

---

**pyEQUIB**

*Release 0.2.0*

**Ashkbiz Danekar**

**Oct 14, 2020**



**CONTENTS:**

<b>1</b>	<b>pyEQUIB API functions</b>	<b>1</b>
	<b>Python Module Index</b>	<b>29</b>



## PYEQUIB API FUNCTIONS

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 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 Å 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_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)
>> 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')

```

(continues on next page)

(continued from previous page)

```

>> 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/'
>> 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

```

(continues on next page)

(continued from previous page)

```

>> 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/'
>> 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')

```

(continues on next page)

(continued from previous page)

```

>> 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/'

```

(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')
>> 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')
>> atom='s'
>> ion='ii'
>> s_ii_elj=atomneb.read_elj(atom_elj_file, atom, ion, level_num=5) #_
↳read Energy Levels (Ej)

```

(continues on next page)

(continued from previous page)

```

>> 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 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')

```

(continues on next page)

(continued from previous page)

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

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

**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
>> 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 function calculates the emissivity for the He II recombination line 4686 Å by using the helium emissivities from Storey & Hummer, 1995 MNRAS. 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_PFS12.
↳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 Kisieliu 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'
>> 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,
```

(continues on next page)

(continued from previous page)

```

>>                                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)
>> 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.
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, '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
```

**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 Fitzpatrick; '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  $H_b=100$ , 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
```

(continues on next page)

(continued from previous page)

```
>> 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 Fitzpatrick; '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)
>> 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:
```

(continues on next page)

(continued from previous page)

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)
>> 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,
```

(continues on next page)

(continued from previous page)

```

>>          aij_data=o_iii_ajj, 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 ( $E_j$ ) data
- **omij\_data** (*array/object*) – collision strengths ( $\omega_{ij}$ ) data
- **aij\_data** (*array/object*) – transition probabilities ( $A_{ij}$ ) 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 Fitzpatrick; '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



## PYTHON MODULE INDEX

### p

pyequib, [1](#)