc_
↳SH95.fits')
>>> atom='h'
>>> ion='ii' # H I
>>> h_i_rc_data=atomneb.read_aeff_sh95(atom_rc_sh95_file, atom,
↳ion)
>>> h_i_aeff_data=h_i_rc_data['aeff'][0]
>>> atom='he'
>>> ion='ii' # He I
>>> he_i_rc_data=atomneb.read_aeff_he_i_pfsd12(atom_rc_he_i_
↳file, atom, ion)
```

(continues on next page)



(continued from previous page)

```

>>> he_i_aeff_data=he_i_rc_data['aeff'][0]
>>> temperature=np.float64(10000.0)
>>> density=np.float64(5000.0)
>>> he_i_4471_flux= 2.104
>>> linenum=10# 4471.50
>>> abund_he_i=pyequib.calc_abund_he_i_
↳rl(temperature=temperature, density=density,
>>>                                     linenum=linenum, line_
↳flux=he_i_4471_flux,
>>>                                     he_i_aeff_data=he_i_aeff_
↳data, h_i_aeff_data=h_i_aeff_data)
>>> print('N(He^+)/N(H^+):', abund_he_i)
      N(He^+)/N(H^+):      0.040848393

```

**Categories** Abundance Analysis, Recombination Lines

**Dirs**

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on improved He I emissivities in the case B from Porter et al. 2012MNRAS.425L..28P

15/12/2013, A. Danehkar, IDL code written.

20/03/2017, A. Danehkar, Integration with AtomNeb.

**10/07/2019, A. Danehkar, Made a new function calc\_emiss\_he\_i\_rl()** for calculating line emissivities and separated it from calc\_abund\_he\_i\_rl().

03/10/2020, A. Danehkar, Transferred from IDL to Python.

```

pyequib.calc_abund_he_ii_rl(temperature=None,          density=None,          line_flux=None,
                             he_ii_aeff_data=None, h_i_aeff_data=None)

```

This function determines the ionic abundance from the observed flux intensity for the He II recombination line 4686 Å by using the helium emissivities from Storey & Hummer, 1995MNRAS.272...41S.

**Returns** type=double. This function returns the ionic abundanc.

**Keywords**

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**line\_flux** [in, required, type=float] line flux intensity

**he\_ii\_aeff\_data** [in, required, type=array/object] He II recombination coefficients

**h\_i\_aeff\_data** [in, required, type=array/object] H I recombination coefficients

**Examples**

For example:

```

>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_he_i_file= os.path.join(base_dir,data_rc_dir, 'rc_
↳he_ii_PFSd12.fits')
>>> atom_rc_sh95_file= os.path.join(base_dir,data_rc_dir, 'rc_
↳SH95.fits')
>>> atom='h'
>>> ion='ii' # H I
>>> h_i_rc_data=atomneb.read_aeff_sh95(atom_rc_sh95_file, atom,
↳ion)
>>> h_i_aeff_data=h_i_rc_data['aeff'][0]
>>> atom='he'
>>> ion='iii' # He II
>>> he_ii_rc_data=atomneb.read_aeff_sh95(atom_rc_sh95_file,
↳atom, ion)
>>> he_ii_aeff_data=he_ii_rc_data['aeff'][0]
>>> temperature=np.float64(10000.0)
>>> density=np.float64(5000.0)
>>> he_ii_4686_flux = 135.833
>>> abund_he_ii=pyequib.calc_abund_he_ii_
↳rl(temperature=temperature, density=density,
>>>                                     line_flux=he_ii_4686_flux,
>>>                                     he_ii_aeff_data=he_ii_aeff_
↳data, h_i_aeff_data=h_i_aeff_data)
>>> print('N(He^2+)/N(H^+):', abund_he_ii)
      N(He^2+)/N(H^+) :      0.11228817

```

**Categories** Abundance Analysis, Recombination Lines

**Dirs**

./ Main routines

**Author** Ashkbiz Daneshkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on He II emissivities from Storey & Hummer, 1995MNRAS.272...41S.

15/12/2013, A. Daneshkar, IDL code written.

02/04/2017, A. Daneshkar, Integration with AtomNeb.

**10/07/2019, A. Daneshkar, Made a new function calc\_emiss\_he\_ii\_rl()** for calculating line emissivities and separated it from calc\_abund\_he\_ii\_rl().

03/10/2020, A. Daneshkar, Transferred from IDL to Python.

```

pyequib.calc_abund_n_ii_rl(temperature=None,          density=None,          wavelength=None,
                           line_flux=None,          n_ii_rc_br=None,          n_ii_rc_data=None,
                           h_i_aeff_data=None)

```

This function determines the ionic abundance from the observed flux intensity for the given wavelength of N II recombination line by using the recombination coefficients from Escalante & Victor 1990ApJS...73..513E.

**Returns** type=double. This function returns the ionic abundanc.

**Keywords**

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**wavelength** [in, required, type=float] Line Wavelength in Angstrom

**line\_flux** [in, required, type=float] line flux intensity

**n\_ii\_rc\_br** [in, required, type=array/object] N II branching ratios (Br)

**n\_ii\_rc\_data** [in, required, type=array/object] N II recombination coefficients

**h\_i\_aeff\_data** [in, required, type=array/object] H I recombination coefficients

**Examples** For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_all_file= os.path.join(base_dir,data_rc_dir, 'rc_
↳collection.fits')
>>> atom_rc_sh95_file= os.path.join(base_dir,data_rc_dir, 'rc_SH95.
↳fits')
>>> atom='h'
>>> ion='ii' # H I
>>> h_i_rc_data=atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)
>>> h_i_aeff_data=h_i_rc_data['aeff'][0]
>>> atom='n'
>>> ion='iii' # N II
>>> n_ii_rc_data=atomneb.read_aeff_collection(atom_rc_all_file, atom,
↳ion)
>>> n_ii_rc_data_br=atomneb.read_aeff_collection(atom_rc_all_file,
↳atom, ion, br=True)
>>> temperature=np.float64(10000.0)
>>> density=np.float64(5000.0)
>>> n_ii_4442_flux = 0.017
>>> wavelength=4442.02
>>> abund_n_ii=pyequib.calc_abund_n_ii_rl(temperature=temperature,
↳density=density,
>>>                                     wavelength=wavelength, line_flux=n_
↳ii_4442_flux,
>>>                                     n_ii_rc_br=n_ii_rc_data_br, n_ii_rc_
↳data=n_ii_rc_data,
>>>                                     h_i_aeff_data=h_i_aeff_data)
>>> print('N(N^2+)/N(H+):', abund_n_ii)
N(N^2+)/N(H+): 0.00069297541
```

**Categories** Abundance Analysis, Recombination Lines

**Dirs**

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on Effective recombination coefficients for N II lines from Escalante & Victor 1990ApJS...73..513E.

Adopted from MIDAS Rnii script written by X.W.Liu.

**Revised based on scripts by Yong Zhang added to MOCASSIN, 02/2003** Ercolano et al. 2005MNRAS.362.1038E.

10/05/2013, A. Danehkar, Translated to IDL code.

25/04/2017, A. Danehkar, Integration with AtomNeb.

**10/07/2019, A. Danehkar, Made a new function calc\_emiss\_n\_ii\_rl()** for calculating line emissivities and separated it from calc\_abund\_n\_ii\_rl().

03/10/2020, A. Danehkar, Transferred from IDL to Python.

```
pyequib.calc_abund_n_iii_rl(temperature=None, density=None, wavelength=None,
                             line_flux=None, n_iii_rc_data=None, h_i_aeff_data=None)
```

This function determines the ionic abundance from the observed flux intensity for the given wavelength of N III recombination line by using the recombination coefficients from Pequignot et al. 1991A&A...251..680P.

**Returns** type=double. This function returns the ionic abundanc.

#### Keywords

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**wavelength** [in, required, type=float] Line Wavelength in Angstrom

**line\_flux** [in, required, type=float] line flux intensity

**n\_iii\_rc\_data** [in, required, type=array/object] N III recombination coefficients

**h\_i\_aeff\_data** [in, required, type=array/object] H I recombination coefficients

#### Examples

For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_ppb91_file=os.path.join(base_dir,data_rc_dir, 'rc_
↳PPB91.fits')
>>> atom_rc_sh95_file= os.path.join(base_dir,data_rc_dir, 'rc_
↳SH95.fits')
>>> atom='h'
>>> ion='ii' # H I
>>> h_i_rc_data=atomneb.read_aeff_sh95(atom_rc_sh95_file, atom,
↳ion)
>>> h_i_aeff_data=h_i_rc_data['aeff'][0]
>>> atom='n'
>>> ion='iv' # N III
>>> n_iii_rc_data=atomneb.read_aeff_ppb91(atom_rc_ppb91_file,
↳atom, ion)
>>> temperature=np.float64(10000.0)
```

(continues on next page)

(continued from previous page)

```

>>> density=np.float64(5000.0)
>>> n_iii_4641_flux = 0.245
>>> wavelength=4640.64
>>> abund_n_iii=pyequib.calc_abund_n_iii_
↳rl(temperature=temperature, density=density,
>>>                                wavelength=wavelength, line_
↳flux=n_iii_4641_flux,
>>>                                n_iii_rc_data=n_iii_rc_data,
↳ h_i_aeff_data=h_i_aeff_data)
>>> print('N(N^3+)/N(H+):', abund_n_iii)
N(N^3+)/N(H+):      6.3366175e-05

```

**Categories** Abundance Analysis, Recombination Lines**Dirs**

./ Main routines

**Author** Ashkbiz Danehkar**Copyright** This library is released under a GNU General Public License.**Version** 0.3.0**History** Based on effective radiative recombination coefficients for N III lines from Pequignot, Petitjean, Boisson, C. 1991A&A...251..680P.

10/05/2013, A. Danehkar, IDL code written.

20/04/2017, A. Danehkar, Integration with AtomNeb.

**10/07/2019, A. Danehkar, Made a new function calc\_emiss\_n\_iii\_rl()** for calculating line emissivities and separated it from calc\_abund\_n\_iii\_rl().

03/10/2020, A. Danehkar, Transferred from IDL to Python.

```

pyequib.calc_abund_ne_ii_rl(temperature=None, density=None, wavelength=None,
                             line_flux=None, ne_ii_rc_data=None, h_i_aeff_data=None)

```

This function determines the ionic abundance from the observed flux intensity for the given wavelength of Ne II recombination line by using the recombination coefficients from Kisielius et al. (1998) & Storey (unpublished).

**Returns** type=double. This function returns the ionic abundanc.**Keywords****temperature** [in, required, type=float] electron temperature**density** [in, required, type=float] electron density**wavelength** [in, required, type=float] Line Wavelength in Angstrom**line\_flux** [in, required, type=float] line flux intensity**ne\_ii\_rc\_data** [in, required, type=array/object] Ne II recombination coefficients**h\_i\_aeff\_data** [in, required, type=array/object] H I recombination coefficients**Examples**

For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_all_file= os.path.join(base_dir,data_rc_dir, 'rc_
↳collection.fits')
>>> atom_rc_sh95_file= os.path.join(base_dir,data_rc_dir, 'rc_
↳SH95.fits')
>>> atom='h'
>>> ion='ii' # H I
>>> h_i_rc_data=atomneb.read_aeff_sh95(atom_rc_sh95_file, atom,
↳ion)
>>> h_i_aeff_data=h_i_rc_data['aeff'][0]
>>> atom='ne'
>>> ion='iii' # Ne II
>>> ne_ii_rc_data=atomneb.read_aeff_collection(atom_rc_all_file,
↳ atom, ion)
>>> temperature=np.float64(10000.0)
>>> density=np.float64(5000.0)
>>> ne_ii_3777_flux = 0.056
>>> wavelength=3777.14
>>> abund_ne_ii=pyequib.calc_abund_ne_ii_
↳rl(temperature=temperature, density=density,
↳wavelength=wavelength, line_
↳flux=ne_ii_3777_flux,
↳ne_ii_rc_data=ne_ii_rc_data,
↳h_i_aeff_data=h_i_aeff_data)
>>> print('N(Ne^2+)/N(H+):', abund_ne_ii)
N(Ne^2+)/N(H+) :      0.00043376850
```

**Categories** Abundance Analysis, Recombination Lines

#### Dirs

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on effective radiative recombination coefficients for Ne II lines from Kisieliuss et al. 1998A&AS..133..257K & Storey (unpublished).

Adopted from MOCASSIN, Ercolano et al. 2005MNRAS.362.1038E.

02/2003, Yong Zhang, scripts added to MOCASSIN.

14/05/2013, A. Danehkar, Translated to IDL code.

10/04/2017, A. Danehkar, Integration with AtomNeb.

**10/07/2019, A. Danehkar, Made a new function calc\_emiss\_ne\_ii\_rl()** for calculating line emissivities and separated it from calc\_abund\_ne\_ii\_rl().

03/10/2020, A. Danehkar, Transferred from IDL to Python.

```
pyequib.calc_abund_o_ii_rl(temperature=None,          density=None,          wavelength=None,
                           line_flux=None,          o_ii_rc_br=None,          o_ii_rc_data=None,
                           h_i_aeff_data=None)
```

This function determines the ionic abundance from the observed flux intensity for the given wavelength of O II recombination line by using the recombination coefficients from Storey 1994A&A...282..999S and Liu et al. 1995MNRAS.272..369L.

**Returns** type=double. This function returns the ionic abundanc.

#### Keywords

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**wavelength** [in, required, type=float] Line Wavelength in Angstrom

**line\_flux** [in, required, type=float] line flux intensity

**o\_ii\_rc\_br** [in, required, type=array/object] O II branching ratios (Br)

**o\_ii\_rc\_data** [in, required, type=array/object] O II recombination coefficients

**h\_i\_aeff\_data** [in, required, type=array/object] H I recombination coefficients

**Examples** For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_all_file= os.path.join(base_dir,data_rc_dir, 'rc_
↳collection.fits')
>>> atom_rc_sh95_file= os.path.join(base_dir,data_rc_dir, 'rc_SH95.
↳fits')
>>> atom='h'
>>> ion='ii' # H I
>>> h_i_rc_data=atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)
>>> h_i_aeff_data=h_i_rc_data['aeff'][0]
>>> atom='o'
>>> ion='iii' # O II
>>> o_ii_rc_data=atomneb.read_aeff_collection(atom_rc_all_file, atom,
↳ion)
>>> o_ii_rc_data_br=atomneb.read_aeff_collection(atom_rc_all_file,
↳atom, ion, br=True)
>>> temperature=np.float64(10000.0)
>>> density=np.float64(5000.0)
>>> o_ii_4614_flux = 0.009
>>> wavelength=4613.68
>>> abund_o_ii=pyequib.calc_abund_o_ii_rl(temperature=temperature,
↳density=density,
>>>                                     wavelength=wavelength, line_flux=o_
↳ii_4614_flux,
>>>                                     o_ii_rc_br=o_ii_rc_data_br, o_ii_rc_
↳data=o_ii_rc_data,
>>>                                     h_i_aeff_data=h_i_aeff_data)
>>> print('N(O^2+)/N(H+):', abund_o_ii)
N(O^2+)/N(H+) : 0.0018886330
```

**Categories** Abundance Analysis, Recombination Lines

#### Dirs

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on recombination coefficients for O II lines from Storey 1994A&A...282..999S and Liu et al. 1995MNRAS.272..369L.

Adopted from MIDAS script R0ii.prg written by X.W.Liu.

**Revised based on scripts by Yong Zhang added to MOCASSIN, 02/2003** Ercolano et al. 2005MNRAS.362.1038E.

10/05/2013, A. Danehkar, Translated to IDL code.

25/04/2017, A. Danehkar, Integration with AtomNeb.

**10/07/2019, A. Danehkar, Made a new function calc\_emiss\_o\_ii\_rl()** for calculating line emissivities and separated it from calc\_abund\_o\_ii\_rl().

03/10/2020, A. Danehkar, Transferred from IDL to Python.

```
pyequib.calc_abundance(temperature=None, density=None, line_flux=None, atomic_levels=None,
                        elj_data=None, omij_data=None, aij_data=None, h_i_aeff_data=None)
```

This function determines the ionic abundance from the observed flux intensity for specified ion with level(s) by solving atomic level populations and line emissivities in statistical equilibrium for given electron density and temperature.

**Returns** type=double. This function returns the ionic abundanc.

#### Keywords

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**line\_flux** [in, required, type=float] line flux intensity

**atomic\_levels** [in, required, type=string] level(s) e.g '1,2/', '1,2,1,3/'

**elj\_data** [in, required, type=array/object] energy levels (Ej) data

**omij\_data** [in, required, type=array/object] collision strengths (omega\_ij) data

**aij\_data** [in, required, type=array/object] transition probabilities (Aij) data

**h\_i\_aeff\_data** [in, required, type=array/object] H I recombination coefficients

**Examples** For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_dir = os.path.join('atomic-data', 'chianti70')
>>> atom_elj_file = os.path.join(base_dir, data_dir, 'AtomElj.fits')
>>> atom_omij_file = os.path.join(base_dir, data_dir, 'AtomOmij.fits')
>>> atom_aij_file = os.path.join(base_dir, data_dir, 'AtomAij.fits')
>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_sh95_file= os.path.join(base_dir, data_rc_dir, 'rc_SH95.
↳fits')
>>> atom='o'
>>> ion='iii'
```

(continues on next page)



(continued from previous page)

```

>>> o_iii_elj=atomneb.read_elj(atom_elj_file, atom, ion, level_num=5)
↳ # read Energy Levels (Ej)
>>> o_iii_omij=atomneb.read_omij(atom_omij_file, atom, ion) # read_
↳ Collision Strengths (Omegaij)
>>> o_iii_aij=atomneb.read_aij(atom_aij_file, atom, ion) # read_
↳ Transition Probabilities (Aij)
>>> atom='h'
>>> ion='ii' # H I
>>> hi_rc_data=atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)
>>> h_i_aeff_data=hi_rc_data['aeff'][0]
>>> temperature=np.float64(10000.0)
>>> density=np.float64(5000.0)
>>> atomic_levels='3,4/'
>>> iobs5007=np.float64(1200.0)
>>> abb5007=np.float64(0.0)
>>> abb5007=pyequib.calc_abundance(temperature=temperature,
↳ density=density,
>>>                                line_flux=iobs5007, atomic_levels=atomic_
↳ levels,
>>>                                elj_data=o_iii_elj, omij_data=o_iii_omij,
>>>                                aij_data=o_iii_aij, h_i_aeff_data=h_i_rc_
↳ data['aeff'][0])
>>> print('N(O^2+)/N(H+):', abb5007)
N(O^2+)/N(H+): 0.00041256231

```

**Categories** Abundance Analysis, Collisionally Excited Lines

## Dirs

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** 15/09/2013, A. Danehkar, Translated from FORTRAN to IDL code.

20/10/2016, A. Danehkar, Replaced str2int with strnumber.

**20/10/2016, A. Danehkar, Replaced CFY, SPLMAT, and CFD with IDL function INTERPOL( /SPLINE).**

**20/10/2016, A. Danehkar, Replaced LUSLV with IDL LAPACK function LA\_LINEAR\_EQUATION.**

**15/11/2016, A. Danehkar, Replaced LA\_LINEAR\_EQUATION (not work in GDL) with IDL function LUDC & LUSOL.**

**19/11/2016, A. Danehkar, Replaced INTERPOL (not accurate) with SPL\_INIT & SPL\_INTERP.**

**20/11/2016, A. Danehkar, Made a new function calc\_populations() for solving atomic level populations and separated it from calc\_abundance(), calc\_density() and calc\_temperature().**

**21/11/2016, A. Danehkar, Made a new function calc\_emissivity() for calculating line emissivities and separated it from calc\_abundance().**

**10/03/2017, A. Danehkar, Integration with AtomNeb, now uses atomic data input elj\_data, omij\_data, aij\_data.**

**12/06/2017, A. Danekar, Cleaning the function, and remove unused variables** from `calc_abundance()`.

03/10/2020, A. Danekar, Transferred from IDL to Python.

#### FORTTRAN HISTORY:

03/05/1981, I.D.Howarth, Version 1.

05/05/1981, I.D.Howarth, Minibug fixed!

07/05/1981, I.D.Howarth, Now takes collision rates or strengths.

03/08/1981, S.Adams, Interpolates collision strengths.

07/08/1981, S.Adams, Input method changed.

**19/11/1984, R.E.S.Clegg, SA files entombed in scratch disk. Logical** filenames given to SA's data files.

08/1995, D.P.Ruffle, Changed input file format. Increased matrices.

**02/1996, X.W.Liu, Tidy up. SUBROUTINES SPLMAT, HGEN, CFY and CFD** modified such that matrix sizes (i.e. maximum of Te and maximum no of levels) can now be changed by modifying the parameters NDIM1, NDIM2 and N in the Main program. EASY! Now takes collision rates as well. All variables are declared explicitly. Generate two extra files (ionpop.lis and ionra) of plain stream format for plotting.

**06/1996, C.J.Pritchett, Changed input data format for cases IBIG=1,2.** Fixed reading bug for IBIG=2 case. Now reads reformatted upsilons (easier to see and the 0 0 0 data end is excluded for these cases). The A values have a different format for IBIG=.

2006, B.Ercolano, Converted to F90.

```
pyequib.calc_crit_density(temperature=None, elj_data=None, omij_data=None, aij_data=None,  
                           level_num=None, irats=None)
```

This function calculates critical densities in statistical equilibrium for given electron temperature.

**Returns** type=array/object. This function returns the critical densities.

#### Keywords

**temperature** [in, required, type=float] electron temperature

**elj\_data** [in, required, type=array/object] energy levels (Ej) data

**omij\_data** [in, required, type=array/object] collision strengths (omega\_ij) data

**aij\_data** [in, required, type=array/object] transition probabilities (Aij) data

**level\_num** [in, type=int] Number of levels

**irats** [in, type=int] Else Coll. rates = tabulated values \* 10 \*\* irats

**Examples** For example:

```
>>> import pyequib  
>>> import atomneb  
>>> import os  
>>> base_dir = '../externals/atomneb/'  
>>> data_dir = os.path.join('atomic-data', 'chianti70')  
>>> atom_elj_file = os.path.join(base_dir, data_dir, 'AtomElj.fits')  
>>> atom_omij_file = os.path.join(base_dir, data_dir, 'AtomOmicron.fits')  
>>> atom_aij_file = os.path.join(base_dir, data_dir, 'AtomAij.fits')
```

(continues on next page)

(continued from previous page)

```

>>> atom='s'
>>> ion='ii'
>>> s_ii_elj=atomneb.read_elj(atom_elj_file, atom, ion, level_num=5)
↳ # read Energy Levels (Ej)
>>> s_ii_omij=atomneb.read_omij(atom_omij_file, atom, ion) # read_
↳ Collision Strengths (Omegaij)
>>> s_ii_aij=atomneb.read_aij(atom_aij_file, atom, ion) # read_
↳ Transition Probabilities (Aij) >>> temperature=np.
↳ float64(10000.0)
>>> n_crit=pyequib.calc_crit_density(temperature=temperature,
>>>                                  elj_data=s_ii_elj, omij_data=s_ii_omij,
>>>                                  aij_data=s_ii_aij)
>>> print('Critical Densities:', n_crit)
Critical Densities:      0.0000000      5007.8396      1732.
↳ 8414      1072685.0      2220758.1

```

**Categories** Plasma Diagnostics, Abundance Analysis, Collisionally Excited Lines

### Dirs

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** 15/09/2013, A. Danehkar, Translated from FORTRAN to IDL code.

20/10/2016, A. Danehkar, Replaced str2int with strnumber.

**20/10/2016, A. Danehkar, Replaced CFY, SPLMAT, and CFD with IDL function INTERPOL( /SPLINE).**

**20/10/2016, A. Danehkar, Replaced LUSLV with IDL LAPACK function LA\_LINEAR\_EQUATION.**

**15/11/2016, A. Danehkar, Replaced LA\_LINEAR\_EQUATION (not work in GDL) with IDL function LUDC & LUSOL.**

**19/11/2016, A. Danehkar, Replaced INTERPOL (not accurate) with SPL\_INIT & SPL\_INTERP.**

**20/11/2016, A. Danehkar, Made a new function calc\_populations() for solving atomic level populations and separated it from calc\_abundance(), calc\_density() and calc\_temperature().**

**10/03/2017, A. Danehkar, Integration with AtomNeb, now uses atomic data input elj\_data, omij\_data, aij\_data.**

**12/06/2017, A. Danehkar, Cleaning the function, and remove unused varibales from calc\_populations().**

**27/02/2019, A. Danehkar, Simplify the calc\_populations() routine for external usage.**

**01/03/2019, A. Danehkar, Create the calc\_crit\_density() routine from the calc\_populations() routine.**

04/03/2019, A. Danehkar, Use the get\_omij\_temp() routine.

03/10/2020, A. Danehkar, Transferred from IDL to Python.

FORTRAN HISTORY:

03/05/1981, I.D.Howarth, Version 1.

05/05/1981, I.D.Howarth, Minibug fixed!

07/05/1981, I.D.Howarth, Now takes collision rates or strengths.

03/08/1981, S.Adams, Interpolates collision strengths.

07/08/1981, S.Adams, Input method changed.

**19/11/1984, R.E.S.Clegg, SA files entombed in scratch disk.** Logical filenames given to SA's data files.

08/1995, D.P.Ruffle, Changed input file format. Increased matrices.

**02/1996, X.W.Liu, Tidy up.** SUBROUTINES SPLMAT, HGEN, CFY and CFD modified such that matrix sizes (i.e. maximum of Te and maximum no of levels) can now be changed by modifying the parameters NDIM1, NDIM2 and N in the Main program. EASY! Now takes collision rates as well. All variables are declared explicitly. Generate two extra files (ionpop.lis and ionra) of plain stream format for plotting.

**06/1996, C.J.Pritchett, Changed input data format for cases IBIG=1,2.** Fixed reading bug for IBIG=2 case. Now reads reformatted upsilons (easier to see and the 0 0 0 data end is excluded for these cases). The A values have a different format for IBIG=.

2006, B.Ercolano, Converted to F90.

```
pyequib.calc_density(line_flux_ratio=None, temperature=None, upper_levels=None,
                    lower_levels=None, elj_data=None, omij_data=None, aij_data=None,
                    low_density=None, high_density=None, num_density=None,
                    min_temperature=None)
```

This function determines electron density from given flux intensity ratio for specified ion with upper level(s) lower level(s) by solving atomic level populations and line emissivities in statistical equilibrium for given electron temperature.

**Returns** type=double. This function returns the electron density.

#### Keywords

**line\_flux\_ratio** [in, required, type=float] flux intensity ratio

**temperature** [in, required, type=float] electron temperature

**upper\_levels** [in, required, type=string] upper atomic level(s) e.g '1,2', '1,2,1,3'

**lower\_levels** [in, required, type=string] lower atomic level(s) e.g '1,2', '1,2,1,3'

**elj\_data** [in, required, type=array/object] energy levels (Ej) data

**omij\_data** [in, required, type=array/object] collision strengths (omega\_ij) data

**aij\_data** [in, required, type=array/object] transition probabilities (Aij) data

**low\_density** [in, optional, type=float] lower density range

**high\_density** [in, optional, type=float] upper density range

**num\_density** [in, optional, type=integer] number of the iteration step

**min\_temperature** [in, optional, type=float] minimum temperature

**Examples** For example:

```

>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_dir = os.path.join('atomic-data', 'chianti70')
>>> atom_elj_file = os.path.join(base_dir, data_dir, 'AtomElj.fits')
>>> atom_omij_file = os.path.join(base_dir, data_dir, 'AtomOmij.fits')
>>> atom_aij_file = os.path.join(base_dir, data_dir, 'AtomAij.fits')
>>> atom='s'
>>> ion='ii'
>>> s_ii_elj=atomneb.read_elj(atom_elj_file, atom, ion, level_num=5)
↳ # read Energy Levels (Ej)
>>> s_ii_omij=atomneb.read_omij(atom_omij_file, atom, ion) # read_
↳ Collision Strengths (Omegaij)
>>> s_ii_aij=atomneb.read_aij(atom_aij_file, atom, ion) # read_
↳ Transition Probabilities (Aij) >>> upper_levels='1,2/'
>>> lower_levels='1,3/'
>>> temperature=np.float64(7000.0) #
>>> line_flux_ratio=np.float64(1.506) #
>>> density=pyequib.calc_density(line_flux_ratio=line_flux_ratio,
↳ temperature=temperature,
>>> upper_levels=upper_levels, lower_
↳ levels=lower_levels,
>>> elj_data=s_ii_elj, omij_data=s_ii_omij,
>>> aij_data=s_ii_aij)
>>> print("Electron Density:", density)
Electron Density: 2312.6395

```

**Categories** Plasma Diagnostics, Collisionally Excited Lines

#### Dirs

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** 15/09/2013, A. Danehkar, Translated from FORTRAN to IDL code.

20/10/2016, A. Danehkar, Replaced str2int with strnumber.

20/10/2016, A. Danehkar, Replaced CFY, SPLMAT, and CFD with IDL function INTERPOL( /SPLINE).

20/10/2016, A. Danehkar, Replaced LUSLV with IDL LAPACK function LA\_LINEAR\_EQUATION.

15/11/2016, A. Danehkar, Replaced LA\_LINEAR\_EQUATION (not work in GDL) with IDL function LUDC & LUSOL.

19/11/2016, A. Danehkar, Replaced INTERPOL (not accurate) with SPL\_INIT & SPL\_INTERP.

20/11/2016, A. Danehkar, Made a new function calc\_populations() for solving atomic level populations and separated it from calc\_abundance(), calc\_density() and calc\_temperature().

10/03/2017, A. Danehkar, Integration with AtomNeb, now uses atomic data input elj\_data, omij\_data, aij\_data.

**12/06/2017, A. Danekar, Cleaning the function, and remove unused variables** from `calc_density()`.

**27/02/2019, A. Danekar, Fix a bug in the atomic level assumption, and** use the simplified `calc_populations()` routine.

04/03/2019, A. Danekar, Use the `get_omij_temp()` routine.

24/05/2019, A. Danekar, Add the optional density range.

03/10/2020, A. Danekar, Transferred from IDL to Python.

#### FORTTRAN HISTORY:

03/05/1981, I.D.Howarth, Version 1.

05/05/1981, I.D.Howarth, Minibug fixed!

07/05/1981, I.D.Howarth, Now takes collision rates or strengths.

03/08/1981, S.Adams, Interpolates collision strengths.

07/08/1981, S.Adams, Input method changed.

**19/11/1984, R.E.S.Clegg, SA files entombed in scratch disk. Logical** filenames given to SA's data files.

08/1995, D.P.Ruffle, Changed input file format. Increased matrices.

**02/1996, X.W.Liu, Tidy up. SUBROUTINES SPLMAT, HGEN, CFY and CFD** modified such that matrix sizes (i.e. maximum of Te and maximum no of levels) can now be changed by modifying the parameters NDIM1, NDIM2 and N in the Main program. EASY! Now takes collision rates as well. All variables are declared explicitly. Generate two extra files (ionpop.lis and ionra) of plain stream format for plotting.

**06/1996, C.J.Pritchett, Changed input data format for cases IBIG=1,2.** Fixed readin bug for IBIG=2 case. Now reads reformatted upsilons (easier to see and the 0 0 0 data end is excluded for these cases). The A values have a different format for IBIG=.

2006, B.Ercolano, Converted to F90.

```
pyequib.calc_emiss_c_ii_rl(temperature=None, density=None, wavelength=None,
                           c_ii_rc_data=None)
```

This function calculates the emissivity for the given wavelength of C II recombination line by using the recombination coefficients from from Davey et al. (2000) 2000A&AS..142...85D.

**Returns** type=double. This function returns the line emissivity.

#### Keywords

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**wavelength** [in, required, type=float] Line Wavelength in Angstrom

**c\_ii\_rc\_data** [in, required, type=array/object] C II recombination coefficients

**Examples** For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
```

(continues on next page)

(continued from previous page)

```

>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_all_file= os.path.join(base_dir,data_rc_dir, 'rc_
↳collection.fits')
>>> atom_rc_sh95_file= os.path.join(base_dir,data_rc_dir, 'rc_SH95.
↳fits')
>>>
>>> atom='c'
>>> ion='iii' # C II
>>> c_ii_rc_data=atomneb.read_aeff_collection(atom_rc_all_file, atom,
↳ion)
>>> temperature=np.float64(10000.0)
>>> density=np.float64(5000.0)
>>> wavelength=6151.43
>>> emiss_c_ii=pyequib.calc_emiss_c_ii_rl(temperature=temperature,
↳density=density,
>>>                                     wavelength=wavelength,
>>>                                     c_ii_rc_data=c_ii_rc_data)
>>> print('Emissivity:', emiss_c_ii)
Emissivity: 5.4719511e-26

```

**Categories** Abundance Analysis, Recombination Lines, Emissivity

#### Dirs

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on recombination coefficients for C II lines from Davey et al. 2000A&AS..142...85D.

Adopted from MOCASSIN, Ercolano et al. 2005MNRAS.362.1038E.

02/2003, Yong Zhang, added to MOCASSIN.

10/05/2013, A. Danehkar, Translated to IDL code.

15/04/2017, A. Danehkar, Integration with AtomNeb.

**10/07/2019, A. Danehkar, Made a new function calc\_emiss\_c\_ii\_rl()** for calculating line emissivities and separated it from calc\_abund\_c\_ii\_rl().

03/10/2020, A. Danehkar, Transferred from IDL to Python.

```

pyequib.calc_emiss_c_iii_rl(temperature=None,      density=None,      wavelength=None,
                             c_iii_rc_data=None)

```

This function calculates the emissivity for the given wavelength of C III recombination line by using the recombination coefficients from Pequignot et al. 1991A&A...251..680P.

**Returns** type=double. This function returns the line emissivity.

#### Keywords

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**wavelength** [in, required, type=float] Line Wavelength in Angstrom

**c\_iii\_rc\_data** [in, required, type=array/object] C III recombination coefficients

### Examples

For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_ppb91_file=os.path.join(base_dir,data_rc_dir, 'rc_
↳PPB91.fits')
>>> atom_rc_sh95_file= os.path.join(base_dir,data_rc_dir, 'rc_
↳SH95.fits')
>>>
>>> atom='c'
>>> ion='iv' # C III
>>> c_iii_rc_data=atomneb.read_aeff_ppb91(atom_rc_ppb91_file,
↳atom, ion)
>>> temperature=np.float64(10000.0)
>>> density=np.float64(5000.0)
>>> wavelength=4647.42
>>> emiss_c_iii=pyequib.calc_emiss_c_iii_
↳rl(temperature=temperature, density=density,
>>>                                     wavelength=wavelength,
>>>                                     c_iii_rc_data=c_iii_rc_data)
>>> print('Emissivity:', emiss_c_iii)
Emissivity: 7.5749632e-25
```

**Categories** Abundance Analysis, Recombination Lines, Emissivity

### Dirs

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on effective radiative recombination coefficients for C III lines from Pequignot, Petitjean, Boisson, C. 1991A&A...251..680P.

18/05/2013, A. Danehkar, Translated to IDL code.

06/04/2017, A. Danehkar, Integration with AtomNeb.

**10/07/2019, A. Danehkar, Made a new function calc\_emiss\_c\_iii\_rl()** for calculating line emissivities and separated it from calc\_abund\_c\_iii\_rl().

03/10/2020, A. Danehkar, Transferred from IDL to Python.

`pyequib.calc_emiss_h_beta` (*temperature=None, density=None, h\_i\_aeff\_data=None*)

This function calculates the emissivity for H\_beta 4861A  $\text{Emis}(\text{Hbeta}) = 4\pi j(\text{HBeta } 4861 \text{ \AA})/N_p N_e$  for the given temperature and density by using the helium emissivities from Storey & Hummer, 1995MNRAS.272...41S.

### Private



**Returns** type=double. This function returns the H beta emissivity  $4\pi j(\text{HBeta } 4861)/N_p \text{ Ne}$ .

**Keywords**

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**h\_i\_aeff\_data** [in, required, type=array/object] H I recombination coefficients

**Categories** Abundance Analysis, Recombination Lines, Emissivity

**Dirs**

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on H I emissivities from Storey & Hummer, 1995MNRAS.272...41S.

25/08/2012, A. Danehkar, IDL code written.

11/03/2017, A. Danehkar, Integration with AtomNeb.

10/07/2019, A. Danehkar, Change from logarithmic to linear

03/10/2020, A. Danehkar, Transferred from IDL to Python.

```
pyequib.calc_emiss_he_i_rl (temperature=None,          density=None,          linenum=None,
                             he_i_aeff_data=None)
```

This function calculates the emissivity for the given wavelength of He I recombination line by using the recombination coefficients from Porter et al. 2012MNRAS.425L..28P.

**Returns** type=double. This function returns the line emissivity.

**Keywords**

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**linenum** [in, required, type=int] Line Number for Wavelength

Wavelength=4120.84:linenum=7,

Wavelength=4387.93: linenum=8,

Wavelength=4437.55: linenum=9,

Wavelength=4471.50: linenum=10,

Wavelength=4921.93: linenum=12,

Wavelength=5015.68: linenum=13,

Wavelength=5047.74: linenum=14,

Wavelength=5875.66: linenum=15,

Wavelength=6678.16: linenum=16,

Wavelength=7065.25: linenum=17,

Wavelength=7281.35: linenum=18.

**line\_flux** [in, required, type=float] line flux intensity

**he\_i\_aeff\_data** [in, required, type=array/object] He I recombination coefficients

**Examples** For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_he_i_file= filepath('rc_he_ii_PFS12.fits', root_dir=base_
↳dir, subdir=data_rc_dir )
>>> atom_rc_sh95_file= os.path.join(base_dir,data_rc_dir, 'rc_SH95.
↳fits')
>>>
>>> atom='he'
>>> ion='ii' # He I
>>> he_i_rc_data=atomneb.read_aeff_he_i_pfsd12(atom_rc_he_i_file,
↳atom, ion)
>>> he_i_aeff_data=he_i_rc_data['aeff'][0]
>>> temperature=np.float64(10000.0)
>>> density=np.float64(5000.0)
>>> linenum=10# 4471.50
>>> emiss_he_i=pyequib.calc_emiss_he_i_rl(temperature=temperature,
↳density=density,
>>>                                     linenum=linenum,
>>>                                     he_i_aeff_data=he_i_aeff_data)
>>> print('Emissivity:', emiss_he_i)
Emissivity: 6.3822830e-26
```

**Categories** Abundance Analysis, Recombination Lines, Emissivity

**Dirs**

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on improved He I emissivities in the case B from Porter et al. 2012MNRAS.425L..28P

15/12/2013, A. Danehkar, IDL code written.

20/03/2017, A. Danehkar, Integration with AtomNeb.

**10/07/2019, A. Danehkar, Made a new function calc\_emiss\_he\_i\_rl()** for calculating line emissivities and separated it from calc\_abund\_he\_i\_rl().

03/10/2020, A. Danehkar, Transferred from IDL to Python.

`pyequib.calc_emiss_he_ii_rl(temperature=None, density=None, he_ii_aeff_data=None)`

This function calculates the emissivity for the He II recombination line 4686 Å by using the helium emissivities from Storey & Hummer, 1995MNRAS.272...41S.

**Returns** type=double. This function returns the line emissivity.

**Keywords**

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**he\_ii\_aeff\_data** [in, required, type=array/object] He II recombination coefficients

**Examples** For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_he_i_file= os.path.join(base_dir,data_rc_dir, 'rc_he_ii_
↳PFSD12.fits')
>>> atom_rc_sh95_file= os.path.join(base_dir,data_rc_dir, 'rc_SH95.
↳fits')
>>>
>>> atom='he'
>>> ion='iii' # He II
>>> he_ii_rc_data=atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)
>>> he_ii_aeff_data=he_ii_rc_data['aeff'][0]
>>> temperature=np.float64(10000.0)
>>> density=np.float64(5000.0)
>>> he_ii_4686_flux = 135.833
>>> emiss_he_ii=pyequib.calc_emiss_he_ii_rl(temperature=temperature,
↳density=density,
                                     he_ii_aeff_data=he_ii_aeff_data)
>>> print('Emissivity:', emiss_he_ii)
Emissivity: 1.4989134e-24
```

**Categories** Abundance Analysis, Recombination Lines, Emissivity

**Dirs**

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on He II emissivities from Storey & Hummer, 1995MNRAS.272...41S.

15/12/2013, A. Danehkar, IDL code written.

02/04/2017, A. Danehkar, Integration with AtomNeb.

10/07/2019, A. Danehkar, Change from logarithmic to linear

**10/07/2019, A. Danehkar, Made a new function calc\_emiss\_he\_ii\_rl()** for calculating line emissivities and separated it from calc\_abund\_he\_ii\_rl().

03/10/2020, A. Danehkar, Transferred from IDL to Python.

```
pyequib.calc_emiss_n_ii_rl(temperature=None, density=None, wavelength=None,
                             n_ii_rc_br=None, n_ii_rc_data=None)
```

This function calculates the emissivity for the given wavelength of N II recombination line by using the recombination coefficients from Escalante & Victor 1990ApJS...73..513E.

**Returns** type=double. This function returns the line emissivity.

**Keywords**

**temperature** [in, required, type=float] electron temperature  
**density** [in, required, type=float] electron density  
**wavelength** [in, required, type=float] Line Wavelength in Angstrom  
**n\_ii\_rc\_br** [in, required, type=array/object] N II branching ratios (Br)  
**n\_ii\_rc\_data** [in, required, type=array/object] N II recombination coefficients

**Examples** For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_all_file= os.path.join(base_dir,data_rc_dir, 'rc_
↳collection.fits')
>>> atom_rc_sh95_file= os.path.join(base_dir,data_rc_dir, 'rc_SH95.
↳fits')
>>> atom='h'
>>> ion='ii' # H I
>>> h_i_rc_data=atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)
>>> h_i_aeff_data=h_i_rc_data['aeff'][0]
>>> atom='n'
>>> ion='iii' # N II
>>> n_ii_rc_data=atomneb.read_aeff_collection(atom_rc_all_file, atom,
↳ion)
>>> n_ii_rc_data_br=atomneb.read_aeff_collection(atom_rc_all_file,
↳atom, ion, br=True)
>>> temperature=np.float64(10000.0)
>>> density=np.float64(5000.0)
>>> wavelength=4442.02
>>> emiss_n_ii=pyequib.calc_emiss_n_ii_rl(temperature=temperature,
↳density=density,
>>>                                     wavelength=wavelength,
>>>                                     n_ii_rc_br=n_ii_rc_data_br, n_ii_rc_
↳data=n_ii_rc_data,
>>>                                     h_i_aeff_data=h_i_aeff_data)
>>> print('Emissivity:', emiss_n_ii)
Emissivity: 3.0397397e-26
```

**Categories** Abundance Analysis, Recombination Lines, Emissivity

**Dirs**

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on Effective recombination coefficients for N II lines from Escalante & Victor 1990ApJS...73..513E.

Adopted from MIDAS Rnii script written by X.W.Liu.

**Revised based on scripts by Yong Zhang added to MOCASSIN, 02/2003** Ercolano et al. 2005MNRAS.362.1038E.

10/05/2013, A. Danehkar, Translated to IDL code.

25/04/2017, A. Danehkar, Integration with AtomNeb.

**10/07/2019, A. Danehkar, Made a new function `calc_emiss_n_ii_rl()`** for calculating line emissivities and separated it from `calc_abund_n_ii_rl()`.

03/10/2020, A. Danehkar, Transferred from IDL to Python.

```
pyequib.calc_emiss_n_iii_rl(temperature=None, density=None, wavelength=None,
                             n_iii_rc_data=None)
```

This function calculates the emissivity for the given wavelength of N III recombination line by using the recombination coefficients from Pequignot et al. 1991A&A...251..680P.

**Returns** type=double. This function returns the line emissivity.

#### Keywords

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**wavelength** [in, required, type=float] Line Wavelength in Angstrom

**n\_iii\_rc\_data** [in, required, type=array/object] N III recombination coefficients

#### Examples

For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_ppb91_file=os.path.join(base_dir,data_rc_dir, 'rc_
↳PPB91.fits')
>>> atom_rc_sh95_file= os.path.join(base_dir,data_rc_dir, 'rc_
↳SH95.fits')
>>>
>>> atom='n'
>>> ion='iv' # N III
>>> n_iii_rc_data=atomneb.read_aeff_ppb91(atom_rc_ppb91_file,
↳atom, ion)
>>> temperature=np.float64(10000.0)
>>> density=np.float64(5000.0)
>>> wavelength=4640.64
>>> emiss_n_iii=pyequib.calc_abund_n_iii_
↳rl(temperature=temperature, density=density,
>>>                                     wavelength=wavelength,
>>>                                     n_iii_rc_data=n_iii_rc_data)
>>> print('Emissivity:', emiss_n_iii)
Emissivity: 4.7908644e-24
```

**Categories** Abundance Analysis, Recombination Lines, Emissivity

#### Dirs

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on effective radiative recombination coefficients for N III lines from Pequignot, Petitjean, Boisson, C. 1991A&A...251..680P.

10/05/2013, A. Danehkar, IDL code written.

20/04/2017, A. Danehkar, Integration with AtomNeb.

**10/07/2019, A. Danehkar, Made a new function `calc_emiss_n_iii_rl()` for calculating line emissivities and separated it from `calc_abund_n_iii_rl()`.**

03/10/2020, A. Danehkar, Transferred from IDL to Python.

```
pyequib.calc_emiss_ne_ii_rl(temperature=None, density=None, wavelength=None,
                             ne_ii_rc_data=None)
```

This function calculates the emissivity for the given wavelength of Ne II recombination line by using the recombination coefficients from Kisielius et al. (1998) & Storey (unpublished).

**Returns** type=double. This function returns the line emissivity.

#### Keywords

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**wavelength** [in, required, type=float] Line Wavelength in Angstrom

**ne\_ii\_rc\_data** [in, required, type=array/object] Ne II recombination coefficients

#### Examples

For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_all_file = os.path.join(base_dir, data_rc_dir, 'rc_
↳ collection.fits')
>>> atom_rc_sh95_file = os.path.join(base_dir, data_rc_dir, 'rc_
↳ SH95.fits')
>>>
>>> atom='ne'
>>> ion='iii' # Ne II
>>> ne_ii_rc_data=atomneb.read_aeff_collection(atom_rc_all_file,
↳ atom, ion)
>>> temperature=np.float64(10000.0)
>>> density=np.float64(5000.0)
>>> wavelength=3777.14
>>> emiss_ne_ii=pyequib.calc_emiss_ne_ii_
↳ rl(temperature=temperature, density=density,
>>>                                     wavelength=wavelength,
>>>                                     ne_ii_rc_data=ne_ii_rc_data,
↳ h_i_aeff_data=h_i_aeff_data)
>>> print('Emissivity:', emiss_ne_ii)
Emissivity: 1.5996881e-25
```

**Categories** Abundance Analysis, Recombination Lines, Emissivity

**Dirs**

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on effective radiative recombination coefficients for Ne II lines from Kisielius et al. 1998A&AS..133..257K & Storey (unpublished).

Adopted from MOCASSIN, Ercolano et al. 2005MNRAS.362.1038E.

02/2003, Yong Zhang, scripts added to MOCASSIN.

14/05/2013, A. Danehkar, Translated to IDL code.

10/04/2017, A. Danehkar, Integration with AtomNeb.

**10/07/2019, A. Danehkar, Made a new function `calc_emiss_ne_ii_rl()` for calculating line emissivities and separated it from `calc_abund_ne_ii_rl()`.**

03/10/2020, A. Danehkar, Transferred from IDL to Python.

```
pyequib.calc_emiss_o_ii_rl(temperature=None, density=None, wavelength=None,
                           o_ii_rc_br=None, o_ii_rc_data=None)
```

This function calculates the emissivity for the given wavelength of O II recombination line by using the recombination coefficients from Storey 1994A&A...282..999S and Liu et al. 1995MNRAS.272..369L.

**Returns** type=double. This function returns the line emissivity.

**Keywords**

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**wavelength** [in, required, type=float] Line Wavelength in Angstrom

**o\_ii\_rc\_br** [in, required, type=array/object] O II branching ratios (Br)

**o\_ii\_rc\_data** [in, required, type=array/object] O II recombination coefficients

**Examples** For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_all_file= os.path.join(base_dir,data_rc_dir, 'rc_
↳collection.fits')
>>> atom_rc_sh95_file= os.path.join(base_dir,data_rc_dir, 'rc_SH95.
↳fits')
>>>
>>> atom='o'
>>> ion='iii' # O II
>>> o_ii_rc_data=atomneb.read_aeff_collection(atom_rc_all_file, atom,
↳ion)
```

(continues on next page)

(continued from previous page)

```
>>> o_ii_rc_data_br=atomneb.read_aeff_collection(atom_rc_all_file,
↳atom, ion, br=True)
>>> temperature=np.float64(10000.0)
>>> density=np.float64(5000.0)
>>> wavelength=4613.68
>>> emiss_o_ii=pyequib.calc_emiss_o_ii_rl(temperature=temperature,
↳density=density,
>>>                                     wavelength=wavelength,
>>>                                     o_ii_rc_br=o_ii_rc_data_br, o_ii_rc_
↳data=o_ii_rc_data,
>>>                                     h_i_aeff_data=h_i_aeff_data)
>>> print('Emissivity:', emiss_o_ii)
Emissivity: 5.9047319e-27
```

**Categories** Abundance Analysis, Recombination Lines, Emissivity

### Dirs

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on recombination coefficients for O II lines from Storey 1994A&A...282..999S and Liu et al. 1995MNRAS.272..369L.

Adopted from MIDAS script Roi.i.prg written by X.W.Liu.

**Revised based on scripts by Yong Zhang added to MOCASSIN, 02/2003** Ercolano et al. 2005MNRAS.362.1038E.

10/05/2013, A. Danehkar, Translated to IDL code.

25/04/2017, A. Danehkar, Integration with AtomNeb.

**10/07/2019, A. Danehkar, Made a new function calc\_emiss\_o\_ii\_rl()** for calculating line emissivities and separated it from calc\_abund\_o\_ii\_rl().

03/10/2020, A. Danehkar, Transferred from IDL to Python.

`pyequib.calc_emissivity(temperature=None, density=None, atomic_levels=None, elj_data=None, omij_data=None, aij_data=None)`

This function calculates line emissivities for specified ion with level(s) by solving atomic level populations and in statistical equilibrium for given electron density and temperature.

**Returns** type=double. This function returns the line emissivity.

### Keywords

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**atomic\_levels** [In, required, type=string] level(s) e.g '1,2/', '1,2,1,3/'

**elj\_data** [in, required, type=array/object] energy levels (Ej) data

**omij\_data** [in, required, type=array/object] collision strengths (omega\_ij) data

**aij\_data** [in, required, type=array/object] transition probabilities (Aij) data



**Examples** For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_dir = os.path.join('atomic-data', 'chianti70')
>>> atom_elj_file = os.path.join(base_dir, data_dir, 'AtomElj.fits')
>>> atom_omij_file = os.path.join(base_dir, data_dir, 'AtomOmij.fits')
>>> atom_aij_file = os.path.join(base_dir, data_dir, 'AtomAij.fits')
>>> atom='o'
>>> ion='iii'
>>> o_iii_elj=atomneb.read_elj(atom_elj_file, atom, ion, level_num=5)
↪# read Energy Levels (Ej)
>>> o_iii_omij=atomneb.read_omij(atom_omij_file, atom, ion) # read_
↪Collision Strengths (Omegaij)
>>> o_iii_aij=atomneb.read_aij(atom_aij_file, atom, ion) # read_
↪Transition Probabilities (Aij)
>>> temperature=np.float64(10000.0)
>>> density=np.float64(5000.0)
>>> atomic_levels='3,4/'
>>> emiss5007=np.float64(0.0)
>>> emiss5007=pyequib.calc_emissivity(temperature=temperature,
↪density=density,
>>>                                atomic_levels=atomic_levels,
>>>                                elj_data=o_iii_elj, omij_data=o_iii_
↪omij,
>>>                                aij_data=o_iii_aij
>>> print('Emissivity(O III 5007):', emiss5007)
Emissivity(O III 5007): 3.6041012e-21
```

**Categories** Abundance Analysis, Collisionally Excited Lines, Emissivity

**Dirs**

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** 15/09/2013, A. Danehkar, Translated from FORTRAN to IDL code.

20/10/2016, A. Danehkar, Replaced str2int with strnumber.

**20/10/2016, A. Danehkar, Replaced CFY, SPLMAT, and CFD with IDL function INTERPOL( /SPLINE).**

**20/10/2016, A. Danehkar, Replaced LUSLV with IDL LAPACK function LA\_LINEAR\_EQUATION.**

**15/11/2016, A. Danehkar, Replaced LA\_LINEAR\_EQUATION (not work in GDL) with IDL function LUDC & LUSOL.**

**19/11/2016, A. Danehkar, Replaced INTERPOL (not accurate) with SPL\_INIT & SPL\_INTERP.**

**20/11/2016, A. Danehkar, Made a new function calc\_populations() for solving atomic level populations and separated it from calc\_abundance(), calc\_density() and calc\_temperature().**

**21/11/2016, A. Danehkar, Made a new function `calc_emissivity()`** for calculating line emissivities and separated it from `calc_abundance()`.

**10/03/2017, A. Danehkar, Integration with AtomNeb, now uses atomic data** input `elj_data`, `omij_data`, `aij_data`.

**12/06/2017, A. Danehkar, Cleaning the function, and remove unused variables** from `calc_emissivity()`.

27/06/2019, A. Danehkar, Use the simplified `calc_populations()` routine.

03/10/2020, A. Danehkar, Transferred from IDL to Python.

#### FORTRAN HISTORY:

03/05/1981, I.D.Howarth, Version 1.

05/05/1981, I.D.Howarth, Minibug fixed!

07/05/1981, I.D.Howarth, Now takes collision rates or strengths.

03/08/1981, S.Adams, Interpolates collision strengths.

07/08/1981, S.Adams, Input method changed.

**19/11/1984, R.E.S.Clegg, SA files entombed in scratch disk. Logical** filenames given to SA's data files.

08/1995, D.P.Ruffle, Changed input file format. Increased matrices.

**02/1996, X.W.Liu, Tidy up. SUBROUTINES SPLMAT, HGEN, CFY and CFD** modified such that matrix sizes (i.e. maximum of Te and maximum no of levels) can now be changed by modifying the parameters NDIM1, NDIM2 and N in the Main program. EASY! Now takes collision rates as well. All variables are declared explicitly. Generate two extra files (`ionpop.lis` and `ionra`) of plain stream format for plotting.

**06/1996, C.J.Pritchett, Changed input data format for cases IBIG=1,2.** Fixed readin bug for IBIG=2 case. Now reads reformatted upsilons (easier to see and the 0 0 0 data end is excluded for these cases). The A values have a different format for IBIG=.

2006, B.Ercolano, Converted to F90.

`pyequib.calc_populations` (*temperature=None, density=None, elj\_data=None, omij\_data=None, aij\_data=None, eff\_omij=None, level\_num=None, irats=None*)

This function solves atomic level populations in statistical equilibrium for given electron temperature and density.

**Returns** `type=array/object`. This function returns the atomic level populations.

#### Keywords

**temperature** [in, required, `type=float`] electron temperature

**density** [in, required, `type=float`] electron density

**elj\_data** [in, required, `type=array/object`] energy levels (Ej) data

**omij\_data** [in, required, `type=array/object`] collision strengths ( $\omega_{ij}$ ) data

**aij\_data** [in, required, `type=array/object`] transition probabilities (Aij) data

**eff\_Omij** [in, `type=array/object`] effective collision strengths ( $\Omega_{ij,T}$ ) at given temperature

**level\_num** [in, `type=int`] Number of levels

**irats** [in, `type=int`] Else Coll. rates = tabulated values \*  $10^{**irats}$

**Examples** For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_dir = os.path.join('atomic-data', 'chianti70')
>>> atom_elj_file = os.path.join(base_dir, data_dir, 'AtomElj.fits')
>>> atom_omij_file = os.path.join(base_dir, data_dir, 'AtomOmiJ.fits')
>>> atom_aij_file = os.path.join(base_dir, data_dir, 'AtomAij.fits')
>>> atom='s'
>>> ion='ii'
>>> s_ii_elj=atomneb.read_elj(atom_elj_file, atom, ion, level_num=5)
↳ # read Energy Levels (Ej)
>>> s_ii_omij=atomneb.read_omij(atom_omij_file, atom, ion) # read_
↳ Collision Strengths (Omegaij)
>>> s_ii_aij=atomneb.read_aij(atom_aij_file, atom, ion) # read_
↳ Transition Probabilities (Aij) >>> density = np.float64(1000)
>>> temperature=np.float64(10000.0) #
>>> nlj=pyequib.calc_populations(temperature=temperature,
↳ density=density,
>>>                               elj_data=s_ii_elj, omij_data=s_ii_omij,
>>>                               aij_data=s_ii_aij)
>>> print('Atomic Level Populations:', nlj)
Atomic Level Populations: 0.96992832 0.0070036315 0.
↳ 0.23062261 2.6593671e-06 3.1277019e-06
```

**Categories** Plasma Diagnostics, Abundance Analysis, Collisionally Excited Lines

**Dirs**

./ Subroutines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** 15/09/2013, A. Danehkar, Translated from FORTRAN to IDL code.

20/10/2016, A. Danehkar, Replaced str2int with strnumber.

**20/10/2016, A. Danehkar, Replaced CFY, SPLMAT, and CFD with IDL function INTERPOL (/SPLINE).**

**20/10/2016, A. Danehkar, Replaced LUSLV with IDL LAPACK function LA\_LINEAR\_EQUATION.**

**15/11/2016, A. Danehkar, Replaced LA\_LINEAR\_EQUATION (not work in GDL) with IDL function LUDC & LUSOL.**

**19/11/2016, A. Danehkar, Replaced INTERPOL (not accurate) with SPL\_INIT & SPL\_INTERP.**

**20/11/2016, A. Danehkar, Made a new function calc\_populations() for solving atomic level populations and separated it from calc\_abundance(), calc\_density() and calc\_temperature().**

**10/03/2017, A. Danehkar, Integration with AtomNeb, now uses atomic data input elj\_data, omij\_data, aij\_data.**

**12/06/2017, A. Danehkar, Cleaning the function, and remove unused varibales from calc\_populations().**

27/02/2019, A. Danekar, Simplify the `calc_populations()` routine for external usage.

04/03/2019, A. Danekar, Use the `get_omij_temp()` routine.

03/10/2020, A. Danekar, Transferred from IDL to Python.

#### FORTRAN HISTORY:

03/05/1981, I.D.Howarth, Version 1.

05/05/1981, I.D.Howarth, Minibug fixed!

07/05/1981, I.D.Howarth, Now takes collision rates or strengths.

03/08/1981, S.Adams, Interpolates collision strengths.

07/08/1981, S.Adams, Input method changed.

19/11/1984, R.E.S.Clegg, SA files entombed in scratch disk. Logical filenames given to SA's data files.

08/1995, D.P.Ruffle, Changed input file format. Increased matrices.

02/1996, X.W.Liu, Tidy up. SUBROUTINES SPLMAT, HGEN, CFY and CFD modified such that matrix sizes (i.e. maximum of Te and maximum no of levels) can now be changed by modifying the parameters NDIM1, NDIM2 and N in the Main program. EASY! Now takes collision rates as well. All variables are declared explicitly. Generate two extra files (ionpop.lis and ionra) of plain stream format for plotting.

06/1996, C.J.Pritchett, Changed input data format for cases IBIG=1,2. Fixed readin bug for IBIG=2 case. Now reads reformatted upsilons (easier to see and the 0 0 0 data end is excluded for these cases). The A values have a different format for IBIG=.

2006, B.Ercolano, Converted to F90.

```
pyequib.calc_temperature (line_flux_ratio=None,      density=None,      upper_levels=None,
                          lower_levels=None,      elj_data=None,      omij_data=None,
                          aij_data=None,      low_temperature=None,      high_temperature=None,
                          num_temperature=None, min_density=None)
```

This function determines electron temperature from given flux intensity ratio for specified ion with upper level(s) lower level(s) by solving atomic level populations and line emissivities in statistical equilibrium for given electron density.

**Returns** type=double. This function returns the electron temperature.

#### Keywords

**line\_flux\_ratio** [in, required, type=float] flux intensity ratio

**density** [in, required, type=float] electron density

**upper\_levels** [in, required, type=string,] upper atomic level(s) e.g '1,2', '1,2,1,3/'

**lower\_levels** [in, required, type=string] lower atomic level(s) e.g '1,2/', '1,2,1,3/'

**elj\_data** [in, required, type=array/object] energy levels (Ej) data

**omij\_data** [in, required, type=array/object] collision strengths (omega\_ij) data

**aij\_data** [in, required, type=array/object] transition probabilities (Aij) data

**low\_temperature** [in, optional, type=float] lower temperature range

**high\_temperature** [in, optional, type=float] upper temperature range

**num\_temperature** [in, optional, type=integer] number of the iteration step

**min\_density** [in, optional, type=float] lower density range

**Examples** For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_dir = os.path.join('atomic-data', 'chianti70')
>>> atom_elj_file = os.path.join(base_dir, data_dir, 'AtomElj.fits')
>>> atom_omij_file = os.path.join(base_dir, data_dir, 'AtomOmi.j.fits')
>>> atom_aij_file = os.path.join(base_dir, data_dir, 'AtomAij.fits')
>>> atom='s'
>>> ion='ii'
>>> s_ii_elj=atomneb.read_elj(atom_elj_file, atom, ion, level_num=5)
↪ # read Energy Levels (Ej)
>>> s_ii_omij=atomneb.read_omij(atom_omij_file, atom, ion) # read_
↪ Collision Strengths (Omegaij)
>>> s_ii_aij=atomneb.read_aij(atom_aij_file, atom, ion) # read_
↪ Transition Probabilities (Aij)
>>> upper_levels='1,2,1,3/'
>>> lower_levels='1,5/'
>>> density = np.float64(2550)
>>> line_flux_ratio=np.float64(10.753)
>>> temperature=pyequib.calc_temperature(line_flux_ratio=line_flux_
↪ ratio, density=density,
>>>                                     upper_levels=upper_levels, lower_
↪ levels=lower_levels,
>>>                                     elj_data=s_ii_elj, omij_data=s_ii_
↪ omij,
>>>                                     aij_data=s_ii_aij)
>>> print("Electron Temperature:", temperature)
Electron Temperature:      7920.2865
```

**Categories** Plasma Diagnostics, Collisionally Excited Lines

**Dirs**

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** 15/09/2013, A. Danehkar, Translated from FORTRAN to IDL code.

20/10/2016, A. Danehkar, Replaced str2int with strnumber.

**20/10/2016, A. Danehkar, Replaced CFY, SPLMAT, and CFD with IDL function INTERPOL (/SPLINE).**

**20/10/2016, A. Danehkar, Replaced LUSLV with IDL LAPACK function LA\_LINEAR\_EQUATION.**

**15/11/2016, A. Danehkar, Replaced LA\_LINEAR\_EQUATION (not work in GDL) with IDL function LUDC & LUSOL.**

**19/11/2016, A. Danehkar, Replaced INTERPOL (not accurate) with SPL\_INIT & SPL\_INTERP.**

**20/11/2016, A. Danehkar, Made a new function `calc_populations()`** for solving atomic level populations and separated it from `calc_abundance()`, `calc_density()` and `calc_temperature()`.

**10/03/2017, A. Danehkar, Integration with AtomNeb, now uses atomic data** input `elj_data`, `omij_data`, `aij_data`.

**12/06/2017, A. Danehkar, Cleaning the function, and remove unused variables** from `calc_temperature()`.

**27/02/2019, A. Danehkar, Fix a bug in the atomic level assumption, and** use the simplified `calc_populations()` routine.

04/03/2019, A. Danehkar, Use the `get_omij_temp()` routine.

24/05/2019, A. Danehkar, Add the optional temperature range.

03/10/2020, A. Danehkar, Transferred from IDL to Python.

#### FORTTRAN HISTORY:

03/05/1981, I.D.Howarth, Version 1.

05/05/1981, I.D.Howarth, Minibug fixed!

07/05/1981, I.D.Howarth, Now takes collision rates or strengths.

03/08/1981, S.Adams, Interpolates collision strengths.

07/08/1981, S.Adams, Input method changed.

**19/11/1984, R.E.S.Clegg, SA files entombed in scratch disk. Logical** filenames given to SA's data files.

08/1995, D.P.Ruffle, Changed input file format. Increased matrices.

**02/1996, X.W.Liu, Tidy up. SUBROUTINES SPLMAT, HGEN, CFY and CFD** modified such that matrix sizes (i.e. maximum of Te and maximum no of levels) can now be changed by modifying the parameters NDIM1, NDIM2 and N in the Main program. EASY! Now takes collision rates as well. All variables are declared explicitly. Generate two extra files (`ionpop.lis` and `ionra`) of plain stream format for plotting.

**06/1996, C.J.Pritchett, Changed input data format for cases IBIG=1,2.** Fixed readin bug for IBIG=2 case. Now reads reformatted upsilons (easier to see and the 0 0 0 data end is excluded for these cases). The A values have a different format for IBIG=.

2006, B.Ercolano, Converted to F90.

`pyequib.deredden_flux(wavelength, flux, m_ext, ext_law=None, rv=None, fmlaw=None)`

This function dereddens absolute flux intensity based on the reddening law.

**Returns** `type=double`. This function returns the deredden flux intensity.

#### Params

**wavelength** [in, required, `type=float/array`] Wavelength in Angstrom

**flux** [in, required, `type=float,`] absolute flux intensity

**m\_ext** [in, required, `type=float,`] logarithmic extinction

#### Keywords

**ext\_law** [in, optional, `type=string`, `default='GAL'`] the extinction law:

'GAL' for Howarth Galactic.

‘GAL2’ for Savage and Mathis.

‘CCM’ for CCM galactic.

‘JBK’ for Whitford, Seaton, Kaler.

‘FM’ for Fitzpatrick.

‘SMC’ for Prevot SMC.

‘LMC’ for Howarth LMC.

**rv** [in, optional, type=float, default=3.1] the optical total-to-selective extinction ratio,  $RV = A(V)/E(B-V)$ .

**fmlaw** [in, optional, type=string, default=‘GAL’] the fmlaw keyword is used only in the redlaw\_fm function:

‘GAL’ for the default fit parameters for the R-dependent Galactic extinction curve from Fitzpatrick & Massa (Fitzpatrick, 1999, PASP, 111, 63).

‘LMC2’ for the fit parameters are those determined for reddening the LMC2 field (inc. 30 Dor) from Misselt et al. (1999, ApJ, 515, 128).

‘AVGLMC’ for the fit parameters are those determined for reddening in the general Large Magellanic Cloud (LMC) field by Misselt et al. (1999, ApJ, 515, 128).

## Examples

For example:

```
>>> import pyequib
>>> wavelength=6563.0
>>> ext_law='GAL'
>>> r_v=3.1
>>> m_ext=1.0
>>> flux=1.0
>>> flux_deredden=pyequib.deredden_flux(wavelength, flux, m_ext,
↳ ext_law=ext_law, rv=r_v) # deredden absolute flux intensity
>>> print('dereddened flux(6563):', flux_deredden)
dereddened flux(6563): 4.7847785
```

**Categories** Interstellar Extinction

**Dirs**

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** 31/08/2012, A. Danehkar, IDL code.

03/10/2020, A. Danehkar, Transferred from IDL to Python.

`pyequib.deredden_relflux(wavelength, relflux, m_ext, ext_law=None, rv=None, fmlaw=None)`

This function dereddens flux intensity relative to Hb=100, based on the reddening law.

**Returns** type=double. This function returns the deredden flux intensity relative to Hb=100.

**Params**

**wavelength** [in, required, type=float/array] Wavelength in Angstrom  
**relflux** [in, required, type=float,] flux intensity relative to Hb=100  
**m\_ext** [in, required, type=float,] logarithmic extinction

**Keywords**

**ext\_law** [in, optional, type=string, default='GAL'] the extinction law:

‘GAL’ for Howarth Galactic.

‘GAL2’ for Savage and Mathis.

‘CCM’ for CCM galactic.

‘JBK’ for Whitford, Seaton, Kaler.

‘FM’ for Fitzpatrick.

‘SMC’ for Prevot SMC.

‘LMC’ for Howarth LMC.

**rv** [in, optional, type=float, default=3.1] the optical total-to-selective extinction ratio,  
 $RV = A(V)/E(B-V)$ .

**fmlaw** [in, optional, type=string, default='GAL'] the fmlaw keyword is used only in  
the redlaw\_fm function:

**‘GAL’ for the default fit parameters for the R-dependent** Galactic extinction  
curve from Fitzpatrick & Massa (Fitzpatrick, 1999, PASP, 111, 63).

**‘LMC2’ for the fit parameters are those determined for** reddening the LMC2  
field (inc. 30 Dor) from Misselt et al. (1999, ApJ, 515, 128).

**‘AVGLMC’ for the fit parameters are those determined for** reddening in the  
general Large Magellanic Cloud (LMC) field by Misselt et al. (1999, ApJ, 515,  
128).

**Examples**

For example:

```
>>> import pyequib
>>> wavelength=6563.0
>>> ext_law='GAL'
>>> r_v=3.1
>>> m_ext=1.0
>>> flux=1.0
>>> flux_deredden=pyequib.deredden_relflux(wavelength, flux, m_
↳ext, ext_law=ext_law, rv=r_v) # deredden absolute flux_
↳intensity
>>> print('dereddened relative flux(6563):', flux_deredden)
dereddened relative flux(6563):      0.47847785
```

**Categories** Interstellar Extinction

**Dirs**

./ Main routines

**Author** Ashkbiz Danehkar



**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** 31/08/2012, A. Danehkar, IDL code.

03/10/2020, A. Danehkar, Transferred from IDL to Python.

```
pyequib.get_omij_temp(temperature=None, omij_data=None, elj_data=None, level_num=None,
                    irats=None)
```

This function derives the effective collision strengths (Omij\_T) from the collision strengths (omega\_ij) data for the given temperature.

**Returns** type=array/object. This function returns the effective collision strengths (Omij\_T).

**Keywords**

**temperature** [in, required, type=float] electron temperature

**omij\_data** [in, required, type=array/object] collision strengths (omega\_ij) data

**level\_num** [in, type=int] Number of levels

**irats** [in, type=int] Else Coll. rates = tabulated values \* 10 \*\* irats

**Examples** For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_dir = os.path.join('atomic-data', 'chianti70')
>>> atom_elj_file = os.path.join(base_dir, data_dir, 'AtomElj.fits')
>>> atom_omij_file = os.path.join(base_dir, data_dir, 'AtomOmij.fits')
>>> atom_aij_file = os.path.join(base_dir, data_dir, 'AtomAij.fits')
>>> atom='s'
>>> ion='ii'
>>> s_ii_elj=atomneb.read_elj(atom_elj_file, atom, ion, level_num=5)
↪# read Energy Levels (Ej)
>>> s_ii_omij=atomneb.read_omij(atom_omij_file, atom, ion) # read_
↪Collision Strengths (Omegaij)
>>> s_ii_aij=atomneb.read_aij(atom_aij_file, atom, ion) # read_
↪Transition Probabilities (Aij) >>> temperature=np.
↪float64(10000.0)#
>>> omij_t=pyequib.get_omij_temp(temperature=temperature, omij_data=s_
↪ii_omij)
>>> print('Effective Collision Strengths: ')
>>> print(omij_t)
Effective Collision Strengths:
0.0000000 0.0000000 0.0000000 0.0000000 0.
↪0000000
2.7800000 0.0000000 0.0000000 0.0000000 0.
↪0000000
4.1600000 7.4600000 0.0000000 0.0000000 0.
↪0000000
1.1700000 1.8000000 2.2000000 0.0000000 0.
↪0000000
2.3500000 3.0000000 4.9900000 2.7100000 0.
↪0000000
```

**Categories** Plasma Diagnostics, Abundance Analysis, Collisionally Excited Lines

**Dirs**

./ Subroutines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** 04/03/2019, A. Danehkar, create the get\_omij\_temp() routine.

03/10/2020, A. Danehkar, Transferred from IDL to Python.

```
pyequib.print_ionic(temperature=None, density=None, elj_data=None, omij_data=None,
                    aij_data=None, h_i_aeff_data=None, printemissivity=None, printpopulations=None, printcritdensity=None)
```

This function prints the atom's transitions information, atomic level populations, critical densities, and emissivities for given temperature and density.

**Keywords**

**temperature** [in, required, type=float] electron temperature

**density** [in, required, type=float] electron density

**elj\_data** [in, required, type=array/object] energy levels (Ej) data

**omij\_data** [in, required, type=array/object] collision strengths (omega\_ij) data

**aij\_data** [in, required, type=array/object] transition probabilities (Aij) data

**h\_i\_aeff\_data** [in, type=array/object] H I recombination coefficients

**printEmissivity** [in, type=boolean] Set for printing Emissivities

**printPopulations** [in, type=boolean] Set for printing Populations

**printCritDensity** [in, type=boolean] Set for printing Critical Densities

**Examples** For example:

```
>>> import pyequib
>>> import atomneb
>>> import os
>>> base_dir = '../externals/atomneb/'
>>> data_dir = os.path.join('atomic-data', 'chianti70')
>>> atom_elj_file = os.path.join(base_dir, data_dir, 'AtomElj.fits')
>>> atom_omij_file = os.path.join(base_dir, data_dir, 'AtomOmij.fits')
>>> atom_aij_file = os.path.join(base_dir, data_dir, 'AtomAij.fits')
>>> data_rc_dir = os.path.join('atomic-data-rc')
>>> atom_rc_sh95_file = os.path.join(base_dir, data_rc_dir, 'rc_SH95.
↳fits')
>>> atom='o'
>>> ion='iii'
>>> o_iii_elj=atomneb.read_elj(atom_elj_file, atom, ion, level_num=5)
↳# read Energy Levels (Ej)
>>> o_iii_omij=atomneb.read_omij(atom_omij_file, atom, ion) # read_
↳Collision Strengths (Omegaij)
>>> o_iii_aij=atomneb.read_aij(atom_aij_file, atom, ion) # read_
↳Transition Probabilities (Aij)
>>> atom='h'
>>> ion='ii' # H I
```

(continues on next page)

(continued from previous page)

```

>>> hi_rc_data=atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)
>>> temperature=np.float64(10000.0)#
>>> density = np.float64(1000.)
>>> pyequib.print_ionic, temperature=temperature, density=density,
>>>             elj_data=o_iii_elj, omij_data=o_iii_omij,
>>>             aij_data=o_iii_aij, h_i_aeff_data=hi_rc_data['aeff
↪'] [0]
Temperature = 10000.0 K
Density = 1000.0 cm-3

Level      Populations      Critical Densities
Level 1:    3.063E-01      0.000E+00
Level 2:    4.896E-01      4.908E+02
Level 3:    2.041E-01      3.419E+03
Level 4:    4.427E-05      6.853E+05
Level 5:    2.985E-09      2.547E+07

2.597E-05
88.34um
(2-->1)
2.859E-22

0.000E+00  9.632E-05
32.66um    51.81um
(3-->1)    (3-->2)
0.000E+00  7.536E-22

2.322E-06  6.791E-03  2.046E-02
4932.60A   4960.29A   5008.24A
(4-->1)    (4-->2)    (4-->3)
4.140E-25  1.204E-21  3.593E-21

0.000E+00  2.255E-01  6.998E-04  1.685E+00
2315.58A   2321.67A   2332.12A   4364.45A
(5-->1)    (5-->2)    (5-->3)    (5-->4)
0.000E+00  5.759E-24  1.779E-26  2.289E-23

H-beta emissivity: 1.237E-25 N(H+) Ne [erg/s]

```

**Categories** Plasma Diagnostics, Abundance Analysis, Collisionally Excited Lines**Dirs**

./ Main routines

**Author** Ashkbiz Danehkar**Copyright** This library is released under a GNU General Public License.**Version** 0.3.0**History** 04/03/2019, A. Danehkar, create the print\_ionic() routine.

03/10/2020, A. Danehkar, Transferred from IDL to Python.

pyequib.**redlaw** (*wavelength*, *ext\_law=None*, *rv=None*, *fmlaw=None*)

This function determines the reddening law function of the line at the given wavelength for the used extinction law.

**Returns** type=double/array. This function returns the reddening law function value for the given wavelength.

**Params**

**wavelength** [in, required, type=float/array] Wavelength in Angstrom

**Keywords**

**ext\_law** [in, optional, type=string, default='GAL'] the extinction law:

‘GAL’ for Howarth Galactic.

‘GAL2’ for Savage and Mathis.

‘CCM’ for CCM galactic.

‘JBK’ for Whitford, Seaton, Kaler.

‘FM’ for Fitzpatrick.

‘SMC’ for Prevot SMC.

‘LMC’ for Howarth LMC.

**rv** [in, optional, type=float, default=3.1] the optical total-to-selective extinction ratio,  $RV = A(V)/E(B-V)$ .

**fmlaw** [in, optional, type=string, default='GAL'] the fmlaw keyword is used only in the redlaw\_fm function:

**‘GAL’ for the default fit parameters for the R-dependent** Galactic extinction curve from Fitzpatrick & Massa (Fitzpatrick, 1999, PASP, 111, 63).

**‘LMC2’ for the fit parameters are those determined for** reddening the LMC2 field (inc. 30 Dor) from Misselt et al. (1999, ApJ, 515, 128).

**‘AVGLMC’ for the fit parameters are those determined for** reddening in the general Large Magellanic Cloud (LMC) field by Misselt et al. (1999, ApJ, 515, 128).

**Examples**

For example:

```
>>> import pyequib
>>> wavelength=6563.0
>>> r_v=3.1
>>> fl=pyequib.redlaw(wavelength, rv=r_v)
>>> print('fl(6563)', fl)
fl(6563)      -0.32013816
```

**Categories** Interstellar Extinction

**Dirs**

./ Main routines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Originally from IRAF STSDAS SYNPHOT redlaw.x, ebmvxfunc.x

31/08/2012, A. Danekkar, Converted to IDL code.

03/10/2020, A. Danekkar, Transferred from IDL to Python.

`pyequib.redlaw_ccm(wavelength, rv=None)`

This function determines the reddening law function of Cardelli, Clayton & Mathis.

**Returns** type=double/array. This function returns the reddening law function value for the given wavelength.

#### Params

**wavelength** [in, required, type=float/array] Wavelength in Angstrom

#### Keywords

**RV** [in, optional, type=float, default=3.1] the optical total-to-selective extinction ratio,  $RV = A(V)/E(B-V)$ .

#### Examples

For example:

```
>>> import pyequib
>>> wavelength=6563.0
>>> r_v=3.1
>>> fl=pyequib.redlaw_ccm(wavelength, rv=r_v)
>>> print('fl(6563)', fl)
fl(6563)          -0.29756615
```

**Categories** Interstellar Extinction

#### Dirs

./ Subroutines

**Author** Ashkbiz Danekkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on Formulae by Cardelli, Clayton & Mathis 1989, ApJ 345, 245-256. 1989ApJ...345..245C

Originally from IRAF STSDAS SYNPHOT redlaw.x

**18/05/1993, R. A. Shaw, Initial IRAF implementation, based upon CCM module** in onedspec.deredden.

31/08/2012, A. Danekkar, Converted to IDL code.

03/10/2020, A. Danekkar, Transferred from IDL to Python.

`pyequib.redlaw_fm(wavelength, rv=None, fmlaw=None)`

This function determines the reddening law function by Fitzpatrick & Massa for the line at the given wavelength.

**Returns** type=double/array. This function returns the reddening law function value for the given wavelength.

**Params**

**wavelength** [in, required, type=float/array] Wavelength in Angstrom

**Keywords**

**RV** [in, optional, type=float, default=3.1] the optical total-to-selective extinction ratio,  $RV = A(V)/E(B-V)$ .

**fmlaw** [in, optional, type=string, default='GAL'] the fmlaw keyword is used only in the redlaw\_fm function:

**'GAL'** for the default fit parameters for the R-dependent Galactic extinction curve from Fitzpatrick & Massa (Fitzpatrick, 1999, PASP, 111, 63).

**'LMC2'** for the fit parameters are those determined for reddening the LMC2 field (inc. 30 Dor) from Misselt et al. (1999, ApJ, 515, 128).

**'AVGLMC'** for the fit parameters are those determined for reddening in the general Large Magellanic Cloud (LMC) field by Misselt et al. (1999, ApJ, 515, 128).

**Examples**

For example:

```
>>> import pyequib
>>> wavelength=6563.0
>>> r_v=3.1
>>> fl=pyequib.redlaw_fm(wavelength, rv=r_v)
>>> print('fl(6563)', fl)
fl(6563)          -0.35054942
```

**Categories** Interstellar Extinction

**Dirs**

/ Subroutines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on Formulae by Fitzpatrick 1999, PASP, 11, 63 1999PASP..111...63F, Fitzpatrick & Massa 1990, ApJS, 72, 163, 1990ApJS...72..163F

Adopted from NASA IDL Library & PyAstronomy.

30/12/2016, A. Danehkar, Revised in IDL code.

03/10/2020, A. Danehkar, Transferred from IDL to Python.

`pyequib.redlaw_gal(wavelength, rv=None)`

This function determines the reddening law function of the line at the given wavelength for Galactic Seaton1979+Howarth1983+CCM1983.

**Returns** type=double/array. This function returns the reddening law function value(s) for the given wavelength(s).

**Params**

**wavelength** [in, required, type=float] Wavelength in Angstrom

### Keywords

**rv** [in, optional, type=float, default=3.1] the optical total-to-selective extinction ratio,  $RV = A(V)/E(B-V)$ .

**Examples** For example:

```
>>> import pyequib
>>> wavelength=6563.0
>>> r_v=3.1
>>> fl=pyequib.redlaw_gal(wavelength, rv=r_v)
>>> print('fl(6563)', fl)
fl(6563)      -0.32013816
```

**Categories** Interstellar Extinction

### Dirs

./ Subroutines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on the UV Formulae from Seaton 1979, MNRAS, 187, 73 1979MNRAS.187P..73S, the opt/NIR from Howarth 1983, MNRAS, 203, 301 the FIR from Cardelli, Clayton and Mathis 1989, ApJ, 345, 245 1989ApJ...345..245C

Originally from IRAF STSDAS SYNPHOT ebmvxfunc.x, pyneb.extinction

31/08/2012, A. Danehkar, Converted to IDL code.

03/10/2020, A. Danehkar, Transferred from IDL to Python.

`pyequib.redlaw_gal2(wavelength)`

This function determines the reddening law function of the line at the given wavelength for Galactic Savage & Mathis 1979.

**Returns** type=double/array. This function returns the reddening law function value(s) for the given wavelength(s).

### Params

**wavelength** [in, required, type=float] Wavelength in Angstrom

**Examples** For example:

```
>>> import pyequib
>>> wavelength=6563.0
>>> fl=pyequib.redlaw_gal2(wavelength)
>>> print('fl(6563)', fl)
fl(6563)      -0.30925984
```

**Categories** Interstellar Extinction

### Dirs

./ Subroutines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on Savage & Mathis 1979, ARA&A, vol. 17, 73-111

Originally from IRAF STSDAS SYNPHOT ebmvxfunc.x

20/09/1994, R. A. Shaw, Initial IRAF implementation.

04/03/1995, R. A. Shaw, Return A(lambda)/A(V) instead.

31/08/2012, A. Danehkar, Converted to IDL code.

03/10/2020, A. Danehkar, Transferred from IDL to Python.

`pyequib.redlaw_jbk(wavelength)`

This function determines the reddening law function for Galactic Whitford1958 + Seaton1977 + Kaler1976.

**Returns** type=double/array. This function returns the reddening law function value(s) for the given wavelength(s).

**Params**

**wavelength** [in, required, type=float] Wavelength in Angstrom

**Examples** For example:

```
>>> import pyequib
>>> wavelength=6563.0
>>> fl=pyequib.redlaw_jbk(wavelength)
>>> print('fl(6563)', fl)
fl(6563)      -0.33113684
```

**Categories** Interstellar Extinction

**Dirs**

./ Subroutines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on Whitford (1958), extended to the UV by Seaton (1977), adapted by Kaler (1976).

Originally from IRAF STSDAS SYNPHOT redlaw.x

13/05/1993, R. A. Shaw, Initial IRAF implementation.

31/08/2012, A. Danehkar, Converted to IDL code.

03/10/2020, A. Danehkar, Transferred from IDL to Python.

`pyequib.redlaw_lmc(wavelength)`

This function determines the reddening law function of the line at the given wavelength for the Large Magellanic Cloud.

**Returns** type=double/array. This function returns the reddening law function value(s) for the given wavelength(s).



**Params**

**wavelength** [in, required, type=float] Wavelength in Angstrom

**Examples** For example:

```
>>> import pyequib
>>> wavelength=6563.0
>>> fl=pyequib.redlaw_lmc(wavelength)
>>> print('fl(6563)', fl)
fl(6563)      -0.30871187
```

**Categories** Interstellar Extinction

**Dirs**

./ Subroutines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on Formulae by Howarth 1983, MNRAS, 203, 301 1983MNRAS.203..301H

Originally from IRAF STSDAS SYNPHOT ebmvfunc.x, redlaw.x

18/10/1994, R. A. Shaw, Initial IRAF implementation.

14/03/1995, R. A. Shaw, Return A(lambda)/A(V) instead.

31/08/2012, A. Danehkar, Converted to IDL code.

03/10/2020, A. Danehkar, Transferred from IDL to Python.

`pyequib.redlaw_smc(wavelength)`

This function determines the reddening law function of the line at the given wavelength for Small Magellanic Cloud.

**Returns** type=double/array. This function returns the reddening law function value(s) for the given wavelength(s).

**Params**

**wavelength** [in, required, type=float] Wavelength in Angstrom

**Examples** For example:

```
>>> import pyequib
>>> wavelength=6563.0
>>> fl=pyequib.redlaw_smc(wavelength)
>>> print('fl(6563)', fl)
fl(6563)      -0.22659261
```

**Categories** Interstellar Extinction

**Dirs**

./ Subroutines

**Author** Ashkbiz Danehkar

**Copyright** This library is released under a GNU General Public License.

**Version** 0.3.0

**History** Based on Prevot et al. (1984), A&A, 132, 389-392 1984A%26A...132..389P

Originally from IRAF STSDAS SYNPHOT redlaw.x, ebmvxfunc.x

20/09/1994, R. A. Shaw, Initial IRAF implementation.

04/03/1995, R. A. Shaw, Return  $A(\lambda)/A(V)$  instead.

31/08/2012, A. Danekar, Converted to IDL code.

03/10/2020, A. Danekar, Transferred from IDL to Python.

## INDICES AND TABLES

- genindex
- modindex
- search



## PYTHON MODULE INDEX

### p

pyequib, [25](#)