pyEQUIB

Release 0.3.0

Ashkbiz Danehkar

Oct 15, 2020

USER DOCUMENTATION

CHAPTER

ONE

INTRODUCTION

pyEQUIB package is a collection of Python programs developed to perform plasma diagnostics and abundance analysis using emission line fluxes measured in ionzed 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 follwing 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 functiosn 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.

CHAPTER

TWO

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

CHAPTER

THREE

USAGE

The Documentation of the Python functions provides in detail in the *API Documentation* (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)
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_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)

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)
```

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_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)

atom = 'he'
ion = 'ii' # He I
he_i_rc_data = atomneb.read_aeff_he_i_pfsd12(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(N^2+)/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(N^3+)/N(H+):', abund_n_iii)
```

which gives:

```
N(N^3+)/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_PFSD12.fits')

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)

```
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_relflux(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
```

CHAPTER

FOUR

REFERENCES

- Danehkar, A. (2018). proEQUIB: IDL Library for Plasma Diagnostics and Abundance Analysis. *J. Open Source Softw.*, **3**, 899. doi:10.21105/joss.00899 ads:2018JOSS....3..899D.

CHAPTER

FIVE

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.

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
>> 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)
>> 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
```

Returns This function returns the ionic abundanc.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **wavelength** (*float*) – Line Wavelength in Angstrom
- **line_flux** (*float*) – line flux intensity
- **c_ii_rc_data** (*array/object*) – C II recombination coefficients
- **h_i_aeff_data** (*array/object*) – H I recombination coefficients

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.

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
>> 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
```

Returns This function returns the ionic abundanc.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **wavelength** (*float*) – Line Wavelength in Angstrom
- **line_flux** (*float*) – line flux intensity

- **c_iii_rc_data** (*array/object*) – C III recombination coefficients
- **h_i_aeff_data** (*array/object*) – H I recombination coefficients

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.

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
>> 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
>> 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
```

Returns This function returns the ionic abundanc.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **linenum** (*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** (*float*) – line flux intensity

- **he_i_aeff_data** (*array/object*) – He I recombination coefficients
- **h_i_aeff_data** (*array/object*) – H I recombination coefficients

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 A by using the helium emissivities from Storey & Hummer, 1995MNRAS.272...41S.

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
>> 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
>> 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
```

Returns This function returns the ionic abundanc.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **line_flux** (*float*) – line flux intensity
- **he_ii_aeff_data** (*array/object*) – He II recombination coefficients
- **h_i_aeff_data** (*array/object*) – H I recombination coefficients

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.

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
>> 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
```

Returns This function returns the ionic abundanc.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **wavelength** (*float*) – Line Wavelength in Angstrom
- **line_flux** (*float*) – line flux intensity
- **n_ii_rc_br** (*array/object*) – N II branching ratios (Br)
- **n_ii_rc_data** (*array/object*) – N II recombination coefficients
- **h_i_aeff_data** (*array/object*) – H I recombination coefficients

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.

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
>> 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)
>> 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
```

Returns This function returns the ionic abundanc.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **wavelength** (*float*) – Line Wavelength in Angstrom
- **line_flux** (*float*) – line flux intensity
- **n_iii_rc_data** (*array/object*) – N III recombination coefficients
- **h_i_aeff_data** (*array/object*) – H I recombination coefficients

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).

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
>> 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
```

Returns This function returns the ionic abundanc.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **wavelength** (*float*) – Line Wavelength in Angstrom
- **line_flux** (*float*) – line flux intensity
- **ne_ii_rc_data** (*array/object*) – Ne II recombination coefficients
- **h_i_aeff_data** (*array/object*) – H I recombination coefficients

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.

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='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
>> 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
```

Returns This function returns the ionic abundanc.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **wavelength** (*float*) – Line Wavelength in Angstrom
- **line_flux** (*float*) – line flux intensity
- **o_ii_rc_br** (*array/object*) – O II branching ratios (Br)
- **o_ii_rc_data** (*array/object*) – O II recombination coefficients
- **h_i_aeff_data** (*array/object*) – H I recombination coefficients

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.

For example:

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

(continues on next page)

(continued from previous page)

```
>> 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
>> hi_rc_data=atomneb.read_aeff_sh95(atom_rc_sh95_file, atom, ion)
>> h_i_aeff_data=hi_rc_data.aeff
>> 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=hi_rc_data.
↪aeff)
>> print('N(O^2+)/N(H+):', abb5007)
   N(O^2+)/N(H+):   0.00041256231
```

Returns This function returns the ionic abundanc.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **line_flux** (*float*) – line flux intensity
- **atomic_levels** (*str*) – level(s) e.g '1,2/', '1,2,1,3/'
- **elj_data** (*array/object*) – energy levels (Ej) data
- **omij_data** (*array/object*) – collision strengths (omega_ij) data
- **aij_data** (*array/object*) – transition probabilities (Aij) data
- **h_i_aeff_data** (*array/object*) – H I recombination coefficients

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.

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)
>> 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
```

Returns This function returns the critical densities.

Return type array/object

Parameters

- **temperature** (*float*) – electron temperature
- **elj_data** (*array/object*) – energy levels (Ej) data
- **omij_data** (*array/object*) – collision strengths (omega_ij) data
- **aij_data** (*array/object*) – transition probabilities (Aij) data
- **level_num** (*int, optional*) – Number of levels
- **irats** (*int, optional*) – Else Coll. rates = tabulated values * 10 ** irats

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.

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')
```

(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)        >> 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
```

Returns This function returns the electron density.

Return type float64

Parameters

- **line_flux_ratio** (*float*) – flux intensity ratio
- **temperature** (*float*) – electron temperature
- **upper_levels** (*str*) – upper atomic level(s) e.g '1,2/', '1,2,1,3/'
- **lower_levels** (*str*) – lower atomic level(s) e.g '1,2/', '1,2,1,3/"
- **elj_data** (*array/object*) – energy levels (Ej) data
- **omij_data** (*array/object*) – collision strengths (omega_ij) data
- **aij_data** (*array/object*) – transition probabilities (Aij) data
- **low_density** (*float, optional*) – lower density range
- **high_density** (*float, optional*) – upper density range
- **num_density** (*int, optional*) – number of the iteration step
- **min_temperature** (*float, optional*) – minimum temperature

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.

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
```

Returns This function returns the line emissivity.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **wavelength** (*float*) – Line Wavelength in Angstrom
- **c_ii_rc_data** (*array/object*) – C II recombination coefficients

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.

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)
```

(continues on next page)

(continued from previous page)

```
>> print('Emissivity:', emiss_c_iii)
   Emissivity:   7.5749632e-25
```

Returns This function returns the line emissivity.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **wavelength** (*float*) – Line Wavelength in Angstrom
- **c_iii_rc_data** (*array/object*) – C III recombination coefficients

pyequib.**calc_emiss_h_beta**(*temperature=None*, *density=None*, *h_i_aeff_data=None*)

This function calculates the emissivity for H_beta 4861A Emis(Hbeta)= 4pi j(HBeta 4861 A)/Np Ne) for the given temperature and density by using the helium emissivities from Storey & Hummer, 1995MNRAS.272...41S.

Returns This function returns the H beta emissivity 4pi j(HBeta 4861)/Np Ne).

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **h_i_aeff_data** (*array/object*) – H I recombination coefficients

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.

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_PFSD12.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
>> temperature=np.float64(10000.0)
>> density=np.float64(5000.0)
>> linenum=10# 4471.50
```

(continues on next page)

(continued from previous page)

```
>> 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
```

Returns This function returns the line emissivity.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **linenum** (*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** (*float*) – line flux intensity
- **he_i_aeff_data** (*array/object*) – He I recombination coefficients

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

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

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
>> 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
```

Returns This function returns the line emissivity.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **he_ii_aeff_data** (*array/object*) – He II recombination coefficients

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.

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
>> 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
```

Returns This function returns the line emissivity.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **wavelength** (*float*) – Line Wavelength in Angstrom
- **n_ii_rc_br** (*array/object*) – N II branching ratios (Br)
- **n_ii_rc_data** (*array/object*) – N II recombination coefficients

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.

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
```

Returns This function returns the line emissivity.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **wavelength** (*float*) – Line Wavelength in Angstrom
- **n_iii_rc_data** (*array/object*) – N III recombination coefficients

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).

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'
```

(continues on next page)

(continued from previous page)

```
>> 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
```

Returns This function returns the line emissivity.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **wavelength** (*float*) – Line Wavelength in Angstrom
- **ne_ii_rc_data** (*array/object*) – Ne II recombination coefficients

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.

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)
>> 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)
```

(continues on next page)

(continued from previous page)

```
>> print('Emissivity:', emiss_o_ii)
   Emissivity:   5.9047319e-27
```

Returns This function returns the line emissivity.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **wavelength** (*float*) – Line Wavelength in Angstrom
- **o_ii_rc_br** (*array/object*) – O II branching ratios (Br)
- **o_ii_rc_data** (*array/object*) – O II recombination coefficients

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.

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
```

Returns This function returns the line emissivity.

Return type float64

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **atomic_levels** (*str*) – level(s) e.g '1,2/', '1,2,1,3/'
- **elj_data** (*array/object*) – energy levels (Ej) data
- **omij_data** (*array/object*) – collision strengths (omega_ij) data
- **aij_data** (*array/object*) – transition probabilities (Aij) data

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.

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.
↪023062261   2.6593671e-06   3.1277019e-06
```

Returns This function returns the atomic level populations.

Return type array/object

Parameters

- **line_flux_ratio** (*float*) – flux intensity ratio
- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **elj_data** (*array/object*) – energy levels (Ej) data
- **omij_data** (*array/object*) – collision strengths (omega_ij) data
- **aij_data** (*array/object*) – transition probabilities (Aij) data
- **eff_Omij** (*array/object*) – effective collision strengths (Omij_T) at given temperature

- **level_num** (*int, optional*) – Number of levels
- **irats** (*int, optional*) – Else Coll. rates = tabulated values * 10 ** irats

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.

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='1,2,1,3/'
>> 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
```

Returns This function returns the electron temperature.

Return type float64

Parameters

- **line_flux_ratio** (*float*) – flux intensity ratio
- **density** (*float*) – electron density
- **upper_levels** (*str*) – upper atomic level(s) e.g '1,2/', '1,2,1,3/'
- **lower_levels** (*str*) – lower atomic level(s) e.g '1,2/', '1,2,1,3/"
- **elj_data** (*array/object*) – energy levels (Ej) data
- **omij_data** (*array/object*) – collision strengths (omega_ij) data

- **aij_data** (*array/object*) – transition probabilities (Aij) data
- **low_temperature** (*float, optional*) – lower temperature range
- **high_temperature** (*float, optional*) – upper temperature range
- **num_temperature** (*int, optional*) – number of the iteration step
- **min_density** (*float, optional*) – lower density range

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.

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
```

Returns This function returns the deredden flux intensity.

Return type float64

Parameters

- **wavelength** (*float/array*) – Wavelength in Angstrom
- **flux** (*float*) – absolute flux intensity
- **m_ext** (*float*) – logarithmic extinction
- **ext_law** (*str, optional*) – the extinction law (default='GAL'): 'GAL' for Howarth Galactic; 'GAL2' for Savage and Mathis; 'CCM' for CCM galactic; 'JBK' for Whitford, Seaton, Kaler; 'FM' for Fitxpatrick; 'SMC' for Prevot SMC; 'LMC' for Howarth LMC.
- **rv** (*float, optional*) – the optical total-to-selective extinction ratio, RV = A(V)/E(B-V), default=3.1
- **fmlaw** (*str, optional*) – the fmlaw keyword is used only in the redlaw_fm function (default='GAL'): '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).

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.

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
```

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

Return type float64

Parameters

- **wavelength** (*float/array*) – Wavelength in Angstrom
- **relflux** (*float*) – flux intensity relative to Hb=100
- **m_ext** (*float*) – logarithmic extinction
- **ext_law** (*str, optional*) – the extinction law (default='GAL'): 'GAL' for Howarth Galactic; 'GAL2' for Savage and Mathis; 'CCM' for CCM galactic; 'JBK' for Whitford, Seaton, Kaler; 'FM' for Fitxpatrick; 'SMC' for Prevot SMC; 'LMC' for Howarth LMC.
- **rv** (*float, optional*) – the optical total-to-selective extinction ratio, RV = A(V)/E(B-V), default=3.1
- **fmlaw** (*str, optional*) – the fmlaw keyword is used only in the redlaw_fm function (default='GAL'): '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).

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.

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)
```

(continues on next page)

(continued from previous page)

```
>> 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
```

Returns This function returns the effective collision strengths (Omij_T).

Return type array/object

Parameters

- **temperature** (*float*) – electron temperature
- **omij_data** (*array/object*) – collision strengths (omega_ij) data
- **level_num** (*int*) – Number of levels
- **irats** (*int*) – Else Coll. rates = tabulated values * 10 ** irats

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.

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)
```

(continues on next page)

(continued from previous page)

```
>> atom='h'
>> ion='ii' # H I
>> 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
   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]
```

Parameters

- **temperature** (*float*) – electron temperature
- **density** (*float*) – electron density
- **elj_data** (*array/object*) – energy levels (Ej) data
- **omij_data** (*array/object*) – collision strengths (omega_ij) data
- **aij_data** (*array/object*) – transition probabilities (Aij) data
- **h_i_aeff_data** (*array/object*) – H I recombination coefficients
- **printEmissivity** (*boolean*) – Set for printing Emissivities
- **printPopulations** (*boolean*) – Set for printing Populations
- **printCritDensity** (*boolean*) – Set for printing Critical Densities

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.

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
```

Returns This function returns the reddening law function value(s) for the given wavelength(s).

Return type float64/array

Parameters

- **wavelength** (*float/array*) – Wavelength in Angstrom
- **ext_law** (*str, optional*) – the extinction law (default='GAL'): 'GAL' for Howarth Galactic; 'GAL2' for Savage and Mathis; 'CCM' for CCM galactic; 'JBK' for Whitford, Seaton, Kaler; 'FM' for Fitxpatrick; 'SMC' for Prevot SMC; 'LMC' for Howarth LMC.
- **rv** (*float, optional*) – the optical total-to-selective extinction ratio, RV = A(V)/E(B-V), default=3.1
- **fmlaw** (*str, optional*) – the fmlaw keyword is used only in the redlaw_fm function (default='GAL'): '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).

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

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

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
```

Returns This function returns the reddening law function value(s) for the given wavelength(s).

Return type float64/array

Parameters

- **wavelength** (*float/array*) – Wavelength in Angstrom
- **rv** (*float, optional*) – the optical total-to-selective extinction ratio, RV = A(V)/E(B-V), default=3.1

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.

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
```

Returns This function returns the reddening law function value(s) for the given wavelength(s).

Return type float64/array

Parameters

- **wavelength** (*float/array*) – Wavelength in Angstrom
- **rv** (*float, optional*) – the optical total-to-selective extinction ratio, RV = A(V)/E(B-V), default=3.1
- **fmlaw** (*str, optional*) – the fmlaw keyword is used only in the redlaw_fm function (default='GAL'): '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).

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.

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
```

Returns This function returns the reddening law function value(s) for the given wavelength(s).

Return type float64/array

Parameters

- **wavelength** (*float/array*) – Wavelength in Angstrom
- **rv** (*float, optional*) – the optical total-to-selective extinction ratio, RV = A(V)/E(B-V), default=3.1

pyequib.**redlaw_gal2**(*wavelength*)

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

For example:

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

Returns This function returns the reddening law function value(s) for the given wavelength(s).

Return type float64/array

Parameters **wavelength** (*float/array*) – Wavelength in Angstrom

pyequib.**redlaw_jbk**(*wavelength*)

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

For example:

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

Returns This function returns the reddening law function value(s) for the given wavelength(s).

Return type float64/array

Parameters **wavelength** (*float/array*) – Wavelength in Angstrom

pyequib.**redlaw_lmc**(*wavelength*)

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

For example:

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

Returns This function returns the reddening law function value(s) for the given wavelength(s).

Return type float64/array

Parameters **wavelength** (*float/array*) – Wavelength in Angstrom

pyequib.**redlaw_smc**(*wavelength*)

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

For example:

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

Returns This function returns the reddening law function value(s) for the given wavelength(s).

Return type float64/array

Parameters **wavelength** (*float/array*) – Wavelength in Angstrom