ChiantiPy Documentation

Release 0.15.0

Ken Dere

Dec 01, 2022

CONTENTS

Welcome to the ChiantiPy documentation. ChiantiPy is a pure Python package for performing calculations of astrophysical spectra using the CHIANTI atomic database.

The latest version of ChiantiPy is 0.15.0 and is compatible with CHIANTI database version 10.0.X. It is not compatible with previous versions

ChiantiPy v0.15.0 is released under the OSI approved ISC license. From Wikipedia: The ISC license is a permissive free software license written by the Internet Software Consortium (ISC). It is functionally equivalent to the simplified BSD and MIT/Expat licenses, ...

CHIANTI consists of a database of atomic data that can be used to interpret spectral lines and continua emitted from high-temperature, optically-thin astrophysical sources.

CHIANTI is developed and maintained by scientists at George Mason University (USA), the University of Michigan (USA), and the University of Cambridge (UK). The first version of CHIANTI was released in 1997 and version 10.0 in 2021.

CHAPTER

ONE

GETTING STARTED WITH CHIANTIPY

1.1 Prerequisites

- CHIANTI, the atomic database for astrophysical spectroscopy (Version 10 or later)
- Python3 (3.8 is the current development version)
- Numpy (currently developed with 1.20)
- Scipy (currently developed with 1.6)
- Matplotlib (currently developed with 3.3.4)
- Matplotlib requires a GUI library

  PyQt5

  Once one of these is installed, it must be set as the backend in your matplotlibrc file, e.g., backend: Qt5Agg
- ipyparallel (required for multiprocessing with ipymspectrum)
- (not really a prerequisite but **extremely** useful) IPython version 7.21 and Jupyter

1.2 Install the CHIANTI database

The gzipped *data* tar ball can be downloaded from the CHIANTI website

- put the file in a convenient directory, cd to the directory and untar the file
- ChiantiPy uses the environment variable *XUVTOP* to find the database. Set XUVTOP to the name of the directory where the CHIANTI data tarball was placed. For example

```
setenv XUVTOP /data1/xuv/directory.where.the.tarball.was.placed
```

or on Windows: To set the environment variable, go to Control Panel -> System -> Advanced System Properties -> Environment Variables.

Some sites have the CHIANTI database maintained as part of a SolarSoft distribution. In that case, simply set XUVTOP to the directory were it resides, usually something like $SSW/packages/chianti/dbase

1.3 Install the Prerequisites

On **Linux** systems this can usually be done with your package manager.

On **Windows**, **Linux** and **Mac** systems, it is possible to use

- the Anaconda distribution from from Continuum, or,
- the Canopy distribution from Enthought.

On **Windows**, it is also possible to use:

- WinPython

All of these packages are free, at least for noncommercial use (I believe) and have a considerable amount of documentation. **You shoud check the version of IPython that is provided**.

1.4 Install the ChiantiPy package

In order to be compatible with the latest version (10) of the CHIANTI atomic database, it is necessary to install the latest version (0.10.0) of ChiantiPy

```
pip install ChiantiPy
```

or

```
pip3 install ChiantiPy
```

I have not tried this with ChiantiPy, myself.

The ChiantiPy package can be downloaded from the ChiantiPy project page at Sourceforge, untar it, cd to the directory where it was unpacked, and then, as root

```
python setup.py install
```

If you do not have root privileges, simply put the ChiantiPy directory in your PYTHONPATH

```
python setup.py install --prefix=somewhere_in_my_PYTHONPATH
```

or on a Mac, with the Anaconda package

```
python setup.py install --prefix=/Users/your_user_name/anaconda/
```

Thanks to Peter Young (GMU) for providing the instructions for installation on Mac and Windows

1.5 Note - ChiantiPy interactions with various GUI backends

First, Matplotlib requires a GUI backend and can be specified by the user in the matplotlibrc file. Matplotlib expects to find this file in $HOME/.config/matplotlib, although it might require that you copy it to that directory.

ChiantiPy also uses a GUI dialog widget set. Selections can also be made via the command line. The user choice is specified in the chiantirc file. One is included with the distribution. On Linux, if it is copied into either the $HOME/.config or $HOME/.chianti directory, it will automatically be picked up. It is a text file and can be edited. On Windows, it should be copied to $PROFILEHOME/.config or $PROFILEHOME/.chianti where it will also be picked up. Otherwise, the default GUI is to use the command line. A default chiantirc file is included with the distribution.

In order for the ChiantiPy dialog widget to be used, a backend for them must be initiated. If you choose the same backend for matplotlib (PyQt5 is best) as for the ChiantiPy widgets, then running %matplotlib in an IPython or Jupyter session will do the trick. In an interactive Python session, invoking a matplotlib or matplotlib qt command first should also do the trick.

```
matplotlib inline
```

or

```
matplotlib qt
```

if you are using Qt5

If you choose to use a GUI backend other than that used for matplotlib, then in an IPython or a Jupyter command the following magic commands are also available to start the backend:

```
%gui qt
```

ChiantiPy has mostly been tested with the Qt5 backend for Matplotlib and using the ChiantiPy Qt5 widgets.

CHAPTER

TWO

QUICK START

This short tutorial will demonstrate some of the capabilities of ChiantiPy and the CHIANTI database. It assumes that you know what the CHIANTI database provides and why you want to use it. It is useful to begin by exploring the properties of the **ion class**, as much of ChiantiPy is based on it. An ion such as Fe XIV is specified by the string 'fe_14', in the usual CHIANTI notation.

Perhaps the easiest way is to use a jupyter-notebook is to load the quick start notebook file QuickStart.ipynb in the directory jupyter_notebooks. Then, just run each cell step by step. If you are not familiar with notebooks, then you can cut and paste the following code into a Python/IPython session.

N.B.: in the time some of the plots and data were produced, there have been some changes to ChiantiPy and CHIANTI. It is possible that you might find differences (hopefully small).

Bring up a Python session (using > Python -i), or better yet, an IPython session

```
import os
```

the following will show the XUVTOP directory

```
os.environ['XUVTOP']
```

```
import ChiantiPy
import ChiantiPy.core as ch
import ChiantiPy.tools.filters as chfilters
import ChiantiPy.tools.io as chio
import numpy as np
import matplotlib.pyplot as plt
```

```
matplotlib qt
```

```
autoreload 2
```

to see the ChiantiPy version

```
ChiantiPy.__version__
```

to see the Ipython version

```
import IPython
print(' IPython version = %i.%i.%i'%(IPython.version_info[0],IPython.version_info[1],
↪IPython.version_info[2]))
```

It is useful to open a qtconsole where are the calculations can be easily examined

```
qtconsole
```

to see the version of the CHIANTI database

```
chianti_version = chio.versionRead()
```

```
chianti_version
```

2.1 Setting default values

ChiantiPy determines a number of default setting on instantiation. To use the default values list below, it is not necssary to do anything.

setting	default	possible values
wavelength	angstrom	angstrom, nm, ev, kev
flux	energy	energy, photon
abundfile	sun_photospheric_2015_scott	any.abund
ioneqfile	chianti	any.ioneq

to use any of the other possible values, check out the notes/setting_default_values in the documentation

the defaults can be checked

```
chdata.Defaults.keys()
```

```
chdata.Defaults['wavelength']
```

2.2 Level populations

As a start, we will examine the various properties of the Fe XIV emissivities as a function of temperature and density. So, let's define a numpy array of temperatures

```
temp = 10.**(5.8 + 0.05*np.arange(21.))
```

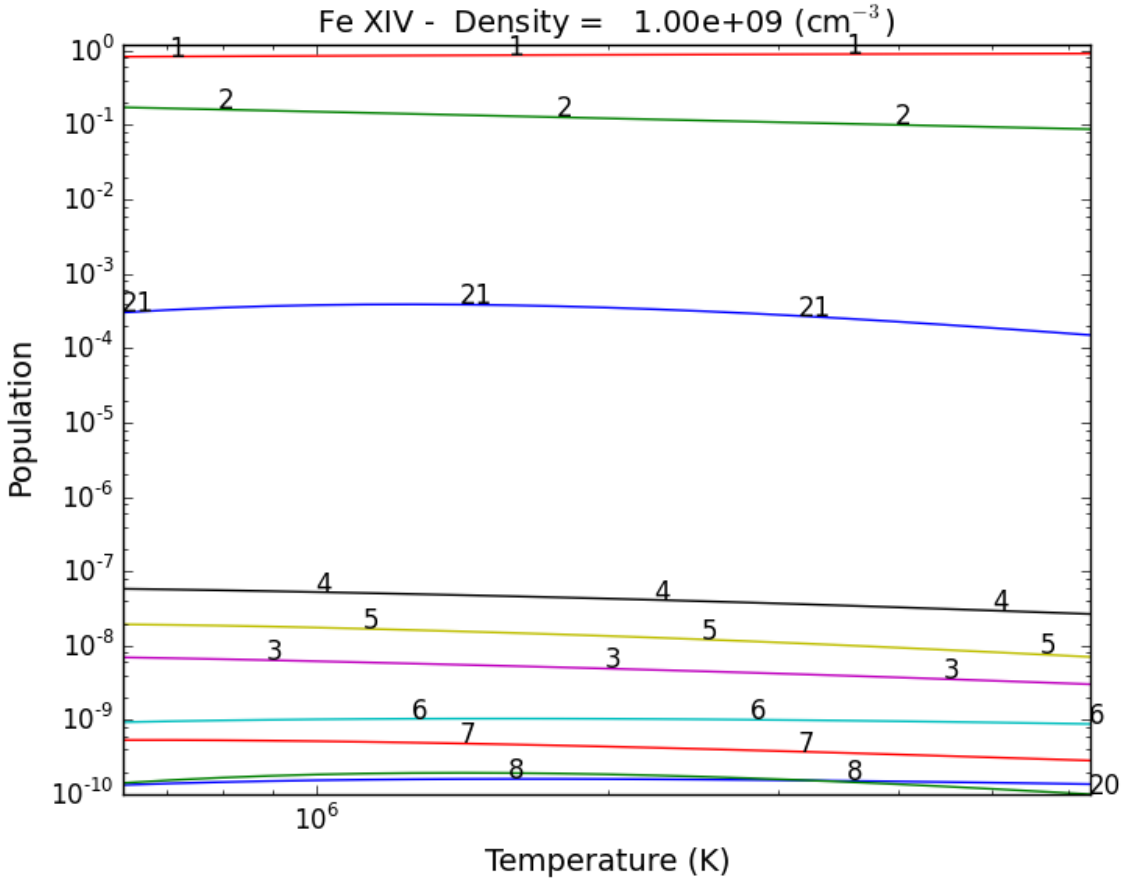
In ChiantiPy, temperatures are currently given in degrees Kelvin and densities as the number electron density per cubic cm. Then, construct fe14 as would be typically done

```
fe14 = ch.ion('fe_14', temperature=temp, eDensity=1.e+9, em=1.e+27)
```

note that eDensity is the new keyword for electron density

```
fe14.popPlot(addLegend=False)
```

produces a matplotlib plot window were the population of the top 10 (the default) levels are plotted as a function of temperature.



If the level populations had not already been calculated, popPlot() would have invoked the populate() method which calculates the level populations and stores them in the Population dictionary, with keys = [‘protonDensity’, ‘population’, ‘temperature’, ‘density’].

2.3 A ChiantiPy Convention

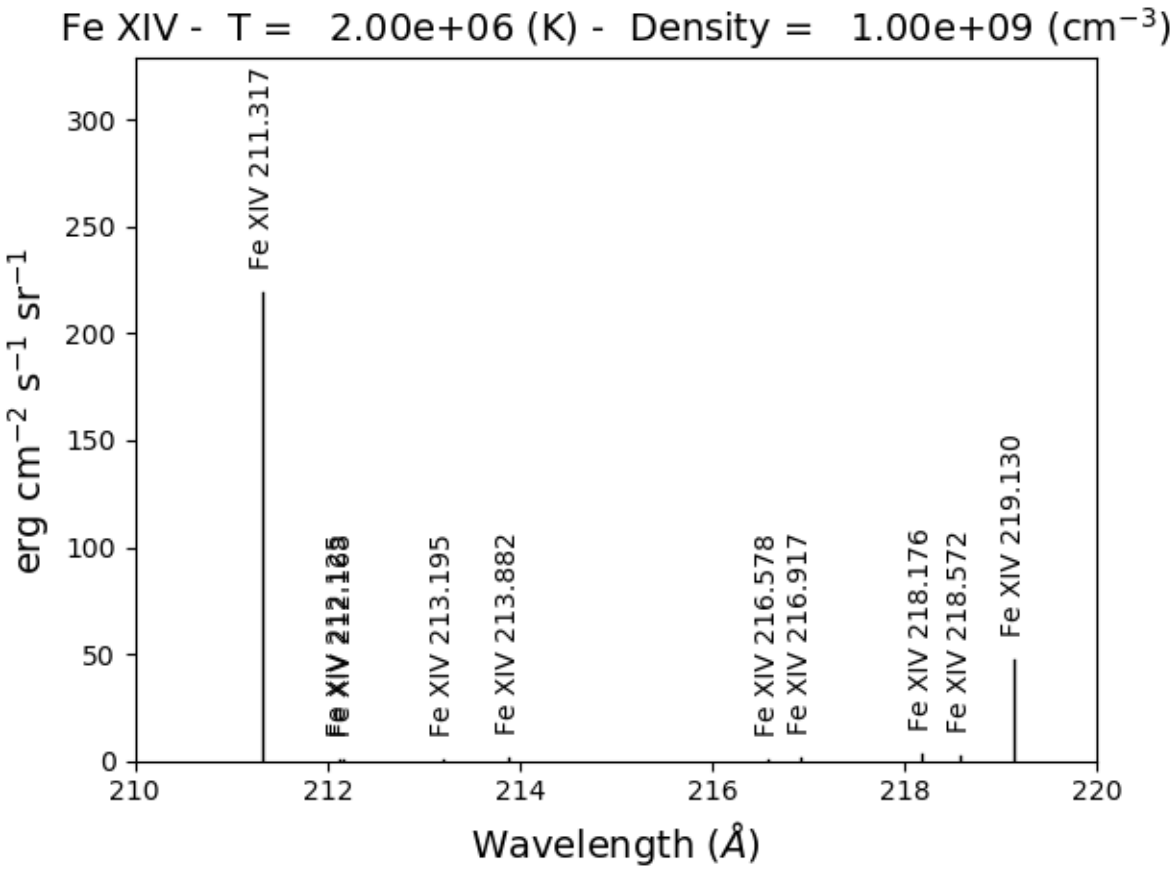
Classes and function of ChiantiPy start with lower case letters. Data/attributes that are attached to the instantiation of a class will start with a capital letter. For example,

```
fe14.populate() creates fe14.Population containing the level population information

fe14.intensity() created fe14.Intensity contain the line intensities information

fe14.spectrum() creates fe14.Spectrum contain the line and continuum spectrum information
```

2.4 Spectral Line Intensities

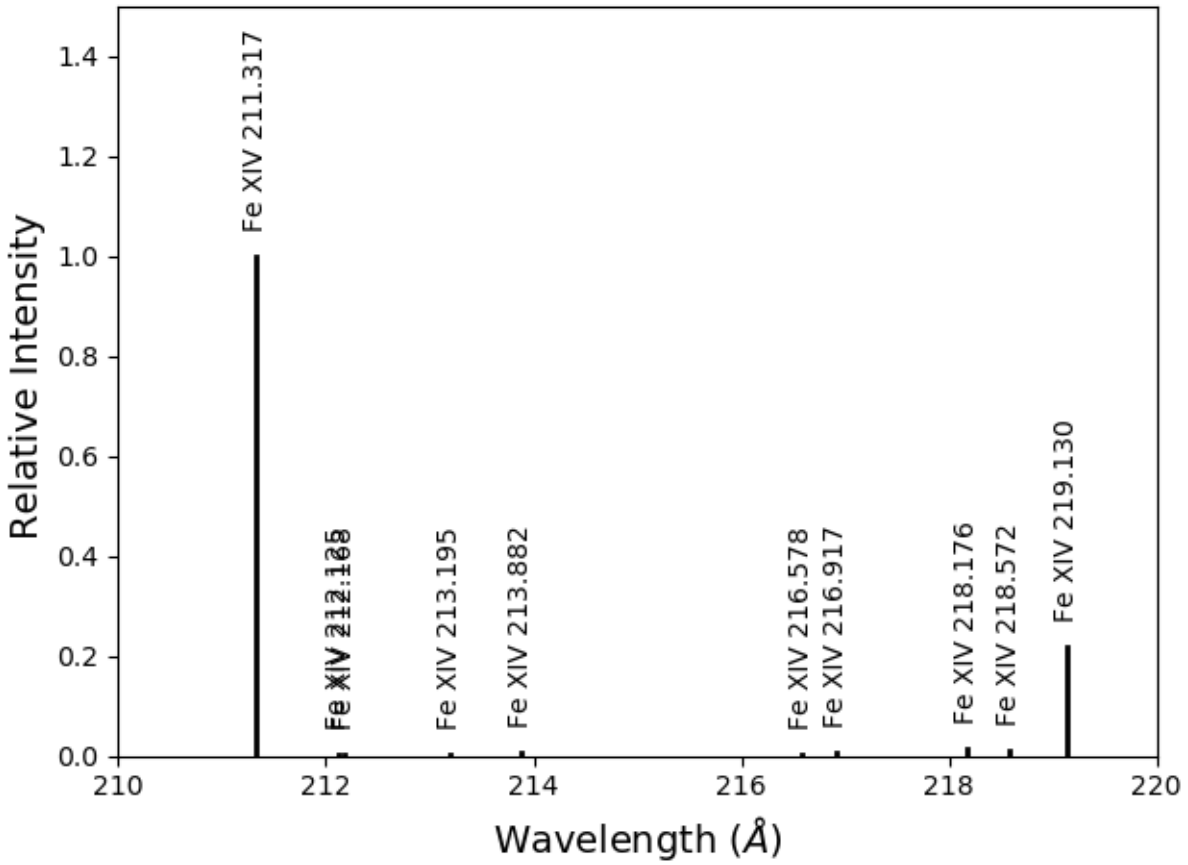
```
fe14.intensityPlot(wvlRange=[210.,220.])
```

will plot the intensities for the top (default = 10) lines in the specified wavelength range. If the **Intensity** attribute has not yet been calculated, it will calculate it. Since there are 21 temperature involved, a single temperature is selected (21//2 = 10). Otherwise,



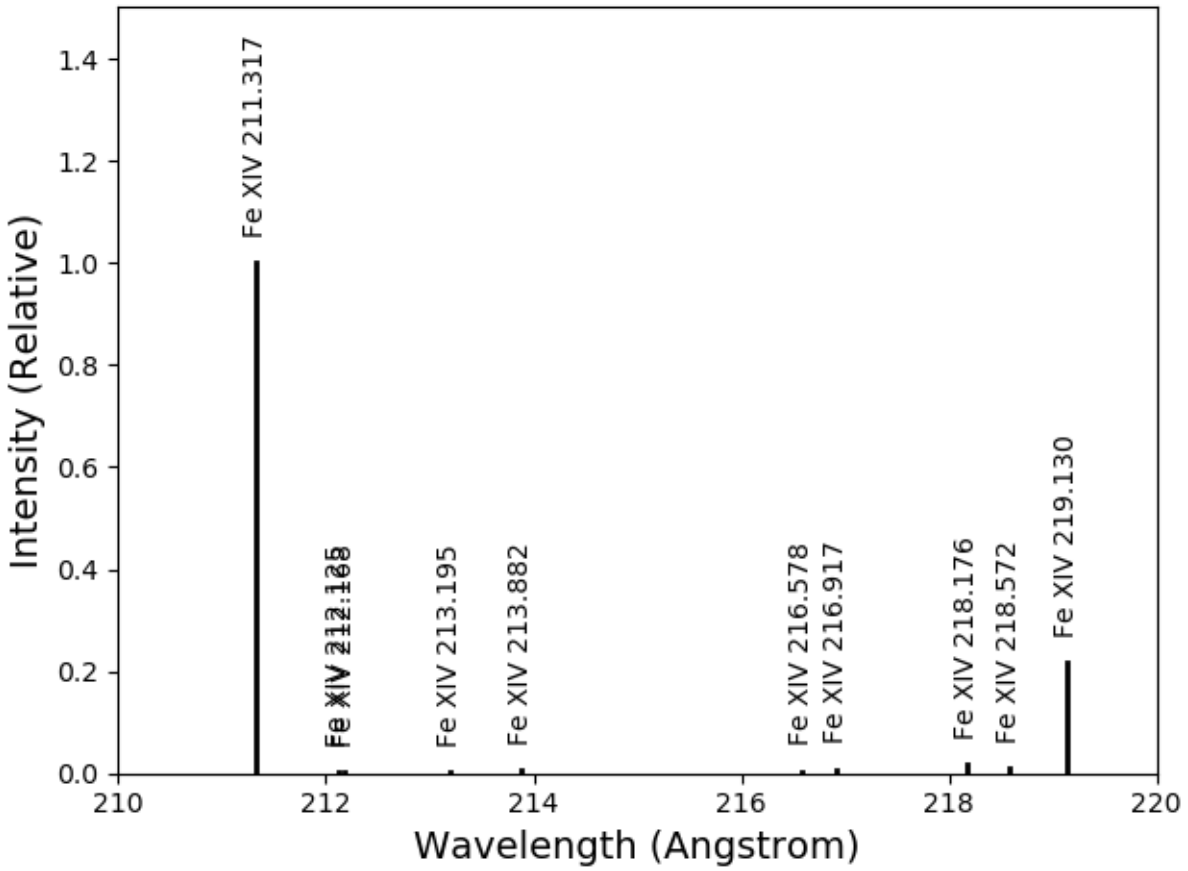

```
fe14.intensityPlot(index=10, wvlRange=[210., 220.], relative=True)
```

plots the intensities for a temperature = t[10] = 2.e+6, in this case. And, by specifying relative = True, the emissivities will be plotted relative to the strongest line.




```
fe14.intensityPlot(index=10, wvlRange=[210., 220.], relative=True, doTitle=False, lw=2)
```

plots the intensities for a temperature = t[10] = 2.e+6, in this case. And, by specifying relative = True, the emissivities will be plotted relative to the strongest line, doTitle=False, stops the title from appearing and lw sets the line width to 2.




```
fe14.intensityList(wvlRange=[200,220], index=10)
```

gives the following terminal output:

```
 using index =    10 specifying temperature =   2.00e+06, eDensity =    1.00e+09 em =
↪1.00e+27

 -----------------------------------------

 Ion  lvl1  lvl2                    lower - upper                          Wvl(A)
↪Intensity      A value Obs
 fe_14     1    11              3s2.3p 2P0.5 - 3s2.3d 2D1.5                211.3172
↪2.336e+02     3.81e+10 Y
 fe_14     4    27              3s.3p2 4P1.5 - 3s.3p(3P).3d 4P1.5          212.1255
↪5.355e-01     2.21e+10 Y
 fe_14     4    28              3s.3p2 4P1.5 - 3s.3p(3P).3d 4D2.5          212.1682
↪4.039e-01     1.15e+10 Y
 fe_14     3    24              3s.3p2 4P0.5 - 3s.3p(3P).3d 4D0.5          213.1955
↪8.073e-01     4.26e+10 Y
 fe_14     3    23              3s.3p2 4P0.5 - 3s.3p(3P).3d 4D1.5          213.8822
↪1.393e+00     2.97e+10 Y
 fe_14     5    28              3s.3p2 4P2.5 - 3s.3p(3P).3d 4D2.5          216.5786
↪9.736e-01     2.83e+10 Y
```

(continues on next page)

(continued from previous page)

```
 fe_14     5    25              3s.3p2 4P2.5 - 3s.3p(3P).3d 4D3.5             216.9173
↪1.730e+00     4.29e+10 Y
 fe_14     7    32              3s.3p2 2D2.5 - 3s.3p(3P).3d 2F3.5             218.1767
↪3.734e+00     1.70e+10 Y
 fe_14     4    22              3s.3p2 4P1.5 - 3s.3p(3P).3d 4P2.5             218.5725
↪2.391e+00     2.65e+10 Y
 fe_14     2    12              3s2.3p 2P1.5 - 3s2.3d 2D2.5                   219.1305
↪5.077e+01     4.27e+10 Y

-----------------------------------------
```

optionally, an output file could also be created by setting the keyword outFile to the name of the desired name

```
fe14.intensityList(wvlRange=[210.,220.], relative=1, index=11)
```

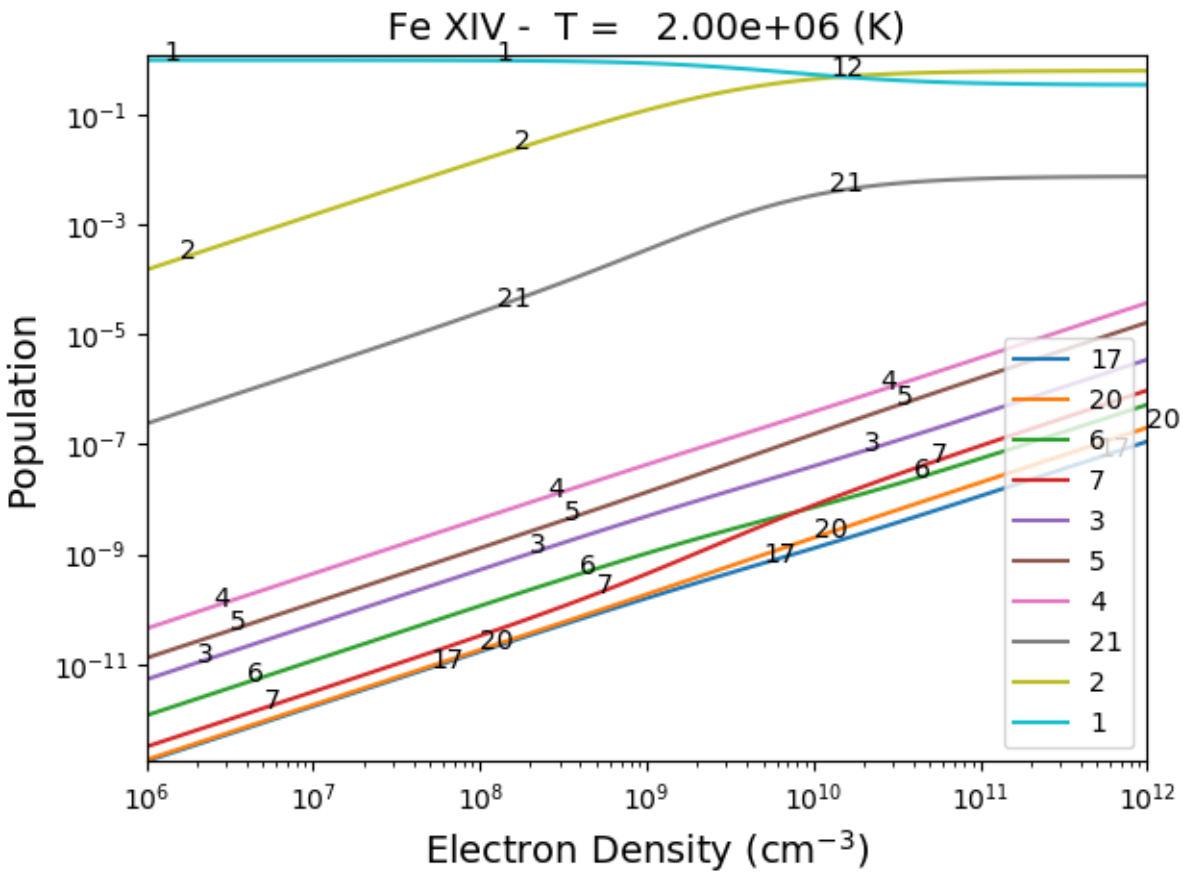
give the following terminal/notebook output

```
 using index =    11 specifying temperature =   2.24e+06, eDensity =    1.00e+09 em =
↪1.00e+27

 -----------------------------------------

 Ion  lvl1  lvl2                    lower - upper                          Wvl(A)
↪Intensity      A value Obs
 fe_14     1    11              3s2.3p 2P0.5 - 3s2.3d 2D1.5                   211.3172
↪1.000e+00     3.81e+10 Y
 fe_14     4    27              3s.3p2 4P1.5 - 3s.3p(3P).3d 4P1.5             212.1255
↪2.267e-03     2.21e+10 Y
 fe_14     4    28              3s.3p2 4P1.5 - 3s.3p(3P).3d 4D2.5             212.1682
↪1.694e-03     1.15e+10 Y
 fe_14     3    24              3s.3p2 4P0.5 - 3s.3p(3P).3d 4D0.5             213.1955
↪3.390e-03     4.26e+10 Y
 fe_14     3    23              3s.3p2 4P0.5 - 3s.3p(3P).3d 4D1.5             213.8822
↪5.891e-03     2.97e+10 Y
 fe_14     5    28              3s.3p2 4P2.5 - 3s.3p(3P).3d 4D2.5             216.5786
↪4.083e-03     2.83e+10 Y
 fe_14     5    25              3s.3p2 4P2.5 - 3s.3p(3P).3d 4D3.5             216.9173
↪7.085e-03     4.29e+10 Y
 fe_14     7    32              3s.3p2 2D2.5 - 3s.3p(3P).3d 2F3.5             218.1767
↪1.557e-02     1.70e+10 Y
 fe_14     4    22              3s.3p2 4P1.5 - 3s.3p(3P).3d 4P2.5             218.5725
↪1.009e-02     2.65e+10 Y
 fe_14     2    12              3s2.3p 2P1.5 - 3s2.3d 2D2.5                   219.1305
↪2.096e-01     4.27e+10 Y

-----------------------------------------
```

2.5 The effect of electron density on line intensities

```
temp = 2.e+6
dens = 10.**(6. + 0.1*np.arange(61))
fe14 = ch.ion('fe_14', temp, dens)
fe14.popPlot()
```

a plot of the population of the top 10 levels is produced as a function of the electron density

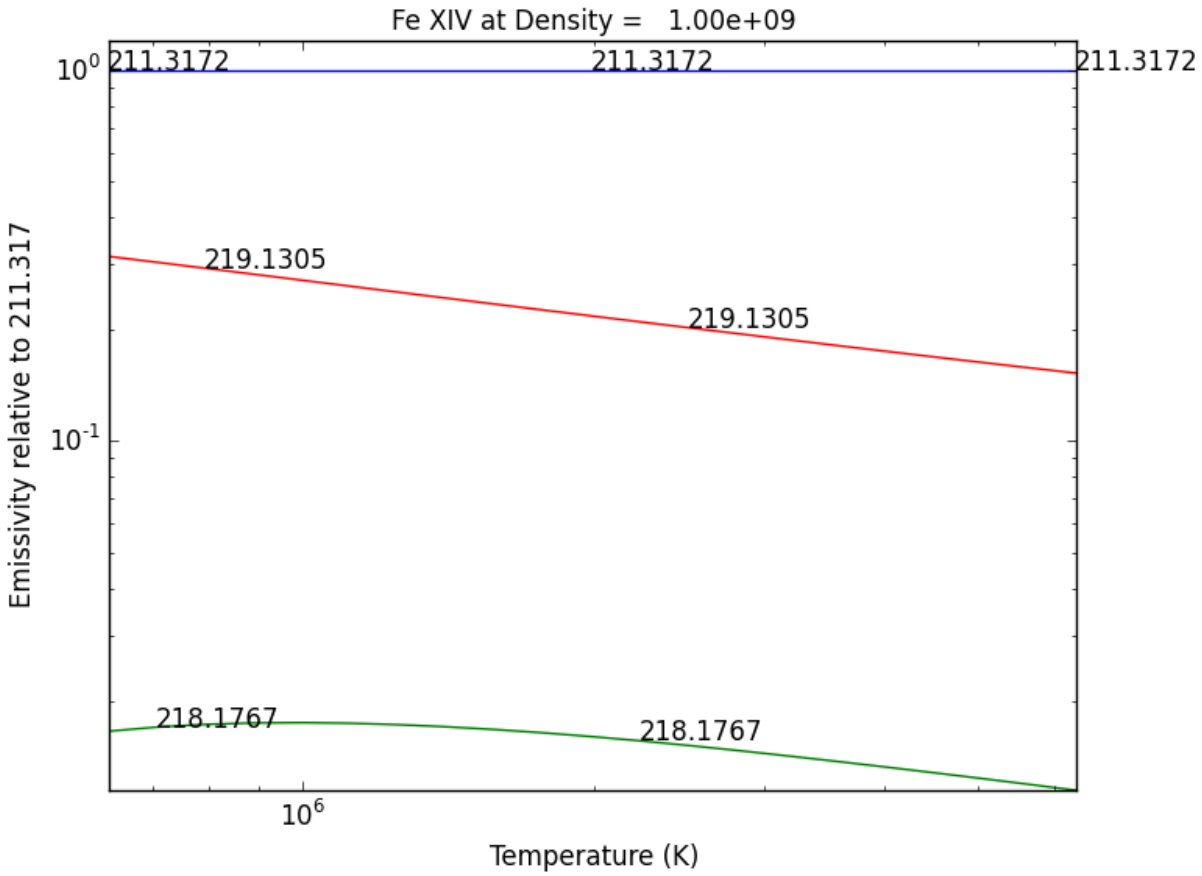


2.6 G(n,T) function

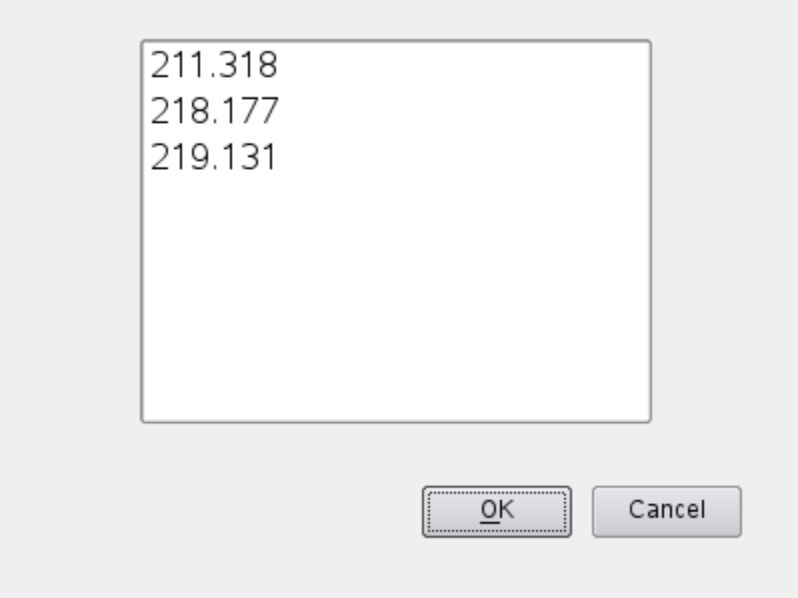
```
temp = 10.**(5.8 + 0.05*np.arange(21.))
dens = 1.e+9
fe14 = ch.ion('fe_14', temp, dens)
```

```
fe14.gofnt(wvlRange=[210., 220.],top=3)
```

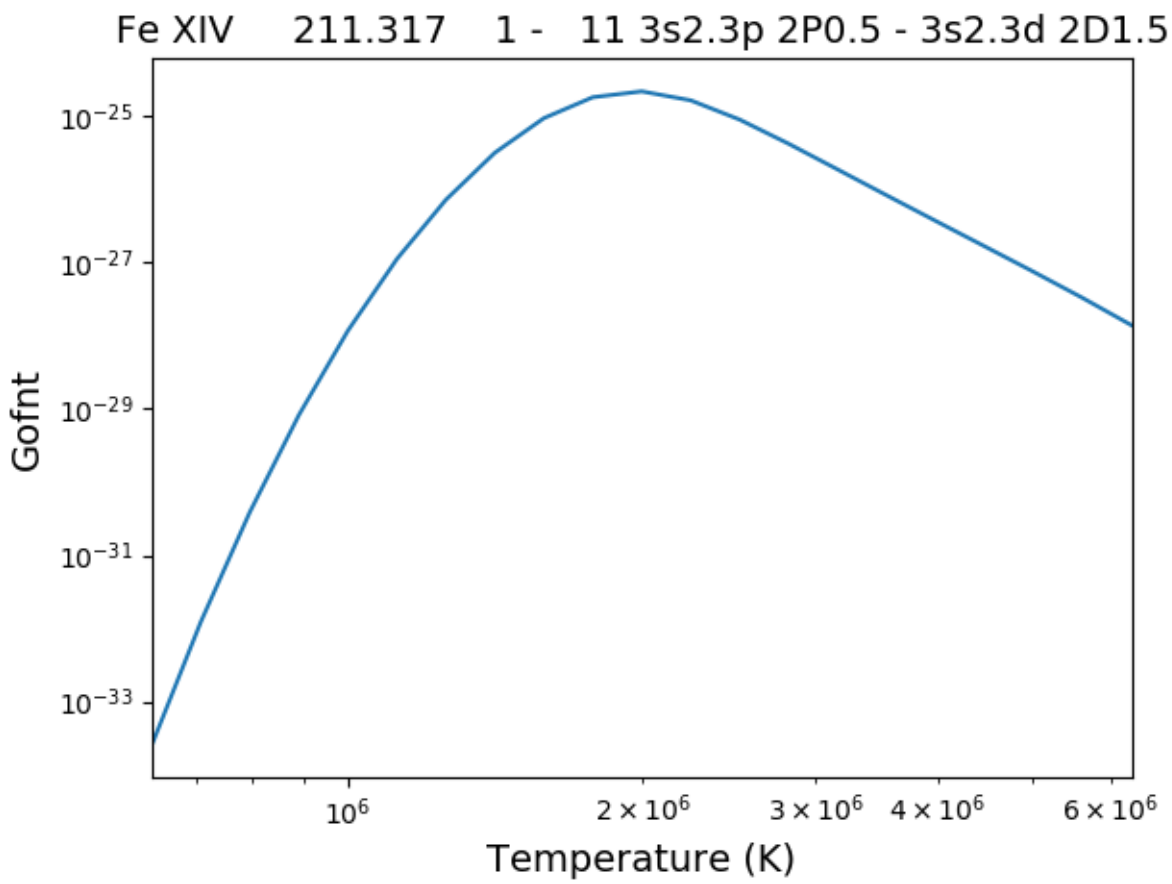
brings up a matplotlib plot window which shows the emissivities of the top (strongest) 3 lines in the wavelength region from 210 to 220 Angstroms.



quickly followed by a dialog where the line(s) of interest can be specified



and finally a plot of the G(n,T) function for the specified lines(s).



The G(n,T) calculation is stored in the Gofnt dictionary, with keys = [‘gofnt’, ‘temperature’, ‘density’]

while the is a fairly straightforward way to get a G(T) function, it is not very practical to use for a more than a handful of lines. For if the fe_14 line at 211.3172 is in a list of lines to be analyzed, a more practical way is the following

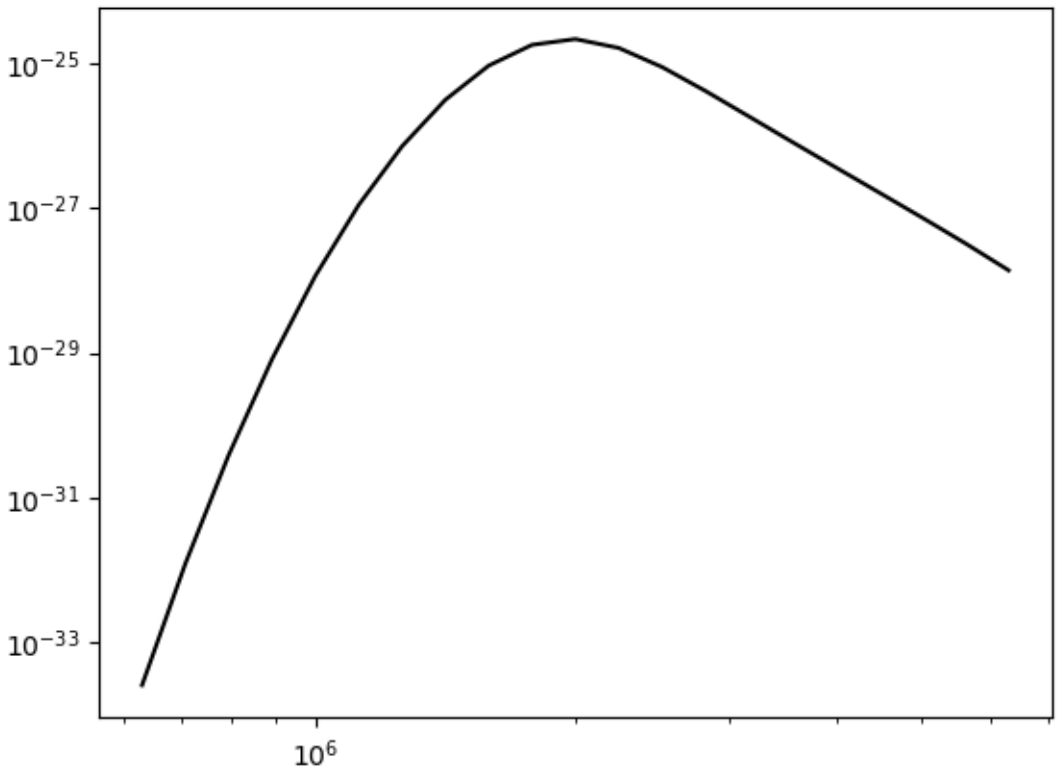
```
fe14.intensity()
dist = np.abs(np.asarray(fe14.Intensity['wvl']) - 211.3172)
idx = np.argmin(dist)
print(' wvl = %10.3f '%(fe14.Intensity['wvl'][idx]))
```

prints

wvl = 211.317

```
plt.loglog(temp,fe14.Intensity['intensity'][:,idx])
```

once the axes are properly scaled, this produces the same values as fe14.Gofnt[‘gofnt’]

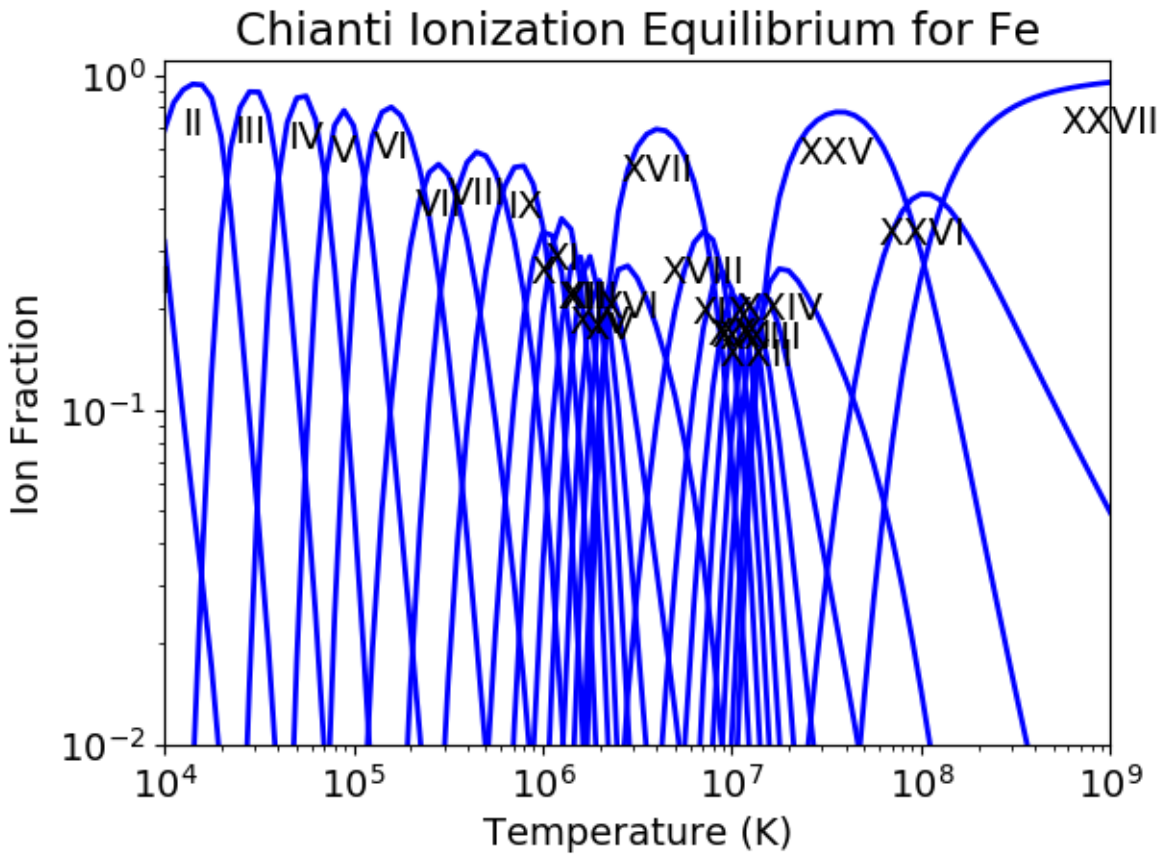


2.7 Ionization Equilibrium

For the Fe XIV example, the temperature was chosen to center around 2.e+6. It was not immediately apparent why this was done but in most of the following examples it is necessary to pick an appropriate temperature. This can be done with the **ioneq** class. To look at the ionization equilibrium for the iron ions (Z = 26, or 'fe')

```
fe = ch.ioneq(26)
fe.load()
fe.plot()
plt.tight_layout()
```

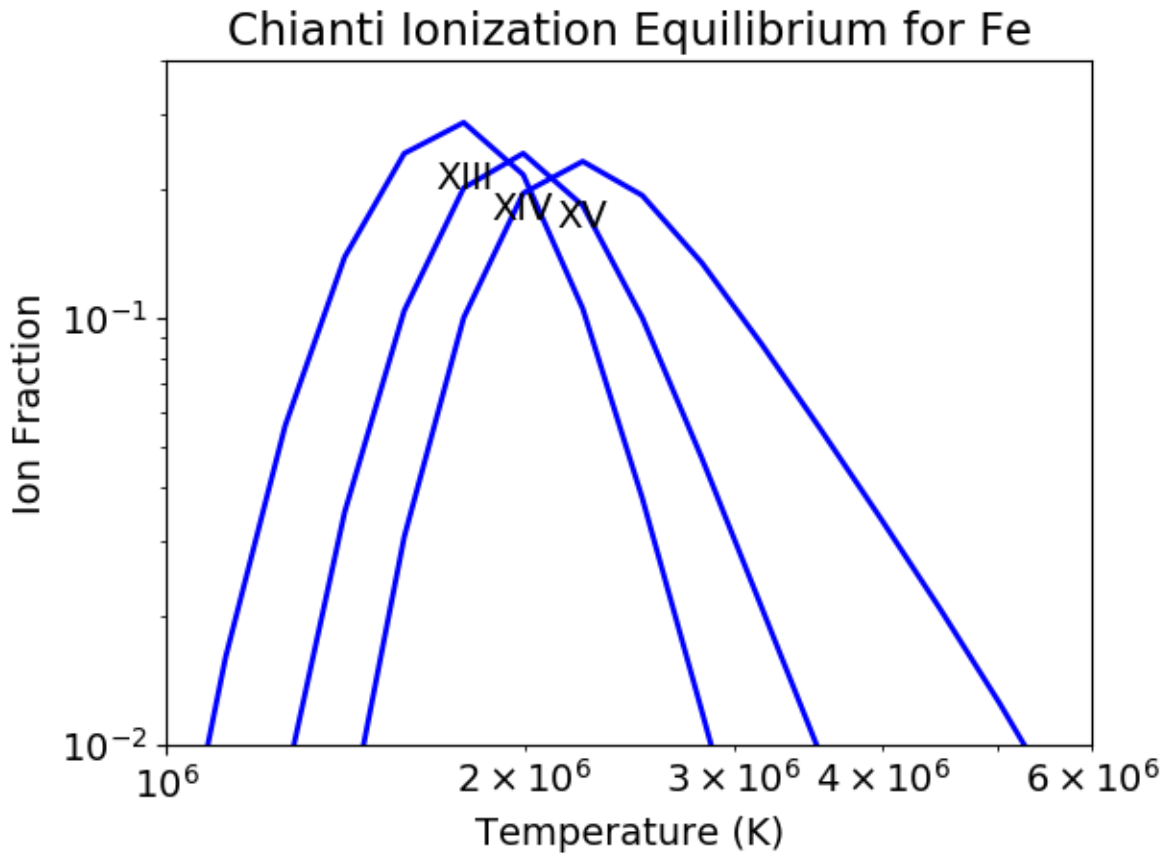
brings up a plot showing the ionization equilibrium for all of the stages of iron as a function of temperature



This is pretty crowded and we are only interested in Fe XIV (fe_14), so

```
plt.figure()
fe.plot(stages=[13,14,15],tRange=[1.e+6, 6.e+6], yr = [1.e-2, 0.4])
plt.tight_layout()
```

produces a plot of the ionization equilibria of Fe XIII, XIV and XV over a limited temperature range (tRange) and vertical range (yr)



from this it is clear that Fe XIV (fe_14) is formed at temperatures near 2×10^6 K

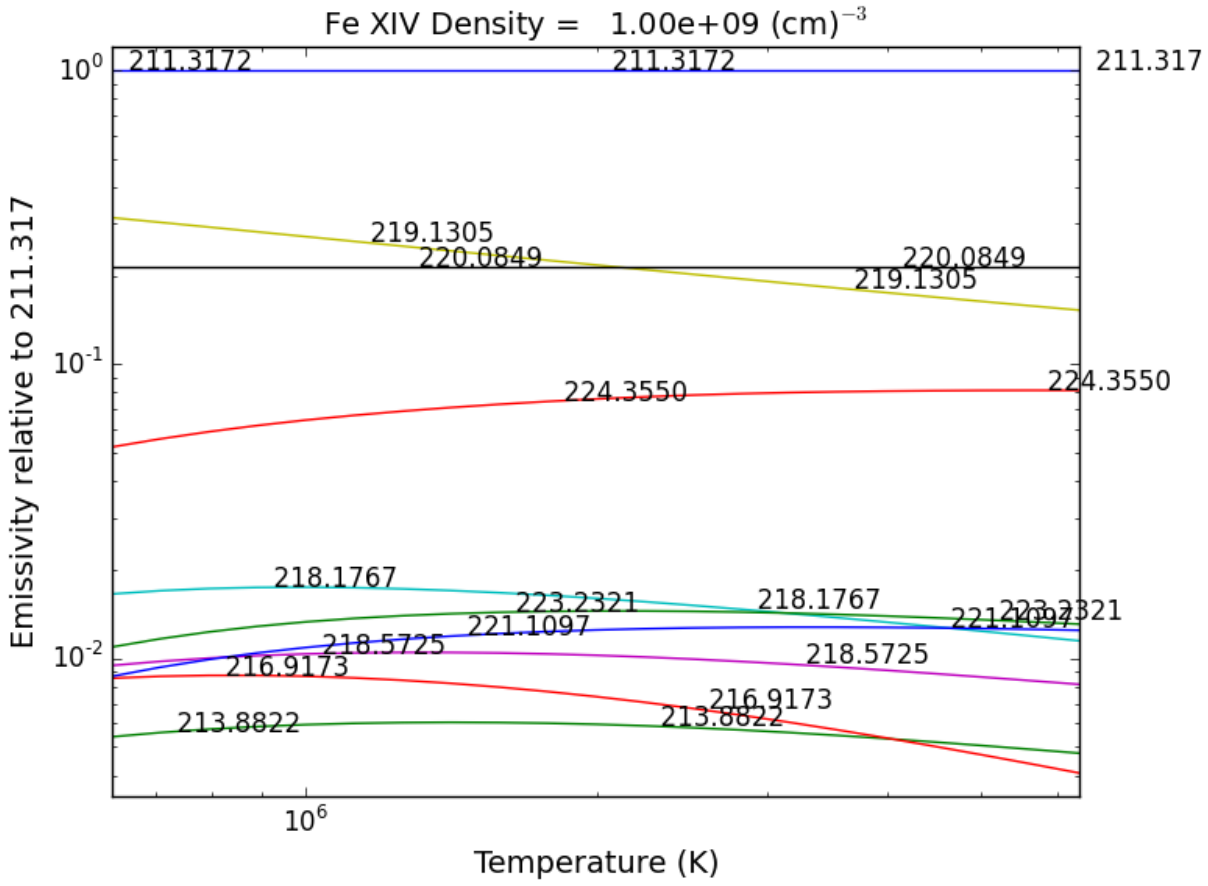
2.8 Intensity Ratios

```
temp = 10.**(5.8 + 0.05*np.arange(21.))
dens = 1.e+9
```

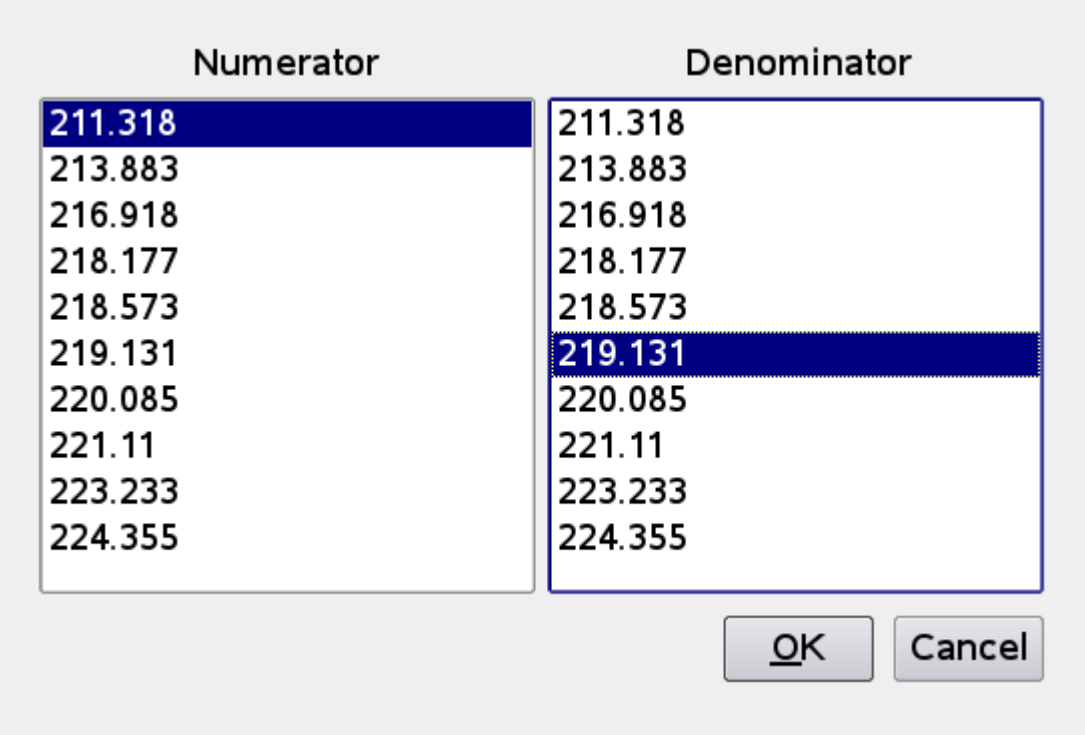
```
fe14 = ch.ion('fe_14', temperature = temp, eDensity = dens)
```

```
fe14.intensityRatio(wvlRange=[210., 225.])
```

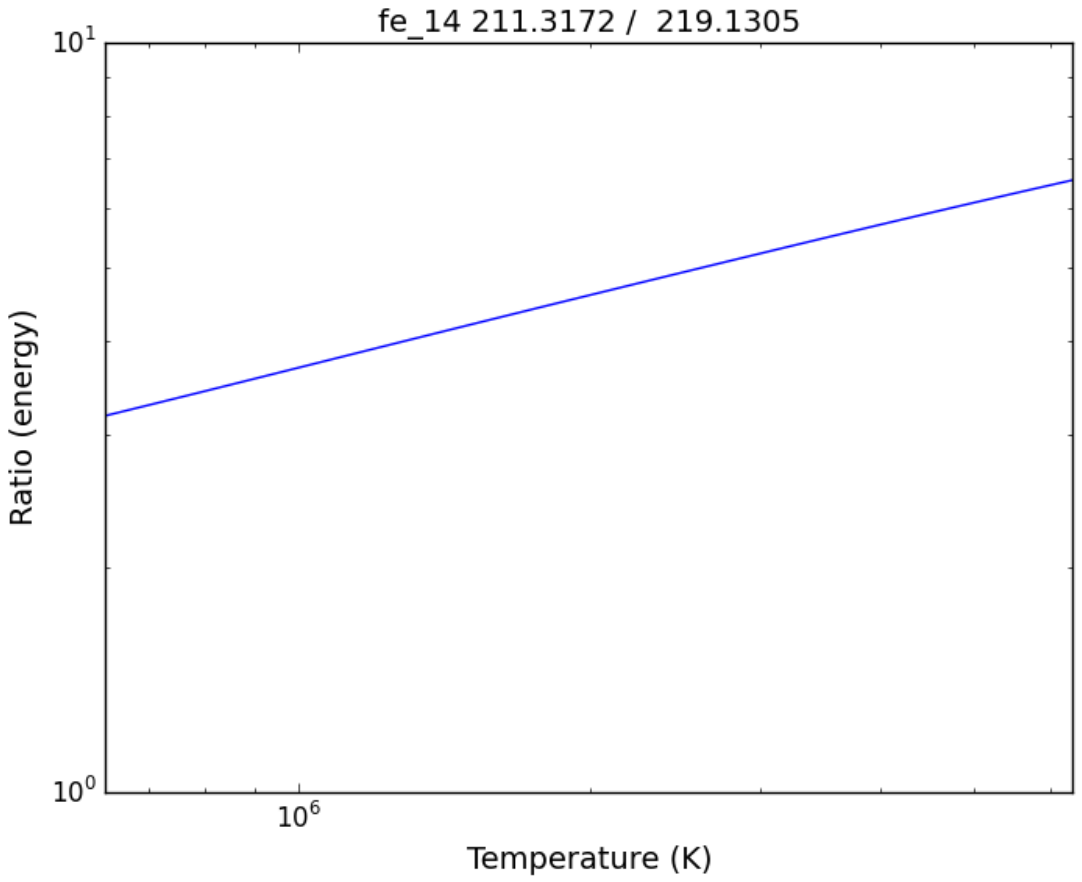
this brings up a plot showing the relative emissivities on the Fe XIV lines



following by a dialog where you can selector the numerator(s) and denominator(s) of the desired intensity ratio



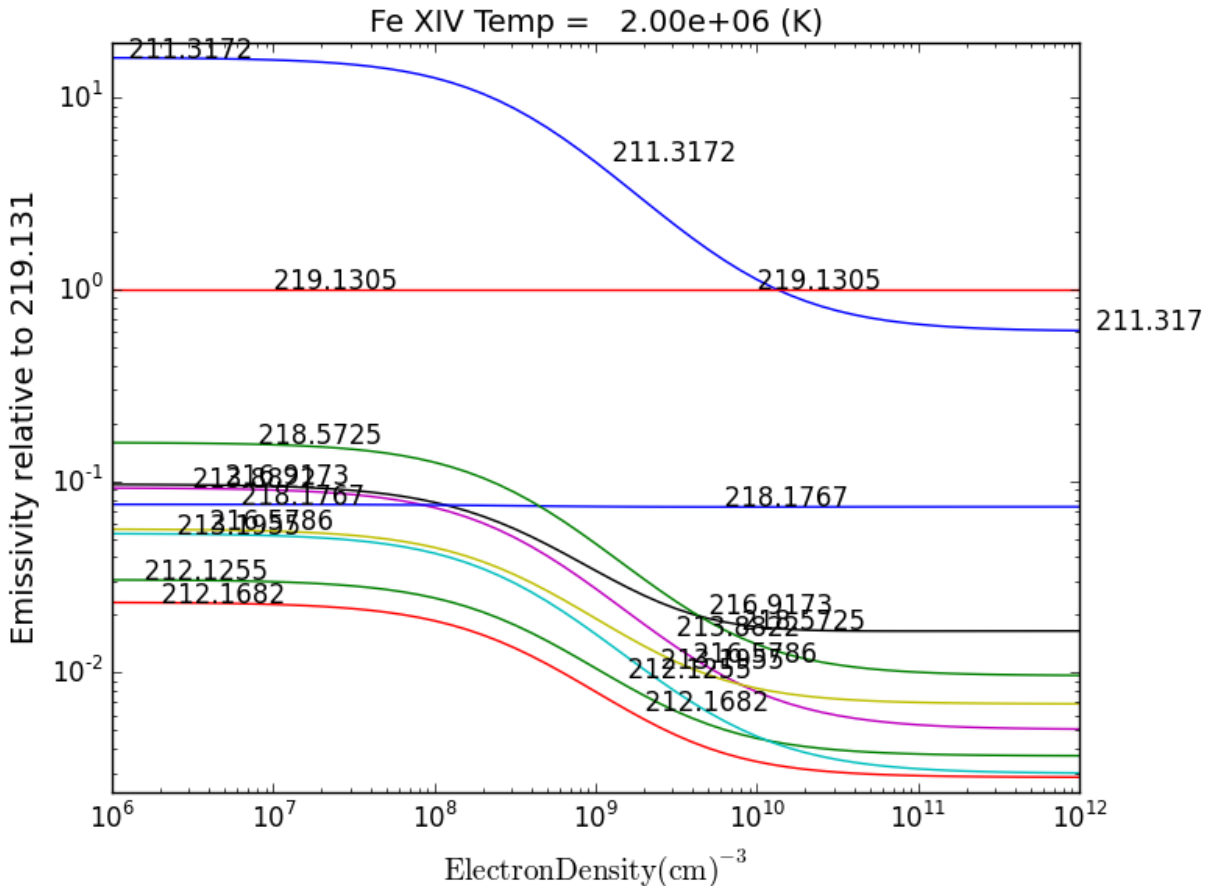
so the specified ratio is then plotted



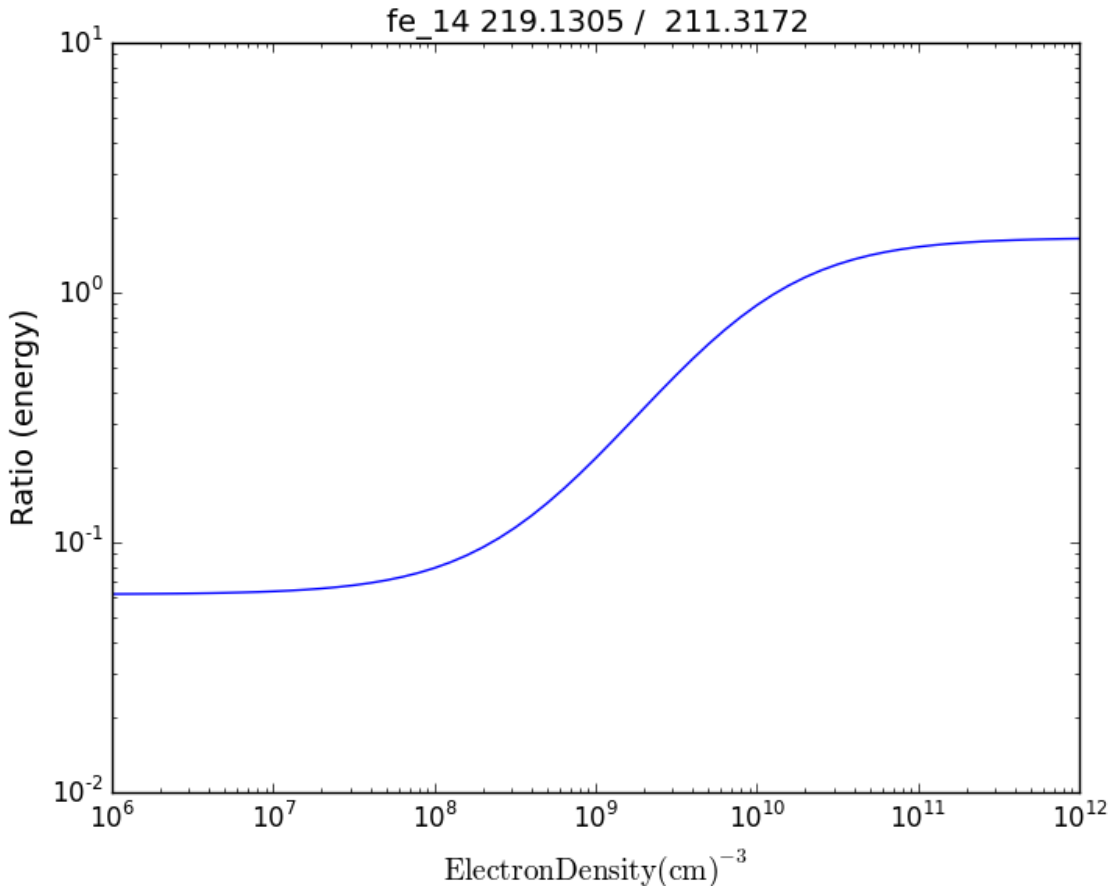
if previously, we had done

```
dens = 10.**(6. + 0.1*arange(61))
fe14 = ch.ion('fe_14', 2.e+6, dens)
fe14.intensityRatio(wvlRange=[210., 225.])
```

then the plot of relative intensities vs density would appear

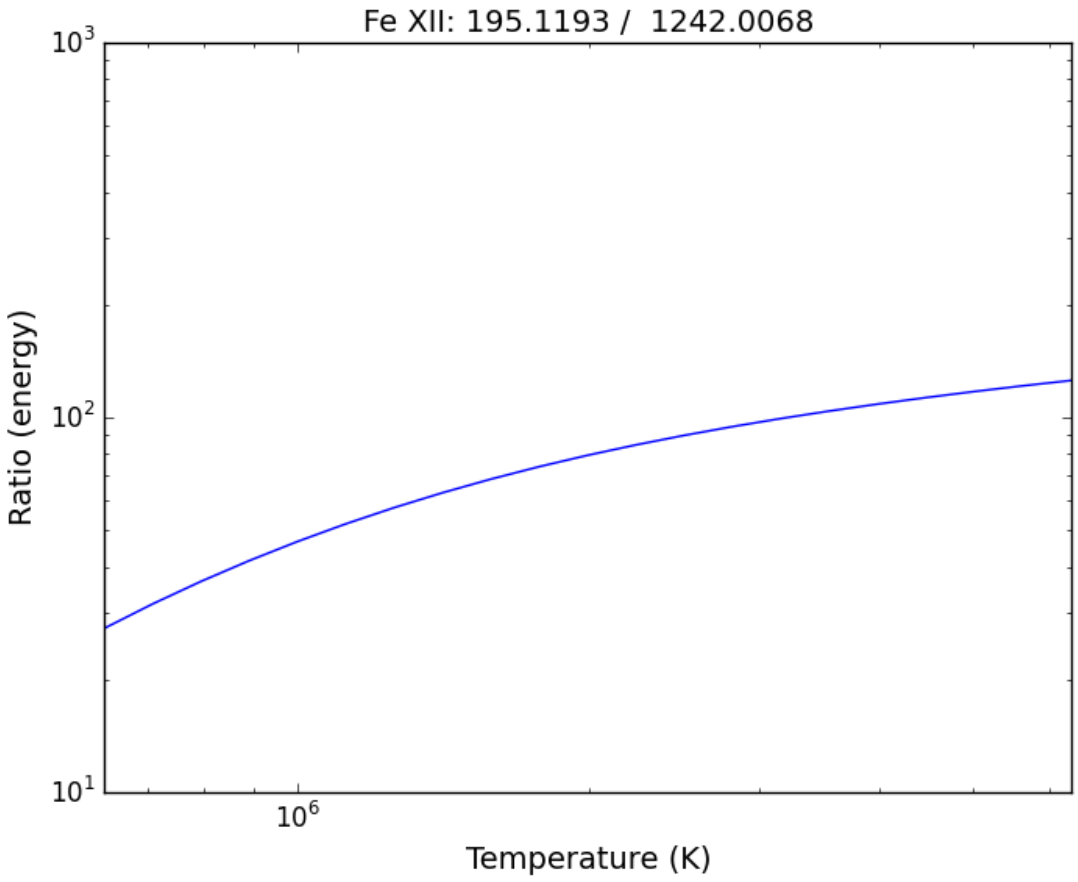


the same numerator/denominator selector dialog would come up and when 2 or more lines are selected, the intensity ratio versus density appears.



to obtain ratios of lines widely separated in wavelength, the wvlRanges keyword can be used:

```
fe12 = ch.ion('fe_12', temperature=t, eDensity=1.e+9
fe12.intensityRatio(wvlRanges=[[190.,200.],[1240.,1250.]])
```



2.9 Spectra of a single ion

```
fe14 = ch.ion('fe_14', temperature = 2.e+6, density = 1.e+9)
```

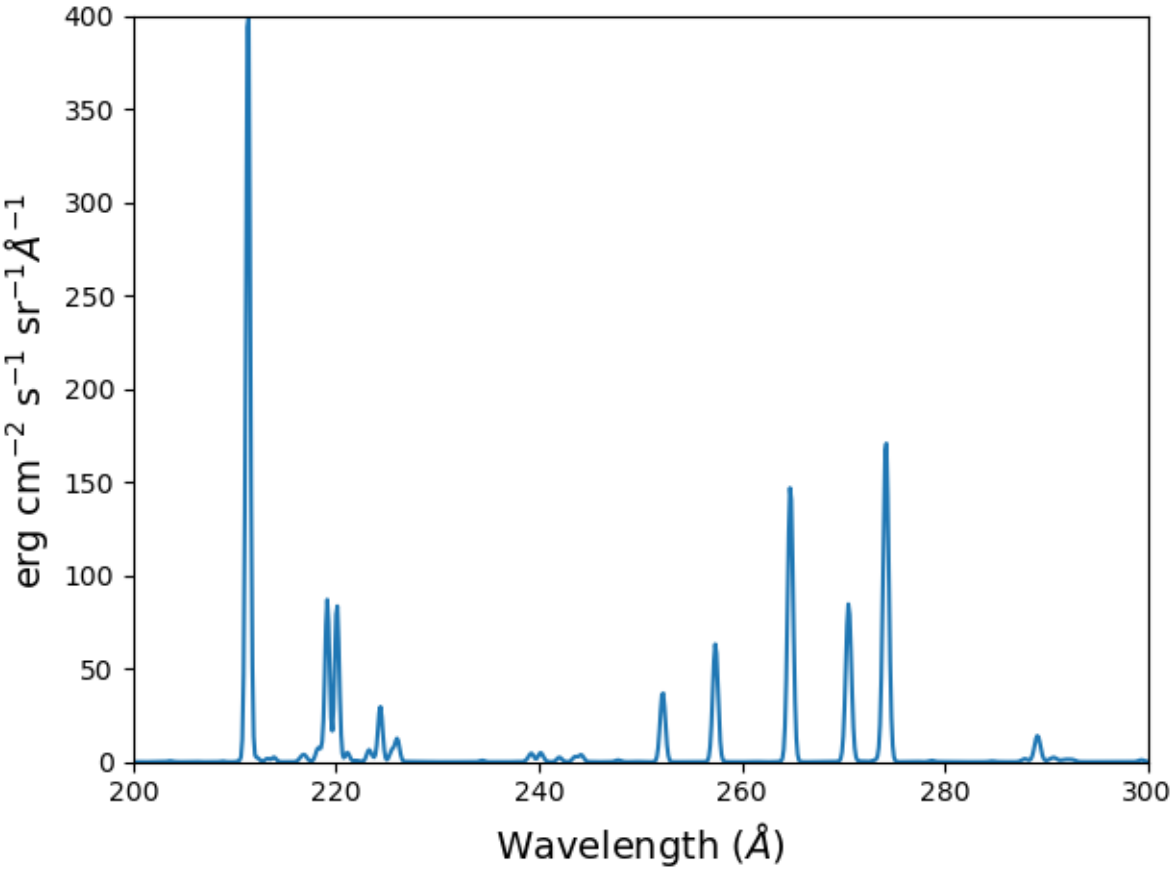
```
wvl = wvl=200. + 0.125*arange(801)
```

:: fe14.spectrum(wvl, em=1.e+27)

```
plt.figure()
plt.plot(wvl, fe14.Spectrum['intensity'])
xy = plt.axis()
xy
```

```
plt.axis([200., 300., 0., 400.])
plt.xlabel(fe14.Spectrum['xlabel'], fontsize=14)
plt.ylabel(fe14.Spectrum['ylabel'], fontsize=14)
plt.tight_layout()
```

this will calculate the spectrum of fe_14 over the specified wavelength range and filter it with the default filter which is a gaussian (filters.gaussianR) with a 'resolving power' of 1000 which gives a gaussian width of wvl/1000.



other filters available in chianti.tools.filters include a boxcar filter and a gaussian filter where the width can be specified directly

```
if hasattr(fe14,'Em'):
    print(' Emission Measure = %12.2e'%(fe14.Em))
else:
    print(' the value for the emission measure is unspecified')
```

Emission Measure = 1.00e+27

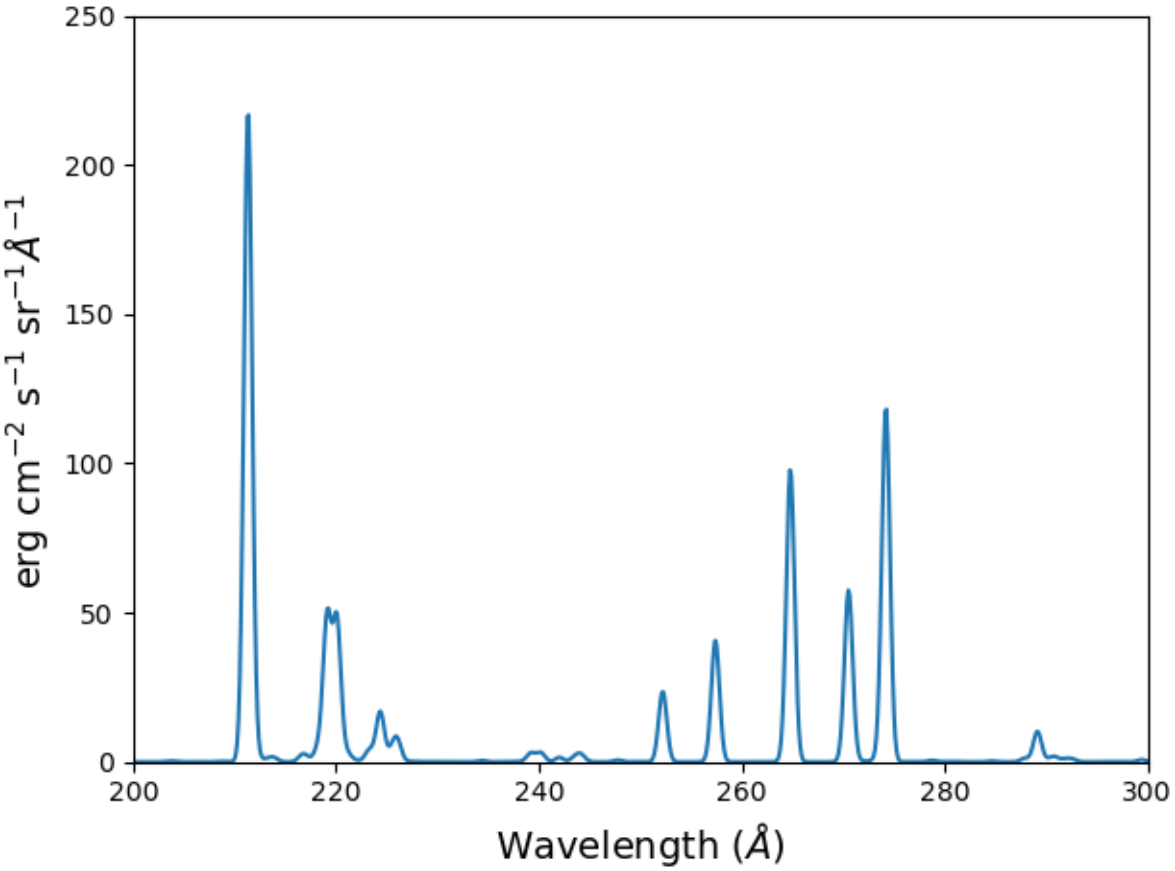
```
import chianti.tools.filters as chfilters
fe14.spectrum(wvl,filter=(chfilters.gaussian,.04))
```

calculates the spectrum of fe_14 for a gaussian filter with a width of 0.04 Angstroms. The current value of the spectrum is kept in fe14.Spectrum with the following keys:

```
for akey in sorted(fe14.Spectrum.keys()):
    print(' %10s'%(akey))
```

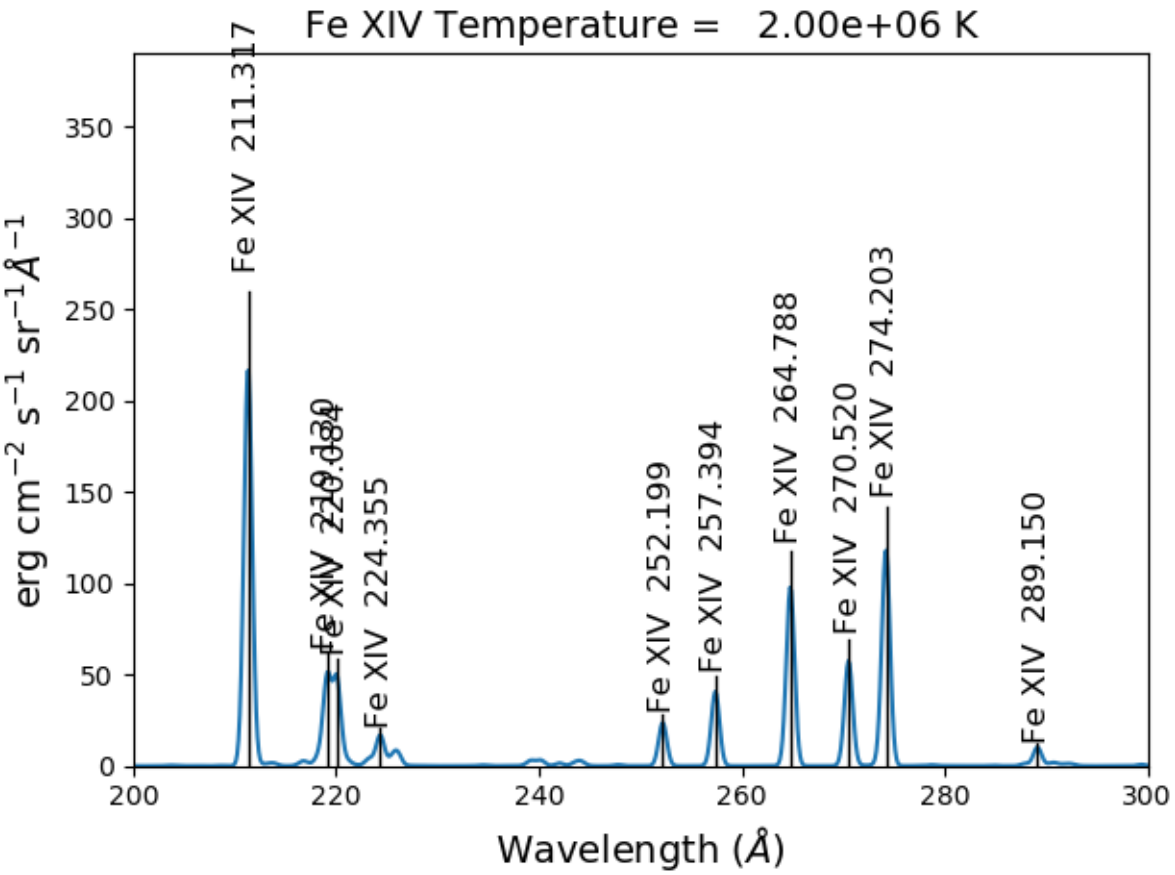
allLines em filter filterWidth intensity wvl xlabel ylabel

```
plot(wvl,fe14.Spectrum['intensity'])
plt.xlabel(fe14.Spectrum['xlabel'])
plt.ylabel(fe14.Spectrum['ylabel'])
```



As of **ChiantiPy 0.14.0**, the **ion** class inherits the spectrumPlot method.

```
wvlRange = [wvl[0], wvl[-1]
fe14.spectrumPlot(wvlRange=wvlRange, index=5)
```



Also in 0.14.0 is the saveData method and the redux class. Using the **saveData** method, the calculations can be save and the restored later with the **redux** class

```
saveName = 'fe14_save.pkl'
fe14.saveData(saveName, verbose=True)
```

the attributes are used to create a dict and saved as a pickle file. If verbose is set to True, these attributes are listed

```
with open(saveName,'rb') as inpt:
    fe14Dict = pickle.load(inpt)
```

```
for akey in fe14Dict:
  print(' key = %s'%(akey))
```

```
for akey in fe14Dict['Spectrum']:
    print(' key = %s'%(akey))
```

it is possible to work directly with the saved data

```
plt.figure()
plt.plot(fe14Dict['Spectrum']['wavelength'], fe14Dict['Spectrum']['intensity'])
```

with version 0.14.0, there is a new class, **redux**

with this class, the saved data can be restored and all of the apprpriated inherited methods are available

```
rdx = ch.redux(saveName, verbose=True)
```

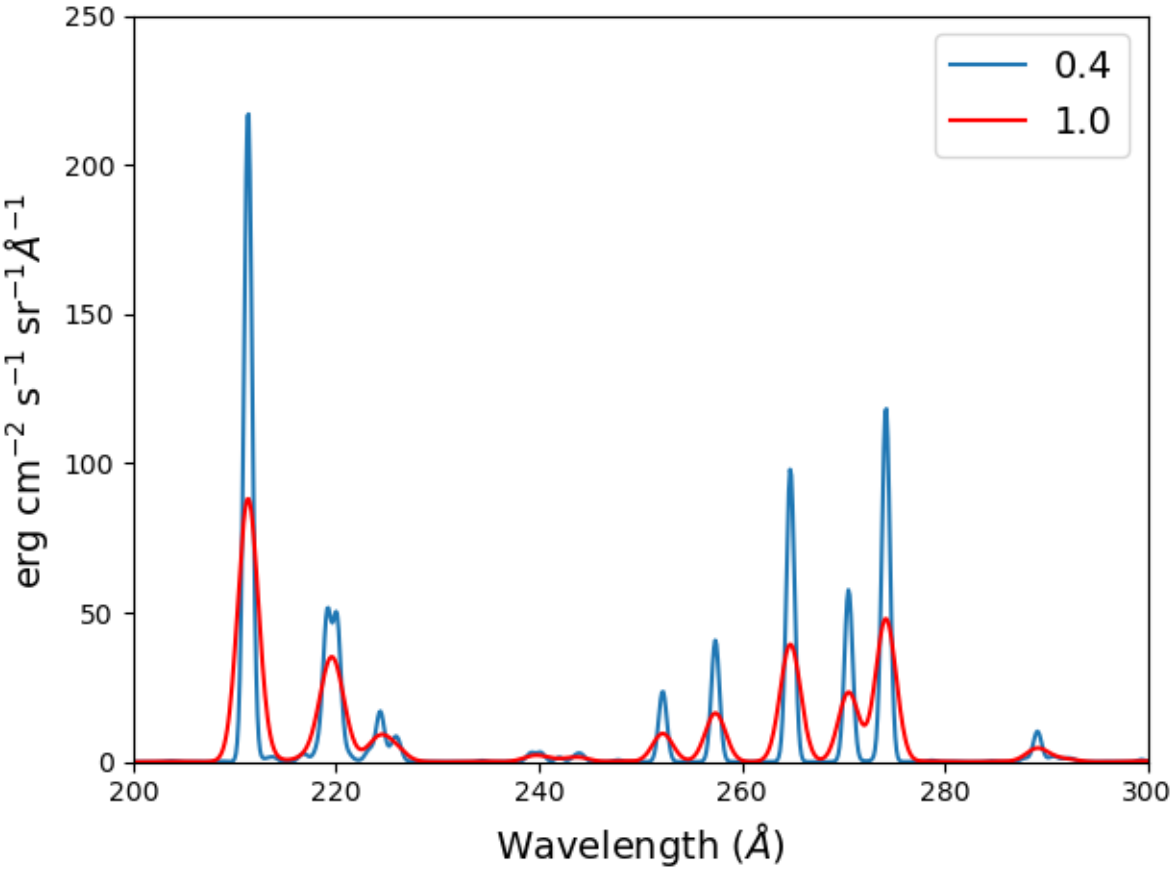
The save data are loaded as attributes. With verbose=True, they are listed

```
rdx.spectrumPlot(wvlRange=wvlRange, index=5)
```

Returns the previous plot

New in **ChiantiPy 0.6**, the *label* keyword has been added to the ion.spectrum method, and also to the other various spectral classes. This allows several spectral calculations for different filters to be saved and compared. However, when the *label* keyword is specified, the intensityPlot and spectrumPlot methods do not work, as of version 0.14.0

```
temp = 10.**(5.8 + 0.1*np.arange(11.))
dens = 1.e+9
fe14 = ch.ion('fe_14', temp, dens)
emeas = np.ones(11,'float64')*1.e+27
wvl = 200. + 0.125*np.arange(801)
fe14.spectrum(wvl,filter=(chfilters.gaussian,.4),label='.4',em=emeas, label='0.4')
fe14.spectrum(wvl,filter=(chfilters.gaussian,1.),label='1.', label-'1.0')
plt.plot(wvl,fe14.Spectrum['.4']['intensity'][5])
plt.plot(wvl,fe14.Spectrum['1.']['intensity'][5],'-r')
plt.xlabel(fe14.Spectrum['.4']['xlabel'])
plt.ylabel(fe14.Spectrum['.4']['ylabel'])
plt.legend(loc='upper right')
plt.tight_layout()
```



2.10 Using emission measures (EM)

the line-of-sight emission measure is given by $\int \mathrm{n_e\, n_H\, dl}\ (\mathrm{cm^{-5}})$

the volumetric emission measure is give by $\int \mathrm{n_e\, n_H\, dV}\ (\mathrm{cm^{-3}})$

where the integrations are performed over the source region

```
emDir = os.path.join(os.environ['XUVTOP'], 'em')
emList = os.listdir(emDir)
for idx, emFile in enumerate(emList):
  print('%i  %s'%(idx, emFile))
```

the following is printed

0 quiet_sun_1993_serts_4T.em
1 active_region_1993_serts_4T.em

Beginning with CHIANTI version 10, a new directory, em, as been added to contain emission measure files.

At this time, there only 2 files available and we can pick the active region file

```
arDict = chio.emRead(emList[1])
arDict.keys()
```

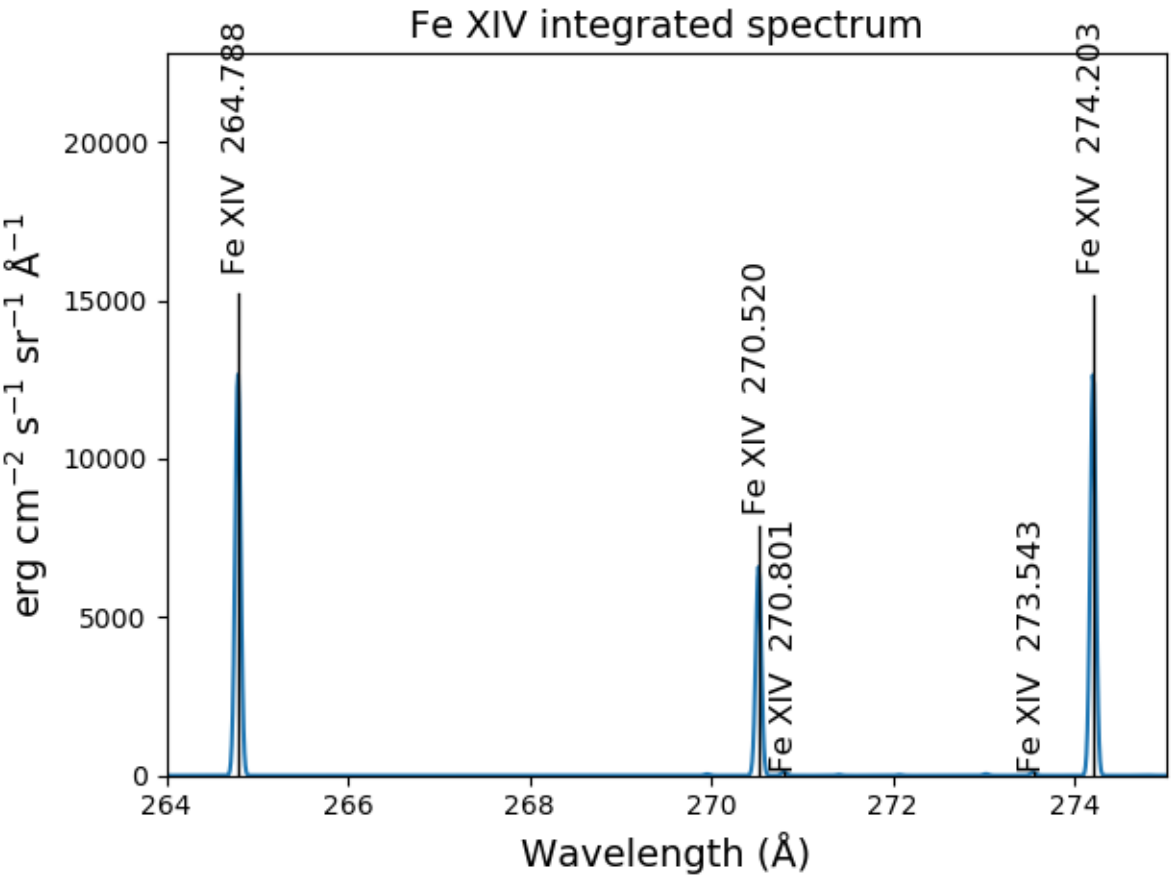
dict_keys([‘temperature’, ‘density’, ‘em’, ‘ref’, ‘filename’])

```
arTemp = arDict['temperature']
arDens = arDict['density']
arEm = arDict['em']
```

```
for idx, atemp in enumerate(arTemp):
    print('%i %10.2e %10.2e %10.2e'%(idx, atemp, arDens[idx], arEm[idx]))
```

0 6.17e+05 2.00e+09 4.97e+26
1 1.12e+06 2.00e+09 2.09e+27
2 1.86e+06 2.00e+09 7.89e+27
3 3.16e+06 2.00e+09 1.46e+28

```
fe14  = ch.ion('fe_14', arTemp, arDens, em=arEm)
wvl = np.linspace(200., 300., 10001)
fe14.spectrum(wvl, filter=(chfilters.gaussian, .03))
fe14.spectrumPlot(wvlRange=[264., 275.], integrated=True, top=5)
```



2.11 Free-free and free-bound continuum

The module continuum provides the ability to calculate the free-free and free-bound spectrum for a large number of individual ions. The two-photon continuum is produced only by the hydrogen-like and helium-like ions

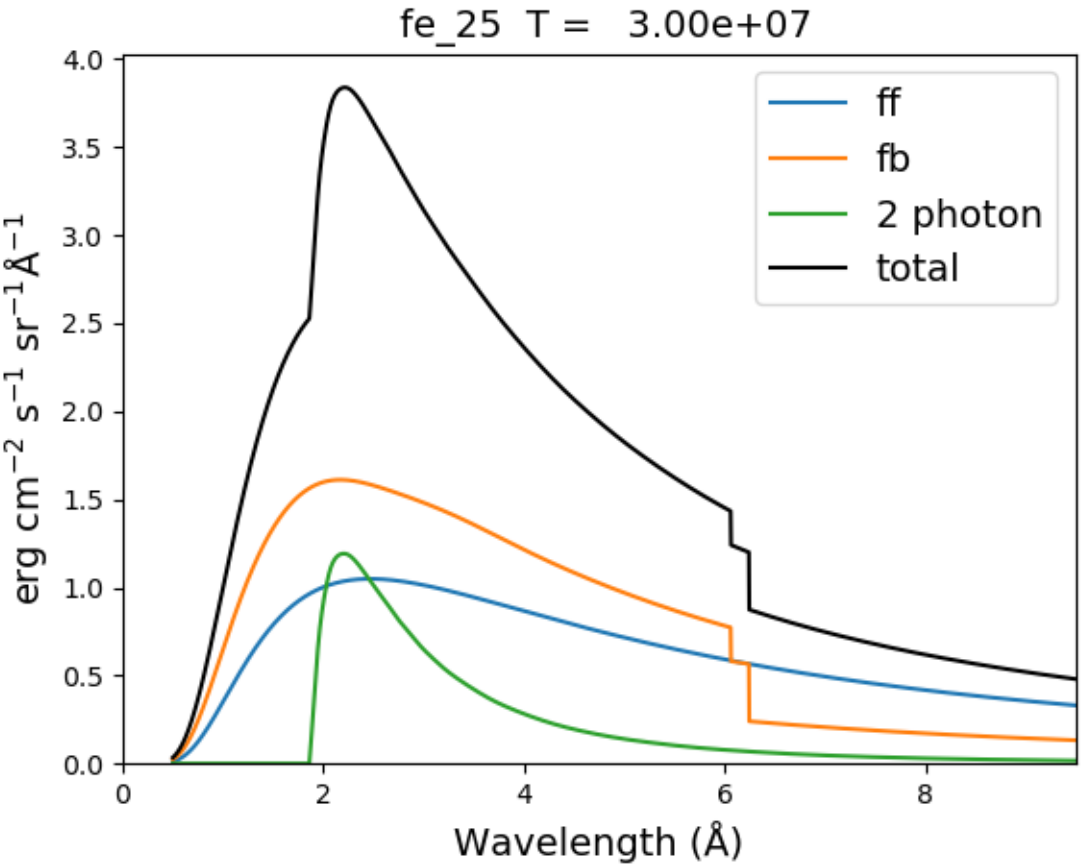
```
myIon = 'fe_25'
```

```
temperature = [2.e+7, 3.e+7, 6.e+7]
density = 1.e+9
em = [1.e+27, 1.e+27, 1.e+27]
wvl = 0.5 + 0.002*np.arange(4501)
```

```
c = ch.continuum(myIon, temperature = temperature, em=em)
c.freeFree(wvl)
c.freeBound(wvl)
fe25=ch.ion(myIon, temperature, density, em=em)
fe25.twoPhoton(wvl)
total = c.FreeFree['intensity'][itemp] + c.FreeBound['intensity'][itemp] + fe25.
↪TwoPhoton['intensity'][itemp]
```

```
itemp = 1
plt.figure()
plt.plot(wvl, c.FreeFree['intensity'][itemp],label='ff')
plt.plot(wvl, c.FreeBound['intensity'][itemp],label='fb')
plt.plot(wvl,fe25.TwoPhoton['intensity'][itemp],label='2 photon')
plt.plot(wvl, total, 'k', label='total')
plt.xlabel(c.FreeFree['xlabel'], fontsize=14)
plt.ylabel(c.FreeFree['ylabel'], fontsize=14)
plt.legend(loc='upper right', fontsize=14)
plt.title(' %s  T = %10.2e'%(fe25.IonStr, temperature[itemp]), fontsize=14)
plt.ylim(bottom=0.)
plt.xlim([0., wvl[-1]])
plt.tight_layout
```

produces



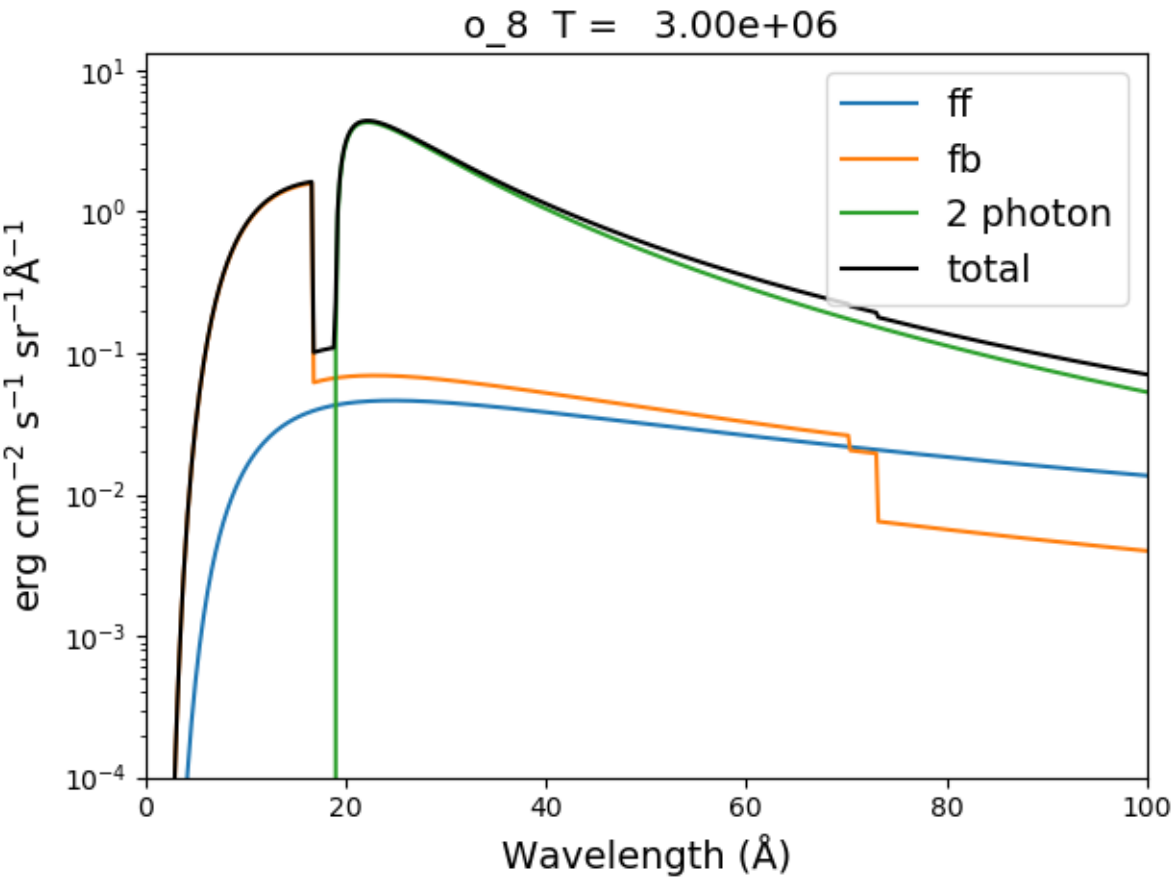

```
myIon = 'o_8'
```

```
temperature = [3.e+6, 6.e+6]
density = 1.e+9
em = [2.e+27,1.e+27]
wvl = 2. + 0.2*np.arange(701)
```

```
c = ch.continuum(myIon, temperature = temperature, em=em)
c.freeFree(wvl)
c.freeBound(wvl)
o8 = ch.ion(myIon, temperature, density, em=em)
o8.twoPhoton(wvl)
total = c.FreeFree['intensity'][itemp] + c.FreeBound['intensity'][itemp] + o8.TwoPhoton[
↪'intensity'][itemp]
```

```
itemp = 1
plt.figure()
plt.semilogy(wvl, c.FreeFree['intensity'][index],label='ff')
plt.semilogy(wvl, c.FreeBound['intensity'][index],label='fb')
plt.semilogy(wvl,o8.TwoPhoton['intensity'][index],label='2 photon')
plt.plot(wvl, total[itemp], 'k', label='total')
plt.ylim(bottom=1.e-4, top=1.)
plt.xlabel(c.FreeFree['xlabel'], fontsize=14)
plt.ylabel(c.FreeFree['ylabel'], fontsize=14)
plt.title(' %s  T = %10.2e'%(o8.IonStr, temperature[itemp]), fontsize=14)
plt.legend(loc='upper right', fontsize=14)
plt.tight_layout()
```

produces



In the continuum calculations, the specified ion, Fe XXV in this case, is the target ion for the free-free calculation. For

the free-bound calculation, specified ion is also the target ion. In this case, the radiative recombination spectrum of Fe XXV recombining to form Fe XXIV is returned.

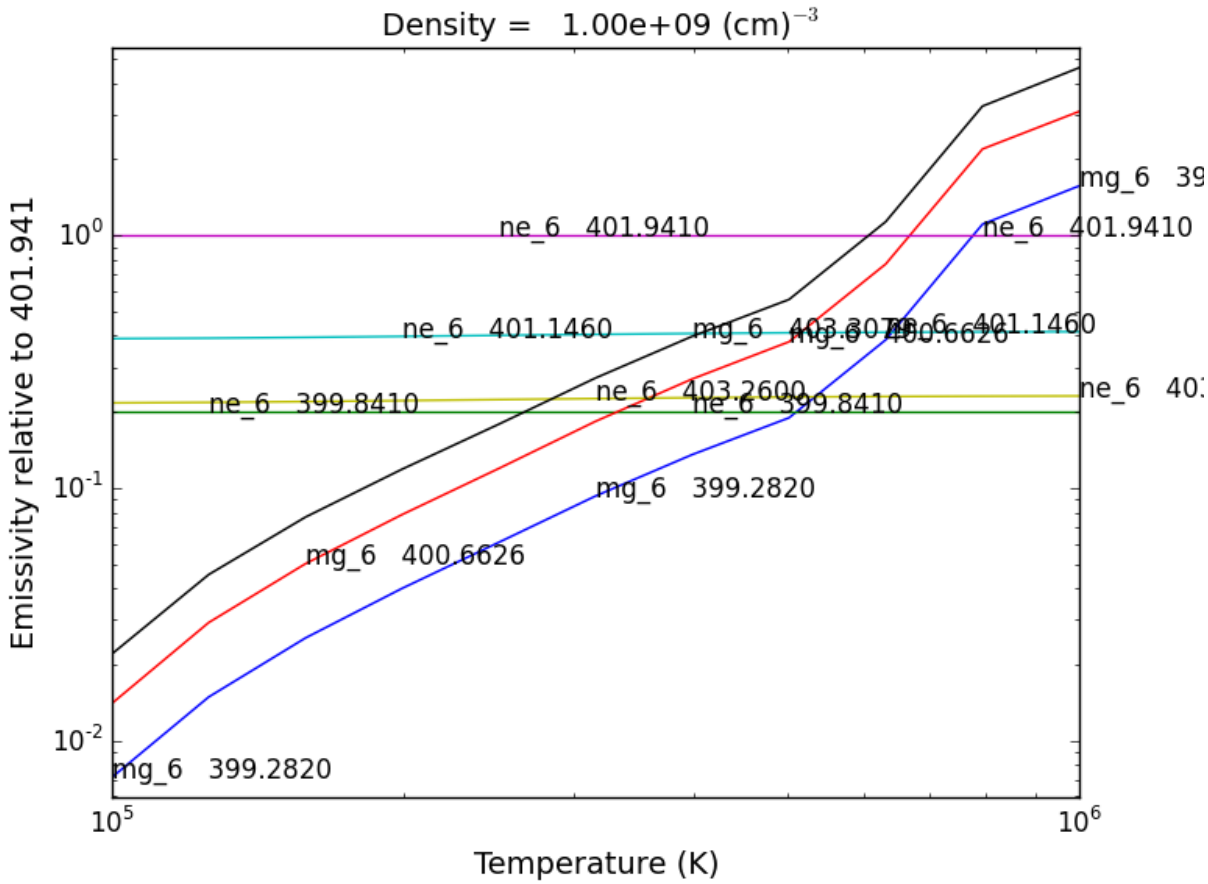
2.12 The multi-ion class Bunch

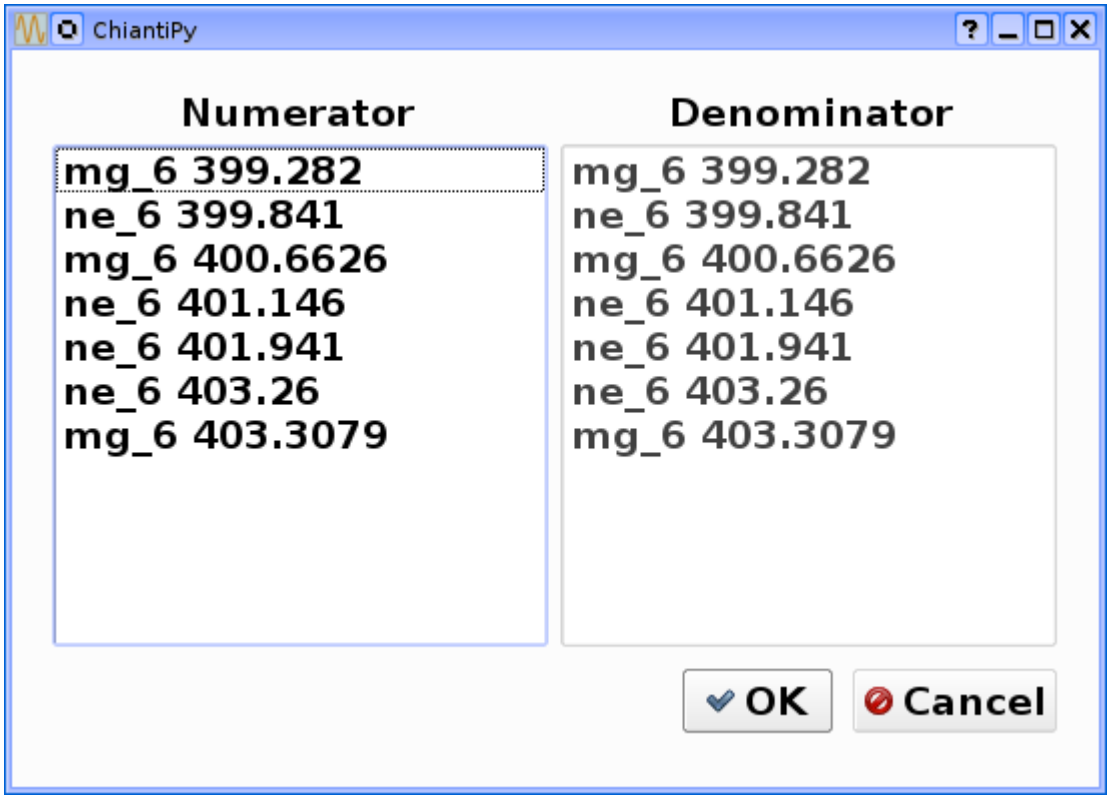
The multi-ion class **bunch** [new in v0.6] inherits a number of the same methods inherited by the ion class, for example *intensityList*, *intensityRatio*, and *intensityRatioSave*. As a short demonstration of its usefulness, Widing and Feldman (1989, ApJ, 344, 1046) used line ratios of Mg VI and Ne VI as diagnostics of elemental abundance variations in the solar atmosphere. For that to be accurate, it is necessary that the lines of the two ions have the same temperature response.

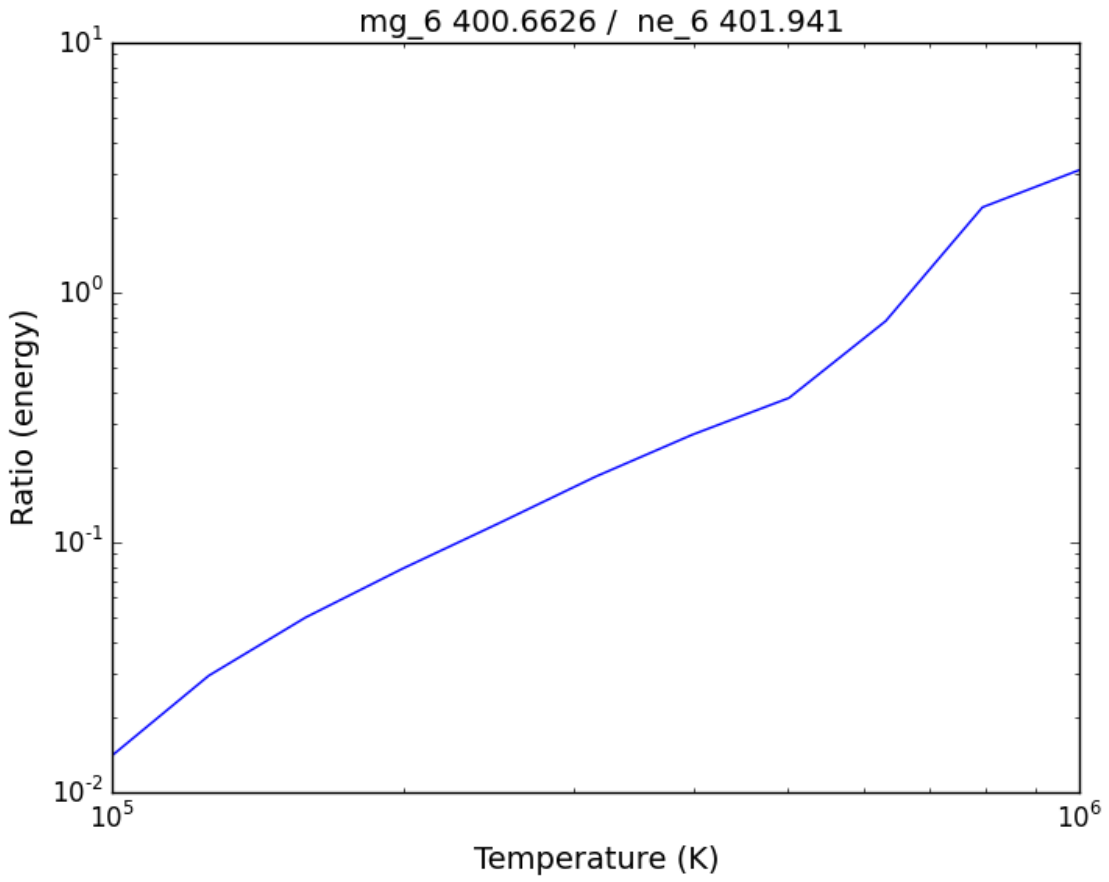
```
temp = 10.**(5.0+0.1*np.arange(11))
dens = 1.e+9
wvlRange = [wvl.min(),wvl.max()]
```

```
bnch=ch.bunch(temp, 1.e+9, wvlRange=wvlRange, ionList=['ne_6','mg_6'], abundance='unity',
↪ em=1.e+27)
bnch.intensityRatio(wvlRange=[395.,405.], top=7)
```

produces and initial plot of the selected lines, a selection widget and finally a plot of the ratio




ChiantiPy
Numerator
Denominator
mg_6 399.282
ne_6 399.841
mg_6 400.6626
ne_6 401.146
ne_6 401.941
ne_6 403.26
mg_6 403.3079
mg_6 399.282
ne_6 399.841
mg_6 400.6626
ne_6 401.146
ne_6 401.941
ne_6 403.26
mg_6 403.3079
OK
Cancel

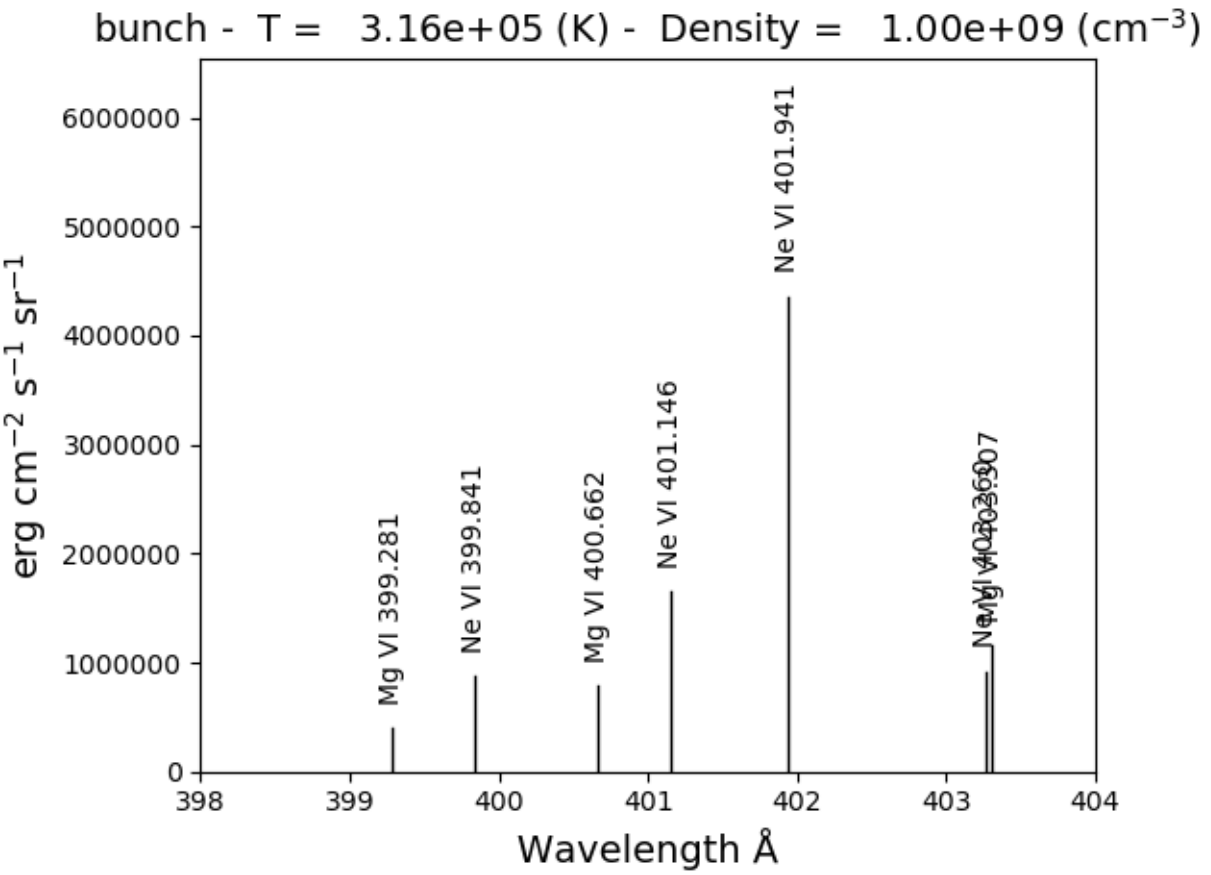


there seems to be a significant temperature dependence to the ratio, even though both are formed near 4.e+5 K.

The intensityPlot method can also be used with the bunch class

```
bnch.intensityPlot(index=5, wvlRange=[398., 404.])
```

results in



with version 0.13.0 it is possible to save multi-ion calculations as a pickle file with the saveData method

```
dataName = 'mybunch.pkl'
bnch.saveData(dataName, verbose=True)
```

the saveData method creates a dict of all of the attributes of the bnch instance. The pickle file can be loaded an it is possible to work directly with the data.

```
with open(dataName, 'rb') as inpt:
    mybnch = pickle.load(inpt)
```

```
mybnch.keys()
```

```
mybnch['Intensity']['intensity'].shape
```

with version 0.14.1, the redux class is introduced to allow the use of the pickled data inside a class that inherits such methods as intensityPlot and spectrumPlot

```
rebnch = ch.redux(dataName, verbose=False)
```

```
rebnch.intensityPlot(index=5, wvlRange=[398., 404.])
```

then returns the above plot

A new keyword argument **keepIons** has been added in v0.6 to the bunch and the 3 spectrum classes. It should be used with some care as it can lead to very large instances in the case of a large number of ions, temperature, or densities.

```
temp = 10.**(5.0+0.2*np.arange(6))
dens = 1.e+9
```

```
dwvl = 0.01
nwvl = (406.-394.)/dwvl
wvl = 394. + dwvl*np.arange(nwvl+1)
```

```
bnch2=ch.bunch(temp, 1.e+9, wvlRange=[wvl.min(),wvl.max()], elementList=['ne','mg'], \
  keepIons=1,em=1.e+27)
```

```
bnch2.convolve(wvl,filter=(chfilters.gaussian,5.*dwvl))
```

elapsed seconds = 11.000

```
for one in sorted(bnch2.IonInstances.keys()):
  print('%s'%(one))
```

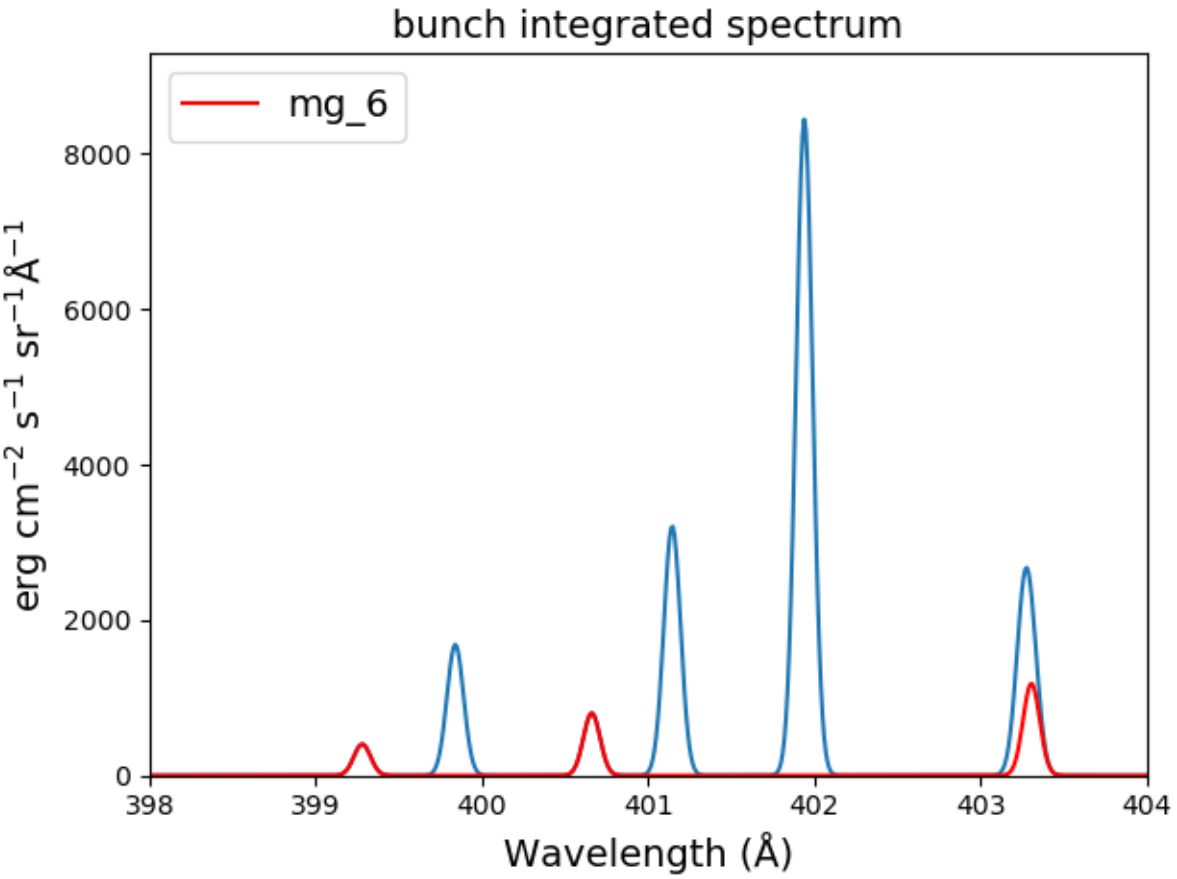
yields:

mg_10
mg_10d
mg_3
mg_4
mg_5
mg_6
mg_8
mg_9
ne_10
ne_2
ne_3
ne_5
ne_6
ne_8

these IonInstances have all the properties of the Ion class for each of these ions. However, this should be used with some caution as it can result in a memory-hogging instance.

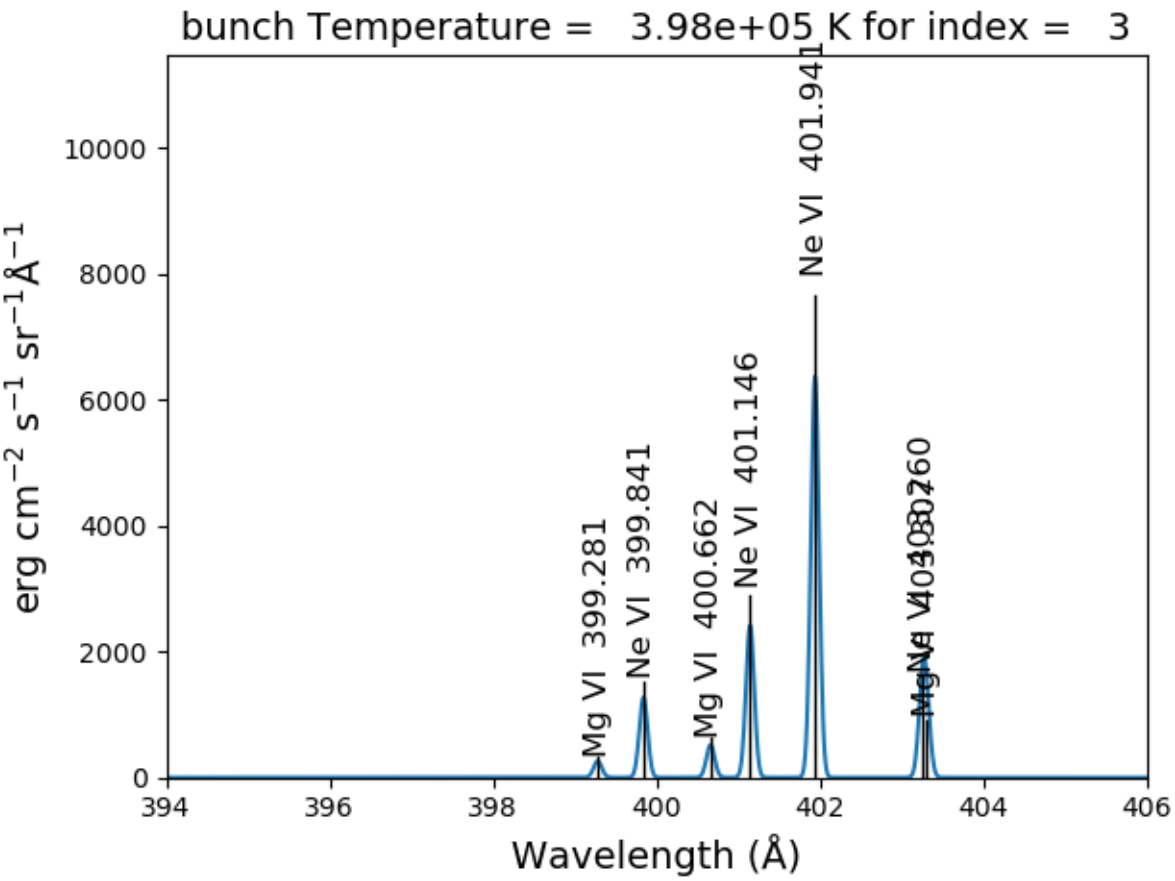
```
bnch2.spectrumPlot(integrated=True, doLabel=False)
plt.plot(wvl,bnch2.IonInstances['mg_6'].Spectrum['integrated'],'r',label='mg_6')
plt.xlim(left=398., right=404.)
plt.legend(loc='upper left', fontsize=14)
```

produces



The spectrumPlot method can also be used with bunch after convolve is run

```
bnch2.spectrumPlot(top=7)
```



2.13 Spectra of multiple ions and continuum

the spectrum for all ions in the CHIANTI database can also be calculated

The spectrum for a selection of all of the ions in the CHIANTI database can also be calculated. There are 3 spectral classes.

- **spectrum** - the single processor implementation that can be used anywhere
- **mspectrum** - uses the Python multiprocessing class and cannot be used in a IPython qtconsole or notebook
- **ipymspectrum** [new in v0.6] - uses the IPython parallel class and can be used in a IPython qtconsole or notebook

As of version 0.13.0, it is now possible to save the calculations with the **saveData** methods, demonstrated with the bunch class above

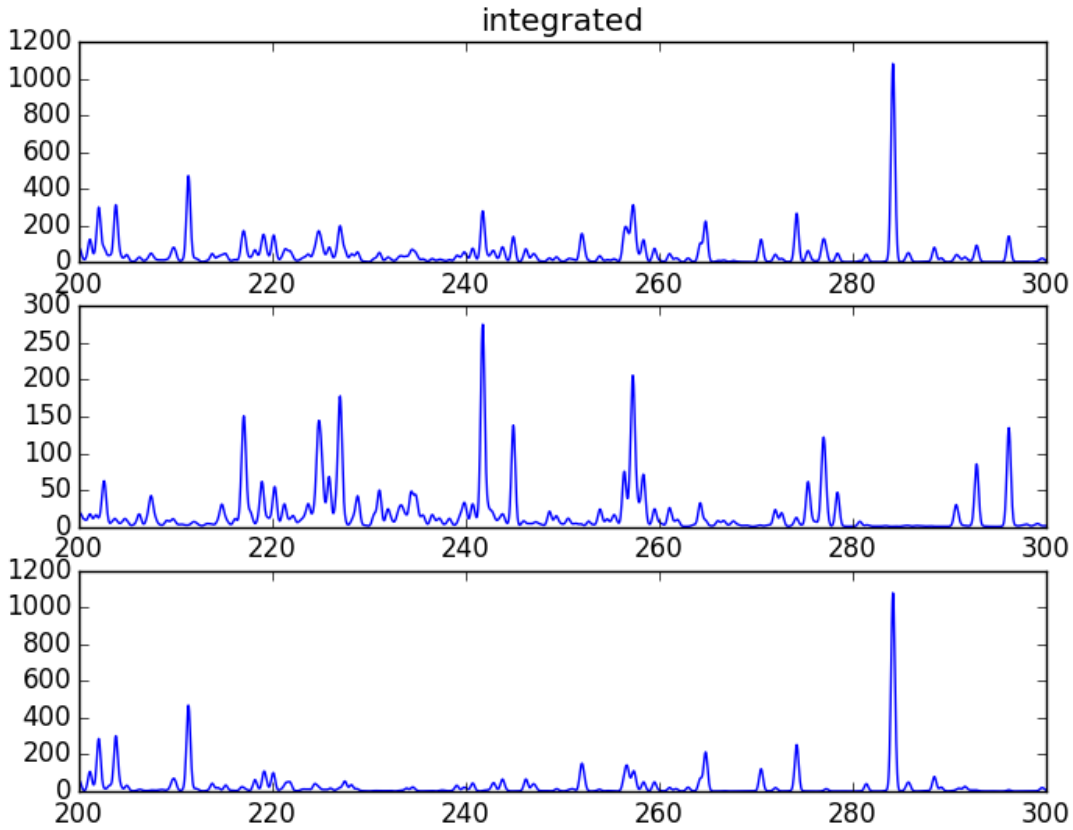
2.13.1 The single processor spectrum class

```
temperature = [1.e+6, 2.e+6]
density = 1.e+9
wvl = 200. + 0.05*arange(2001)
emeasure = [1.e+27 ,1.e+27]
```

```
s = ch.spectrum(temperature, density, wvl, filter = (chfilters.gaussian,.2), em =␣
↪emeasure, doContinuum=0, minAbund=1.e-5)
```

```
subplot(311)
plot(wvl, s.Spectrum['integrated'])
subplot(312)
plot(wvl, s.Spectrum['intensity'][0])
subplot(313)
plot(wvl, s.Spectrum['intensity'][1])
```

produces



The integrated spectrum is formed by summing the spectra for all temperatures.

- For minAbund=1.e-6, the calculatation takes 209 s on a 3.5 GHz processor.
- For minAbund=1.e-5, the calculatation takes 122 s on a 3.5 GHz processor.

The filter is not applied to the continuum.

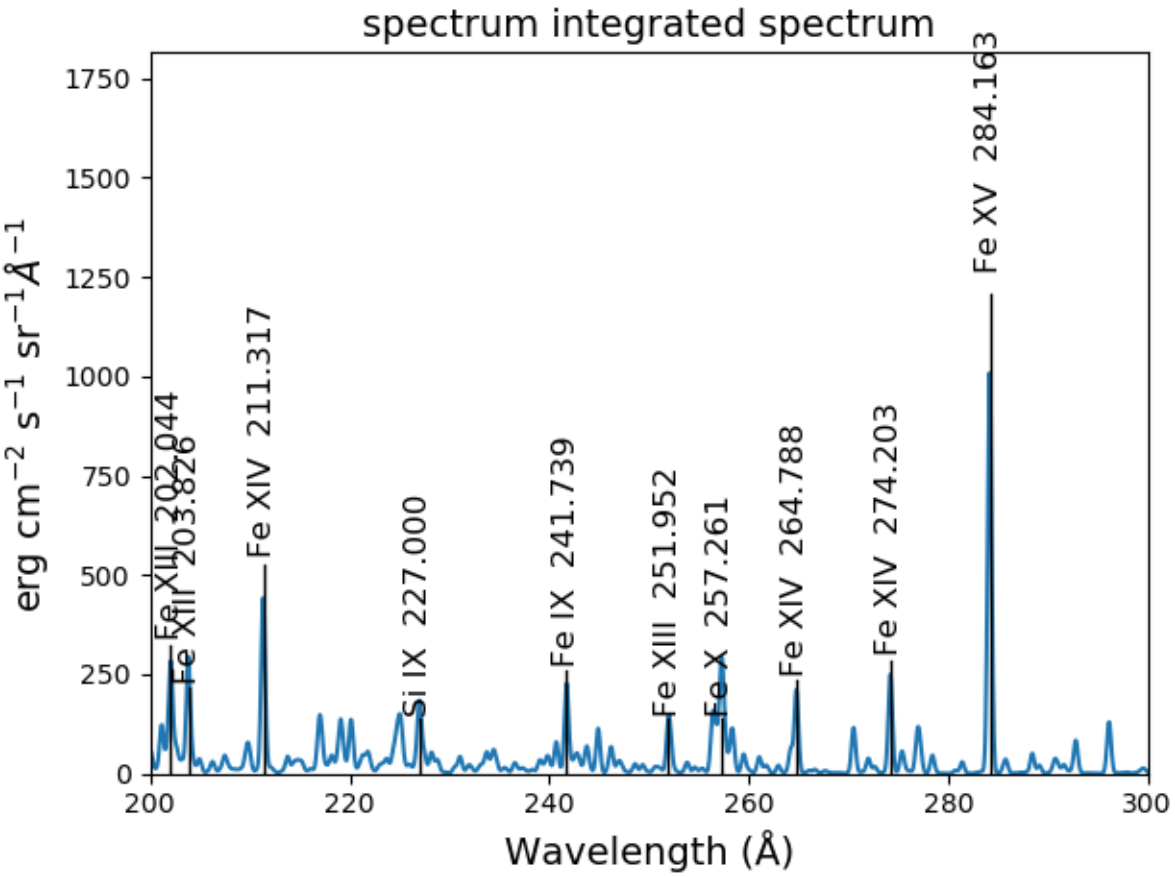
Save the calculations

```
saveName = 'spectrum.pkl'
s.saveData(saveName, verbose=True)
```

The spectrumPlot method is also available

```
s.spectrumPlot(integrated=True)
```

yields



One can return to the saved data at a later date and reload it with the redux class

```
rdx = ch.redux(saveName)
```

The inherited spectrumPlot is again available

```
rdx.spectrumPlot(index=1)
```

produces a figure like above

Calculations with the Spectrum module can be time consuming. One way to control the length of time the calculations take is to limit the number of ions with the ionList keyword and to avoid the continuum calculations by setting the doContinuum keyword to 0 or False. Another way to control the length of time the calculations take is with the

minAbund keyword. It sets the minimum elemental abundance that an element can have for its spectra to be calculated. The default value is set include all elements. Some usefull values of minAbund are:

- minAbund = 1.e-4, will include H, He, C, O, Ne
- minAbund = 2.e-5 adds N, Mg, Si, S, Fe
- minAbund = 1.e-6 adds Na, Al, Ar, Ca, Ni

2.13.2 The multiple processor mspectrum class

Another way to speed up calculations is to use the *mspectrum* class which uses multiple cores on your local computer. It requires the Python *multiprocessing* module which is available with Python versions 2.6 and later. *mspectrum* is called in the same way as *spectrum* but you can specify the number of cores with the *proc* keyword. The default is 3 but it will not use more cores than are available on your machine. For example,

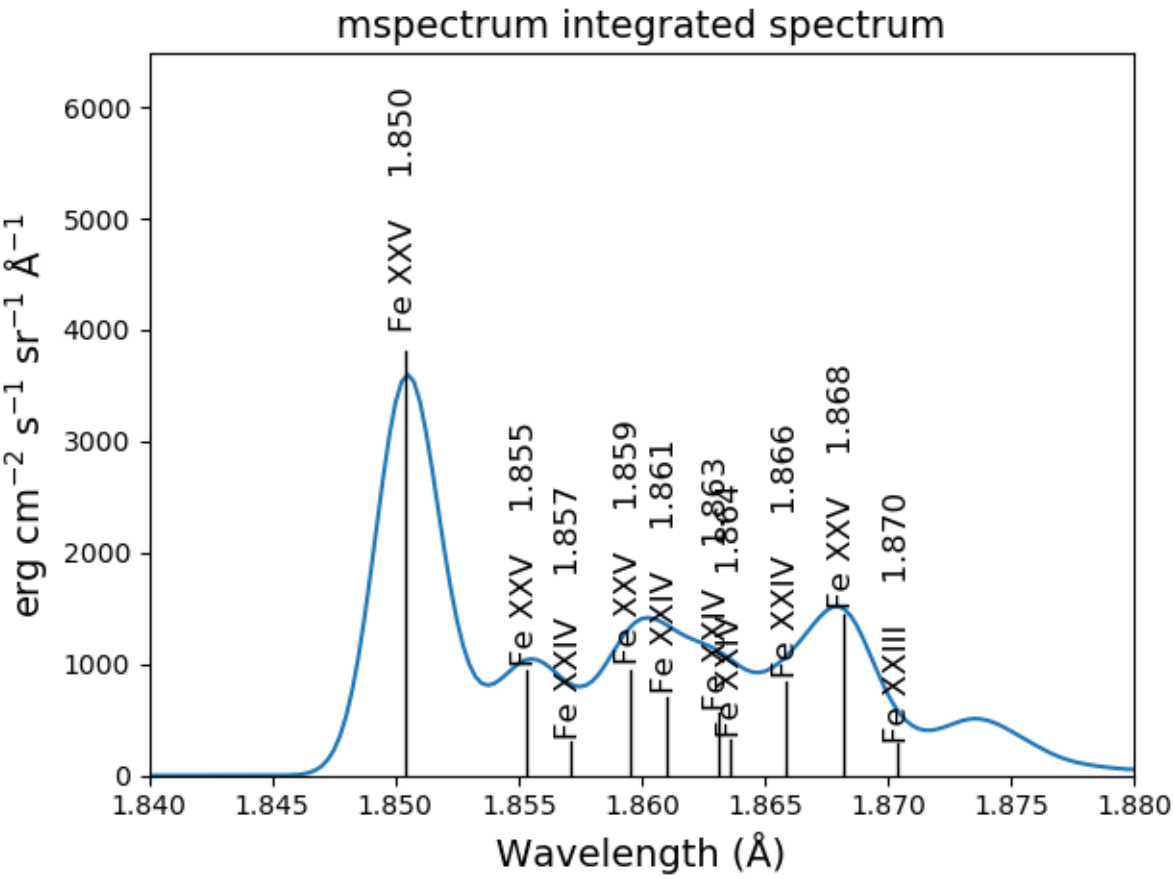
```
temp = [1.e+7, 2.e+7, 3.e+7]
dens = 1.e+9
wvl = np.linspace(1.5, 4., 10001)
emeasure = 1.e+27
core=6
```

```
dwvl = wvl[1] - wvl[0]
' dwvl:  %8.4f'%(dwvl)
```

```
sm = ch.mspectrum(temperature, density ,wvl, em=emeasure, filter = (chfilters.gaussian,␣
↪5.*dwvl), proc=core)
```

```
sm.spectrumPlot(wvlRange=[1.84, 1.88], index=2)
```

yields



another example

2.14 Using differential emission measures (DEM)

Beginning with CHIANTI version 14.1, the io.demRead function has been added to read dem file in the existing XUVTOP/dem directory

```
demDir = os.path.join(os.environ['XUVTOP'], 'dem')
demList = os.listdir(demDir)
```

```
for idx, demFile in enumerate(demList):
    print('%i  %s'%(idx, demFile))
```

produces

0 quiet_sun_eis.dem
1 version_3
2 coronal_hole.dem
3 flare.dem
4 flare_ext.dem
5 AU_Mic.dem

6 quiet_sun.dem
7 active_region.dem
8 prominence.dem

select the desired file by index

```
flDict = chio.demRead(demList[3])
```

```
flDict.keys()
```

dict_keys([‘temperature’, ‘density’, ‘dem’, ‘em’, ‘dt’, ‘ref’, ‘filename’])

since we will be looking at X-ray wavelengths, select only the highest temperatures

```
flTemp = flDict['temperature'][20:]
flDens = flDict['density'][20:]
flEm = flDict['em'][20:]
```

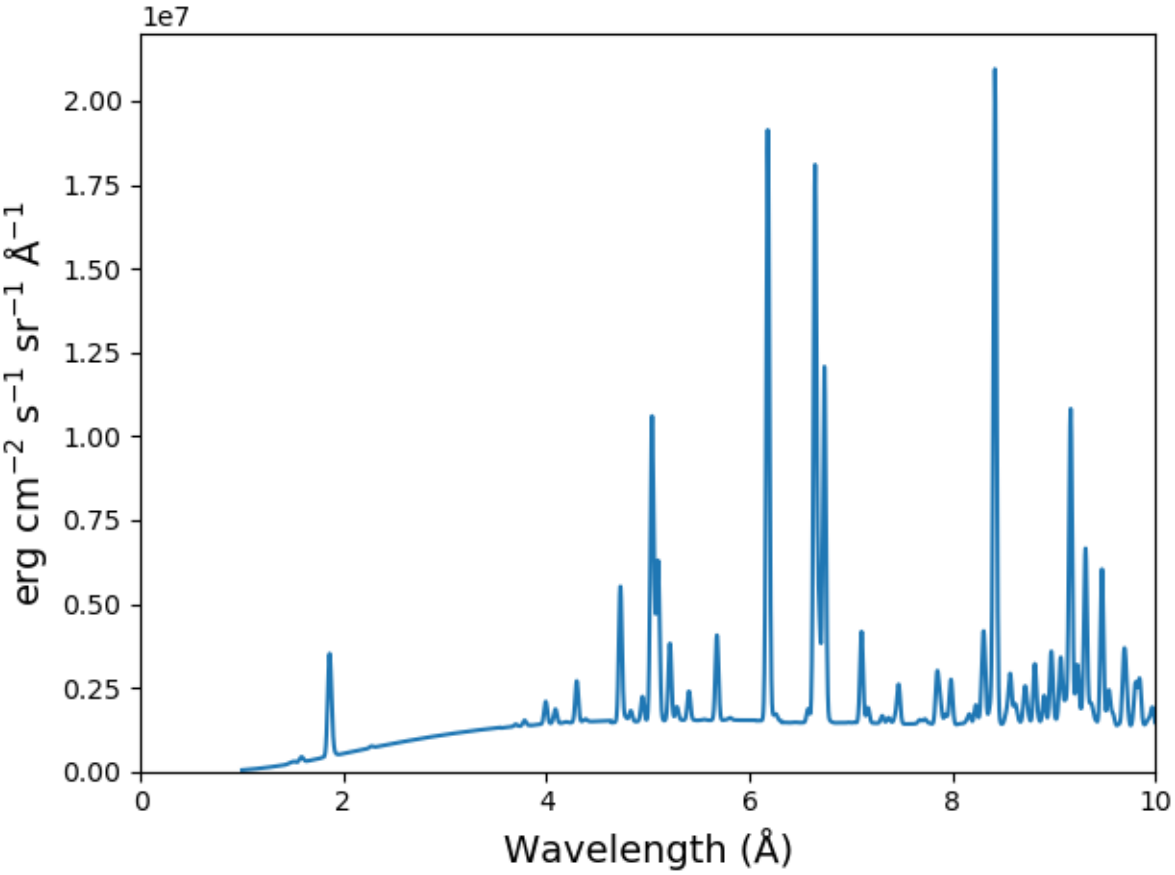
```
wvl = 1. + 0.002*np.arange(4501)
core = 6
```

```
s3 = ch.mspectrum(flTemp, flDens, wvl, filter = (chfilters.gaussian,.015), em=flEm,␣
↪minAbund=1.e-5, proc=core, verbose=0)
```

2.14.1 save the calculations

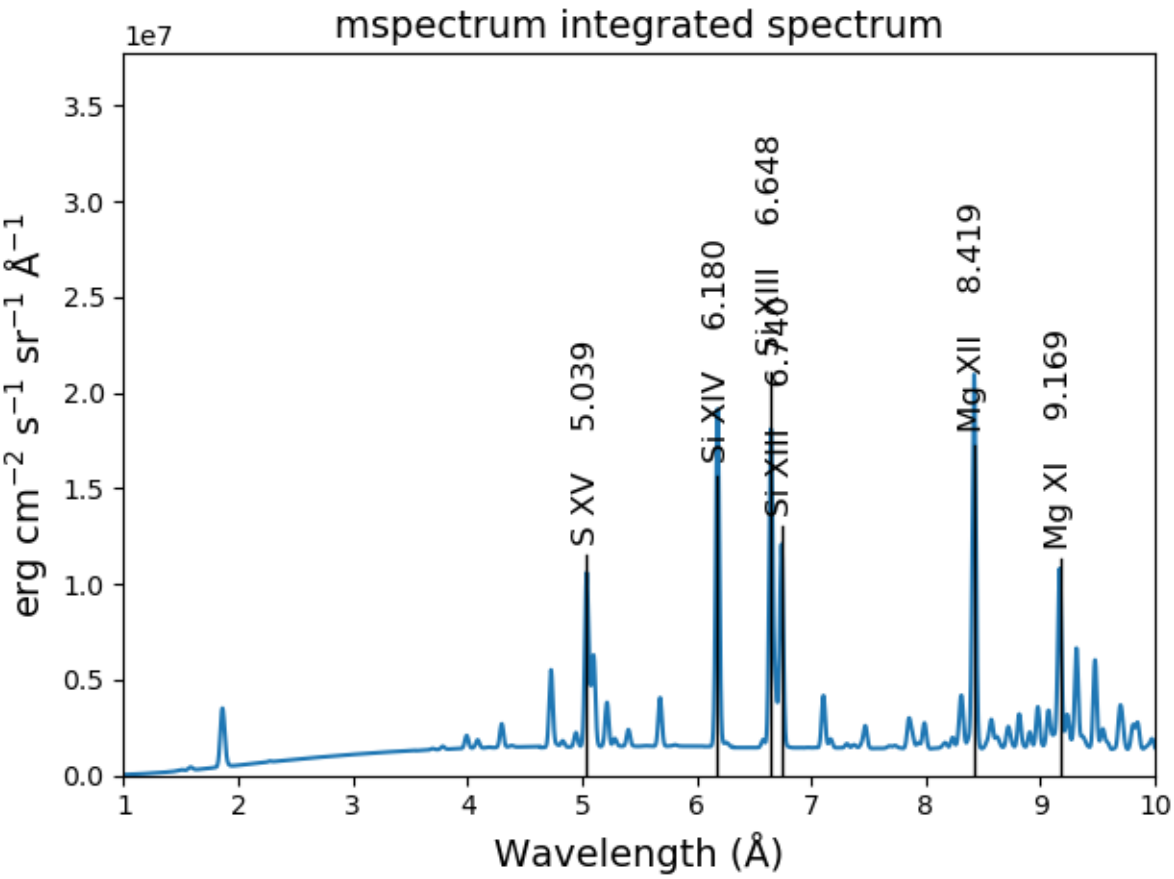
```
saveName = 'mspectrum3_dem.pkl'
s3.saveData(saveName)
```

```
plt.figure()
plt.plot(wvl, s3.Spectrum['intensity'].sum(axis=0))
plt.xlabel(s3.Spectrum['xlabel'], fontsize=14)
plt.ylabel(s3.Spectrum['ylabel'], fontsize=14)
plt.ylim(bottom = 0.)
plt.xlim([0., wvl[-1]])
plt.tight_layout()
```



The spectrumPlot method can also be used

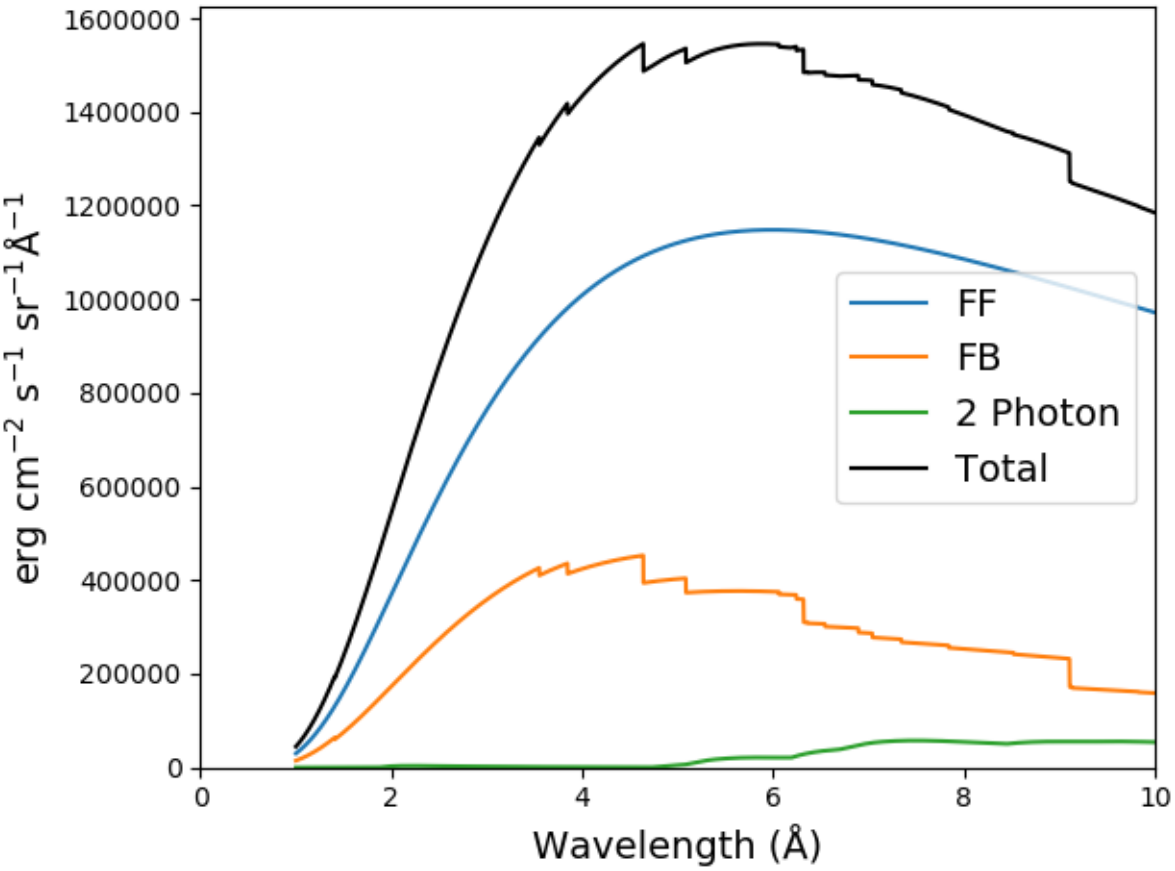
```
s3.spectrumPlot(top=6)
```



the default value for doContinuum is True, so, the continuum can be plotted separately

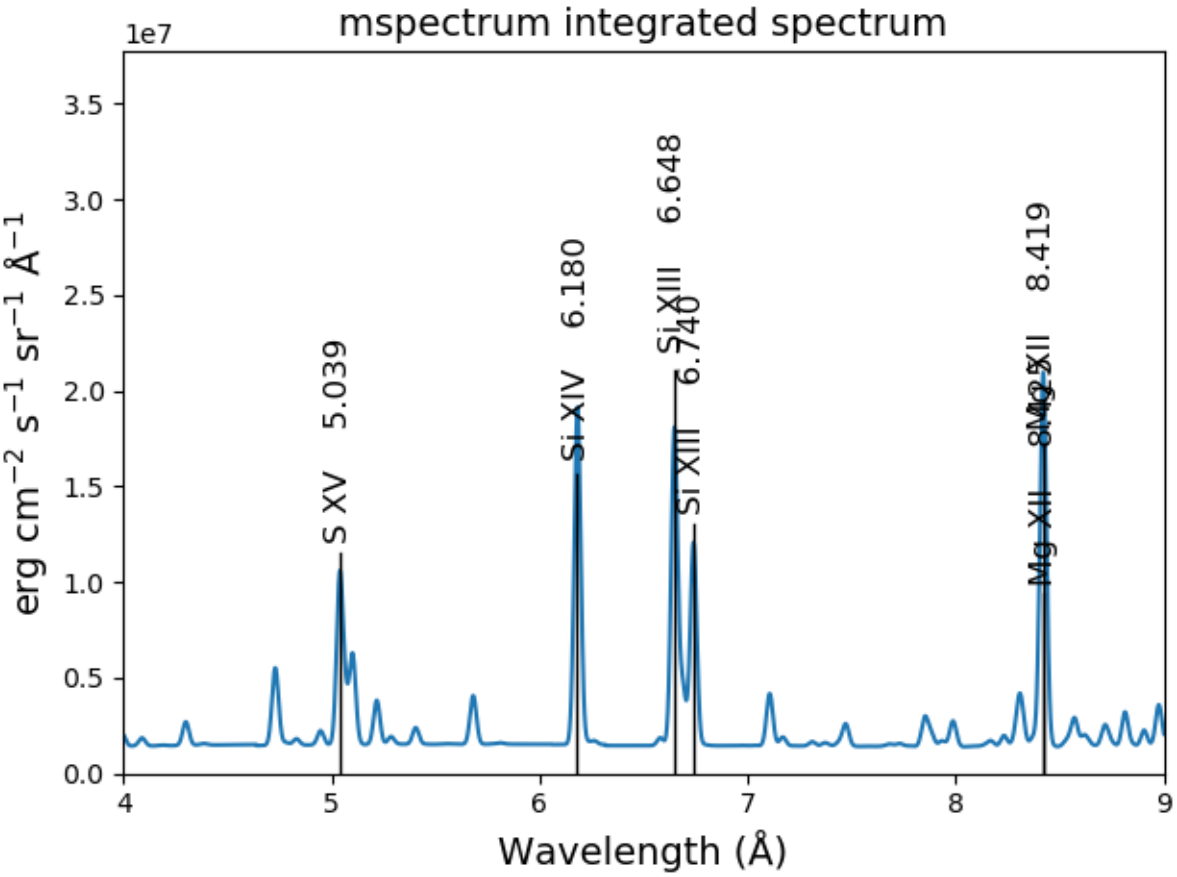
```
plt.figure()
plt.plot(wvl, s3.FreeFree['intensity'], label='FF')
plt.plot(wvl, s3.FreeBound['intensity'], label='FB')
plt.plot(wvl, s3.TwoPhoton['intensity'], label='2 Photon')
plt.plot(wvl, s3.Continuum['intensity'].sum(axis=0), 'k', label='Total')
plt.xlabel(s3.Spectrum['xlabel'], fontsize=14)
plt.ylabel(s3.Spectrum['ylabel'], fontsize=14)
plt.ylim(bottom = 0.)
plt.xlim([0., wvl[-1]])
plt.legend(loc='upper right', fontsize=14)
plt.tight_layout()
```

produces




```
s3.spectrumPlot(wvlRange=[4., 9.], top=6, integrated=True)
```

produces

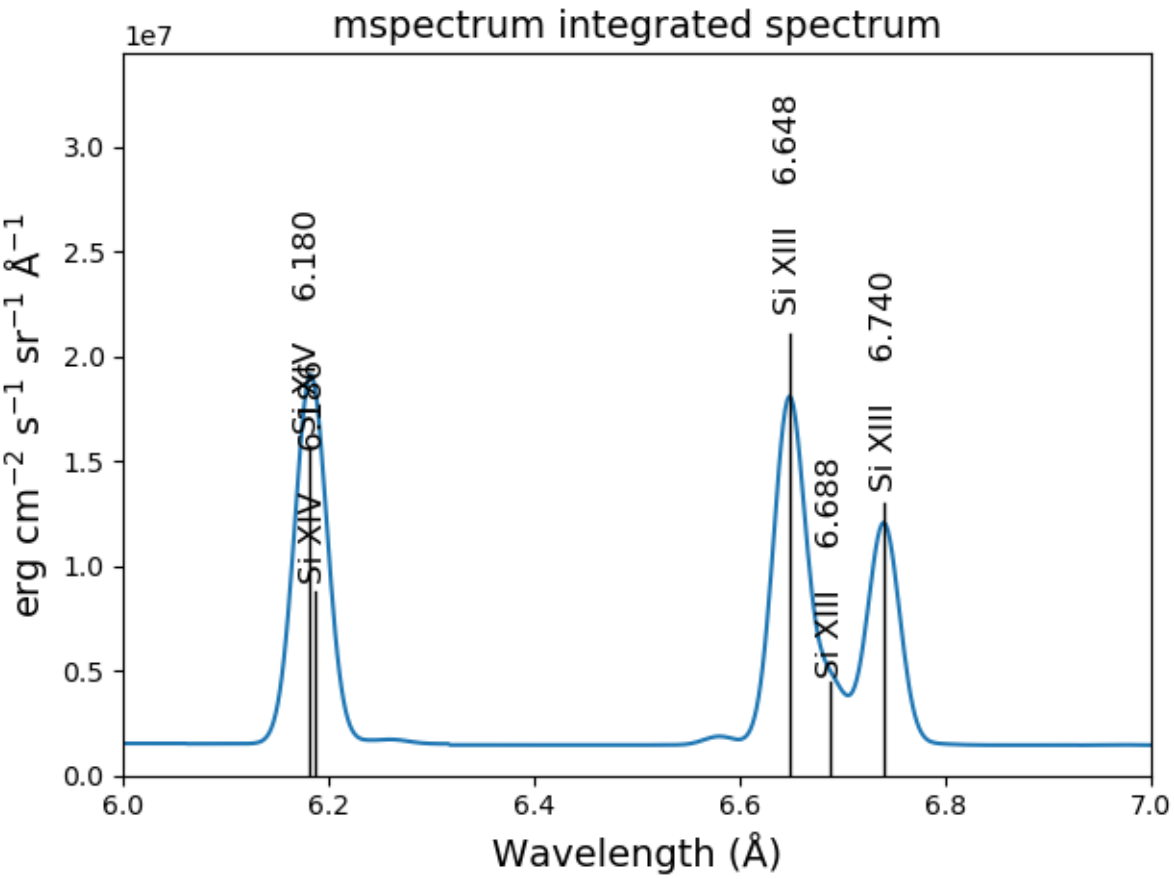


2.14.2 With the redux class, the save calculations can be restored

```
s3r = ch.redux(saveName, verbose=True)
```

the redux class inherits the intensityPlot and spectrumPlot methods as well as a few others

```
s3r.spectrumPlot(wvlRange=[6., 7.], integrated=True, top=5)
```



2.14.3 The multiple processor ipymspectrum class

next, we will use the ipymspectrum class. First, it is necessary to start up the cluster. In some shell

> ipcluster start –n=4

or, if you are using Python3

> ipcluster3 start –n=4

this will start 4 engines if you have 4 cores but it won't start more than you have

then in an IPython notebook or qtconsole

```
temp = [1.e+6, 2.e+6]
dens = 1.e+9
wvl = 200. + 0.05*np.arange(2001)
emeasure = [1.e+27 ,1.e+27]
```

```
s = ch.ipymspectrum(temp, dens, wvl, filter = (chfilters.gaussian,.2), \
  em = emeasure, doContinuum=1, minAbund=1.e-5, verbose=True)
```

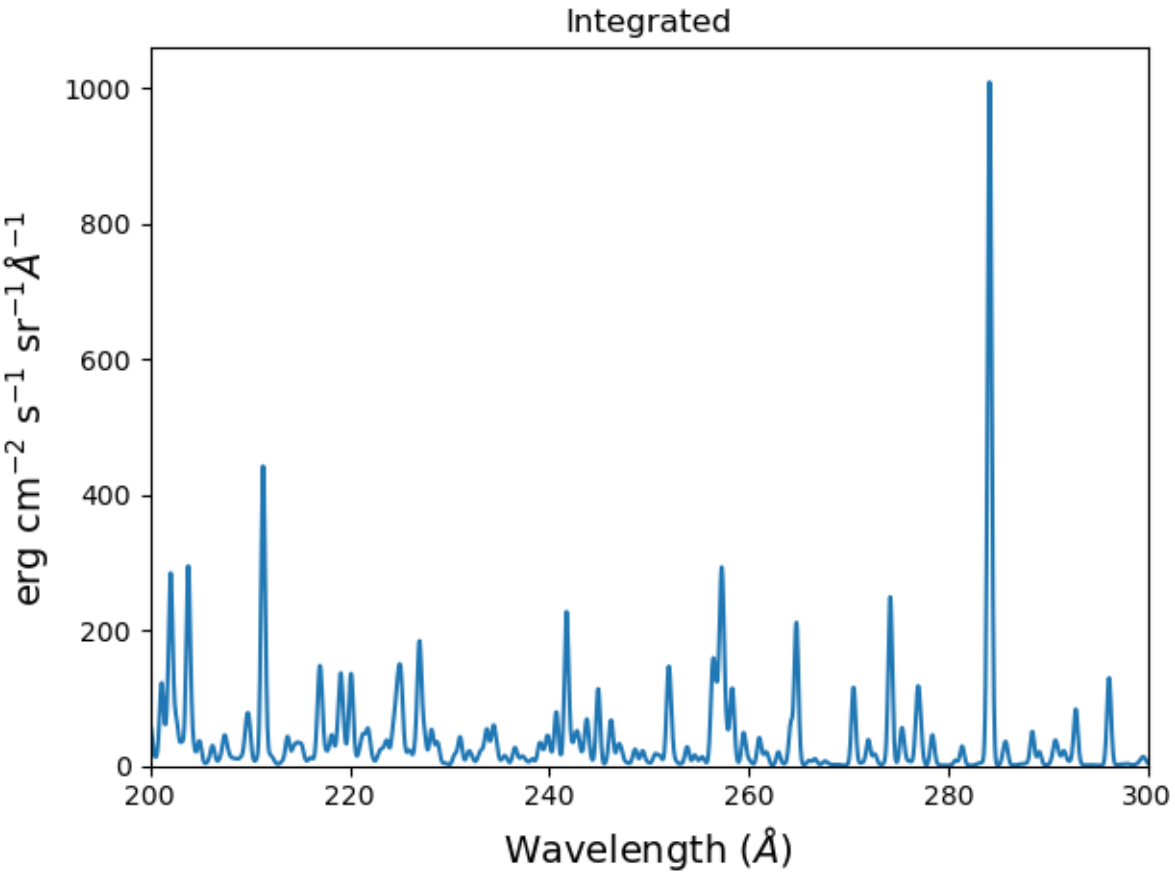
```
plt.figure()
plt.plot(wvl, s.Spectrum['integrated'])
```

(continues on next page)

(continued from previous page)

```
plt.ylim(bottom=0.)
plt.xlim([wvl[0], wvl[-1]])
plt.title('Integrated')
plt.xlabel(s.Xlabel, fontsize=14)
plt.ylabel(s.Ylabel, fontsize=14)
plt.tight_layout()
```

produces



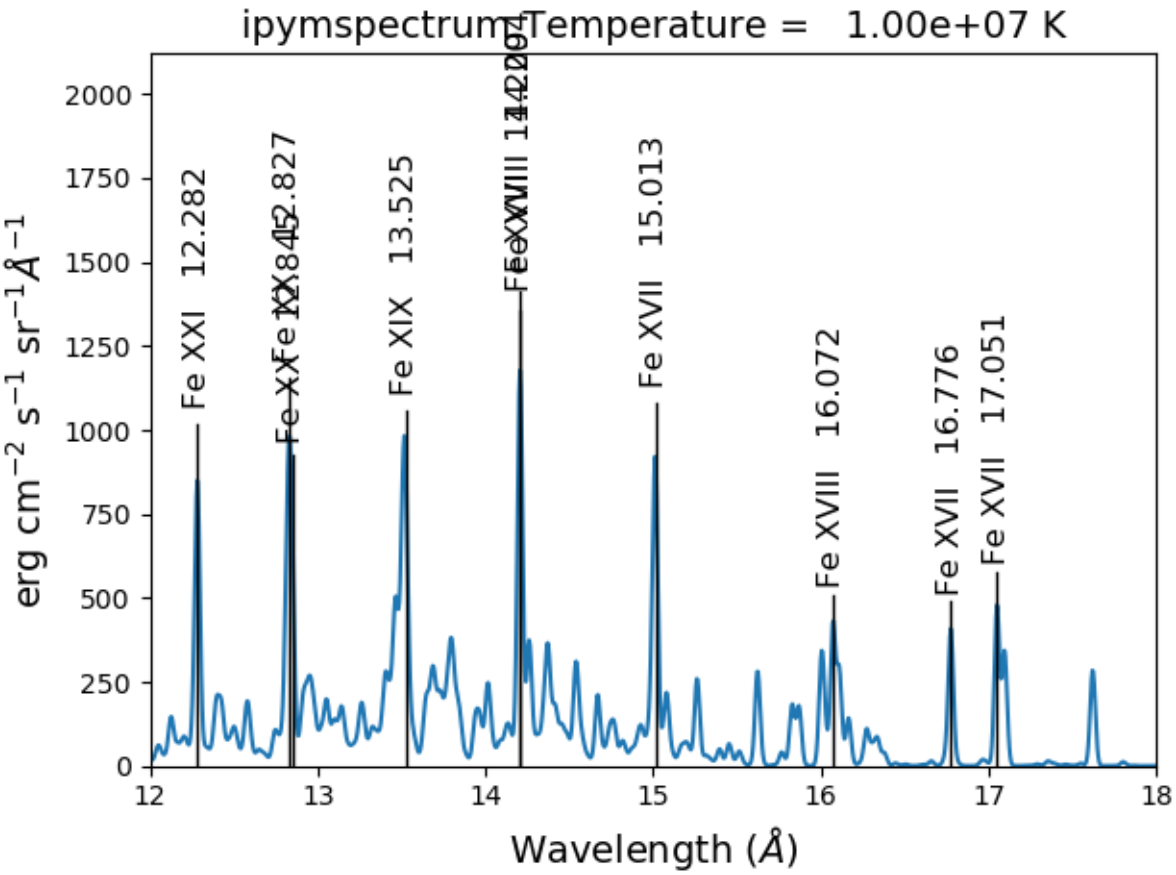
spectrum, mspectrum and ipymspectrum can all be instantiated with the same arguments and keyword arguments. Most of the examples below use the ipymspectrum class for speed.

```
temperature = 1.e+7
dens = 1.e+9
wvl = 10. + 0.005*np.arange(2001)
```

```
s = ch.ipymspectrum(temp, dens, wvl, filter = (chfilters.gaussian,.015), \
  elementList=['fe'])
```

```
s.spectrumPlot()
```

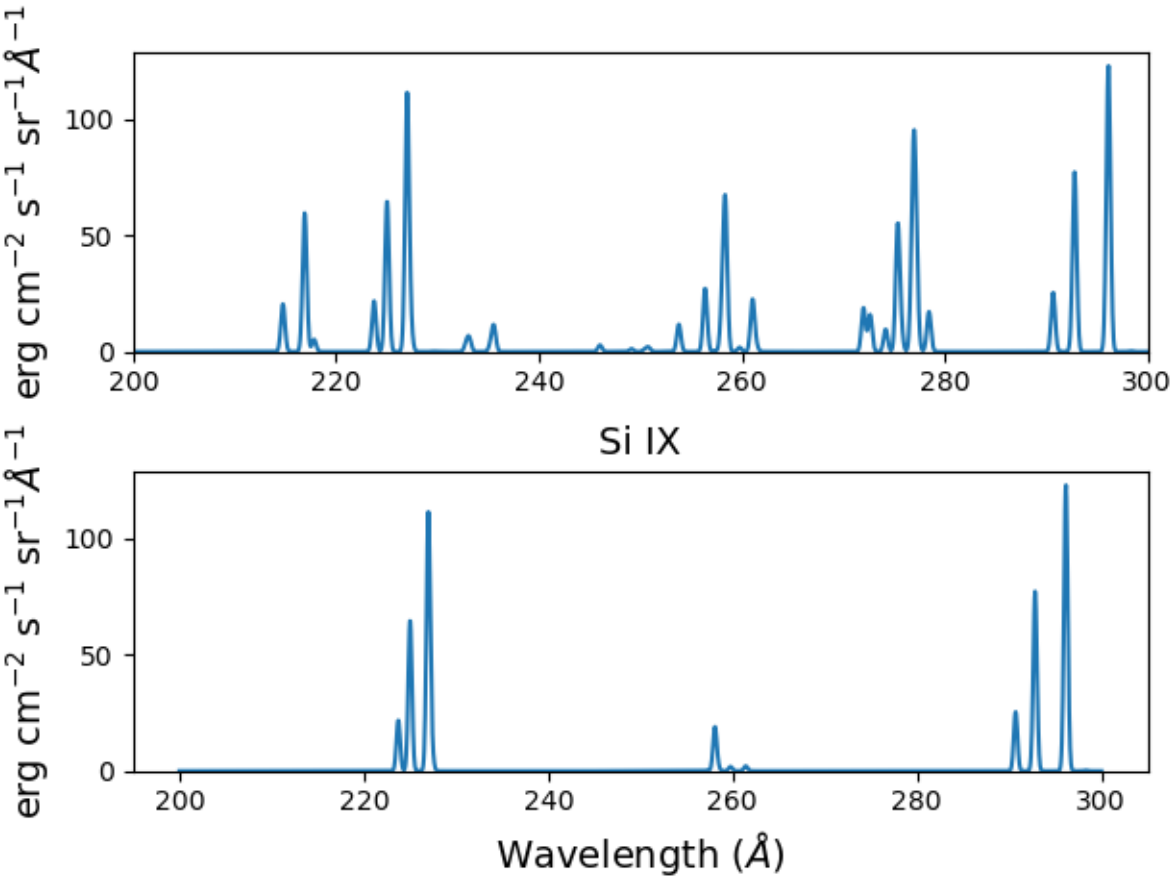
produces



It is also possible to specify a selection of ions by means of the *ionList* keyword, for example, *ionList=['fe_11','fe_12','fe_13']*

```
s2 = ch.ipymspectrum(temp, dens, wvl, filter = (chfilters.gaussian,.2), \
  em = emeasure, doContinuum=0, keepIons=1, elementList=['si'])
```

```
fig, [ax1, ax2] = plt.subplots(2,1)
ax1.plot(wvl,s2.Spectrum['intensity'][0])
ax1.set_ylim(bottom=0.)
ax1.set_xlim([wvl[0], wvl[-1]])
ax1.set_ylabel(r'erg cm$^{-2}$ s$^{-1}$ sr$^{-1} \AA^{-1}$', fontsize=14)
ax2.plot(wvl,s2.IonInstances['si_9'].Spectrum['intensity'][0])
ax2.set_ylim(bottom=0.)
ax1.set_xlim([wvl[0], wvl[-1]])
ax2.set_ylabel(r'erg cm$^{-2}$ s$^{-1}$ sr$^{-1} \AA^{-1}$', fontsize=14)
ax2.set_xlabel(r'Wavelength ($\AA$)', fontsize=14)
ax2.set_title('Si IX', fontsize=14)
fig.tight_layout()
```

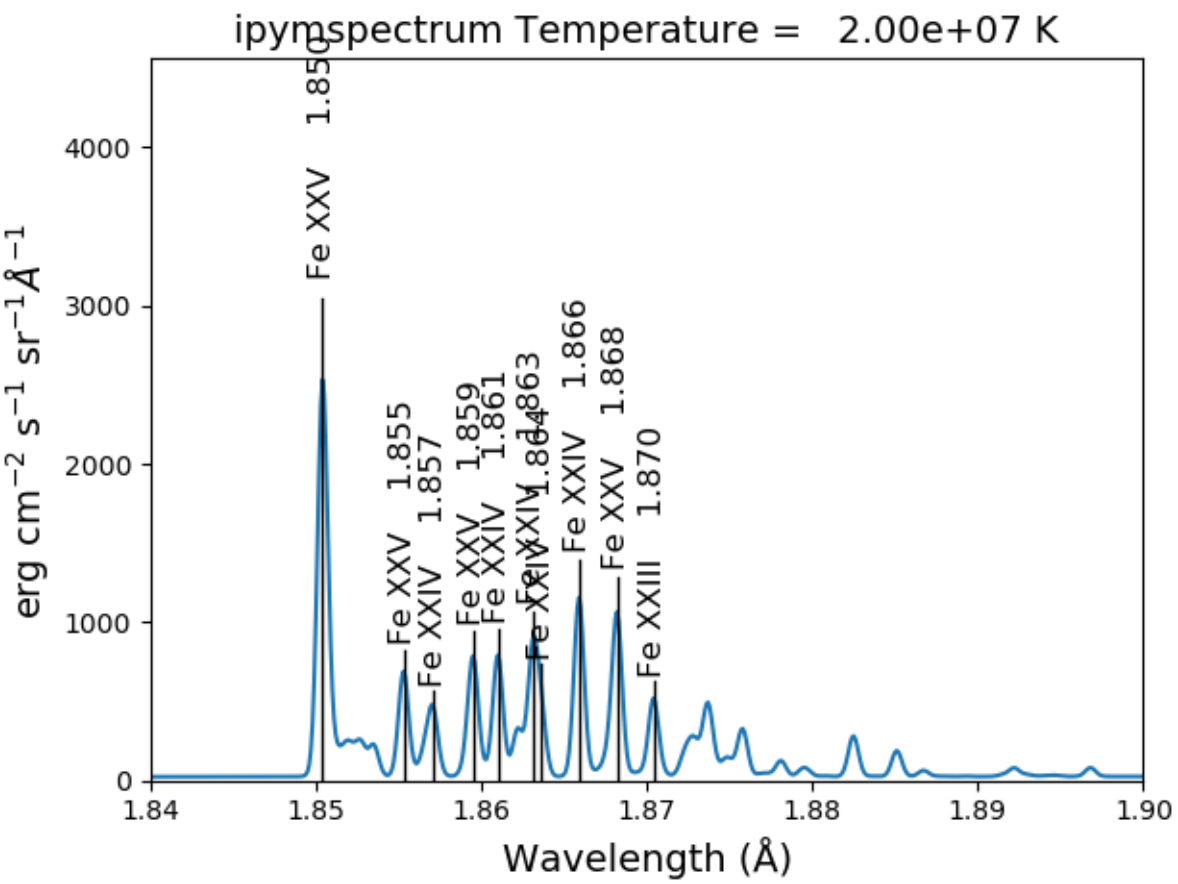


Because **keepIons** has been set, the ion instances of all of the ions are maintained in the s2.IonInstances dictionary. It has been possible to compare the spectrum of all of the ions with the spectrum of a single ion. It should be used with some care as it can lead to very large instances in the case of a large number of ions, temperature, or densities.

```
temperature = 2.e+7
density = 1.e+9
em = 1.e+27
wvl = 1.84 + 0.0001*arange(601)
s4 = ch.ipymspectrum(temperature, density ,wvl, filter = (chfilters.gaussian,.0003), \
  doContinuum=1, minAbund=1.e-5, em=em, verbose=0)
```

```
s4.spectrumPlot()
```

produces



There are two demo notebooks, spectrum_demo.ipynb and spectrum_demo_2.ipynb in the jupyter_notebooks directory on github.

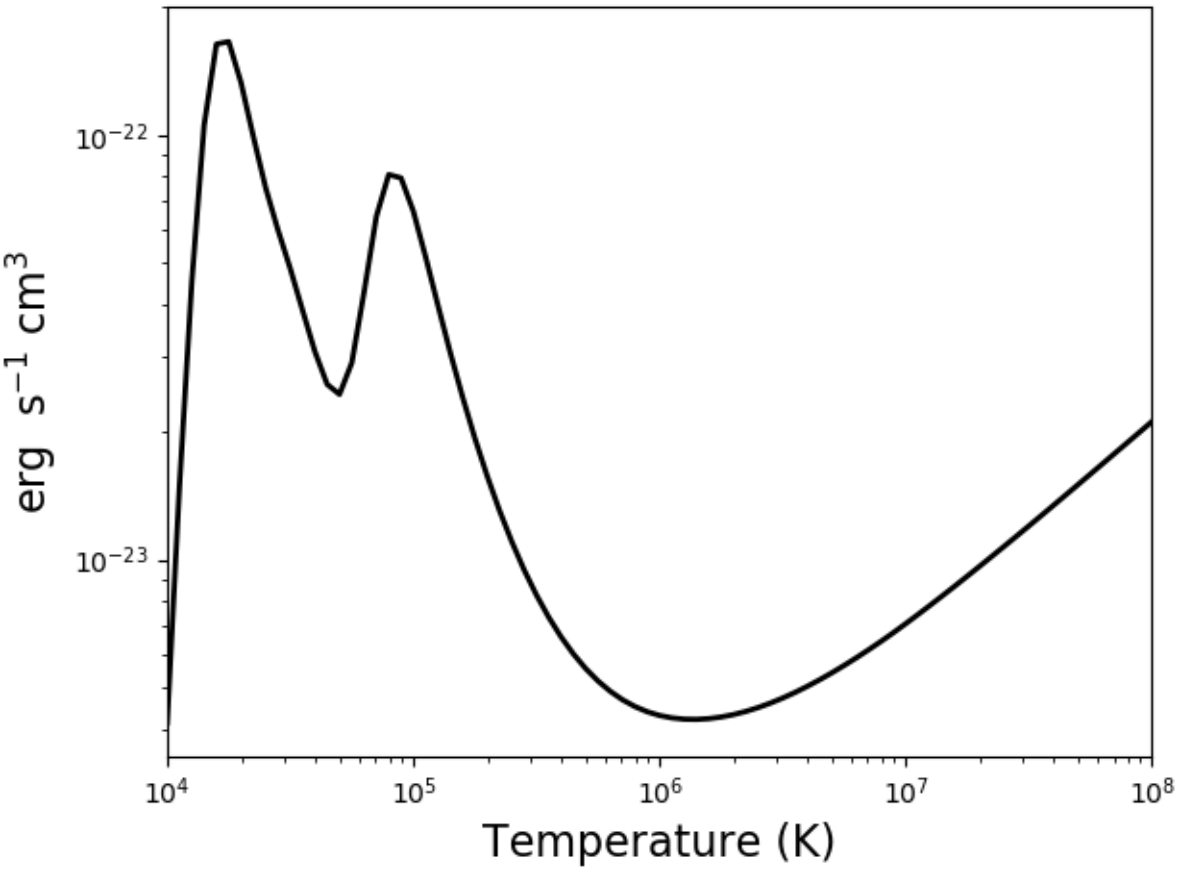
2.15 Radiative loss rate

the radiative loss rate can be calculated as a function of temperature and density. If all elements are included, the calculation can take some time. So, for a shorter example:

```
temp = 10.**(4.+0.05*np.arange(81))
dens = 1.e+9
rlhhe = ch.radLoss(temp, dens, elementList=['h', 'he'])
```

```
plt.figure()
rl.radLossPlot()
```

produces, in 2s:



with version 0.15.0, the class mradLoss is available for doing a multiprocessor calculation of the radiation loss

```
temp = 10.**(4.+0.05*np.arange(81))
dens = 1.e+9
```

the following will calculate the radiation loss for elements with an abundance greater the 1.e-5 that of the hydrogen abundance. In this case the default abundance file is for photospheric abundances

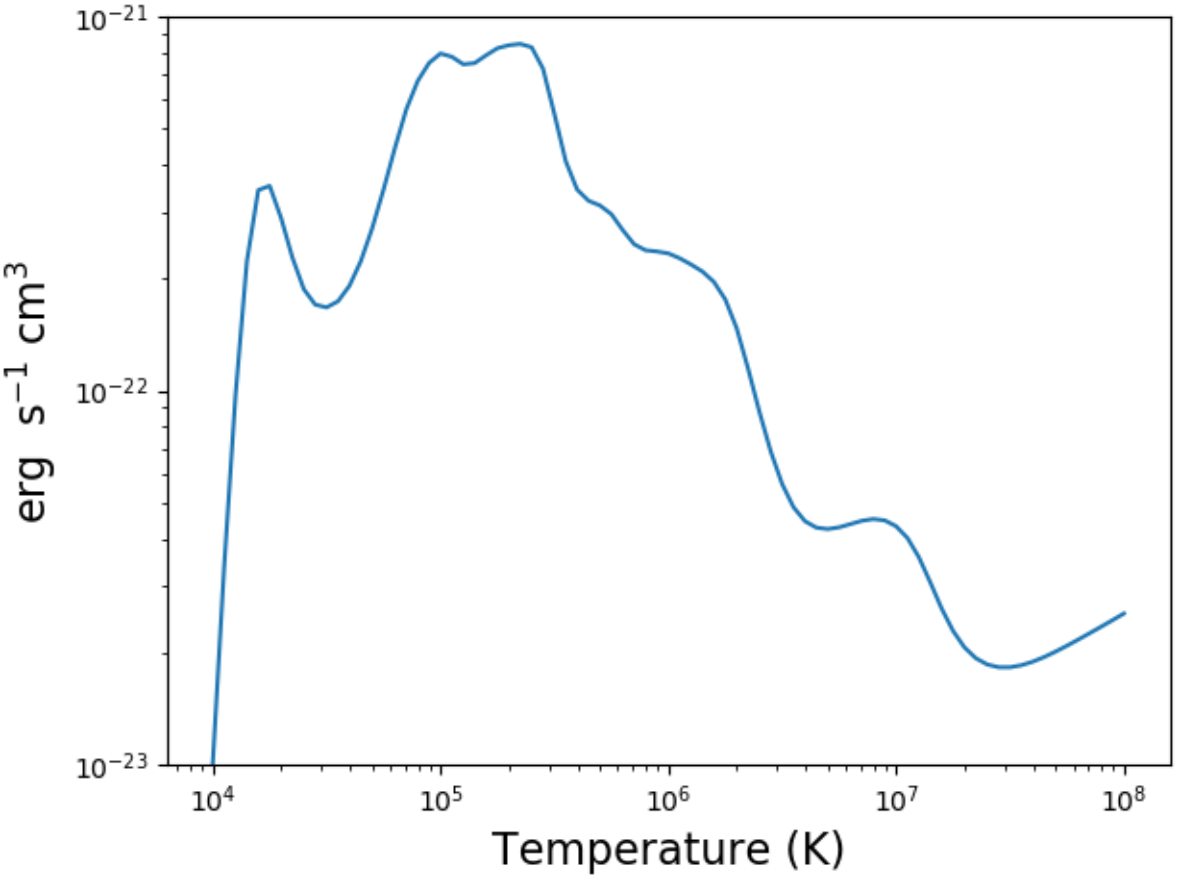
```
mrl = ch.mradLoss(temp, dens, minAbund=1.e-5)
```

these calculations can take some time so it is a good idea to save them

```
mrl.saveData('rl_phot_1m5.pkl')
```

```
plt.figure()
mrl.radLossPlot()
```

produces, produces after 250s on a 3.5 GHz 4 core processor:



the radiative losses are kept in the rl.RadLoss dictionary

the **abundance** keyword argument can be set to the name of an available abundance file in XUVTOP/abund

if abundance='abc', or some name that does not match an abundance name, a dialog will come up so that a abundance file can be selected

or:

```
abundDir = os.path.join(os.environ['XUVTOP'], 'abundance')
```

```
abundList = os.listdir(abundDir)
```

```
for idx, aname in enumerate(abundList):
  print('%5i  %s'%(idx, aname))
```

to select photospheric abundances

```
myAbund = abundList[4]
```

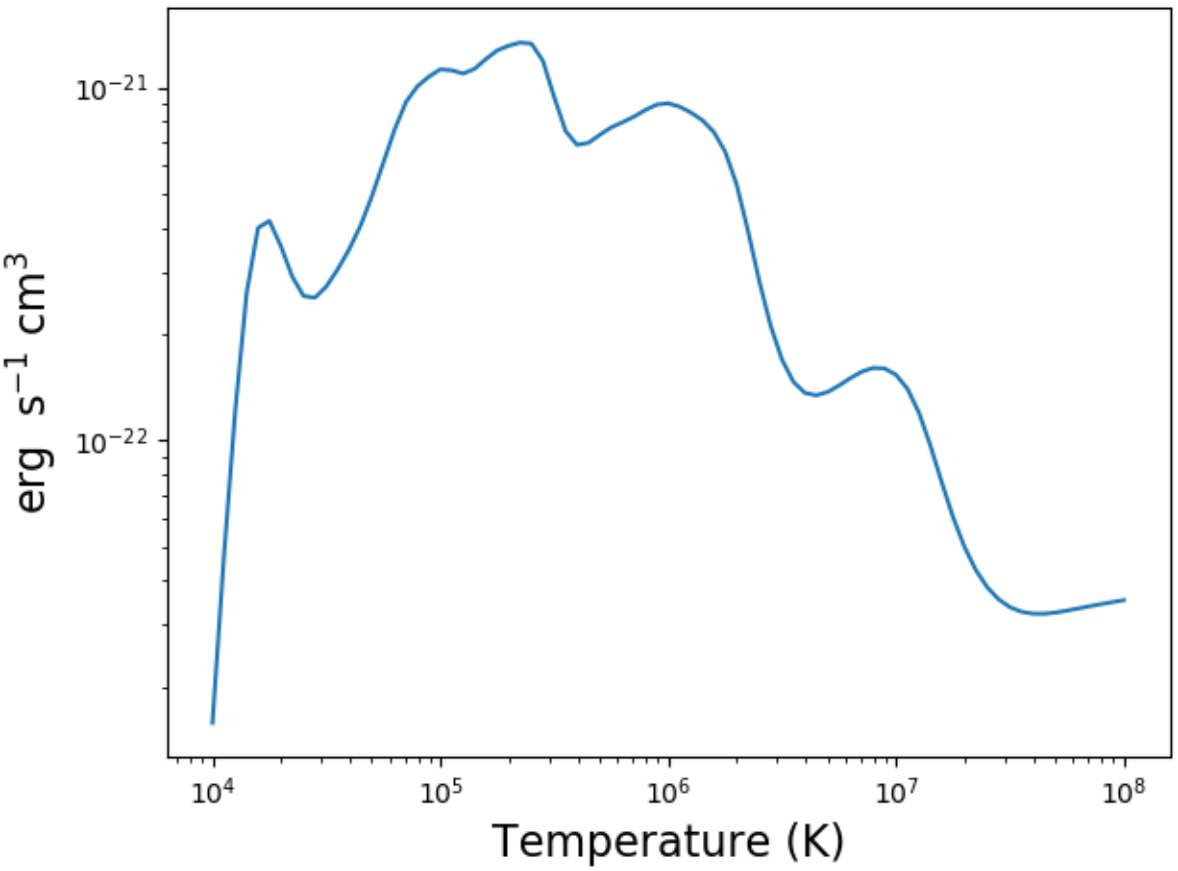
```
myAbund
```

```
mrl2 = ch.radLoss(temp, dens, minAbund=1.e-5, abundance=myAbund, verbose=1)
```

```
saveName = 'rl_coronal_1m5.pkl'
mrl2.saveData(saveName)
```

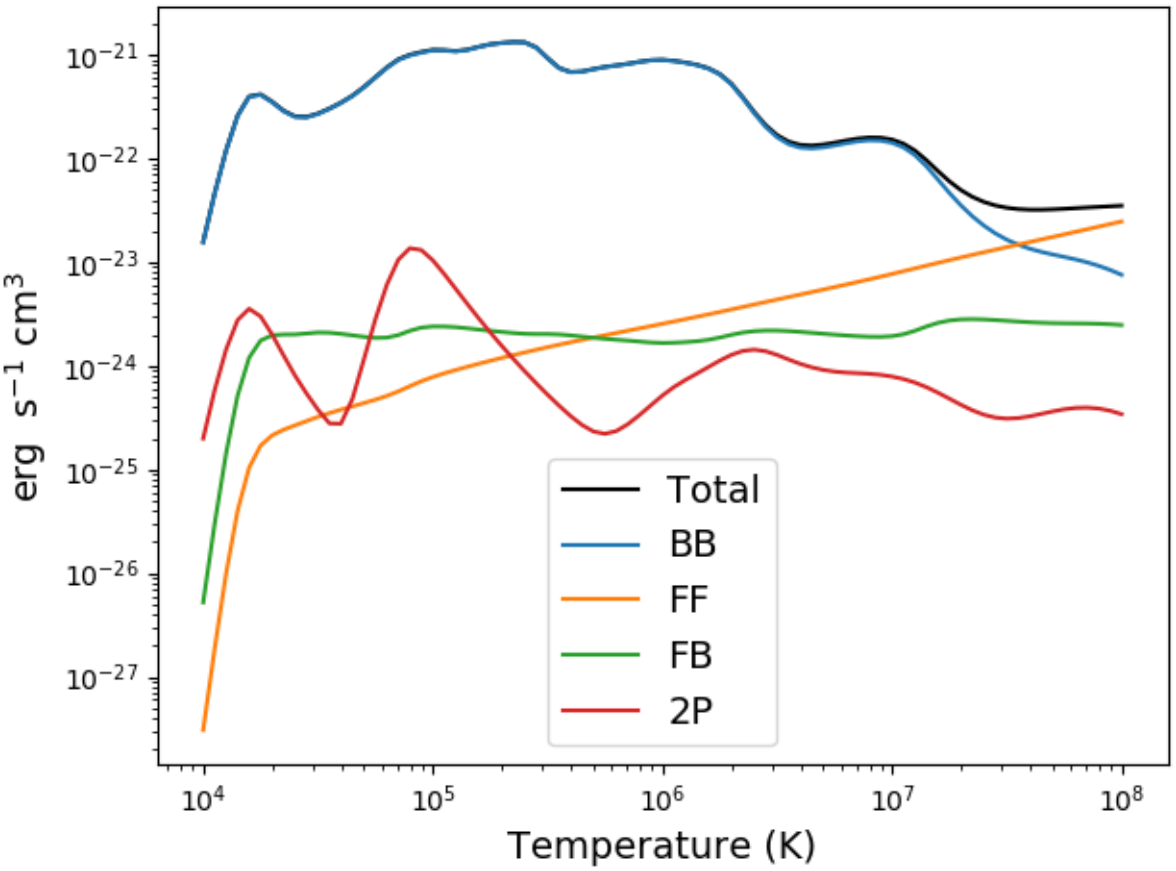
```
plt.figure()
mrl2.radLossPlot()
```

produces after 250s on a 3.5 GHz 4 core processor




```
plt.figure()
plt.loglog(temp, mrl2.RadLoss['rate'], 'k', label='Total')
plt.loglog(temp, mrl2.BoundBoundLoss, label = 'BB')
plt.loglog(temp, mrl2.FreeFreeLoss, label = 'FF')
plt.loglog(temp, mrl2.FreeBoundLoss, label = 'FB')
plt.loglog(temp, mrl2.TwoPhotonLoss, label = '2P')
plt.xlabel(mrl2.RadLoss['xlabel'], fontsize=14)
plt.ylabel(mrl2.RadLoss['ylabel'], fontsize=14)
plt.legend(loc='lower center', fontsize=14)
plt.tight_layout()
```

```
produces
```

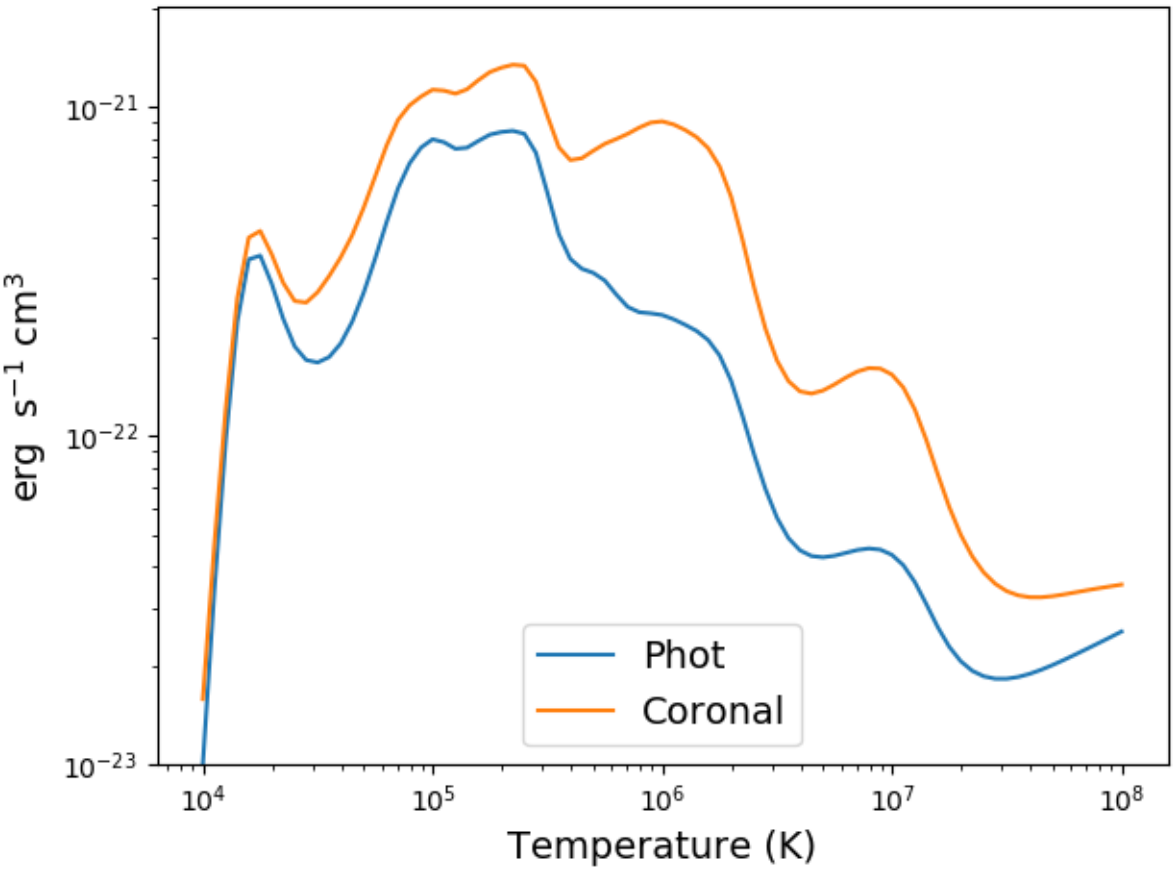


to compare photospheric and coronal radiation losses

```
rlph = ch.redux(saveNamePhot)
rlco = ch.redux(saveNameCoronal)
```

```
plt.figure()
plt.loglog(temp, rlph.RadLoss['rate'], label='Phot')
plt.loglog(temp, rlco.RadLoss['rate'], label='Coronal')
plt.ylim(bottom=1.e-23, top=2.e-21)
plt.xlabel(rlph.RadLoss['xlabel'], fontsize=14)
plt.ylabel(rlph.RadLoss['ylabel'], fontsize=14)
plt.legend(loc='lower center', fontsize=14)
plt.tight_layout()
```

produces



2.16 Jupyter Notebooks

There are 9 jupyter notebooks in the jupyter_notebooks directory that demonstrate the capabilities of ChiantyPy together with the CHIANTI database. There is also a README.txt file that provides a short explanation of the notebooks.

2.16.1 A summary of the notebooks

This directory contains 9 Jupyter IPython notebooks that demonstrate some of the ways to use ChiantiPy and the CHIANTI database

QuickStart.ipynb - a notebook that generally follows the Quick-Start guide in the docs

A demo of the bunch class is found in: bunch_demo.ipynb

These notebook show some of the characteristics and capabilities of the bunch class. Among other things, it shows how to apply labels to plots of line intensities.

Two demos of the spectrum class are found in: spectrum_demo.ipynb and spectrum_demo_2.ipynb

As with the bunch notebook, these notebook show some of the characteristics and capabilities of the spectrum class. Among other things, it shows how to apply labels to plots of spectra as a funcition of wavelength.

The directory also contains 5 other notebooks and a json file. These are demo files for reproducing some of the analyses in the paper "Electron densities and their uncertainties derived from spectral emission line intensities" by Kenneth Dere.

This paper has been published in the Monthly Notices of the Royal Astronomical Society, 2020, 496, 2334.

The notebook file '1_fe_13_demo_make_model.ipynb' constructs the model that is used by 2_fe_13_demo_check_model.ipynb and 3_fe_13_demo_chi2_search.ipynb by reading the 'tab2_1993_qs_fe_13.json' file. It is easiest if all files are placed in the same directory. This files contains the Fe XIII line intensities from Brosius et al., 1996, Astrophysical Journal Supplement Series, 106, 143. This notebook file needs to be run first.

The next notebook to run is: 2_fe_13_demo_check_model.ipynb – This notebook load the previously created pickle containing the match attribute. In this notebook a density and emission measure are guessed from an 'em loci' plot and the predictions compard with the observations.

The next notebook to run is: 3_fe_13_demo_chi2_search.ipynb – This performs a brute force chi-squared search over the density range and finds the best fit density and emission measure. These best values are then inserted into the model, a prediction made and compared with the observations.

The next notebook to run is: 4_fe_13_demo_mcmc.ipynb – This performs a MCMC analysis of the spectra to determine the most likely density and emission measure from an analysis of the trace. The trace is also save for futher analysis.

The next notebook to run is: 5_fe_13_demo_mcmc_trace_analyze.ipynb – the load the trace so that it can be re-analyzed

2.16.2 QuickStart.ipynb

2.16.3 bunch_demo.ipynb

2.16.4 spectrum_demo.ipynb

2.16.5 spectrum_demo_2.ipynb

2.16.6 1_fe_13_demo_make_match.ipynb

2.16.7 2_fe_13_demo_check_model.ipynb

2.16.8 3_fe_13_demo_chi2_search.ipynb

2.16.9 4_fe_13_demo_mcmc.ipynb

2.16.10 5_fe_13_demo_mcmc_trace_analyze.ipynb

CHAPTER

THREE

TUTORIAL

3.1 The ChiantiPy Approach

Python is a modern, object-oriented programming language. It provides a number of features that are useful to the programmer and the end user, such as, classes with methods (function-like) and attributes (data), and functions, among other things. ChiantiPy has been constructed so that the primary means to calculate the spectral properties of ions and groups of ions is by way of Python classes.

More detailed information can be found in the **API** Reference.

3.1.1 ChiantiPy Classes

There are 7 basic classes that are provided by ChiantiPy.

ion

this class is very useful in itself and is the basic unit employed by all of the other classes

continuum

for calculating the free-free (*bremstrahlung*), free-bound (radiative recombination) continuum as well as the radiative loss rates due to these processes.

bunch

allows the user to specify a *bunch* of ions and to calculate the radiative properties of the selected ions in a group. The ions can be specified by list of individual ions, list of elements, as well as by the minimum elemental abundance. The properties of each ion are available, as with members of the **ion** class. Among other things, the ratios of lines of different ions and elements can be calculated and then displayed.

spectrum

the **spectrum** class calculates the intensities of the line and continuum and them convolves the complete spectrum with a filter such as a Gaussian of specified width

there are actually 3 *spectrum* classes. Two of these all the use of multiple cpu cores to speed the calculation. The basic **spectrum** class does not do multi-processes and is therefore compatible with most Python environments

mspectrum

mspectrum duplicates the calculations of the **spectrum** class but it employes the Python multiprocessing package in order to use multiple cpu cores to calculate the spectrum. This class can be used in a basic Python shell, in a Python script, or in an IPython terminal. It can not be used in either the Jupyter qtconsole or the Jupyter notebook.

ipymspectrum

this class employes the IPython ipyparallel module to provide access to multiple cpu cores. It can only be used in the Jupyter qtconsole and the Jupyter notebook.

ioneq

this class allows the user to load and plot the ionization equilibria of a specfic element. It can read the ionization equililbrium files in the CHIANTI $XUVTOP/ioneq directory. Different ionization equilibria can be plotted against each other

it is also possible to calculate the ionization equilibria of an individual element using the ionization and recombination rates in the CHIANTI database. The results of this calculation can also be plotted agains existing calculations in the CHIANTI $XUVTOP/ioneq directory.

3.1.2 ChiantiPy Classes, Methods and Attributes

Each of the ChiantiPy classes listed above has a number of methods for calculating various properties. The results of these calculations are stored as attributes of the class that has been instantiatied (created). In Python, all objects, which includes everything in Python, have introspection so that all methods and attributes can be discovered and used. The IPython terminal and the Jupyter qtconsole both provide their own methods of easily displaying the methods and attributes.

Some methods in each class are more useful to the user than others. For example, below the populate() method is demonstrated below. However, it is generally not necessary for the user to use the populate() method. Methods that need the ion population will make use that the Population attribute is available and, if not, use the populate() method to create it.

Below, the methods that are most likely of interest to users are listed below. All of the available methods are presented and documented in the API section, a part of the ChiantiPy documentation.

ion

popPlot() method

plots the level populations of the *top* (most highly populated) levels as a function of temperature and/or density.

gofnt() method

an interactive method that plots the most intense lines of an ion in a give wavelength range (*wvlRange*) and allows the user to select a line or several lines, that will be summed, and then plots the *GofT* function for the selected lines and saves these values in the **Gofnt** dictionary as an attribute of the ion object.

emiss() method

calculates the spectral line emissivities of the ion and saves these in the **Emiss** dictionary as an attribute of the ion object.

intensity() method

calculates the *intensity* of the specified ion as a function of temperature and density. These properties are saved in the **Intensity** dictionary, available as an attribute of the ion.

intensityList() method

lists the spectral line intensities in a given wavelength range (*wvlRange*) in an interactive terminal or notebook

intensityPlot() method

plots the spectral line intensites in a given wavelength range (*wvlRange*) for the *top* most intensity lines.

intensityRatio() method

an interactive method that plots the most intense lines of an ion in a give wavelength range (*wvlRange*) and allows the user to select a pair of lines or a pair of lines to be summed and then plots the intensity ratio as a function of temperature and/or density. The ratio is saved in the **IntensityRatio** dictionary as an attribute of the ion.

spectrum() method

calculates the spectrum of the ion as a function of wavelength. The spectral line intensities are pass through a selectable filter to simulate the spectrometer line profile. The spectrum is save in the **Spectrum** dictionary as an attribute of the ion.

ionizRate() method

calculates the ionization rate coefficient as a function of temperature. The rate coefficient is save in the **IonizRate** dictionary as an attribute. Uses the methods **diRate()** and **eaRate()** to first calculate the direct and excitation-ionization (ea) rate coefficients and sums them.

recombRate() method

calculates the recombination rate coefficient as a function of temperature. The rate coefficient is save in the **RecombRate** dictionary as an attribute. Uses the methods **rrRate()** and **drRate()** to first calculate the radiative recombination and dielectronic recombination rate coefficients and sums them.

ioneq

load() method

reads a selected, existing ionization equilibrium calculation for a given element and saves it as a numpy array **Ioneq** as an attribute of the object.

calculate() method

calculates the ionization equilibrium of a selected element from the CHIANTI ionization and recombination rates for a specified temperature(s) and saves it as a numpy array **Ioneq** as an attribute of the object.

plot() method

plots the loaded or calculated ionization equilibrium. Various parameters can be specified to plot only those aspects that are desired. Can also plot an additional existing ionization equilibrium for comparison

bunch

the init method calculates the spectral line intensities for the selection of ions save the information in the **Intensity** dictionary as an attribute . It does not calculate the continuum.

beyond the init method, the **bunch** class inherits all of the following methods that are described under the **ion** class above

intensityList()

intensityPlot()

intensityRatio()

in addition, it inherits the following methods that are described under the **spectrum** class below

convolve()

lineSpectrumPlot()

spectrumPlot()

spectrum

the init method calculates the spectral line intensities and the continuum due to the free-free (*bremstrahlung*), free-bound (radiative recombination), and two-photon processes. The line intensities are convolved using the **convolve()** method (below). The sum is saved in the **Spectrum** dictionary as an attribute.

beyond the init method, the **spectrum** class also inherits the same methods as the **bunch** class including **intensityList()**, **intensityPlot**, and **intensityRatio**.

convolve()

convolves the line spectrum with specified filter from **ChiantiPy.tools.filters** using a specified width.

lineSpectrumPlot()

plots the convolved line spectrum as a function wavelength

spectrumPlot()

plots the spectrum calculated by the init method. The summed (integrated) spectrum can be plotted or the spectrum for a specific temperature can be plotted.

mspectrum

the mspectrum behaves in the same way as the **spectrum** class except that it invokes the Python multiprocessing module so that the calculations are made using a specified number of cpu cores. **mspectrum** can not be used in the Jupyter qtconsole or notebook.

ipymspectrum

the ipymspectrum behaves in the same way as the **spectrum** class except that it invokes the IPython ipyparallel module so that the calculations are made using a specified number of cpu cores. **ipymspectrum** can only be used in the IPython terminal or the Jupyter qtconsole or notebook.

3.2 The ion class, basic properties

Bring up a Python session, or better yet, an IPython session

```
import ChiantiPy.core as ch
fe14 = ch.ion('fe_14', setup=False)
```

The fe14 object is instantiated with a number of methods and data. Methods start with lowercase letters and attributes start with uppercase letters. It is best not to simply import ion as there is a method with the same name in matplotlib. A few examples:

```
fe14.IonStr
  >> 'fe_14'
fe14.Spectroscopic
  >> 'Fe XIV'
```

CHIANTI and spectroscopic notation for the ion

```
fe14.Z
  >> 26
fe14.Ion
  >> 14
```

nuclear charge Z and the ionization stage (in spectroscopic notation) for the ion

```
fe14.Ip
  >> 392.16196
```

this is the ionization potential in electron volts.

```
fe14.FIP
  >> 7.9023801573028294
```

this is the first ionization potential (FIP) in electron volts - the ionization potential of the neutral (Fe I).

```
fe14.Abundance
  >> 0.00012589265
fe14.AbundanceName
  >> 'sun_photospheric_1998_grevesse'
```

this is the abundance of iron relative to hydrogen for the specified elemental abundance set. For the ion class, the abundance can be specified by the *abuncance* keyword argument or the *abundanceName* keyword argument. In the case the abundance is taken from default abundcance set. The specified defaults can be examined by

```
fe14.Defaults
  >>  {'abundfile': 'sun_photospheric_1998_grevesse', 'flux': 'energy', 'ioneqfile':
↪'chianti', 'wavelength': 'angstrom'}
```

the defaults can be specified by the user in the chiantrc file. One is included in the distribution but needs to be place in the proper directory to be picked up. On Linux, it should be placed either in $HOME/.config or $HOME/.chianti. On Windows, it should be copied to $PROFILEHOME/.config or $PROFILEHOME/.chianti. If it is not found, a set of coded default values are used.

```
fe14.Elvlc.keys()
>>  ['ecmth', 'term', 'ref', 'pretty', 'spd', 'ecm', 'j', 'l', 'erydth', 'conf', 'lvl',
↪'spin', 'eryd', 'mult']
```

fe14.Elvlc is a dictionary that describes the energy levels of the Fe XIV ion. The key 'ecm' provides the energies, relative to the ground level, in inverse cm. The 'ref' key provides the references in the scientific literature where the data were provided.

```
fe14.Elvlc['ref']
>> ['%filename: fe_14.elvlc',
     %observed energy levels: Churilov S.S., Levashov V.E., 1993, Physica Scripta 48,␣
↪425,
     %observed energy levels: Redfors A., Litzen U., 1989, J.Opt.Soc.Am.B 6, #8, 1447,
     %theoretical energy levels: Storey P.J., Mason H.E., Young P.R., 2000, A&ASS 141,␣
↪28,
     %comment,
     Only level 16 does not have an observed energy. I have placed in,
     the third energy column a recommended value for the energy value of,
     this level, based on the theoretical and observed splittings of the,
```

(continues on next page)

(continued from previous page)

```
      4F levels. It is this energy value which is used to compute the,
      wavelengths of transitions involving level 16 given in the .wgfa,
      file.,
      %produced as part of the Arcetri/Cambridge/GMU/NRL 'CHIANTI' atomic data base␣
↪collaboration,
      %,
      %   P.R.Young Feb 99']
```

If the fe14 **ion** object had be instantiated (created) with a temperature and an electron density, then many more attributes can be calculated. For example, if the populate() method is used, it creates a dictionary attribute *Population*. One thing to remember with Python is that capitalization matters.

```
import numpy as np
temp = 10.**(5.8+0.1*np.arange(11))
dens = 1.e+9
fe14 = ch.ion('fe_14', temp, dens)
fe14.populate()
fe14.Population.keys()
>>['ci', 'protonDensity', 'popmat', 'eDensity', 'rec', 'population', 'temperature']

fe14.Population['population'].shape
>>(21, 739)

'%10.2e'%(fe14.Temperature[10])
>> '  2.00e+06'

fe14.Population['population'][10,:5]
>>array([ 8.71775703e-01, 1.27867444e-01, 4.91230626e-09, 4.29120495e-08, 1.35517895e-
↪08])
```

gives the population of the first 5 of 739 levels of Fe XIV at a temperature of 2.00e+6

to be continued

CHAPTER

FOUR

CHIANTIPY'S API DOCUMENTATION

4.1 ChiantiPy package

4.1.1 Subpackages

ChiantiPy.Gui package

Subpackages

ChiantiPy.Gui.gui_cl package

Submodules

ChiantiPy.Gui.gui_cl.gui module

Command line selection dialogs.

class ChiantiPy.Gui.gui_cl.gui.**choice2Dialog**(*items*, *label=None*, *parent=None*)
Bases: object

Make a single or multiplee selection from a list of items and another single or multiple selection from the same list.

Useful for picking numerators and denominators.

expects the input of an array of items, will select one or more from both widgets the keywords label and parent are there for consistency with real gui dialogs

ChiantiPy.Gui.gui_cl.gui.**chpicker**(*path*, *filter='*.*'*, *label=None*)
Select a filename from using a command line dialog.

the label keyword is included for consistency but does nothing

class ChiantiPy.Gui.gui_cl.gui.**selectorDialog**(*items*, *label=None*, *parent=None*, *multiChoice=False*)
Bases: object

Make a single or multiple selection from a list of items.

expects the input of an array of items, will select one or more the label and parent keywords are for consistency with other modules but do nothing

Module contents

command-line selection dialogs

ChiantiPy.Gui.gui_qt5 package

Submodules

ChiantiPy.Gui.gui_qt5.gui module

PyQt5 widget selection dialogs

class ChiantiPy.Gui.gui_qt5.gui.**choice2Dialog**(*items*, *label=None*, *parent=None*, *multi=True*)
Bases: PyQt5.QtWidgets.QDialog

Make a single or multiple selection from a list of items and another single or multiple selection from the same list.

Useful for picking numerators and denominators.

expects the input of an array of items, will select one or more from both widgets.

accept(*self*)

reject(*self*)

class ChiantiPy.Gui.gui_qt5.gui.**chpicker**(*dir*, *label*)
Bases: PyQt5.QtWidgets.QWidget

dialog to select a single file name under the directory code largely taken from pythonspot.com

initUI()

openFileNameDialog()

class ChiantiPy.Gui.gui_qt5.gui.**selectorDialog**(*items*, *label=None*, *parent=None*, *multiChoice=False*)
Bases: PyQt5.QtWidgets.QDialog

Make a single or multiple selection from a list of items.

expects the input of an array of items, will select one or more

accept(*self*)

reject(*self*)

ChiantiPy.Gui.gui_qt5.ui module

class ChiantiPy.Gui.gui_qt5.ui.**Ui_choice2DialogForm**
Bases: object

retranslateUi(*choice2DialogForm*)

setupUi(*choice2DialogForm*)

class ChiantiPy.Gui.gui_qt5.ui.**Ui_selectorDialogForm**
Bases: object

retranslateUi(*selectorDialogForm*)

setupUi(*selectorDialogForm*)

Module contents

PyQt5 selection dialog widgets

Module contents

Select GUI package

ChiantiPy.base package

Submodules

ChiantiPy.base._IonTrails module

Base classes used in the *ChiantiPy.core.ion* and *ChiantiPy.core.spectrum* classes. Mostly printing, plotting and saving routines.

class ChiantiPy.base._IonTrails.**ionTrails**

Bases: object

Base class for *ChiantiPy.core.ion* and *ChiantiPy.core.spectrum*

argCheck(*temperature=None*, *eDensity=None*, *pDensity='default'*, *em=None*, *verbose=0*)

to check the compatibility of the three arguments and put them into numpy arrays of atleast_1d and create attributes to the object

intensityList(*index=None*, *wvlRange=None*, *wvlRanges=None*, *top=10*, *integrated=False*, *relative=0*, *outFile=0*, *rightDigits=4*)

List the line intensities. Checks to see if there is an existing Intensity attribute. If it exists, then those values are used. Otherwise, the *intensity* method is called.

This method prints an ASCII table with the following columns:

1. Ion: the CHIANTI style notation for the ion, e.g. 'c_4' for C IV
2. lvl1: the lower level of the transition in the CHIANTI .elvlc file
3. lvl2: the upper level of the transition in the CHIANTI .elvlc file
4. lower: the notation, usually in LS coupling, of the lower fine structure level
5. upper: the notation, usually in LS coupling, of the upper fine structure level
6. Wvl(A): the wavelength of the transition in units as specified in the chiantirc file.
7. Intensity
8. A value: the Einstein coefficient for spontaneous emission from level 'j' to level 'i'
9. Obs: indicates whether the CHIANTI database considers this an observed line or one obtained from theoretical energy levels

Regarding the intensity column, if 'flux' in the chiantirc file is set to 'energy', the intensity is given by,

$$I = \Delta E_{ij} n_j A_{ij} \mathrm{Ab} \frac{1}{N_e} \frac{N(X^{+m})}{N(X)} \mathrm{EM},$$

in units of ergs cm^{-2} s^{-1} sr^{-1}. If 'flux' is set to 'photon',

$$I = n_j A_{ij} \mathrm{Ab} \frac{1}{N_e} \frac{N(X^{+m})}{N(X)} \mathrm{EM},$$

where,

- ΔE_{ij} is the transition energy (ergs)
- n_j is the fractions of ions in level j
- A_{ij} is the Einstein coefficient for spontaneous emission from level j to level i (in s^{-1})
- Ab is the abundance of the specified element relative to hydrogen
- N_e is the electron density (in cm^{-3})
- $N(X^{+m})/N(X)$ is the fractional ionization of ion as a function of temperature
- EM is the emission measure integrated along the line-of-sight, $\int \mathrm{d}l\, N_e N_H$ (cm^{-5}) where N_H is the density of hydrogen (neutral + ionized) (cm^{-3})

Note that if *relative* is set, the line intensity is relative to the strongest line and so the output will be unitless.

Parameters

- **index** (*int*,optional) – Index the temperature or eDensity array to use. -1 (default) sets the specified value to the middle of the array
- **wvlRange** (*tuple*) – Wavelength range
- **wvlRanges** (*a tuple, list or array that contains at least 2*) – 2 element tuples, lists or arrays so that multiple wavelength ranges can be specified
- **top** (*int*) – Number of lines to plot, sorted by descending magnitude.
- **integrated** (ool`) – if set to True, the integrated/summed spectrum is used
- **relative** (*int*) – specifies whether to normalize to strongest line default (relative = 0) specified that the intensities should be their calculated values
- **outFile** (*str*) – specifies the file that the intensities should be output to default(outFile = 0) intensities are output to the terminal
- **rightDigits** (*int*) – specifies the format for the wavelengths for the number of digits to right of the decimal place

intensityPlot(*wvlRange=None*, *index=None*, *top=10*, *integrated=False*, *linLog='lin'*, *relative=False*, *doLabel=True*, *doTitle=True*, *lw=1*, *verbose=False*, *plotFile=0*, *em=0*)

Plot the line intensities. Uses *Intensity* if it already exists. If not, calls the *intensity* method.

Keyword Arguments

- **wvlRange** (*2 element tuple, list or array determines the wavelength range to plot*) –
- **index** (*integer*) – specified which value of the temperature array or eDensity array to use. default (index=-1) sets the specified value to the middle of the array
- **top** (*integer*) – specifies to plot only the top strongest lines, default = 10
- **linLog** (*str*) – default('lin') produces a plot where the intensity scale is linear if set to 'log', produces a plot where the intensity scale is logarithmic
- **relative** (*= True specifies whether to normalize to strongest line*) – default (relative = False) specified that the intensities should be their calculated values

- **doLabel** (*= True, then lines are labeled with ion and wavelength (default=True)*) –
- **doTitle** (*= True, then a title is applied to the plot (default=True)*) –
- **lw** (*int*, width of the label line in matplotlib units (default=1)) –
- **plotFile** – default=0, the plot is not saved to a file othewise, the plot is saved to the 'plotFile'
- **em** (*emission measure*) – if an Intensity attribute needs be created, then the emission measure is applied

intensityRatio(*wvlRange=None*, *wvlRanges=None*, *top=10*, *doTitle=True*)

Plot the intensity ratio of 2 lines or sums of lines. Shown as a function of density and/or temperature. For a single wavelength range, set wvlRange = [wMin, wMax] For multiple wavelength ranges, set wvlRanges = [[wMin1,wMax1],[wMin2,wMax2], ...] A plot of relative emissivities is shown and then a dialog appears for the user to choose a set of lines.

Note: if the default value for gui is set to False, then it is usually necessary to invoke this method twice to get the desired result.

Parameters

- **wvlRange** (*array-like*) – Wavelength range, i.e. min and max
- **wvlRanges** (*a tuple, list or array that contains at least 2*) – 2 element tuples, lists or arrays so that multiple wavelength ranges can be specified
- **top** (*int*) – specifies to plot only the top strongest lines, default = 10
- **title** (*bool*) – if True, places a title on the plot

intensityRatioSave(*outFile=0*)

Save the intensity ratio to a file.

The intensity ratio as a function to temperature and eDensity is saved to an asciii file. Descriptive information is included at the top of the file.

Parameters **outFile** – default(0): the plot of the intensity ratio is not saved str/unicode: the plot is saved to the file names 'outFile'

ChiantiPy.base._SpecTrails module

Base class used in several ChiantiPy objects

class ChiantiPy.base._SpecTrails.**specTrails**

Bases: object

a collection of methods for use in spectrum calculations

ionGate(*elementList=None*, *ionList=None*, *minAbund=None*, *doLines=1*, *doContinuum=1*, *doWvlTest=1*, *doIoneqTest=1*, *includeDiel=False*, *verbose=0*)

creates a list of ions for free-free, free-bound, and line intensity calculations if doing the radiative losses, accept all wavelength -> doWvlTest=0 the list is a dictionary self.Todo

lineSpectrumPlot(*index=0*, *integrated=False*, *saveFile=False*, *linLog='lin'*)

to plot the line spectrum as a function of wavelength

Keyword Arguments

- **index** (*int*) – selects the temperature of the calculated line spectrum

- **integrated** (*bool*) – if True, plots the integrated/summed line spectrum
- **saveFile** (*bool*, *str*) – if set, saves the plot to 'saveFile'
- **linLog** (*str*) – should be either 'lin' for linear, or 'log' for logarithmic base10

restoreData(*filename*)

Parameters **filename** (*str*) – filename where the pickle file of the saved data to be loaded

saveData(*filename*, *verbose=False*)

Parameters **filename** (*str*) – filename where the pickle file of the saved data will be stored

following running a multi-ion calculation (bunch, spectrum, mspectrum, ipymspectrum, radloss, save the calculation as a dictionary to a pickle file

spectrumPlot(*wvlRange=None*, *index=None*, *integrated=False*, *saveFile=False*, *linLog='lin'*, *doLabel=True*, *lw=1*, *doTitle=True*, *top=10*)
to plot the spectrum as a function of wavelength

Keyword Arguments

- **wvlRange** (*2 element tuple, list or array determines the wavelength range to plot*) –
- **index** (*int*) – selects the temperature of the calculated spectrum
- **integrated** (*bool*) – if True, plots the integrated/summed spectrum
- **saveFile** (*bool*, *str*) – if set, saves the plot to 'saveFile'
- **linLog** (*str*) – should be either 'lin' for linear, or 'log' for logarithmic base10
- **doLabel** (*bool*) – if set to True, labels of the top spectral lines are added
- **lw** (*int*, width of the label line in matplotlib units (default=1)) –
- **doTitle** (*bool* if True, then a title is applied to the plot (default=True)) –

top: integer specifies to plot only the top strongest lines, default = 10 if set to None, all lines are plotted

Module contents

Base classes for ion- and spectrum-related objects.

ChiantiPy.core package

Subpackages

ChiantiPy.core.tests package

Submodules

ChiantiPy.core.tests.test_Continuum module

ChiantiPy.core.tests.test_Ion module

ChiantiPy.core.tests.test_Ioneq module

Tests for ioneq class

ChiantiPy.core.tests.test_Ioneq.**test_calculate_ioneq**()

ChiantiPy.core.tests.test_Ioneq.**test_el_input**()

ChiantiPy.core.tests.test_Ioneq.**test_el_z_inputs_same**()

ChiantiPy.core.tests.test_Ioneq.**test_load_ioneq**()

ChiantiPy.core.tests.test_Ioneq.**test_load_ioneq_alternate_file**()

ChiantiPy.core.tests.test_Ioneq.**test_z_input**()

ChiantiPy.core.tests.test_Spectrum module

Tests for the spectrum and bunch classes

ChiantiPy.core.tests.test_Spectrum.**test_bunch**()

ChiantiPy.core.tests.test_Spectrum.**test_spectrum_array**()

ChiantiPy.core.tests.test_Spectrum.**test_spectrum_scalar**()

Module contents

Submodules

ChiantiPy.core.Continuum module

Continuum module

class ChiantiPy.core.Continuum.**continuum**(*ionStr*, *temperature*, *abundance=None*, *em=None*, *verbose=0*)

Bases: *ChiantiPy.base._IonTrails.ionTrails*

The top level class for continuum calculations. Includes methods for the calculation of the free-free and free-bound continua.

Parameters

- **ionStr** (*str*) – CHIANTI notation for the given ion, e.g. 'fe_12' that corresponds to the Fe XII ion.
- **temperature** (*array-like*) – In units of Kelvin

Keyword Arguments

- **abundance** (*float*, *str*, optional) – Elemental abundance relative to Hydrogen or name of CHIANTI abundance file, without the '.abund' suffix, e.g. 'sun_photospheric_1998_grevesse'.
- **em** (*array-like, optional*) – Line-of-sight emission measure ($\int \mathrm{d}l\, n_e n_H$), in units of cm^{-5}, or the volumetric emission measure ($\int \mathrm{d}V\, n_e n_H$) in units of cm^{-3}.
- **verbose** (*bool*) – if True, prints additional info to the console

Examples

```
>>> import ChiantiPy.core as ch
>>> import numpy as np
>>> temperature = np.logspace(4,9,20)
>>> cont = ch.continuum('fe_15',temperature)
>>> wavelength = np.arange(1,1000,10)
>>> cont.freeFree(wavelength)
>>> cont.freeBound(wavelength, include_abundance=True, include_ioneq=False)
>>> cont.calculate_free_free_loss()
>>> cont.calculate_free_bound_loss()
```

Notes

The methods for calculating the free-free and free-bound emission and losses return their result to an attribute. See the respective docstrings for more information.

References

calculate_free_bound_loss(***kwargs*)

Calculate the free-bound energy loss rate of an ion. The result is returned to the *free_bound_loss* attribute.

The free-bound loss rate can be calculated by integrating the free-bound emission over the wavelength. This is difficult using the expression in *calculate_free_bound_emission* so we instead use the approach of[108] and[106]. Eq. 1a of[106] can be integrated over wavelength to get the free-bound loss rate,

$$\frac{dW}{dtdV} = C_{ff}\frac{k}{hc}T^{1/2}G_{fb},$$

in units of erg $\mathrm{cm}^3\,\mathrm{s}^{-1}$ where G_{fb} is the free-bound Gaunt factor as given by Eq. 15 of[106] (see *mewe_gaunt_factor* for more details) and C_{ff} is the numerical constant as given in Eq. 4 of[?] and can be written in terms of the fine structure constant α,

$$C_{ff}\frac{k}{hc} = \frac{8}{3}\left(\frac{\pi}{6}\right)^{1/2}\frac{h^2\alpha^3}{\pi^2}\frac{k_B}{m_e^{3/2}} \approx 1.43\times10^{-27}$$

freeBound(*wvl*, *includeAbund=True*, *includeIoneq=True*, *verner=True*, *verbose=False*)

Calculates the free-bound (radiative recombination) continuum emissivity of an ion. Provides emissivity in units of ergs $\mathrm{cm}^{-2}\,\mathrm{s}^{-1}\,\mathrm{str}^{-1}\,{}^{-1}$ for an individual ion. If includeAbund is set, the abundance is included. If includeIoneq is set, the ionization equililibrium for the given ion is included

Notes

- Uses the Gaunt factors of[103] for recombination to the excited levels
- Uses the photoionization cross sections of[3] to develop the free-bound cross section
- Include the elemental abundance and ionization fraction by default
- The specified ion is the target ion

[108] Gronenschild, E.H.B.M. and Mewe, R., 1978, A&AS, 32, 283
[106] Mewe, R. et al., 1986, A&AS, 65, 511
[103] Karzas and Latter, 1961, ApJSS, 6, 167
[3] Verner & Yakovlev, 1995, A&AS, 109, 125

- uses the corrected version of the K-L bf gaunt factors available in CHIANTI V10
- revised to calculate the bf cross, fb cross section and the maxwell energy distribution

References

freeBoundLoss(*includeAbund=True*, *includeIoneq=True*, *verner=True*, *verbose=False*)
to calculate the free-bound (radiative recombination) energy loss rate coefficient of an ion, the ion is taken to be the target ion, including the elemental abundance and the ionization equilibrium population uses the Gaunt factors of[?] Karzas, W.J, Latter, R, 1961, ApJS, 6, 167 provides rate = erg cm^-2 s^-1 .. rubric:: Notes

- Uses the Gaunt factors of[?] for recombination to the ground level
- Uses the photoionization cross sections of[2] to develop the free-bound cross section
- Does not include the elemental abundance or ionization fraction
- The specified ion is the target ion
- uses the corrected version of the K-L bf gaunt factors available in CHIANTI V10
- revised to calculate the bf cross, fb cross section and the maxwell energy distribution
- the Verner cross sections are not included for now
- using the RESTART formulation

References

freeBoundLossMao(*includeAbund=False*, *includeIoneq=False*)
to calculate the radiative loss rate from the parameters of Mao J., Kaastra J., Badnell N.R. <Astron. Astrophys. 599, A10 (2017)>=2017A&A...599A..10M

freeBoundLossMewe(***kwargs*)
Calculate the free-bound energy loss rate of an ion. The result is returned to the *free_bound_loss* attribute.

The free-bound loss rate can be calculated by integrating the free-bound emission over the wavelength. This is difficult using the expression in *calculate_free_bound_emission* so we instead use the approach of[?] and[?]. Eq. 1a of[?] can be integrated over wavelength to get the free-bound loss rate,

$$\frac{dW}{dtdV} = C_{ff}\frac{k}{hc}T^{1/2}G_{fb},$$

in units of erg $\mathrm{cm}^3\,\mathrm{s}^{-1}$ where G_{fb} is the free-bound Gaunt factor as given by Eq. 15 of[?] (see *mewe_gaunt_factor* for more details) and C_{ff} is the numerical constant as given in Eq. 4 of[?] and can be written in terms of the fine structure constant α,

$$C_{ff}\frac{k}{hc} = \frac{8}{3}\left(\frac{\pi}{6}\right)^{1/2}\frac{h^2\alpha^3}{\pi^2}\frac{k_B}{m_e^{3/2}} \approx 1.43\times10^{-27}$$

freeBoundRate(*verner=True*, *verbose=False*)
Calculates the free-bound (radiative recombination) rate of an ion. Provides emissivity in units of ergs cm^{-2} s^{-1} str^{-1} $^{-1}$ for an individual ion. If includeAbund is set, the abundance is included. If includeIoneq is set, the ionization equililibrium for the given ion is included This method is only for the purposes of testing. The best rr/FB rates come from the ion class

[2] Verner & Yakovlev, 1995, A&AS, 109, 125

Notes

- Uses the Gaunt factors of[?] for recombination to the excited levels
- Uses the photoionization cross sections of[?] to develop the free-bound cross section
- Include the elemental abundance and ionization fraction by default
- The specified ion is the target ion
- uses the corrected version of the K-L bf gaunt factors available in CHIANTI V10
- revised to calculate the bf cross, fb cross section and the maxwell energy distribution

freeFree(*wavelength*, *includeAbund=True*, *includeIoneq=True*, ***kwargs*)

Calculates the free-free emission for a single ion. The result is returned as a dict to the *FreeFree* attribute. The dict has the keywords *intensity*, *wvl*, *temperature*, *em*.

The free-free emission for the given ion is calculated according Eq. 5.14a of[107], substituting $\nu = c/\lambda$, dividing by the solid angle, and writing the numerical constant in terms of the fine structure constant α,

$$\frac{dW}{dtdVd\lambda} = \frac{c}{3m_e}\left(\frac{\alpha h}{\pi}\right)^3\left(\frac{2\pi}{3m_e k_B}\right)^{1/2}\frac{Z^2}{\lambda^2 T^{1/2}}\exp\left(-\frac{hc}{\lambda k_B T}\right)\bar{g}_{ff},$$

where Z is the nuclear charge, T is the electron temperature in K, and $\bar{g}_{ff}$ is the velocity-averaged Gaunt factor. The Gaunt factor is estimated using the methods of[104] and[101], depending on the temperature and energy regime. See *itoh_gaunt_factor* and *sutherland_gaunt_factor* for more details.

The free-free emission is in units of erg $\mathrm{cm^3 s^{-1} \AA^{-1} str^{-1}}$. If the emission measure has been set, the units will be multiplied by $\mathrm{cm^{-5}}$ or $\mathrm{cm^{-3}}$, depending on whether it is the line-of-sight or volumetric emission measure, respectively.

Parameters

- **wavelength** (*array-like*) – In units of angstroms
- **includeAbund** (*bool*, optional) – If True, include the ion abundance in the final output.
- **includeIoneq** (*bool*, optional) – If True, include the ionization equilibrium in the final output

freeFreeLoss(*includeAbund=True*, *includeIoneq=True*, ***kwargs*)

Calculate the free-free energy loss rate of an ion. The result is returned to the *FreeFreeLoss* attribute.

The free-free radiative loss rate is given by Eq. 5.15a of[?]. Writing the numerical constant in terms of the fine structure constant α,

$$\frac{dW}{dtdV} = \frac{4\alpha^3 h^2}{3\pi^2 m_e}\left(\frac{2\pi k_B}{3m_e}\right)^{1/2} Z^2 T^{1/2}\bar{g}_B$$

where where Z is the ion charge, T is the electron temperature, and $\bar{g}_B$ is the wavelength-averaged and velocity-averaged Gaunt factor. The Gaunt factor is calculated using the methods of[?]. Note that this expression for the loss rate is just the integral over wavelength of Eq. 5.14a of[?], the free-free emission, and is expressed in units of erg $\mathrm{cm^3\ s^{-1}}$.

free_free_loss(*includeAbund=True*, *includeIoneq=True*, ***kwargs*)

Calculate the free-free energy loss rate of an ion. The result is returned to the *free_free_loss* attribute.

[107] Rybicki and Lightman, 1979, Radiative Processes in Astrophysics, (Wiley-VCH)
[104] Itoh, N. et al., 2000, ApJS, 128, 125
[101] Sutherland, R. S., 1998, MNRAS, 300, 321

The free-free radiative loss rate is given by Eq. 5.15a of[?]. Writing the numerical constant in terms of the fine structure constant α,

$$\frac{dW}{dtdV} = \frac{4\alpha^3 h^2}{3\pi^2 m_e}\left(\frac{2\pi k_B}{3m_e}\right)^{1/2} Z^2 T^{1/2}\bar{g}_B$$

where where Z is the nuclear charge, T is the electron temperature, and $\bar{g}_B$ is the wavelength-averaged and velocity-averaged Gaunt factor. The Gaunt factor is calculated using the methods of[?]. Note that this expression for the loss rate is just the integral over wavelength of Eq. 5.14a of[?], the free-free emission, and is expressed in units of erg $\mathrm{cm}^3\,\mathrm{s}^{-1}$.

ioneqOne()

Provide the ionization equilibrium for the selected ion as a function of temperature. Similar to but not identical to ion.ioneqOne() - the ion class needs to be able to handle the 'dielectronic' ions returned in self.IoneqOne

ioneq_one(*stage*, ***kwargs*)

Calculate the equilibrium fractional ionization of the ion as a function of temperature.

Uses the *ChiantiPy.core.ioneq* module and does a first-order spline interpolation to the data. An ionization equilibrium file can be passed as a keyword argument, *ioneqfile*. This can be passed through as a keyword argument to any of the functions that uses the ionization equilibrium.

Parameters **stage** (*int*) – Ionization stage, e.g. 25 for Fe XXV

itoh_gaunt_factor(*wavelength*)

Calculates the free-free gaunt factors of[?].

An analytic fitting formulae for the relativistic Gaunt factor is given by Eq. 4 of[?],

$$g_Z = \sum_{i,j=0}^{10} a_{ij} t^i U^j$$

where,

$$t = \frac{1}{1.25}(\log_{10} T - 7.25),\; U = \frac{1}{2.5}(\log_{10} u + 1.5),$$

$u = hc/\lambda k_B T$, and a_{ij} are the fitting coefficients and are read in using *ChiantiPy.tools.io.itohRead* and are given in Table 4 of[?]. These values are valid for $6 < \log_{10}(T) < 8.5$ and $-4 < \log_{10}(u) < 1$.

See also:

ChiantiPy.tools.io.itohRead Read in Gaunt factor coefficients from[?]

mewe_gaunt_factor(***kwargs*)

Calculate the Gaunt factor according to[?] for a single ion Z_z.

Using Eq. 9 of[?], the free-bound Gaunt factor for a single ion can be written as,

$$G_{fb}^{Z,z} = \frac{E_H}{k_B T}\mathrm{Ab}(Z)\frac{N(Z,z)}{N(Z)} f(Z,z,n)$$

where E_H is the ground-state potential of H, $\mathrm{Ab}(Z)$ is the elemental abundance, $\frac{N(Z,z)}{N(Z)}$ is the fractional ionization, and $f(Z,z,n)$ is given by Eq. 10 and is approximated by Eq 16 as,

$$f(Z,z,n) \approx f_2(Z,z,n_0) = 0.9\frac{\zeta_0 z_0^4}{n_0^5}\exp\left(\frac{E_H z_0^2}{n_0^2 k_B T}\right) + 0.42\frac{z^4}{n_0^{3/2}}\exp\left(\frac{E_H z^2}{(n_0+1)^2 k_B T}\right)$$

where n_0 is the principal quantum number, z_0 is the effective charge (see Eq. 7 of[?]), and ζ_0 is the number of vacancies in the 0th shell and is given in Table 1 of[?]. Here it is calculated in the same manner as in fb_rad_loss.pro of the CHIANTI IDL library. Note that in the expression for G_{fb}, we have not included the N_H/n_e factor.

Raises `ValueError` – If no .fblvl file is available for this ion

`rrRate()`
Provide the radiative recombination rate coefficient as a function of temperature (K). a revised copy of the Ion method

`sutherland_gaunt_factor(wavelength)`
Calculates the free-free gaunt factor calculations of[Page 78, 101].

The Gaunt factors of[Page 78, 101] are read in using *ChiantiPy.tools.io.gffRead* as a function of u and γ^2. The data are interpolated to the appropriate wavelength and temperature values using *~scipy.ndimage.map_coordinates*.

`vernerCross(wvl)`
Calculates the photoionization cross-section using data from[102] for transitions to the ground state.

The photoionization cross-section can be expressed as $\sigma_i^{fb} = F(E/E_0)$ where F is an analytic fitting formula given by Eq. 1 of[102],

$$F(y) = ((y-1)^2 + y_w^2)y^{-Q}(1+\sqrt{y/y_a})^{-P},$$

where E is the photon energy, n is the principal quantum number, l is the orbital quantum number, $Q = 5.5 + l - 0.5P$, and $\sigma_0, E_0, y_w, y_a, P$ are fitting paramters. These can be read in using *ChiantiPy.tools.io.vernerRead*.

Parameters:

wvl [*list*, *ndarray*] wavelength in Angstroms

ChiantiPy.core.Ion module

Ion class

`class ChiantiPy.core.Ion.ion(ionStr, temperature=None, eDensity=None, pDensity='default', radTemperature=None, rStar=None, abundance=None, setup=True, em=None, verbose=0)`

Bases: `ChiantiPy.base._IoneqOne.ioneqOne`, `ChiantiPy.base._IonTrails.ionTrails`, `ChiantiPy.base._SpecTrails.specTrails`

The top level class for performing spectral calculations for an ion in the CHIANTI database.

Parameters

- **ionStr** (*str*) – CHIANTI notation for the given ion, e.g. 'fe_12' that corresponds to the *Fe XII* ion.
- **temperature** (*float* , *tuple*, *list*, *~numpy.ndarray*, optional) – Temperature array (Kelvin)
- **eDensity** (*float* , *tuple*, *list*, or *~numpy.ndarray*, optional) – Electron density array (cm^{-3})
- **pDensity** (*float*, *tuple*, *list* or *~numpy.ndarray*, optional) – Proton density (cm^{-3})
- **radTemperature** (*float* or *~numpy.ndarray*, optional) – Radiation black-body temperature (in Kelvin)

[102] Verner & Yakovlev, 1995, A&AS, 109, 125

- **rStar** (*float* or *~numpy.ndarray*, optional) – Distance from the center of the star (in stellar radii)
- **abundance** (*float* or *str*, optional) – Elemental abundance relative to Hydrogen or name of CHIANTI abundance file to use, without the '.abund' suffix, e.g. 'sun_photospheric_1998_grevesse'.
- **setup** (*bool* or *str*, optional) – If True, run ion setup function. Otherwise, provide a limited number of attributes of the selected ion
- **em** (*float* or *~numpy.ndarray*, optional) – Emission Measure, for the line-of-sight emission measure ($\int \mathrm{n_e}\,\mathrm{n_H}\,\mathrm{dl}$) ($\mathrm{cm^{-5}}$.), for the volumetric emission measure $\int \mathrm{n_e}\,\mathrm{n_H}\,\mathrm{dV}$ ($\mathrm{cm^{-3}}$).

Variables

- **~ion.IonStr** (*str*) – Name of element plus ion, e.g. *fe_12* for Fe XII
- **~ion.Z** (*int*) – the nuclear charge, 26 for *fe_12*.
- **~ion.Ion** (*int*) – the ionization stage, 12 for *fe_12*.
- **~ion.Dielectronic** (*bool*) – true if the ion is a 'dielectronic' ion where the levels are populated by dielectronic recombination.
- **~ion.Spectroscopic** (*str*) – the spectroscopic notation for the ion, such as *Fe XII* for *fe_12*.
- **~ion.Filename** (*str*) – the complete name of the file *generic* filename in the CHIANTI database, such as *$XUVTOP/fe/fe_12/fe_12*.
- **~ion.Ip** (*float*) – the ionization potential of the ion
- **~ion.FIP** (*float*) – the first ionization potential of the element
- **~ion.Defaults** (*dict*) – these are specified by the software unless a *chiantirc* file is found in '$HOME/.chianti':

Notes

The keyword arguments temperature, eDensity, radTemperature, rStar, em must all be either a float or have the same dimension as the rest if specified as lists, tuples or arrays.

The *Defaults* dict should have the following keys:

- *abundfile*, the elemental abundance file, unless specified in *chiantirc* this defaults to *sun_photospheric_1998_grevesse*.
- *ioneqfile*, the ionization equilibrium file name. Unless specified in 'chiantirc' this is defaults to *chianti*. Other choices are availble in $XUVTOP/ioneq
- *wavelength*, the units of wavelength (Angstroms, nm, or kev), unless specified in the 'chiantirc' this is defaults to 'angstrom'.
- *flux*, specified whether the line intensities are give in energy or photon fluxes, unless specified in the 'chiantirc' this is defaults to *energy*.
- *gui*, specifies whether to use gui selection widgets (True) or to make selections on the command line (False). Unless specified in the 'chiantirc' this is defaults to *False*.

boundBoundLoss(*allLines=1*)

Calculate the summed radiative loss rate for all spectral lines of the specified ion.

Parameters

- **allLines** (*bool*) – If True, include losses from both observed and unobserved lines. If False, only include losses from observed lines.
- **includes elemental abundance and ionization fraction.**

Returns

- *creates the attribute*
- **BoundBoundLoss** (*dict* with the keys below.) – *rate* : the radiative loss rate ($\mathrm{erg\,cm^{-3}\,s^{-1}}$) per unit emission measure.

  temperature : (K).

  eDensity : electron density ($\mathrm{cm^{-3}}$)

`diCross`(*energy=None*, *verbose=False*)
Calculate the direct ionization cross section (cm$^2) as a function of the incident electron energy in eV, puts values into attribute DiCross

Parameters

- **energy** (*array-like*) – incident electron energy in eV
- **verbose** (*bool, int*) – with verbose set to True, printing is enabled

Variables `~ion.diCross.DiCross` (*dict*) – keys: energy, cross

`diRate`(*ngl=20*)
Calculate the direct ionization rate coefficient as a function of temperature (K)

`drPopulate`(*popCorrect=1*, *verbose=0*)
Calculate level populations for specified ion. possible keyword arguments include temperature, eDensity, pDensity, radTemperature and rStar different from method populate() in that it includes the dielectronic recombination from all levels specified by the .auto file - consequently, it also calculates the populations of the higher ionization stage

`drRate`()
Provide the dielectronic recombination rate coefficient as a function of temperature (K).

`drRateLvl`(*verbose=0*)
to calculate the level resolved dielectronic rate from the higher ionization stage to the ion of interest rates are determined from autoionizing A-values the dictionary self.DrRateLvl contains rate = the dielectronic rate into an autoionizing level effRate = the dielectronic rate into an autoionizing level mutilplied by the branching ratio for a stabilizing transition totalRate = the sum of all the effRates

`eaCross`(*energy=None*, *verbose=False*)
Provide the excitation-autoionization cross section.

Energy is given in eV.

`eaDescale`()
Calculates the effective collision strengths (upsilon) for excitation-autoionization as a function of temperature.

`eaRate`()
Calculate the excitation-autoionization rate coefficient.

`emiss`(*allLines=True*)
Calculate the emissivities for lines of the specified ion.

units: ergs s^-1 str^-1

Does not include elemental abundance or ionization fraction

Wavelengths are sorted

set allLines = True to include unidentified lines

emissList(*index=None*, *wvlRange=None*, *wvlRanges=None*, *top=10*, *relative=0*, *outFile=0*)
List the emissivities.

wvlRange, a 2 element tuple, list or array determines the wavelength range

Top specifies to plot only the top strongest lines, default = 10

normalize = 1 specifies whether to normalize to strongest line, default = 0

emissPlot(*index=None*, *wvlRange=None*, *top=10*, *linLog='lin'*, *relative=0*, *verbose=0*, *plotFile=0*, *saveFile=0*)
Plot the emissivities.

wvlRange, a 2 element tuple, list or array determines the wavelength range

Top specifies to plot only the top strongest lines, default = 10

linLog specifies a linear or log plot, want either lin or log, default = lin

normalize = 1 specifies whether to normalize to strongest line, default = 0

emissRatio(*wvlRange=None*, *wvlRanges=None*, *top=10*)
Plot the ratio of 2 lines or sums of lines. Shown as a function of density and/or temperature. For a single wavelength range, set wvlRange = [wMin, wMax] For multiple wavelength ranges, set wvlRanges = [[wMin1,wMax1],[wMin2,wMax2], ...] A plot of relative emissivities is shown and then a dialog appears for the user to choose a set of lines.

gofnt(*wvlRange=None*, *top=10*, *verbose=False*, *plot=True*)
Calculate the 'so-called' G(T) function.

Given as a function of both temperature and eDensity.

Only the top(set by 'top') brightest lines are plotted. the G(T) function is returned in a dictionary self.Gofnt

Note: if the default value for gui is set to False, then it is usually necessary to invoke this method twice to get the desired result.

Keyword Arguments

- **wvlRange** (*array-like*) – the wavelength range to be considered, a two element array-type
- **top** (*int*) – specifies to plot only the top strongest lines, default = 10
- **verbose** (*bool*) – if True, additional information is printed to the console
- **plot** (*bool*) – if True, the G(T) functionis plotted, default - True

intensity(*allLines=1*, *verbose=0*)
Calculate the intensities for lines of the specified ion.

units: ergs cm^-3 s^-1 str^-1

includes elemental abundance and ionization fraction.

the emission measure 'em' is included if specified

intensityRatioInterpolate(*data*, *scale='lin'*, *plot=0*, *verbose=0*)
to take a set of date and interpolate against the IntensityRatio the scale can be one of 'lin'/'linear' [default], 'loglog', 'logx', 'logy',

ionizCross(*energy=None*)
Provides the total ionization cross section.

Notes

uses *diCross* and *eaCross*.

ionizRate(*ngl=20*)
Provides the total ionization rate.

Calls diRate and eaRate.

p2eRatio()
Calculates the proton density to electron density ratio using Eq. 7 of[1].

Notes

Uses the abundance and ionization equilibrium.

References

popPlot(*top=10*, *levels=[]*, *scale=0*, *plotFile=0*, *outFile=0*, *pub=0*, *addTitle=None*, *addLegend=True*)
Plots populations vs temperature or eDensity.

top specifies the number of the most highly populated levels to plot (the default)

or can set levels to an array such as a list to set the desired levels to plot

if scale is set, then the population, if plotted vs. density, is divided by density - only useful if plotting level populations vs density

if pub is set, the want publication plots (bw, lw=2).

if addLegend is set, a matplotlib legend is added

populate(*popCorrect=1*, *verbose=0*)
Calculate level populations for specified ion. possible keyword arguments include temperature, eDensity, pDensity, radTemperature and rStar populate assumes that all of the population in the higher ionization stages exists only in the ground level use drPopulate() for cases where the population of various levels in the higher ionization stage figure into the calculation

recombRate()
Provides the total recombination rate coefficient.

Calls *drRate* and *rrRate*

rrRate()
Provide the radiative recombination rate coefficient as a function of temperature (K).

rrlvlDescale(*verbose=1*)
Interpolate and extrapolate rrlvl rates. Used in level population calculations.

setup(*alternate_dir=None*, *verbose=False*)
Setup various CHIANTI files for the ion including .wgfa, .elvlc, .scups, .psplups, .reclvl, .cilvl, and others.

Parameters

- **alternate_dir** (*str*) – directory cotaining the necessary files for a ChiantiPy ion; use to setup an ion with files not in the current CHIANTI directory
- **verbose** (*bool*)

[1] Young, P. R. et al., 2003, ApJS, 144, 135

Notes

If ion is initiated with *setup=False*, call this method to do the setup at a later point.

setupIonrec(*alternate_dir=None*, *verbose=False*)
Setup method for ion recombination and ionization rates.

Notes

Allows a bare-bones ion object to be setup up with just the ionization and recombination rates. For ions without a complete set of files - one that is not in the MasterList.

spectrum(*wavelength*, *filter=(<function gaussianR>*, *1000.0)*, *label=0*, *allLines=1*)
Calculates the line emission spectrum for the specified ion.

Convolves the results of intensity to make them look like an observed spectrum the default filter is the gaussianR filter with a resolving power of 1000. Other choices include chianti.filters.box and chianti.filters.gaussian. When using the box filter, the width should equal the wavelength interval to keep the units of the continuum and line spectrum the same.

includes ionization equilibrium and elemental abundances

can be called multiple times to use different filters and widths uses label to keep the separate applications of spectrum sorted by the label for example, do .spectrum(.... label = 'test1') and do .spectrum(.... label = 'test2') then will get self.Spectrum.keys() = test1, test2 and self.Spectrum['test1'] = {'intensity':aspectrum, 'wvl':wavelength, 'filter':useFilter.__name__, 'filterWidth':useFactor}

Notes

scipy.ndimage.filters also includes a range of filters.

twoPhoton(*wvl*, *verbose=False*)
to calculate the two-photon continuum - only for hydrogen- and helium-like ions includes the elemental abundance and the ionization equilibrium includes the emission measure if specified discussed in[105]

References

twoPhotonEmiss(*wvl*)
To calculate the two-photon continuum rate coefficient - only for hydrogen- and helium-like ions

twoPhotonLoss()
to calculate the two-photon energy loss rate - only for hydrogen- and helium-like ions includes the elemental abundance and the ionization equilibrium does not include the emission measure

upsilonDescale(*prot=0*)
Provides the temperatures and effective collision strengths (upsilons) set prot for proton rates otherwise, ce will be set for electron collision rates uses the new format "scups" files

[105] Young et al., 2003, ApJSS, 144, 135

ChiantiPy.core.Ioneq module

Ionization equilibrium class

class ChiantiPy.core.Ioneq.**ioneq**(*el_or_z*)
Bases: object

Calculation ion fractions as a function of temperature assuming ionization equilibrium.

Parameters **el_or_z** (*int* or *str*) – Atomic number or symbol

---

Note: When either loading or calculating a set of ion fractions, the temperature and ion fractions are returned to the *Temperature* and *Ioneq* attributes, respectively.

---

calculate(*temperature*)
Calculate ion fractions for given temperature array using the total ionization and recombination rates.

load(*ioneqName=None*)
Read temperature and ion fractions from a CHIANTI ".ioneq" file.

plot(*stages=0*, *tRange=0*, *yr=0*, *oplot=False*, *label=1*, *title=1*, *bw=False*, *semilogx=0*, *verbose=0*)
Plots the ionization equilibria.

self.plot(tRange=None, yr=None, oplot=False) stages = sequence of ions to be plotted, neutral == 1, fully stripped == Z+1 tRange = temperature range, yr = ion fraction range

for overplotting: oplot="ioneqfilename" such as 'mazzotta' or if oplot=True or oplot=1 and a widget will come up so that a file can be selected. bw, if True, the plot is made in black and white

plotRatio(*stageN*, *stageD*, *tRange=0*, *yr=0*, *label=1*, *title=1*, *bw=0*, *semilogx=1*, *verbose=0*)
Plots the ratio of the ionization equilibria of two stages of a given element

self.plotRatio(stageN, stageD) stages = sequence of ions to be plotted, neutral == 1, fully stripped == Z+1 stageN = numerator stageD = denominator tRange = temperature range, yr = ion fraction range

ChiantiPy.core.IpyMspectrum module

ChiantiPy.core.IpyMspectrum.**doAll**(*inpt*)
to process ff, fb and line inputs

class ChiantiPy.core.IpyMspectrum.**ipymspectrum**(*temperature*, *eDensity*, *wavelength*, *filter=(<function gaussianR>, 1000.0)*, *label=None*, *elementList=None*, *ionList=None*, *minAbund=None*, *keepIons=0*, *doLines=1*, *doContinuum=1*, *allLines=1*, *em=None*, *abundance=None*, *verbose=0*, *timeout=0.1*)
Bases: *ChiantiPy.base._IonTrails.ionTrails*, *ChiantiPy.base._SpecTrails.specTrails*

this is the multiprocessing version of spectrum for using inside an IPython Qtconsole or notebook.

be for creating an instance, it is necessary to type something like the following into a console

> ipcluster start –n=3

this is the way to invoke things under the IPython 6 notation

Calculate the emission spectrum as a function of temperature and density.

temperature and density can be arrays but, unless the size of either is one (1), the two must have the same size

the returned spectrum will be convolved with a filter of the specified width on the specified wavelength array

the default filter is gaussianR with a resolving power of 100. Other filters, such as gaussian, box and lorentz, are available in ChiantiPy.filters. When using the box filter, the width should equal the wavelength interval to keep the units of the continuum and line spectrum the same.

A selection of elements can be make with elementList a list containing the names of elements that are desired to be included, e.g., ['fe','ni']

A selection of ions can be make with ionList containing the names of the desired lines in Chianti notation, i.e. C VI = c_6

Both elementList and ionList can not be specified at the same time

a minimum abundance can be specified so that the calculation can be speeded up by excluding elements with a low abundance. With solar photospheric abundances -

setting minAbund = 1.e-4 will include H, He, C, O, Ne setting minAbund = 2.e-5 adds N, Mg, Si, S, Fe setting minAbund = 1.e-6 adds Na, Al, Ar, Ca, Ni

Setting em will multiply the spectrum at each temperature by the value of em.

em [for emission measure], can be a float or an array of the same length as the temperature/density. allLines = 1 will include lines with either theoretical or observed wavelengths. allLines=0 will include only those lines with observed wavelengths

timeout - a small but non-zero value seems to be necessary

ChiantiPy.core.Mspectrum module

class ChiantiPy.core.Mspectrum.**mspectrum**(*temperature*, *eDensity*, *wavelength*, *filter=(<function gaussianR>*, *1000.0)*, *label=0*, *elementList=None*, *ionList=None*, *minAbund=None*, *keepIons=0*, *abundance=None*, *doLines=1*, *doContinuum=1*, *allLines=1*, *em=None*, *proc=3*, *verbose=0*, *timeout=0.1*)

Bases: *ChiantiPy.base._IonTrails.ionTrails*, *ChiantiPy.base._SpecTrails.specTrails*

this is the multiprocessing version of spectrum set proc to the desired number of processors, default=3

Calculate the emission spectrum as a function of temperature and density.

temperature and density can be arrays but, unless the size of either is one (1), the two must have the same size

the returned spectrum will be convolved with a filter of the specified width on the specified wavelength array

the default filter is gaussianR with a resolving power of 100. Other filters, such as gaussian, box and lorentz, are available in chianti.filters. When using the box filter, the width should equal the wavelength interval to keep the units of the continuum and line spectrum the same.

A selection of elements can be make with elementList a list containing the names of elements that are desired to be included, e.g., ['fe','ni']

A selection of ions can be make with ionList containing the names of the desired lines in Chianti notation, i.e. C VI = c_6

Both elementList and ionList can not be specified at the same time

a minimum abundance can be specified so that the calculation can be speeded up by excluding elements with a low abundance. With solar photospheric abundances -

setting minAbund = 1.e-4 will include H, He, C, O, Ne setting minAbund = 2.e-5 adds N, Mg, Si, S, Fe setting minAbund = 1.e-6 adds Na, Al, Ar, Ca, Ni

Setting em will multiply the spectrum at each temperature by the value of em.

em [for emission measure], can be a float or an array of the same length as the temperature/density.

allLines = 1 will include lines with either theoretical or observed wavelengths. allLines=0 will include only those lines with observed wavelengths

proc, the number of processors to use

timeout, a small but non-zero value seems to be necessary keepIons: set this to keep the ion instances that have been calculated in a dictionary self.IonInstances with the keywords being the CHIANTI-style ion names

abundance: to select a particular set of abundances, set abundance to the name of a CHIANTI abundance file, without the '.abund' suffix, e.g. 'sun_photospheric_1998_grevesse'

If set to a blank (''), a gui selection menu will popup and allow the selection of an set of abundances

Parameters

- **temperature** (*float*, *list*, *ndarray*) – the temperature(s) in K
- **eDensity** (*float, ndarray*) – eDensity: electron density in cm^{-3}
- **wavelength** (*list* or *ndarray*) – wavelength: array of wavelengths, generally in Angstroms
- **elementList** (*list*) – elementList: list of elements to include, such as 'fe', 'ne', 's'
- **ionList** (*list*) – ionList: list of ions to include, such as 'fe_16', 'ne_10'
- **minAbund** (*float*) – minAbund: minimum abundance (relative to H) to include
- **doLines** (`bool1) – doLines: if true, line intensities are calculated
- **doContinuum** (*bool*) – doContinuum: if true, continuum intensities are calculated only if wavelengths are in angstroms
- **keepIons** (*bool*) –

  keepIons: keep the ion instances used in the calculation should be used with caution otherwise the bunch instance can become quite large
- **em** (*float*, *list*, *ndarray*) – em: the emission measure - the integral of the electron density times the hydrogen density along the line of sight
- **abundance** (*str*) –

  abuncance: the file name of the abuncance set to be used must be one in the $XUVTOP/abund directory
- **allLInes** (*bool*) – allLines: whether or not to include unobserved lines
- **verbose** (*bool*) – verbose: whether to allow certain print statements

ChiantiPy.core.RadLoss module

class ChiantiPy.core.RadLoss.**radLoss**(*temperature*, *eDensity*, *elementList=None*, *ionList=None*, *minAbund=None*, *doContinuum=True*, *doLines=True*, *abundance=None*, *verbose=0*, *allLines=1*)

Bases: *ChiantiPy.base._IonTrails.ionTrails*, *ChiantiPy.base._SpecTrails.specTrails*

Calculate the radiative emission loss rate as a function of temperature and density.

includes elemental abundances or ionization equilibria

temperature and density can be arrays but, unless the size of either is one (1), the two must have the same size

A selection of ions can be make with ionList containing the names of the desired lines in Chianti notation, i.e. C VI = c_6

a minimum abundance can be specified so that the calculation can be speeded up by excluding elements with a low abundance. With solar photospheric abundances -

setting minAbund = 1.e-4 will include H, He, C, O, Ne setting minAbund = 2.e-5 adds N, Mg, Si, S, Fe setting minAbund = 1.e-6 adds Na, Al, Ar, Ca, Ni

Setting em will multiply the spectrum at each temperature by the value of em.

em [for emission measure], can be a float or an array of the same length as the temperature/density.

abundance: to select a particular set of abundances, set abundance to the name of a CHIANTI abundance file, without the '.abund' suffix, e.g. 'sun_photospheric_1998_grevesse' If set to a blank (''), a gui selection menu will popup and allow the selection of an set of abundances

`radLossPlot`(*doTitle=False*)
to plot the radiative losses vs temperature

Parameters **doTitle** (*bool*) – if True, a title is applied to the plot. The default is for no title

ChiantiPy.core.MradLoss module

`class` ChiantiPy.core.MradLoss.`mradLoss`(*temperature*, *eDensity*, *elementList=None*, *ionList=None*, *minAbund=None*, *doContinuum=True*, *doLines=True*, *abundance=None*, *verbose=0*, *allLines=1*, *proc=4*)

Bases: *ChiantiPy.base._IonTrails.ionTrails*, *ChiantiPy.base._SpecTrails.specTrails*

Calculate the radiative emission loss rate as a function of temperature and density.

this is the multiprocessing version of radloss

includes elemental abundances or ionization equilibria

temperature and density can be arrays but, unless the size of either is one (1), the two must have the same size

A selection of ions can be make with ionList containing the names of the desired lines in Chianti notation, i.e. C VI = c_6

a minimum abundance can be specified so that the calculation can be speeded up by excluding elements with a low abundance. With solar photospheric abundances

setting minAbund = 1.e-4 will include H, He, C, O, Ne setting minAbund = 2.e-5 adds N, Mg, Si, S, Fe setting minAbund = 1.e-6 adds Na, Al, Ar, Ca, Ni

Setting em will multiply the spectrum at each temperature by the value of em.

em [for emission measure], can be a float or an array of the same length as the temperature/density.

abundance: to select a particular set of abundances, set abundance to the name of a CHIANTI abundance file, without the '.abund' suffix, e.g. 'sun_photospheric_1998_grevesse' If set to a blank (''), a gui selection menu will popup and allow the selection of an set of abundances

Parameters

- **temperature** (*float*, *list*, *ndarray*) – the temperature(s) in K
- **eDensity** (*float, ndarray*) – eDensity: electron density in cm^{-3}
- **elementList** (*list*) – elementList: list of elements to include, such as 'fe', 'ne', 's'
- **ionList** (*list*) – ionList: list of ions to include, such as 'fe_16', 'ne_10'

- **minAbund** (*float*) – minAbund: minimum abundance (relative to H) to include
- **doLines** (`bool1) – doLines: if true, line intensities are calculated
- **doContinuum** (*bool*) – doContinuum: if true, continuum intensities are calculated only if wavelengths are in angstroms
- **abundance** (*str*) –

  abuncance: the file name of the abuncance set to be used must be one in the $XUVTOP/abund directory
- **allLInes** (*bool*) – allLines: whether or not to include unobserved lines
- **verbose** (*bool*) – verbose: whether to allow certain print statements

`radLossPlot`(*title=0*)
to plot the radiative losses vs temperature

ChiantiPy.core.Bunch module

`class` ChiantiPy.core.Bunch.**bunch**(*temperature*, *eDensity*, *wvlRange*, *elementList=None*, *ionList=None*, *minAbund=None*, *keepIons=False*, *em=None*, *abundance=None*, *verbose=False*, *allLines=True*)

Bases: *ChiantiPy.base._IonTrails.ionTrails*, *ChiantiPy.base._SpecTrails.specTrails*

Calculate the emission line spectrum as a function of temperature and density.

'bunch' is very similar to 'spectrum' except that continuum is not calculated and the spectrum is not convolved over a filter. However, this can be done with the inherited convolve method

one of the convenient things is that all of the instantiated ion classes, determined through such keywords as 'elementList', 'ionList', and 'minAbund' are kept in a dictionary self.IonInstances where self.IonInstances['mg_7'] is the class instance of ChiantiPy.core.ion for 'mg_7'. All its methods and attributes are available.

includes elemental abundances and ionization equilibria

the set of abundances, a file in $XUVTOP/abundance, can be set with the keyword argument 'abundanceName'

temperature and density can be arrays but, unless the size of either is one (1), the two must have the same size

Inherited methods include 'intensityList', 'intensityRatio' (between lines of different ions), and 'intensityRatioSave' and 'convolve'.

A selection of elements can be make with elementList a list containing the names of elements that are desired to be included, e.g., ['fe','ni']

A selection of ions can be make with ionList containing the names of the desired lines in Chianti notation, i.e. C VI = c_6

Both elementList and ionList can not be specified at the same time

a minimum abundance can be specified so that the calculation can be speeded up by excluding elements with a low abundance. With solar photospheric abundances -

setting minAbund = 1.e-4 will include H, He, C, O, Ne setting minAbund = 2.e-5 adds N, Mg, Si, S, Fe setting minAbund = 1.e-6 adds Na, Al, Ar, Ca, Ni

At least one of elementList, ionList, or minAbund must be set in order for 'bunch' to include any ions.

Setting em will multiply the spectrum at each temperature by the value of em.

em [for emission measure], can be a float or an array of the same length as the temperature/density

Parameters

- **temperature** (*float*, *list*, *ndarray*) – the temperature(s) in K
- **eDensity** (*float, ndarray*) – eDensity: electron density in cm^{-3}
- **wvlRange** (2 element *list* or *ndarray*) – wvlRange: range of wavelengths to consider, generally in angstroms
- **elementList** (*list*) – elementList: list of elements to include, such as 'fe', 'ne', 's'
- **ionList** (*list*) – ionList: list of ions to include, such as 'fe_16', 'ne_10'
- **minAbund** (*float*) – minAbund: minimum abundance (relative to H) to include
- **keepIons** (*bool*) –

  keepIons: keep the ion instances used in the calculation should be used with caution otherwise the bunch instance can become quite large
- **em** (*float*, *list*, *ndarray*) – em: the emission measure
- **abundance** (*str*) –

  abuncance: the file name of the abuncance set to be used must be one in the $XUVTOP/abund directory
- **allLInes** (*bool*) – allLines: whether or not to include unobserved lines
- **verbose** (*bool*) – verbose: whether to allow certain print statements

`convolve`(*wavelength=None*, *filter=(<function gaussianR>*, *1000.0)*, *label=None*, *verbose=False*)

the first application of spectrum calculates the line intensities within the specified wavelength range and for set of ions specified

wavelength will not be used if applied to 'spectrum' objects

wavelength IS need for 'bunch' objects - in this case, the wavelength should not extend beyond the limits of the wvlRange used for the 'bunch' calculation

Keyword Arguments

- **`wavelength`** ('int', *list*) – if an *int*, the attribute 'Wavelength' is looked for otherwise, wavelength is used
- **`filter`** (*tuple*) – first elements if one of the ChiantiPy.tools.filters object second element is the width appropriate to the filter
- **`label`** (*str*) – if set, creates a Spectrum[label] attribute
- **`verbose`** (*bool*) – if True, prints info to the terminal

ChiantiPy.core.Spectrum module

`class` ChiantiPy.core.Spectrum.`spectrum`(*temperature*, *eDensity*, *wavelength*, *filter=(<function gaussianR>*, *1000.0)*, *label=None*, *elementList=None*, *ionList=None*, *minAbund=None*, *doLines=1*, *doContinuum=1*, *em=None*, *keepIons=0*, *abundance=None*, *verbose=0*, *allLines=1*)

Bases: *ChiantiPy.base._IonTrails.ionTrails*, *ChiantiPy.base._SpecTrails.specTrails*

Calculate the emission spectrum as a function of temperature and density.

one of the convenient things is that all of the instantiated ion classes, determined through such keywords as 'elementList', 'ionList', and 'minAbund' are kept in a dictionary self.IonInstances where self.IonInstances['mg_7'] is the class instance of ChiantiPy.core.ion for 'mg_7'. All its methods and attributes are available.

includes elemental abundances and ionization equilibria

the set of abundances, a file in $XUVTOP/abundance, can be set with the keyword argument 'abundanceName'

temperature and density can be arrays but, unless the size of either is unity (1), the two must have the same size

the returned spectrum will be convolved with a filter of the specified width on the specified wavelength array

the default filter is gaussianR with a resolving power of 1000. Other filters, such as gaussian, box and lorentz, are available in ChiantiPy.tools.filters. When using the box filter, the width should equal the wavelength interval to keep the units of the continuum and line spectrum the same.

Inherited methods include 'intensityList', 'intensityRatio' (between lines of different ions), 'intensityRatioSave' and 'convolve'

A selection of elements can be make with elementList a list containing the names of elements that are desired to be included, e.g., ['fe','ni']

A selection of ions can be make with ionList containing the names of the desired lines in CHIANTI notation, i.e. C VI = c_6

Both elementList and ionList can not be specified at the same time

a minimum abundance can be specified so that the calculation can be speeded up by excluding elements with a low abundance. The default of minAbund is 1.e-6

It is necessary to specify at least an elementList, an ionList, or a minAbund to select any ions for a spectrum calculation

With solar photospheric abundances

setting minAbund = 1.e-4 will include H, He, C, O, Ne

setting minAbund = 2.e-5 adds N, Mg, Si, S, Fe

setting minAbund = 1.e-6 adds Na, Al, Ar, Ca, Ni

Setting doLines = 0 will skip the calculation of spectral lines. Setting doContinuum =0 will skip the continuum calculation.

Setting em [for emission measure] will multiply the spectrum at each temperature by the value of em.

em [for emission measure] can be a float or an array of the same length as the temperature/density

keepIons set this to keep the ion instances that have been calculated in a dictionary self.IonInstances with the keywords being the CHIANTI-style ion names

abundance - to select a particular set of abundances, set abundance to the name of a CHIANTI abundance file, without the '.abund' suffix, e.g. 'sun_photospheric_1998_grevesse'

If set to a blank (''), a gui selection menu will popup and allow the selection of an set of abundances

Parameters

- **temperature** (*float*, *list*, *ndarray*) – the temperature(s) in K
- **eDensity** (*float, ndarray*) – eDensity: electron density in cm^{-3}
- **wavelength** (*list* or *ndarray*) – wavelength: array of wavelengths, generally in Angstroms
- **elementList** (*list*) – elementList: list of elements to include, such as 'fe', 'ne', 's'
- **ionList** (*list*) – ionList: list of ions to include, such as 'fe_16', 'ne_10'

- **minAbund** (*float*) – minAbund: minimum abundance (relative to H) to include
- **doLines** (`bool1) – doLines: if true, line intensities are calculated
- **doContinuum** (*bool*) – doContinuum: if true, continuum intensities are calculated only if wavelengths are in angstroms
- **keepIons** (*bool*) –

  keepIons: keep the ion instances used in the calculation should be used with caution otherwise the bunch instance can become quite large
- **em** (*float*, *list*, *ndarray*) – em: the emission measure
- **abundance** (*str*) –

  abuncance: the file name of the abuncance set to be used must be one in the $XUVTOP/abund directory
- **allLInes** (*bool*) – allLines: whether or not to include unobserved lines
- **verbose** (*bool*) – verbose: whether to allow certain print statements

Module contents

chianti.core - contains the main classes for ChiantiPy users.

This software is distributed under the terms of the ISC Software License that is found in the LICENSE file

ChiantiPy.fortranformat package

Submodules

ChiantiPy.fortranformat.FortranRecordReader module

class ChiantiPy.fortranformat.FortranRecordReader.**FortranRecordReader**(*format*)

Bases: `object`

Generate a reader object for FORTRAN format strings

Typical use case ...

```
>>> header_line = FortranRecordReader('(A15, A15, A15)')
>>> header_line.read('              x              y              z')
['              x', '              y', '              z']
>>> line = FortranRecordReader('(3F15.3)')
>>> line.read('          1.000          0.000          0.500')
[1.0, 0.0, 0.5]
>>> line.read('          1.100          0.100          0.600')
[1.1, 0.1, 0.6]
```

Note: it is best to create a new object for each format, changing the format causes the parser to reevalute the format string which is costly in terms of performance

property format

get_format()

match(*record*)

read(*record*)
Pass a string representing a FORTRAN record to obtain the relevent values

set_format(*format*)

ChiantiPy.fortranformat.FortranRecordWriter module

class ChiantiPy.fortranformat.FortranRecordWriter.**FortranRecordWriter**(*format*)
Bases: object

Generate a writer object for FORTRAN format strings

Typical use case ...

```
>>> header_line = FortranRecordWriter('(A15, A15, A15)')
>>> header_line.write(['x', 'y', 'z'])
'              x               y               z'
>>> line = FortranRecordWriter('(3F15.3)')
>>> line.write([1.0, 0.0, 0.5])
'          1.000          0.000          0.500'
>>> line.write([1.1, 0.1, 0.6])
'          1.100          0.100          0.600'
```

Note: it is best to create a new object for each format, changing the format causes the parser to reevalute the format string which is costly in terms of performance

property format

get_format()

set_format(*format*)

write(*values*)
Pass a list of values correspoding to the FORTRAN format specified to generate a string

ChiantiPy.fortranformat.config module

ChiantiPy.fortranformat.config.**reset**()

Module contents

ChiantiPy.model package

Submodules

ChiantiPy.model.Maker module

classes and methods to analyze observations of astrophysical spectra

ChiantiPy.model.Maker.**doDemGofntQ**(*inQueue*, *outQueue*)
helper for multiprocessing with maker.mgofnt()

ChiantiPy.model.Maker.**emPlot**(*matchDict*, *vs='T'*, *loc='upper right'*, *fs=10*, *adjust=None*, *position='both'*, *legend=True*, *verbose=0*)

to plot line intensities divided by gofnt adjust is to provide an adjustment to the position of the labels position : one of 'both', 'right', 'left', or 'none'

Keyword Arguments

- **vs** (*str*, either 'T', or 'D') – whether to plot the emission measure vs temperature or density
- **loc** (*str*) – matplotlib argument for plt.legend
- **fs** (*int*) – the fontsize for the legend
- **adjust** (*list*) – a list of multiplicative adjustments to the labels to the plot lines must be the same length as the number of lines
- **position** (*str*) – where the labels to the lines should be placed, *both* for both ends, *left* for the left size only, 'right' for the right side only, or None for no labels
- **label** (*bool*) – whether to apply
- **legend** (*bool*) – whether to include a matplotlib legend
- **fontsize** (*int*) – fontsize for the matplotlib xlabel and ylabel
- **tscale** (*float*) – scale the temperature by dividing by tscale
- **verbose** (*bool*) – if True, additional output is sent to the terminal

class ChiantiPy.model.Maker.**maker**(*specData*, *temperature=None*, *eDensity=None*, *elementList=[]*, *ionList=[]*, *allLines=False*, *abundanceName=None*, *minAbund=10.0*, *wghtFactor=None*, *verbose=False*)

Bases: *ChiantiPy.base._IonTrails.ionTrails*, *ChiantiPy.base._SpecTrails.specTrails*

a class matching observed lines to lines in the CHIANTI database

Parameters **specData** (*dict*) – contains the following keys intensity - a list of observed line intensities wvlObs - a list of observed wavelengths, usually in Angstroms dwvl the expected wavelength different between the observed wvl and CHIANTI

Keyword Arguments

- **temperature** (*float*, *list*, *ndarray*) – the temperature(s) in K
- **eDensity** (*float, ndarray*) – eDensity: electron density in cm^{-3}
- **elementList** (*list*) – a list of elements, such as fe, to be searched
- **ionList** (*list*) – a list of ions, such as fe_14, to be searched
- **allLines** (*bool*) – if true, unobserved lines in CHIANTI are included in the search
- **abundanceName** (*str*) – the name of the elemental abundance file to be used, if not set, the default abundance file is used
- **minAbund** (*float*) – sets the minimum abundance for an element to be included in the search
- **verbose** (*bool*) – if True, additional output is sent to the terminal

argCheck(*temperature=None*, *eDensity=None*, *pDensity='default'*, *verbose=False*)

to check the compatibility of the three arguments and put them into numpy arrays of atleast_1d

Keyword Arguments

- **temperature** (*float*, *list*, *ndarray*) – the temperature(s) in K
- **eDensity** (*float, ndarray*) – eDensity: electron density in cm^{-3}

- **pDensity** (*str*, *float*, *ndarray*) – pDensity: proton density in cm^{-3} defaults to 'default'
- **verbose** (*bool*) – if True, additional output is sent to the terminal

diff(*sort=None*, *verbose=False*)
calculates the weighted and straight differences between observed and predicted creates an attribute self.Dict, a dict with the following keys: 'wvl' = observed wavelength (A) 'relDiff' = (I_obs - I_pred)/(I_obs) 'ionS' the CHIANTI type name for an ion sort be either of none, 'wvl', or 'ion'

Keyword Arguments **sort** (*str* or None) – whether the output should be sorted by *wvl* or *ion* or not

diffPlot(*title=False*, *loc='upper right'*, *fontsize=16*, *figsize=[7.0, 5.0]*)

Parameters

- **title** (*bool*) – whether to plot the title or not
- **fontsize** (*int*) – fontsize for matplotlib plots
- **figsize** (*2d list, ndarray*) – the figure size for the plot

Variables **~maker.diffPlot.DiffPlot** (*dict*) – contains the fig, ax matplotlib objects created

diffPrint(*filename='diffPrint.txt'*, *sort=None*)
calculates the weighted and straight differences between observed and predicted prints the values and saves to a file also created a attribute self.Dict, a dict with the following keys: 'wvl' = observed wavelength (A) 'relDiff' = (I_obs - I_pred)/(I_obs) 'ionS' the CHIANTI type name for an ion

Keyword Arguments

- **filename** (*str*) – the filename where the text should be output
- **sort** (*str*, can be *wvl*, *ion*, or None) – whether the output should be sorted by *wvl* or *ion* or not

emFitPlot()
to plot the emission measures derived from a chi-squared search over temperature

emMake(*filename*, *reference*)
to make a CHIANTI style emission measure file outName does not need the suffix .em reference should be a list of references

Parameters

- **filename** (*str*) – the name of the em file to be produced
- **reference** (*list*) – a list of strings providing a reference at the end of the em file

emPlot(*vs='T'*, *loc='upper right'*, *fs=10*, *adjust=None*, *position='both'*, *label=True*, *legend=True*, *fontsize=16*, *tscale=1.0*, *verbose=True*)
to plot line intensities divided by gofnt adjust is to provide an adjustment to the position of the labels position : one of 'both', 'right', 'left', or 'none'

Keyword Arguments

- **vs** (*str*, either 'T', or 'D') – whether to plot the emission measure vs temperature or density
- **loc** (*str*) – matplotlib argument for plt.legend
- **fs** (*int*) – the fontsize for the legend
- **adjust** (*list*) – a list of multiplicative adjustments to the labels to the plot lines must be the same length as the number of lines

- **position** (*str*) – where the labels to the lines should be placed, *both* for both ends, *left* for the left size only, 'right' for the right side only, or None for no labels
- **label** (*bool*) – whether to apply
- **legend** (*bool*) – whether to include a matplotlib legend
- **fontsize** (*int*) – fontsize for the matplotlib xlabel and ylabel
- **tscale** (*float*) – scale the temperature by dividing by tscale
- **verbose** (*bool*) – if True, additional output is sent to the terminal

emPlotObj(*vs='T'*, *loc='upper right'*, *fs=10*, *adjust=None*, *position='both'*, *label=True*, *legend=True*, *fontsize=16*, *figsize=[7.0, 5.0]*, *tscale=1.0*, *verbose=True*)

the emPlot using the object oriented version of matplotlib - a figure and axis objects are returned to plot line intensities divided by gofnt adjust is to provide an adjustment to the position of the labels position : one of 'both', 'right', 'left', or 'none' this uses the modern object interface fig, ax = plt.subplots(figsize=figsize)

Keyword Arguments

- **vs** (*str*, either 'T', or 'D') – whether to plot the emission measure vs temperature or density
- **loc** (*str*) – matplotlib argument for plt.legend
- **fs** (*int*) – the fontsize for the legend
- **adjust** (*list*) – a list of multiplicative adjustments to the labels to the plot lines must be the same length as the number of lines
- **position** (*str*) – where the labels to the lines should be placed, *both* for both ends, *left* for the left size only, 'right' for the right side only, or None for no labels
- **label** (*bool*) – whether to apply
- **legend** (*bool*) – whether to include a matplotlib legend
- **fontsize** (*int*) – fontsize for the matplotlib xlabel and ylabel
- **figsize** (two element *list* or *ndarray*) – sets the figure size when using matplotlib subplots to initiate the object style plotting
- **tscale** (*float*) – scale the temperature by dividing by tscale
- **verbose** (*bool*) – if True, additional output is sent to the terminal

emSet(*value*)

sets the EM values for a N temperature EM distribution

Parameters **value** (*list*, *ndarray*) – the values of the emission measure to be used when the intensities are predicted

emSetIndices(*indices*, *add=0.0*, *verbose=0*)

to set the indices of the N temperature/density EM distribution can increase the number of paramaters if additional parameters have been used

Parameters **indices** (*list*, *ndarray*) – the indices of the temperature/density arrays for which a set of intensities will be predicted

Keyword Arguments

- **add** (*float*) – to increase the number of parameters used in the calculation of the reduced chi-squared
- **verbose** (*bool*) – if True, additional output is sent to the terminal

findMinMaxIndices(*verbose=0*)
to find the minimum and maximum indices where all match['intensitySum'] are greater than 0

Keyword Arguments **verbose** (*bool*) – if True, additional output is sent to the terminal

fit1t(*initialValue*, *maxfev=0*)
calls leastsq to fit the 1t (single temperature) model used in search1dspace

Parameters **initialValue** (*float*) – the initial value to start the leastsq process

Todo: see if this can be replaced by fitFunct1t or fitNt

fitFunc1t(*em*)
the fitting function for the isothermal model to be called by leastsq called by fit1t

Parameters **em** (*number*) – the log10 value of the emission measure

Returns **weighted chisquared**

Return type float

Todo: see if this can be replaced by fitFuncNt

fitFuncNt(*value*)
the fitting function for the multiple temperature model to be called by leastsq called by fitNt

Parameters **value** (*list*) – the initial values for the em fit

fitNt(*initialValue*, *maxfev=0*)
calls leastsq to fit the multi temperature models called by search2tSpace, search3tSpace etc

Parameters **initialValue** (*list*) – the initial trial value for the emission measure (log1))

Keyword Arguments **maxfev** (*int*) – not sure it is needed

getChisq()
return the weighted chi-squared

getNormalizedChisq()
return normalized chisq: chi-squared divided by the number of observed lines

getWeightedDiff()
to calculated the weighted difference of each of the intensities returns a 1D array

gofnt(*temperature*, *density*, *verbose=1*)
calculate the gofnt function for each of the matched lines do each ion only once

Parameters

- **temperature** (*float*, *list*, *ndarray*) – the temperature(s) in K
- **density** (*float*, *list*, *ndarray*) – density: electron density in cm^{-3}

loadMatch(*filename*)
to open a pickle file, return the match data and make it an attribute

loadSearchData(*filename*)
to load the pickled search data as an attribute self.SearchData

Keyword Arguments **filename** (*str*) – the filename of the pickle file where the search data has been created

makeMatch(*verbose=False*)
to match the CHIANTI lines with the input specdata uses ionTrails.ionGate to sort through ions

Keyword Arguments **verbose** (*bool*) – if True, additional output is sent to the terminal

mgofnt(*temperature*, *density*, *proc=6*, *timeout=0.1*, *verbose=0*)
calculate the gofnt function for each of the matched lines this is the multiprocessing version does each ion only once

Parameters

- **temperature** (*float*, *list*, *ndarray*) – the temperature(s) in K
- **density** (*float*, *list*, *ndarray*) – density: electron density in cm^{-3}

Keyword Arguments

- **proc** (*int*) – the number of cores to be used
- **timeout** (*'float'*) – may not actually be necessary
- **verbose** (*bool*) – if True, additional output is sent to the terminal

predict()
to predict the intensities of the observed lines from an emission measure the emission measure is already specified as self.Em which is an ndarray the temperatures are set by emSetIndices

predictPrint(*minContribution=0.1*, *filename='predictPrint.txt'*, *sort=None*, *verbose=0*)
to predict the intensities of the observed lines from an emission measure the emission measure is already specified as self.Em which is an np array sort can be 'wvl' or 'ion', otherwise, there is no sorting done

Keyword Arguments

- **minContribution** (*float*) – the minimum contribution a blend must supply to be included in the text output
- **filename** (*str*) – the filename where the text should be output
- **sort** (*str*, can be *wvl*, *ion*, or None) – whether the output should be sorted by *wvl* or *ion* or not
- **verbose** (*bool*) – if True, additional output is sent to the terminal

predictPrint1d(*minContribution=0.1*, *filename='predictPrint1d.txt'*, *verbose=False*)
to predict the intensities of the observed lines from an emission measure the emission measure is already specified as self.Em which is an np array

to be used after a 1d search over density

Keyword Arguments

- **minContribution** (*float*) – the minimum contribution a blend must supply to be included in the text output
- **filename** (*str*) – the filename where the text should be output
- **verbose** (*bool*) – if True, additional output is sent to the terminal

saveMatch(*filename*)
to save the attribute Match to a pickle file so that it can be reloaded later

Keyword Arguments **filename** (*str*) – the filename where the text should be output

saveSearchData(*filename*)
to save the attribute SearchData to a pickle file

Keyword Arguments **filename** (*str*) – the filename of the pickle file where the search data is to be created

search1dSpace(*initialEm*, *indxlimits=None*, *verbose=False*, *log=False*, *maxfev=0*)
to conduct a brute force search over electron density for an isothermal-space and find the best fit to the em and density indxlimits give the range of indices to fit over can use self.MinIndex and self.MaxIndex+1 initialEm = log value of the emission measure to begin the searching

Parameters **initialEm** (*float*) – the initial trial value for the log10 emission measure

Keyword Arguments

- **indxlimits** (*list*, None) – the range of indices of the density array to search if None is specified, the whole range is searched
- **verbose** (*bool*) – if True, additional output is sent to the terminal
- **log** (*bool*) – if True, a log file is created - 'search1d.raw'

search1tEmSpace(*verbose=0*)
to find the value of chisq as a function of Em with T = best-fit

Keyword Arguments **verbose** (*bool*) – if True, additional output is sent to the terminal

search2tSpace(*initial*, *indxlimits=None*, *verbose=0*, *log=0*, *maxfev=0*)
to conduct a brute force search of 2 temperature space and find the best fit indxlimits give the range of indices to fit over

Parameters **initial** (*list*) – the initial trial values (2) for the log10 emission measure

Keyword Arguments

- **indxlimits** (*list*, None) – the range of indices of the density array to search if None is specified, the whole range is searched
- **verbose** (*bool*) – if True, additional output is sent to the terminal
- **log** (*bool*) – if True, a log file is created - 'search1d.raw'

search3tSpace(*initial*, *indxlimits=None*, *verbose=0*, *log=0*)
to conduct a brute force search of 3 temperature space and find the best fit

Parameters **initial** (*list*) – the initial trial values (3) for the log10 emission measure

Keyword Arguments

- **indxlimits** (*list*, None) – the range of indices of the density array to search if None is specified, the whole range is searched
- **verbose** (*bool*) – if True, additional output is sent to the terminal
- **log** (*bool*) – if True, a log file is created - 'search1d.raw'

search4tSpace(*initial*, *indxlimits=None*, *verbose=0*, *log=0*)
to conduct a brute force search of 4 temperature space and find the best fit set log to create a log file of the iterations rather that outputting to the jupyter/ipython session

Parameters **initial** (*list*) – the initial trial values (4) for the log10 emission measure

Keyword Arguments

- **indxlimits** (*list*, None) – the range of indices of the density array to search if None is specified, the whole range is searched
- **verbose** (*bool*) – if True, additional output is sent to the terminal
- **log** (*bool*) – if True, a log file is created - 'search1d.raw'

Module contents

ChiantiPy.tests package

Submodules

ChiantiPy.tests.setup_package module

ChiantiPy.tests.setup_package.**get_package_data**()

Module contents

This packages contains affiliated package tests.

ChiantiPy.tools package

Submodules

ChiantiPy.tools.archival module

Functions for reading pre-v8 CHIANTI files

ChiantiPy.tools.archival.**elvlcRead**(*ions*, *filename=0*, *verbose=0*, *useTh=0*)
read a chianti energy level file and returns {"lvl":lvl,"conf":conf,"term":term,"spin":spin,"l":l,"spd":spd,"j":j ,"mult":mult,"ecm":ecm,"eryd":eryd,"ecmth":ecmth,"erydth":erydth,"ref":ref,"pretty":pretty, 'ionS':ions} if a energy value for ecm or eryd is zero(=unknown), the theoretical values (ecmth and erydth) are inserted

ChiantiPy.tools.archival.**elvlcWrite**(*info*, *outfile=None*, *addLvl=0*)
Write Chianti data to .elvlc file.

Parameters

- **info** (*dict*) – Information about the Chianti data to write. Should contain the following keys: ionS, the Chianti style name of the ion such as c_4 conf, an integer denoting the configuration - not too essential term, a string showing the configuration spin, an integer of the spin of the state in LS coupling l, an integer of the angular momentum quantum number spd, an string for the alphabetic symbol of the angular momemtum, S, P, D, etc j, a floating point number, the total angular momentum ecm, the observed energy in inverse cm, if unknown, the value is 0. eryd, the observed energy in Rydbergs, if unknown, the value is 0. ecmth, the calculated energy from the scattering calculation, in inverse cm erydth, the calculated energy from the scattering calculation in Rydbergs ref, the references in the literature to the data in the input info
- **outfile** (*str*) – Output filename. ionS+'.elvlc' (in current directory) if None
- **addLvl** (*int*) – Add a constant value to the index of all levels

Notes

For use with files created before elvlc format change in November 2012

See also:

ChiantiPy.tools.io.elvlcWrite Write .elvlc file using the new format.

ChiantiPy.tools.archival.**wgfaRead**(*ions*, *filename=None*, *elvlcname=- 1*, *total=False*, *verbose=False*)
Read CHIANTI data from a .wgfa file.

Parameters

- **ions** (*str*) – Ion, e.g. 'c_5' for C V
- **filename** (*str*) – Custom filename, will override that specified by *ions*
- **elvlcname** (*str*) – If specified, the lsj term labels are returned in the *pretty1* and *pretty2* keys of *Wgfa*
- **total** (*bool*) – Return the level 2 avalue data in *Wgfa*
- **verbose** (*bool*)

Returns Wgfa – Information read from the .wgfa file. The dictionary structure is {"lvl1","lvl2","wvl","gf","avalue","ref","ionS","filename"}

Return type *dict*

Notes

This is text-wise not different than the v8 version except that it uses the archival elvlcRead in *~ChiantiPy.tools.archival* though this has now been commented out. Can this routine be removed? Should the elvlcRead routine be uncommented?

See also:

ChiantiPy.tools.io.wgfaRead Read .wgfa file with the new format.

ChiantiPy.tools.constants module

A set of physical constants, in (mostly) cgs unit system. Most are from[11].

References

Notes

Many of these can be replaced by the *~astropy.constants* module. Elemental symbols can be removed in favor of the periodictable module. Spectroscopic roman numerals can be removed in favor of roman module. The Gauss-Laguerre weights can be calculated by *~numpy.polynomial.laguerre.laggauss*.

[11] NIST Reference on Constants, Units and Uncertainty (link)

ChiantiPy.tools.data module

Module for collecting various top-level Chianti data

Descriptions for *keywordArgs*:

- *temperature* : temperature (in K)
- *eDensity* : electron density (in cm^{-3})
- *hDensity* : hydrogen density (in cm^{-3})
- *pDensity* : proton density (in cm^{-3})
- *radTemperature* : radiation temperature of central source (in K)
- *rStar* : distance of the plasma from the source (in units of the source's radius)
- *distance* : distance from the central source

Parameters

- **XUVTOP** (*str*) – the root of the CHIANTI database
- **Defaults** (*dict*) – default values used by ChiantiPy, can also be set by an optional HOME/.chianti/chiantirc file
- **Ip** (*np.ndarray*) – the ionization potentials for all ionization stages up to Zn, in eV
- **MasterList** (*list*) – the CHIANTI style names of all ions in the CHIANTI database
- **IoneqAll** (*dict*) – a dict containing the ionization equilibrium values for the default ionization file
- **ChiantiVersion** (*str*) – the version of the CHIANTI database
- **AbundanceDefault** (*dict*) – the elemental abundances in the default abundance file
- **AbundanceList** (*list*) – the names of all abundance files included in the CHIANTI database
- **GrndLevels** (*list*) – the number of levels that should be considered in an ionization calculation

ChiantiPy.tools.filters module

Line profile filters for creating synthetic spectra.

ChiantiPy.tools.filters.**boxcar**(*wvl*, *wvl0*, *factor=None*)
Box-car filter

Parameters

- **wvl** (*~numpy.ndarray*) – Wavelength array
- **wvl0** (*~numpy.float64*) – Wavelength filter should be centered on.
- **factor** (*~numpy.float64*) – Full width of the box-car filter

ChiantiPy.tools.filters.**gaussian**(*wvl*, *wvl0*, *factor=1.0*)
A gaussian filter

Parameters

- **wvl** (*~numpy.ndarray*) – Wavelength array
- **wvl0** (*~numpy.float64*) – Wavelength filter should be centered on.
- **factor** (*~numpy.float64*) – Gaussian width

- **integrated value is unity**

ChiantiPy.tools.filters.**gaussianR**(*wvl*, *wvl0*, *factor=1000.0*)
A gaussian filter where the gaussian width is given by *wvl0/factor*.

Parameters

- **wvl** (*~numpy.ndarray*) – Wavelength array
- **wvl0** (*~numpy.float64*) – Wavelength filter should be centered on.
- **factor** (*~numpy.float64*) – Resolving power

ChiantiPy.tools.filters.**lorentz**(*wvl*, *wvl0*, *factor=1.0*)
Lorentz profile filter with the exception that all factors are in wavelength units rather than frequency as the lorentz profile is usually defined.

Parameters

- **wvl** (*~numpy.ndarray*) – Wavelength array
- **wvl0** (*~numpy.float64*) – Wavelength filter should be centered on.
- **factor** (*~numpy.float64*) – Value of the so-called constant gamma
- **integrated value is unity**
- **the FWHM is 2*gamma**
- **.. math::** – L = frac{1}{pi gamma} frac{ gamma^2}{(lambda - lambda_0)^2 + gamma^2}

ChiantiPy.tools.filters.**moffat**(*wvl*, *wvl0*, *factor=2.5*)
Moffat profile with parameters suited to Chandra Letg spectra

Parameters

- **wvl** (*~numpy.ndarray*) – Wavelength array
- **wvl0** (*~numpy.float64*) – Wavelength the filter is centered on.
- **factor** (*~numpy.float64*) – Resolving power (TODO: correct description)

ChiantiPy.tools.filters.**voigt**(*wvl*, *wvl0*, *factor=(0.5, 1.0)*)
pseudo-Voigt filter the sum of a Gaussian and a Lorentzian

Parameters

- **wvl** (*~numpy.ndarray*) – Wavelength array
- **wvl0** (*~numpy.float64*) – Wavelength the filter is centered on.
- **factor** (*array-type*) – contains the following 2 parameters
- **A** (*~numpy.float64*) – relative size of gaussian and lorentz components must be between 0. and 1. but this is not currently checked
- **sigma** (*~numpy.float64*) – the gaussian width of the gaussian profile (the standard deviation) also creates the lorentz component with the same fwhm

ChiantiPy.tools.io module

Reading and writing functions

TODO: see if klgfbRead is needed or not

ChiantiPy.tools.io.**abundanceRead**(*abundancename=None*, *verbose=False*)
Read abundance file *abundancename* and return the abundance values relative to hydrogen

Keyword Arguments

- **abundancename** (*str*) – the name of the abundance file in the $XUVTOP/abundance directory to be read the default is an empty string and then the 'default' abundance values are read
- **verbose** (*bool*) – if true, prints out some info

ChiantiPy.tools.io.**autoRead**(*ions*, *filename=None*, *total=True*, *verbose=False*)
Read CHIANTI autoionization rates from a .auto file.

Parameters

- **ions** (*str*) – Ion, e.g. 'c_5' for C V
- **filename** (*str*) – Custom filename, will override that specified by *ions*
- **elvlcname** (*str*) – If specified, the lsj term labels are returned in the 'pretty1' and 'pretty2' keys of 'Wgfa' dict
- **total** (*bool*) – Return the summed level 2 autoionization rates in 'Auto'
- **verbose** (*bool*)

Returns **Auto** – Information read from the .wgfa file. The dictionary structure is {"lvl1", "lvl2", "avalue", "pretty1", "pretty2", "ref","ionS", "filename"}

Return type *dict*

ChiantiPy.tools.io.**autoWrite**(*info*, *outfile=None*, *minBranch=None*)
Write data to a CHIANTI .wgfa file

Parameters

- **info** (*dict*) – Should contain the following: ionS, the Chianti style name of the ion such as c_4 for C IV, lvl1, the lower level, the ground level is 1, lvl2, the upper level, wvl, the wavelength (in Angstroms), avalue, the autoionization rate, pretty1, descriptive text of the lower level (optional), pretty2, descriptive text of the upper level (optiona), ref, reference text, a list of strings
- **outfile** (*str*)
- **minBranch** (*~numpy.float64*) – The transition must have a branching ratio greater than the specified to be written to the file

ChiantiPy.tools.io.**cireclvlRead**(*ions*, *filename=None*, *filetype='cilvl'*)
Read Chianti cilvl, reclvl, or rrlvl files and return data

Parameters

- **ions** (*str*) – Ion, e.g. 'c_5' for C V
- **filename** (*str*, optional) – Custom filename, will override that specified by *ions*
- **filetype** (*str*) – {'cilvl', 'reclvl', 'rrlvl'} Type of the file to read

ChiantiPy.tools.io.**convertName**(*name*)
Convert ion name string to Z and Ion and other interesting info

Parameters **name** (*str*) – a generic name of an ion in the CHIANTI database, such as fe_14 for Fe XIV

Todo: Put in separate module to avoid multiple copies

Notes

A duplicate of the routine in *ChiantiPy.tools.util* but needed by masterList Info

ChiantiPy.tools.io.**defaultsRead**(*verbose=False*)
Read in configuration from .chiantirc file or set defaults if one is not found.

ChiantiPy.tools.io.**demRead**(*demName=''*)
Read emission measure file *emName* and return the temperatures, densities and emission measures

Keyword Arguments **demName** (*str*) – the name of the differential emission measure file to read in the $XUVTOP/dem directory

Returns **DEM** – keywords are *temperature*, *density*, *dem*, *em*, *dt*, *filename*

Return type *dict*

ChiantiPy.tools.io.**diRead**(*ions*, *filename=None*)
Read chianti direct ionization .params files and return data.

Parameters

- **ions** (*str*) – Ion, e.g. 'c_5' for C V
- **filename** (*str*, optional) – Custom filename, will override that specified by *ions*

ChiantiPy.tools.io.**drRead**(*ions*, *filename=None*)
Read CHIANTI dielectronic recombination .drparams files if filename is set, then reads that file

Parameters

- **ions** (*str*) – Ion, e.g. 'c_5' for C V
- **filename** (*str*, optional) – Custom filename, will override that specified by *ions*

ChiantiPy.tools.io.**eaRead**(*ions*, *filename=None*)
Read a CHIANTI excitation-autoionization file and calculate the EA ionization rate data derived from splupsRead.

Parameters

- **ions** (*str*) – Ion, e.g. 'c_5' for C V
- **filename** (*str*, optional) – Custom filename, will override that specified by *ions*

ChiantiPy.tools.io.**elvlcRead**(*ions*, *filename=None*, *getExtended=False*, *verbose=False*, *useTh=True*)
Reads the new format elvlc files.

Parameters

- **ions** (*str*) – Ion, e.g. 'c_5' for C V
- **filename** (*str*, optional) – Custom filename, will override that specified by *ions*
- **getExtended** (*bool*)

- **verbose** (*bool*)
- **useTh** (*bool*) – If True, the theoretical values (ecmth and erydth) are inserted when an energy value for ecm or eryd is zero(=unknown)

ChiantiPy.tools.io.**elvlcWrite**(*info*, *outfile=None*, *round=0*, *addLvl=0*, *includeRyd=False*, *includeEv=False*)

Write Chianti data to .elvlc file.

Parameters

- **info** (*dict*) – Information about the Chianti data to write. Should contain the following keys: ionS, the Chianti style name of the ion such as c_4 term, a string showing the configuration spin, an integer of the spin of the state in LS coupling l, an integer of the angular momentum quantum number spd, an string for the alphabetic symbol of the angular momemtum, S, P, D, etc j, a floating point number, the total angular momentum ecm, the observed energy in inverse cm, if unknown, the value is 0. eryd, the observed energy in Rydbergs, if unknown, the value is 0. ecmth, the calculated energy from the scattering calculation, in inverse cm erydth, the calculated energy from the scattering calculation in Rydbergs ref, the references in the literature to the data in the input info
- **outfile** (*str*) – Output filename. ionS + '.elvlc' (in current directory) if None
- **round** (*int*) – input to 'np.round' to round input values to maintain the correct number of significant figures
- **addLvl** (*int*) – Add a constant value to the index of all levels
- **includeRyd** (*bool*) – If True, write the Rydberg energies in the extended area, delimited by a comma
- **includeEv** (*bool*) – If True, write the energies in eV in the extended area, delimited by a comma

Notes

For use with files created after elvlc format change in November 2012

See also:

ChiantiPy.tools.archival.elvlcWrite Write .elvlc file using the old format.

ChiantiPy.tools.io.**emRead**(*emName=''*)

Read emission measure file *emName* and return the temperatures, densities and emission measures

Keyword Arguments **emName** (*str*) – the name of the emission measure file to read in the $XUVTOP/em directory

Returns **EM** – keywords are *temperature*, *density*, *em*, *filename*

Return type *dict*

ChiantiPy.tools.io.**fblvlRead**(*ions*, *filename=None*, *verbose=False*)

Read a Chianti energy level file for calculating the free-bound continuum

Parameters

- **ions** (*str*) – Ion, e.g. 'c_5' for C V
- **filename** (*str*, optional) – Custom filename, will override that specified by *ions*

ChiantiPy.tools.io.**gffRead**()
Read the free-free gaunt factors of[1].

References

Notes

This function reads the file and reverses the values of g2 and u

ChiantiPy.tools.io.**gffintRead**()
Read the integrated free-free gaunt factors of[1].

ChiantiPy.tools.io.**grndLevelsRead**()
to read the grndLevels.dat file give the number of ground levels to sum over in populate and drPopulate

ChiantiPy.tools.io.**ioneqRead**(*ioneqName=''*, *minIoneq=1e-20*, *verbose=False*)
Reads an ioneq file ionization equilibrium values less then minIoneq are returns as zeros

Keyword Arguments

- **ioneqName** (*str*) – reads the file in the $XUVTOP/ioneq directory, if a blank, the default is read
- **minIoneq** (*float*) – sets values to zero if less the minIoneq
- **verbose** (*bool*) – if true, prints into to the terminal

Returns **{'ioneqname','ioneqAll','ioneqTemperature','ioneqRef'}** – Ionization equilibrium values and the reference to the literature

Return type *dict*

ChiantiPy.tools.io.**ipRead**(*verbose=False*)
Reads the ionization potential file

Returns **ip** – Ionization potential (in eV)

Return type array-like

ChiantiPy.tools.io.**itohRead**()
Read in the free-free gaunt factors of[2].

References

ChiantiPy.tools.io.**klgbfnRead**(*filename*)
Read CHIANTI files containing the free-bound gaunt factors for n=1-6 from[14]. This reads the corrected KL files in CHIANTI version 10+

Parameters **filename** (*str*) – the filename for the KL data in the continuum directory, such as klgfb_1.dat, n=1,6

Returns **[{'pe', 'klgfb'}]** – Photon energy and the bound-free gaunt factors

Return type *list*

[1] Sutherland, R. S., 1998, MNRAS, 300, 321
[2] Itoh, N. et al., 2000, ApJS, 128, 125
[14] Karzas and Latter, 1961, ApJSS, 6, 167

References

ChiantiPy.tools.io.**maoParsRead**(*filename=None*)
to read the mao et al par2.dat file for calculating the ratio of rrloss to rrrecomb The electron energy-loss rate due to radiative recombination. Mao J., Kaastra J., Badnell N.R. <Astron. Astrophys. 599, A10 (2017)> =2017A&A…599A..10M 1- 2 I2 — s Isoelectronic sequence number 4- 5 I2 — z Atomic number 7- 16 E10.4 — a0 Primary fitting parameter 18- 27 E10.4 — b0 Primary fitting parameter 29- 38 E10.4 — c0 Addiational fitting parameter 40- 49 E10.4 — a1 Primary fitting parameter 51- 60 E10.4 — b1 Primary fitting parameter 62- 71 E10.4 — a2 Addiational fitting parameter 73- 82 E10.4 — b2 Addiational fitting parameter 84- 86 F3.1 — mdp Maximum deviation in percent

Keyword Arguments **filename** (*str*) –

Returns **data**

Return type *dict*

ChiantiPy.tools.io.**masterListInfo**(*force=False*, *verbose=False*)
Get information about ions in the CHIANTI masterlist.

Keyword Arguments

- **force** (*bool*) – if true, recreates the masterListInfo file
- **verbose** (*bool*) – if true, prints into to the terminal

Returns **masterListInfo** – {'wmin', 'wmax', 'tmin', 'tmax'} Minimum and maximum wavelengths in the wgfa file. Minimum and maximum temperatures for which the ionization balance is nonzero.

Return type *dict*

Notes

This function speeds up multi-ion spectral calculations. The information is stored in a pickled file 'masterlist_ions.pkl' If the file is not found, one will be created.

ChiantiPy.tools.io.**masterListRead**()
Read a CHIANTI masterlist file.

Returns **masterlist** – All ions in Chianti database

Return type *list*

ChiantiPy.tools.io.**rrLossRead**()
to read the Mao 2017 rr loss parameters[12]

References

ChiantiPy.tools.io.**rrRead**(*ions*, *filename=None*)
Read CHIANTI radiative recombination .rrparams files

Parameters **ions** (*str*) – Ion, e.g. 'c_5' for C V

Keyword Arguments **filename** (*str*, optional) – Custom filename, will override that specified by *ions*

Returns **{'rrtype','params','ref'}**

[12] Mao J., Kaastra J., Badnell N.R., 2017 Astron. Astrophys. 599, A10

Return type *dict*

ChiantiPy.tools.io.**scupsRead**(*ions*, *filename=None*, *verbose=False*)
Read the new format v8 scups file containing the scaled temperature and upsilons from[8].

Parameters **ions** (*str*) – Ion, e.g. 'c_5' for C V

Keyword Arguments

- **filename** (*str*, optional) – Custom filename, will override that specified by *ions*
- **verbose** (*bool*) – if True, prints info to the terminal

ChiantiPy.tools.io.**splomRead**(*ions*, *ea=False*, *filename=None*)
Read chianti .splom files

Parameters **ions** (*str*) – Ion, e.g. 'c_5' for C V

Keyword Arguments

- **ea** (*bool*) – if true, reads the .easplom file
- **filename** (*str*, optional) – Custom filename, will override that specified by *ions*

Returns **{'lvl1', 'lvl2', 'ttype', 'gf', 'deryd', 'c', 'splom', 'ref'}**

Return type *dict*

Notes

Still needed for ionization cross sections

ChiantiPy.tools.io.**splupsRead**(*ions*, *filename=None*, *filetype='splups'*)
Read a CHIANTI .splups file

Parameters **ions** (*str*) – Ion, e.g. 'c_5' for C V

Keyword Arguments

- **filename** (*str*, optional) – Custom filename, will override that specified by *ions*
- **filetype** (*str*, optional) – {*psplups*,`cisplups`,`splups`} Type of file to read

Returns **{'lvl1', 'lvl2', 'ttype', 'gf', 'de', 'cups', 'bsplups', 'ref'}**

Return type *dict*

ChiantiPy.tools.io.**twophotonHRead**()
Read the two-photon Einstein A values and distribution function for the H sequence.

Returns **{'y0', 'z0', 'avalue', 'asum', 'psi0'}**

Return type *dict*

ChiantiPy.tools.io.**twophotonHeRead**()
Read the two-photon Einstein A values and distribution function for the He sequence.

Returns **{'y0', 'avalue', 'psi0'}**

Return type *dict*

ChiantiPy.tools.io.**vernerRead**()
Reads the photoionization cross-section data from[6].

[8] Burgess, A. and Tully, J. A., 1992, A&A, 254, 436
[6] Verner & Yakovlev, 1995, A&AS, 109, 125

Returns {'pqn','l','eth','e0','sig0','ya','p', yw'} – *pqn* is the principal quantum number, *l* is the subshell orbital quantum number, *eth* (in eV) is the subshell ionization threshold energy; *sig0*, *ya*, *p*, and *yw* are all fit parameters used in calculating the total photoionization cross-section.

Return type *dict*

References

ChiantiPy.tools.io.**versionRead**()
Read the version number of the CHIANTI database

ChiantiPy.tools.io.**wgfaRead**(*ions*, *filename=None*, *elvlcname=0*, *total=False*, *verbose=False*)
Read CHIANTI data from a .wgfa file.

Parameters **ions** (*str*) – Ion, e.g. 'c_5' for C V

Keyword Arguments

- **filename** (*str*) – Custom filename, will override that specified by *ions*
- **elvlcname** (*str*) – If specified, the lsj term labels are returned in the 'pretty1' and 'pretty2' keys of 'Wgfa' dict
- **total** (*bool*) – Return the summed level 2 avalue data in 'Wgfa'
- **verbose** (*bool*) –

Returns **Wgfa** – Information read from the .wgfa file. The dictionary structure is {"lvl1","lvl2","wvl","gf","avalue","ref","ionS","filename"}

Return type *dict*

See also:

ChiantiPy.tools.archival.wgfaRead Read .wgfa file with the old format.

ChiantiPy.tools.io.**wgfaWrite**(*info*, *outfile=None*, *minBranch=1e-05*, *rightDigits=4*, *maxLvl1=None*)
Write data to a CHIANTI .wgfa file

Parameters

- **info** (*dict*) – Should contain the following keys: ionS, the Chianti style name of the ion such as c_4 for C IV, lvl1, the lower level, the ground level is 1, lvl2, the upper level, wvl, the wavelength (in Angstroms), gf,the weighted oscillator strength, avalue, the A value, pretty1, descriptive text of the lower level (optional), pretty2, descriptive text of the upper level (optiona), ref, reference text, a list of strings
- **outfile** (*str*)
- **minBranch** (*~numpy.float64*) – The transition must have a branching ratio greater than the specified minBranchto be written to the file

ChiantiPy.tools.io.**zion2name**(*z*, *ion*, *dielectronic=False*)
Convert *Z* and *ion* to generic name, e.g. 26, 13 -> fe_13

Parameters

- **z** (*int*) – the nuclear charge, for example 26 for Fe XIV
- **ion** (*int*) – the ion stage, for example, 14 for Fe XIV

Keyword Arguments **dielectronic** (*bool*) – if True, created the name of a dielectronic ion, with a 'd' at the end

Todo: See if dielectronic is still appropriate

Put in separate module to avoid multiple copies

Notes

A duplicate of the routine in *ChiantiPy.tools.util* but needed by masterList Info

ChiantiPy.tools.mputil module

Functions needed for standard Python multiprocessing module mspectrum

ChiantiPy.tools.mputil.**doFbLossQ**(*inQ*, *outQ*)
Multiprocessing helper for *ChiantiPy.core.continuum.freeBound*

Parameters

- **inQ** (*~multiprocessing.Queue*) – Ion free-bound emission jobs queued up by multiprocessing module
- **outQ** (*~multiprocessing.Queue*) – Finished free-bound emission jobs

ChiantiPy.tools.mputil.**doFbQ**(*inQ*, *outQ*)
Multiprocessing helper for *ChiantiPy.core.continuum.freeBound*

Parameters

- **inQ** (*~multiprocessing.Queue*) – Ion free-bound emission jobs queued up by multiprocessing module
- **outQ** (*~multiprocessing.Queue*) – Finished free-bound emission jobs

ChiantiPy.tools.mputil.**doFfLossQ**(*inQ*, *outQ*)
Multiprocessing helper for *ChiantiPy.core.continuum.freeFreeLoss*

Parameters

- **inQ** (*~multiprocessing.Queue*) – Ion free-free emission jobs queued up by multiprocessing module
- **outQ** (*~multiprocessing.Queue*) – Finished free-free emission jobs

ChiantiPy.tools.mputil.**doFfQ**(*inQ*, *outQ*)
Multiprocessing helper for *ChiantiPy.core.continuum.freeFree*

Parameters

- **inQ** (*~multiprocessing.Queue*) – Ion free-free emission jobs queued up by multiprocessing module
- **outQ** (*~multiprocessing.Queue*) – Finished free-free emission jobs

ChiantiPy.tools.mputil.**doIonLossQ**(*inQueue*, *outQueue*)
Multiprocessing helper for *ChiantiPy.core.ion* and *ChiantiPy.core.ion.twoPhoton*

Parameters

- **inQueue** (*~multiprocessing.Queue*) – Jobs queued up by multiprocessing module
- **outQueue** (*~multiprocessing.Queue*) – Finished jobs

ChiantiPy.tools.mputil.**doIonQ**(*inQueue*, *outQueue*)
Multiprocessing helper for *ChiantiPy.core.ion* and *ChiantiPy.core.ion.twoPhoton*

Parameters

- **inQueue** (*~multiprocessing.Queue*) – Jobs queued up by multiprocessing module
- **outQueue** (*~multiprocessing.Queue*) – Finished jobs

ChiantiPy.tools.sources module

Blackbody temperature calculations

class ChiantiPy.tools.sources.**blackStar**(*temperature*, *radius*)
Bases: object

Calculate blackbody radiation

Parameters

- **temperature** (*~numpy.ndarray*) – Temperature in Kelvin
- **radius** (*~numpy.ndarray*) – Stellar radius in cm

Variables

- **~blackStar.Temperature** (*~numpy.ndarray*) – Temperature in Kelvin
- **~blackStar.Radius** (*~numpy.ndarray*) – Stellar radius in cm
- **~blackStar.Incident** (*~numpy.ndarray*) – Blackbody photon distribution

incident(*distance*, *energy*)
Calculate photon distribution times the visible cross-sectional area.

Parameters

- **distance** (*~numpy.ndarray*) – Distance to the stellar object in cm
- **energy** (*~numpy.ndarray*) – Energy range in erg

Notes

This function returns the photon distribution instead of the distribution times the cross-sectional area. Is this correct? Why is the incident photon distribution calculated at all?

ChiantiPy.tools.sources.**blackbody**(*temperature*, *variable*, *hnu=1*)
Calculate the blackbody photon distribution as a function of energy (*hnu* = 1) or as a function of wavelength (*hnu* = 0) in units of $\mathrm{photons\,cm^{-2}\,s^{-1}\,str^{-1}\,erg^{-1}}$

Parameters

- **temperature** (*~numpy.float64*) – Temperature at which to calculate the blackbody photon distribution
- **variable** (*~numpy.ndarray*) – Either energy (in erg) or wavelength (in angstrom)
- **hnu** (*int*) – If 1, calculate distribution as a function of energy. Otherwise, calculate it as a function of wavelength

Returns **{'photons', 'temperature', 'energy'} or {'photons', 'temperature', 'wvl'}**

Return type *dict*

ChiantiPy.tools.util module

Utility functions

Notes

Some of these functions can be replaced by roman numeral and periodic table lookup libraries. some functions using os.walk can be replaced by os.path

ChiantiPy.tools.util.**between**(*array*, *limits*)
Find the indices of *array* corresponding to values in the range given by *limits*

Parameters

- **array** (*list*, *ndarray*) – contains a list of elements such a wavelengths
- **limits** (*list*, *ndarray*) – a 2 element array specifying the lower and upper limits of the array to be included

ChiantiPy.tools.util.**convertName**(*name*)
Convert ion name string (e.g. 'fe_13') to atomic number and ion number.

Parameters **name** (*str*) – the CHIANTI style name for an ion, such as fe_14

Returns **{'Z', 'Ion', 'Dielectronic', 'Element', 'higher', 'lower'}** – *higher* and *lower* are the Chianti-style names for the higher and lower ionization stages, respectively.

Return type *dict*

ChiantiPy.tools.util.**descale_bt**(*bte*, *btomega*, *f*, *ev1*)
Apply excitation descaling of[3] to energy and collision strength

Parameters

- **bte** (*array-like*) – Scaled energy
- **btomega** (*array-like*) – Scaled collision strength
- **f** (*array-like*)
- **ev1** (*array-like*)

Returns **[energy,omega]** – Descaled energy and collision strength

Return type *list*

Notes

Need more details here. Not clear which equations are being used.

See also:

scale_bt Apply scaling to excitation energy and cross-section

[3] Burgess, A. and Tully, J. A., 1992, A&A, 254, 436

References

ChiantiPy.tools.util.**descale_bti**(*bte*, *btx*, *f*, *ev1*)
Apply ionization descaling of[9] to energy and cross-sections of[7].

Parameters

- **bte** (*array-like*) – Scaled energy
- **btx** (*array-like*) – Scaled cross-section
- **f** (*float*) – Scaling parameter
- **ev1** (*float*) – ionization potential - units determine the output energy units

Returns **[energy,cross]** – Descaled energy and cross-section

Return type *list*

Notes

This is the scaling used and discussed in the Dere (2007) calculation[7] of cross sections. It was derived from similar scalings provided by reference [2]

See also:

scale_bti Descale ionization energy and cross-section

References

ChiantiPy.tools.util.**descale_bti_rate**(*btTemperature*, *btRate*, *ip*, *f=2.0*)
Apply ionization descaling of[7], a Burgess-Tully type scaling to bt scaled ionization rates and temperatures. The result is to return a temperature array and a ionization rate array.

Parameters

- **btTemperature** (*array-like*) – the bt scaled temperature
- **btRate** (*array-like*) – the bt scaled ionization rate
- **ip** (*float*) – the ionization potential in eV.
- **f** (*float* (optional)) – the scaling parameter, 1.7 generally works well

ChiantiPy.tools.util.**dilute**(*radius*)
Calculate the dilution factor.

Parameters **radius** (*array-like*) – Distance from the center of a star in units of the stellar radius.

[9] Burgess, A. and Tully, J. A., 1992, A&A, 254, 436
[7] Dere, K. P., 2007, A&A, 466, 771,

Notes

If *radius* is less than 1.0, returns 0.

ChiantiPy.tools.util.**el2z**(*els*)
Convert elemental symbol to atomic number

Parameters **els** (*str*) – the abreviated element name

Returns **z** – the atomic number or nuclear charge of the element

Return type int

ChiantiPy.tools.util.**findFileTypes**(*wpath*, *type='*.txt'*, *verbose=False*)
to find all the files in wpath and below with file names matching fname

ChiantiPy.tools.util.**ion2filename**(*ions*)
Convert ion name string to generic directory-file name. convertName has probably made this redundant

Parameters **ions** (*str*) – the CHIANTI style name for an ion, such as fe_14

Returns **fname** – the full file name of the ion directory in the CHIANTI database assumes a top directory from the environmental variable XUVTOP

Return type str

Todo: this duplicates what 'convertName' does

ChiantiPy.tools.util.**listFiles**(*dir*)
Walks the path and subdirectories to return a list of files.

Parameters **dir** (*str*) – the top directory to search subdirectories are also searched

Returns **listname** – a list of files in dir and subdirectories

Return type list

Notes

This can be replaced by functions in *os.path*, as if 3.4, pathlib is probably better. It is not clear that this function is used anywhere in ChiantiPy

ChiantiPy.tools.util.**listRootNames**(*dir*)
Walks the path and subdirectories to return a list of file root names.

Notes

This can be replaced by functions in *os.path*, as if 3.4, pathlib is probably better. Only seems to be used by

ChiantiPy.tools.util.**qrp**(*z*, *u*)
Calculate $Q'_R(Z,u)$, where $u = \epsilon/I$ is the impact electron energy in threshold units, from Eq. 2.12 of[4].

Parameters

- **z** (*int*) – Atomic number
- **u** (*array-like*) – Impact electron energy in threshold units.

Returns **q** – 1s ionization cross-section, $Q'_R(Z,u)$

[4] Fontes, C. et al., 1999, PhRvA, 59, 1329

Return type array-like

Notes

Used for calculations of direct ionization cross-sections of the H and He sequences in *ChiantiPy.tools.io.twophotonHRead* and *ChiantiPy.tools.io.twophotonHeRead*, respectively.

References

ChiantiPy.tools.util.**scale_bt**(*evin*, *omega*, *f*, *ev1*)

Apply excitation scaling of[?] to energy and collision strength.

Parameters

- **evin** (*array-like*)
- **omega** (*array-like*)
- **f** (*array-like*)
- **ev1** (*array-like*)

Returns **[bte,btomega]** – Scaled energy and collision strength

Return type *list*

Notes

Need more details here. Not clear which equations are being used.

See also:

descale_bt Descale excitation energy and cross-section

ChiantiPy.tools.util.**scale_bt_rate**(*inDict*, *ip*, *f=2.0*)

Apply ionization scaling of[?], a Burgess-Tully type scaling to ionization rates and temperatures. The result of the scaling is to return a scaled temperature between 0 and 1 and a slowly varying scaled rate as a function of scaled temperature. In addition, the scaled rates vary slowly along an iso-electronic sequence.

Parameters

- **inDict** (*dict*) – the input dictionary should have the following key pairs: *temperature*, array-like and *rate*, array-like
- **ip** (*float*) – the ionization potential in eV.
- **f** (*float* (optional)) – the scaling parameter, 1.7 generally works well

Notes

btTemperature and *btRate* keys are added to *inDict*

ChiantiPy.tools.util.**scale_bti**(*evin*, *crossin*, *f*, *ev1*)
Apply ionization scaling of[?],[8]_, to energy and cross-section.

Parameters

- **evin** (*float*) – Energy - same units as ev1
- **crossin** (*array-like*) – Cross-section
- **f** (*float - the scale factor*)
- **ev1** (*float*) – the ionization potential units - the same as evin

Returns **[bte,btx]** – Scaled energy and cross-section

Return type *list*

Notes

This is the scaling used and discussed in the Dere (2007) calculation [1] of cross sections. It was derived from similar scalings derived in reference [2]

See also:

descale_bti Descale ionization energy and cross-section

References

ChiantiPy.tools.util.**scale_classical**(*inDict*, *ip*)
Apply the classical scaling to the input data

Parameters

- **inDict** (*dictionary*) –

  the input dictionary should have the following key pairs energy and cross or temperature and rate
- **energy** (*array-like*) – energy values of the cross-section
- **cross** (*array-like*) – a cross-section
- **temperature** (*array-like*)
- **rate** (*array-like*)
- **ip** (*float*) – the ionization potential. Typically in eV.
- **Returns** – the following keys are added to inDict
- **——-**
- **{'csEnergy', 'csCross', 'ip'} or {'csTemperature', 'csRate', 'ip'}**

ChiantiPy.tools.util.**spectroscopic2name**(*el*, *roman*)
Convert from spectroscopic notation, e.g. Fe XI to 'fe_11'.

Parameters

- **el** (*str*) – Elemental symbol, e.g. Fe for iron

- **roman** (*str*) – Roman numeral spectroscopic symbol

ChiantiPy.tools.util.**splomDescale**(*splom*, *energy*)
Calculate the collision strength for excitation-autoionization as a function of energy.

Parameters

- **energy** (*array-like*) – In eV
- **splom** (*dict*) – Structure returned by *ChiantiPy.tools.io.splomRead*

Returns **omega** – Collision strength

Return type array-like

ChiantiPy.tools.util.**units**(*defaults*)
to create a set of units compatible with ChiantiPy default values

ChiantiPy.tools.util.**z2element**(*z*)
Convert atomic number *z* to its elemental symbol.

Parameters **z** (*int*) – The atomic number/nuclear charge

Returns **element** – the abbreviated element name

Return type *str*

ChiantiPy.tools.util.**zion2dir**(*z*, *ion*, *dielectronic=False*, *xuvtop=''*)
Convert atomic number and ion number to CHIANTI database directory.

Parameters

- **z** (*int*) – The atomic number/nuclear charge
- **ion** (*int*) – the ionization stage, 1 for the neutral, 2 for the first ionization stage, ...
- **dielectronic** (*bool*, optional)
- **xuvtop** (*str*, optional) – Set different CHIANTI database than the default

Returns **fname** – the CHIANTI directory where the file for the ion specified by z and ion are found

Return type str

ChiantiPy.tools.util.**zion2experimental**(*z*, *ion*, *dielectronic=False*)
Convert atomic number and ion number to spectroscopic notation string

Parameters

- **z** (*int*) – The atomic number/nuclear charge
- **ion** (*int*) – the ionization stage, 0 for neutrals, 1 for singly ionized
- **dielectronic** (*bool*, optional)

Returns **expt** – the experimental/laboratory notation for the ion, such as Fe 13+

Return type *str*

ChiantiPy.tools.util.**zion2filename**(*z*, *ion*, *dielectronic=False*, *xuvtop=''*)
Convert nuclear charge/atomic number and ion number to CHIANTI database filename.

Parameters

- **z** (*int*) – The atomic number/nuclear charge
- **ion** (*int*) – the ionization stage, 1 for neutrals, 2 for singly ionized
- **dielectronic** (*bool*, optional) – whether the ion is the simple dielectronic model

- **xuvtop** (*str*, optional) – the top directory of the CHIANTI database to be used

Returns **fname** – CHIANTI database filename

Return type str

ChiantiPy.tools.util.**zion2localFilename**(*z*, *ion*, *dielectronic=False*)
Convert atomic number and ion number to generic file name with current directory at top.

Parameters

- **z** (*int*) – The atomic number/nuclear charge
- **ion** (*int*) – the ionization stage, 1 for neutrals, 2 for singly ionized
- **dielectronic** (*bool*, optional)

ChiantiPy.tools.util.**zion2name**(*z*, *ion*, *dielectronic=False*)
Convert atomic number and ion number to generic name, e.g. (26,13) to 'fe_13'

Parameters

- **z** (*int*) – The atomic number/nuclear charge
- **ion** (*int*) – the ionization stage, 1 for the neutral, 2 for the first ionization stage, ...
- **dielectronic** (*bool*, optional)

Returns **thisone** – the CHIANTI style ion name, such as 'fe_13'

Return type str

ChiantiPy.tools.util.**zion2spectroscopic**(*z*, *ion*, *dielectronic=False*)
Convert atomic number and ion number to spectroscopic notation string

Parameters

- **z** (*int*) – The atomic number/nuclear charge
- **ion** (*int*) – the ionization stage, 1 for neutrals, 2 for singly ionized
- **dielectronic** (*bool*, optional)

Returns **spect** – the spectroscopic notation for the ion, such as Fe XIV

Return type *str*

Module contents

Basic tools and utilities used in ChiantiPy

4.1.2 Submodules

4.1.3 ChiantiPy.version module

the current version of the ChiantiPy package

4.1.4 Module contents

ChiantiPy - CHIANTI Python package Calculates various aspects of emission lines and continua from the CHIANTI atomic database for astrophysical spectroscopy.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

SIX

NOTES

6.1 Setting default values

Several parameter values can be set to define the way ChiantiPy behaves. The following parameter values can be set in your chiantirc file. A chiantirc file is available in the downloaded package but it will only be read if it is located in either ~HOME/.config (probably the best place) or ~HOME/.chianti or ~PROFILEHOME/.config or ~PROFILEHOME/.chianti. It is then possible to modifiy the file to select the default value you wish to use:

These are:

wavelength

The default value is angstrom. Other possible values are *nm* for nanometers, *ev* for electron volts or *kev* for kilo-electron volts. However, the continuum classes can only be used with the wavelengths in angstroms

flux The default value is energy. Acceptable values are *energy* and *photon* and these govern emissivities and intensities. If set to energy, emission units are in erg.

abundfile

The name of the abundance file. Acceptable values are any of the file names in XUVTOP/abundance, such as *cosmic_1973_allen*. The default value is *sun_photospheric_2015_scott* which includes the abundances of Scott et al., 2015, A&A, 573, A25.

ioneqfile the name of the ionization equilibrium file. Acceptable values are any of the file names in XUVTOP/ioneq such as *arnaud_raymond*, *arnaud_rothenflug*, or *chianti*. The default value is *chianti* which includes the ionization equilibrium calculations of K.P. Dere, G. Del Zanna, P.R. Young, E. Landi, R Sutherland, 2019, ApJ, 241, 2 and are considered to be based on the best ionization and recombination rates currently available.

6.2 Setting *minAbund* in spectrum calculations

When calculation spectra with *spectrum* or *mspectrum*, it is often useful to set the "minAbund" keyword which governs the minimum abundance of any element included in the calculation. Below is a list of elemental abundances for the elements through zinc and the elements that will be included by several value of "minAbund". These are for photospheric abundances, some of which may have changed since this table was made.

Element	Abundance	minAbund		
		1.e-6	1.e-5	1.e-4
H	1.00e+00	•	•	•

continues on next page

Table 1 – continued from previous page

Element	Abundance	minAbund		
		1.e-6	1.e-5	1.e-4
He	8.51e-02	•	•	•
Li	1.26e-11			
Be	2.51e-11			
B	3.55e-10			
C	3.31e-04	•	•	•
N	8.32e-05	•	•	
O	6.76e-04	•	•	•
F	3.63e-08			
Ne	1.20e-04	•	•	•
Na	2.14e-06	•		
Mg	3.80e-05	•	•	
Al	2.95e-06	•		
Si	3.55e-05	•	•	
P	2.82e-07			
S	2.14e-05	•	•	
Cl	3.16e-07			
Ar	2.51e-06	•		
K	1.32e-07			
Ca	2.29e-06	•		
Sc	1.48e-09			
Ti	1.05e-07			
V	1.00e-08			
Cr	4.68e-07			
Mn	2.45e-07			

continues on next page

Table 1 – continued from previous page

Element	Abundance	minAbund		
		1.e-6	1.e-5	1.e-4
Fe	3.16e-05	•	•	
Co	8.32e-08			
Ni	1.78e-06	•		
Cu	1.62e-08			
Zn	3.98e-08			

It shoud be noted that CHIANTI does not include a complete set of data for every ion of every element in this list.

CHAPTER

SEVEN

INTRO FOR CHIANTI IDL USERS

ChiantiPy was not developed to be a clone of the CHIANTI IDL code. The IDL code largely consists of functions that can be used to calculate or plot a variety of properties. Structures are often used to carry the results of one function to be used by another.

ChiantiPy is object oriented. For example, the building block of ChiantiPy is the **ion** class. It carries with it all the methods that are needed as well as various calculated properties that are kept as attributes. By following the Quick Start guide, you will become familiar with how ChiantiPy works. Because one can always inquire, for example with *dir*, as to the methods and attributes of a an object, such as an ion, it is easy to remember what you might want to calculate next. For example, you have created an object *myion*. It is possible to then invoke

```
myion.popPlot()
```

to plot the level populations of *myion*. If you have not already calculated the levels populations, the **ion** class knows to calculate the level populations (*myion.populate()*) and save them for later use as the dictionary attribute *myion.Population* and then plot the specified level populations. The level populations are then available as a 2 dimensional *numpy* array

```
myion.Population['population']
```

Python and IPython provide many tools for examining any give object (everything in Python is an object of some sort)

```
mg4 = ch.ion('mg_4',setup=0)
```

By setting the keyword *setup* to 0 or *False* the complete setup of the ion is not performed, but a certain amount of information is retrieved. Since neither a temperature or electron density were specified, none of these attributes know anything related to these two quantitities. The default value for setup is True so that the setup is performed with the specified temperatures and electron densities. In general, this is the way you want to construct the mg_4 *ion*.

```
for attr in vars(mg4):
    print(attr)

IonStr
FIP
Z
AbundanceName
Ip
Ion
IoneqAll
PDensity
Abundance
RadTemperature
FileName
```

(continues on next page)

(continued from previous page)

```
RStar
ProtonDensityRatio
Dielectronic
Defaults
IoneqName
Spectroscopic
```

The Python function *vars* retrieves the attributes of the mg4 object.

```
print('%s'%(mg4.IoneqName))

chianti

print('the abundance file name is %s'%(mg4.AbundanceName))

the abundance file name is sun_coronal_1992_feldman_ext

print('the abundance of %s is %10.2e'%(mg4.Spectroscopic,mg4.Abundance))

the abundance of Mg IV is   1.41e-04
```

One can get a more complete description of the various attributes and methods of the mg4 object

```
for one in dir(mg4):
    print(one)

Abundance
AbundanceName
Defaults
Dielectronic
FIP
FileName
Ion
IonStr
IoneqAll
IoneqName
Ip
PDensity
ProtonDensityRatio
RStar
RadTemperature
Spectroscopic
Z
__class__
__delattr__
__dict__
__dir__
__doc__
__eq__
__format__
__ge__
__getattribute__
```

(continues on next page)

(continued from previous page)

```
__gt__
__hash__
__init__
__le__
__lt__
__module__
__ne__
__new__
__reduce__
__reduce_ex__
__repr__
__setattr__
__sizeof__
__str__
__subclasshook__
__weakref__
boundBoundLoss
cireclvlDescale
convolve
diCross
diRate
drRate
drRateLvl
eaCross
eaDescale
eaRate
emiss
emissList
emissPlot
emissRatio
gofnt
intensity
intensityList
intensityPlot
intensityRatio
intensityRatioInterpolate
intensityRatioSave
ionGate
ioneqOne
ionizCross
ionizRate
lineSpectrumPlot
p2eRatio
popPlot
populate
recombRate
rrRate
setup
setupIonrec
spectrum
spectrumPlot
twoPhoton
```

(continues on next page)

(continued from previous page)

```
twoPhotonEmiss
twoPhotonLoss
upsilonDescale
upsilonDescaleSplups
```

First, at the top of the list are the attributes that were listed by the *vars* function. Then come a number of methods starting with '__'. These are generally not used and called *private* methods although nothing in Python is really private. In IDL, just about everything is private. After the *private* methods is a list of the methods provided by the mg4 *ion* class object. These all start with a lower case letter to separate them from the attributes that start with an upper case letter (this is a ChiantiPy convention).

In the IPython and jupyter-qtconsole, typing

```
mg4.diCross(
```

and then hitting the tab key will bring up the doc-string for the diCross method, also found in the API reference in the documentation. And then

```
mg4.diCross()
```

calculates the direct ionization cross section of Mg IV for a set of energies above the ionization potential *Ip*. The direct ionization cross sections are then provided in the mg4.DiCross dictionary.

Experience using the CHIANTI IDL package will provide the user with a background with what ChiantiPy can do. However, the way to accomplish them are much easier but must be learned. The best way to start is with the Quick Start guide and a book about Python. Book suggestions are *Learning Python* by Mark Lutz and the handy *Python Pocket Reference*, also by Mark Lutz. The first one is alway in reach and copies of the latter is on all of my computer desks.

CHAPTER

EIGHT

RESOURCES FOR CHIANTIPY USERS

8.1 Forums

The CHIANTI Google group

Github

CHAPTER

NINE

CHANGELOG

9.1 Changes from 0.14.1 to 0.15.0

Significant improvement have been made. It is now possible to calculate spectral line intensities in wavelength/energy units of angstroms, nn, eV, or keV

Calculations of the continuum still require that the wavelengths are in angstroms.

The free-bound continuum now includes the photoionization cross sections of Verner for recombination to the ground level

The free-free continuum has been correction to use the ion charge, not the nuclear charge previously used

A new class MradLoss.mradLoss has been created. It allows multiprocessing calculations of the radiative loss rate

The chiantirc file can now also be place in the $HOME/.config directory

9.2 Changes from 0.14.0 to 0.14.1

This relatively minor release adds some new features and corrects some glitches

A function demRead has been to ChiantiPy.tools.io for reading the CHIANTI .dem files in the XUVTOP/dem directory

The spectrumPlot method has been updated to provide more correct labeling of syntheic spectra

The QuickStart guide (html and notebook) have been updated to reflect these changes and show how to use the .dem files

9.3 Changes from 0.13.1 to 0.14.0

a new class 'ch.redux' restores the attributes saved by the saveData methods. It inherits as number of methods, especially, for plotting.

the inherited methods 'intensityPlot' and 'spectrumPlot' have been improved. These are inherited by the ion, bunch, spectrum, mspectrum, ipymspectrum and redux classes.

First, these methods will also display the ion name ('Fe XIV') and wavelength together with the line intensities or spectral intensity.

These are more flexible and several keyword arguments have been added:

> 'doLabel' governs whether to display the ion name and wavelength 'lw' the linewidth of the marker in matplotlib units (default=1) 'doTitle' governs whether to add a title to the plot

The QuickStart (html and notebook) has been updated to demonstrate some of these new features

Import bug fix: the indices for calculating the two-photon continue were update to match the new ordering of the energy levels for the h-like and he-like ions.

The ionization potential array (Ip) has been enlarged to make room for call to zn_31

9.4 Changes from 0.13.0 to 0.13.1

This is primarily a bug fix release to correct a bug in ionGate that was not taken care of in the 0.13.0 release.

9.5 Changes from 0.12.0 to 0.13.0

it is now possible to incorporate a user created abundance file. It needs to be of the same structure as one of the .abund files in the XUVTOP/abundance directory. The file can be located anywhere on the user's computer. It will be read if the abundance keyword is set to a fully qualified file name of the new abundance file, such as '/home/me/myabundance.abund', or equally, '/home/me/myabundance.txt'

in the core classes, bunch and spectrum, now have saveData methods to save calculations to a pickle file

by default, the pyQt widgets are now used a selection tools. The command line selection tools are still available but the ~/.chianti/chiantirc file needs to select them.

Problems with the ionGate method have been fixed. This method help to select which ions will be used in multi-ion calculations

9.6 Changes from 0.11.0 to 0.12.0

The model module is more mature

For Windows users, it is now possible to place the chiantirc file in $PROJECTHOME/.config or $PROJECTHOME/.chianti

Many improved docstrings for the documentation

the bunch class has been moved to a new module core.Bunch

a number of jupyter ipython notebooks have been created/improved to demonstrate the use of the bunch, spectrum and model.Maker classes. A short README.txt can be found in the same directory provides an introduction to these notebooks

A bug in the inherited method base._IntensityRatio() was not properly corrected in v0.10.0. This is fixed here

9.7 Changes from 0.10.0 to 0.11.0

Calculations of the free-bound/radiative recombination continuum and radiation losses depend on a file that provides and LS description of the bound and singly excited energy levels. This file is called c_5.fblvl in the case of C V. Not all ions have an associated .fblvl files and is was necessary to revise ChiantiPy to ignore the free-bound calculation for these ions.

In addition, it was found that under extreme conditions, such as very low temperatures for highly ionized species, that bad values would arise (Nans and infinities). These are now detected and removed.

9.8 Changes from 0.9.5 to 0.10.0

The Karzas and Latter (1965) bound-free gaunt factors in the CHIANTI database have been corrected as of CHIANTI version 10. This effects the calculation of the free-bound continuum. The continuum.freeBound method has been updated to uses these new data.

The freeBound and the freeFree methods now have 2 new keyword variables: **includeAbund** and **includeIoneq**. Their initial values are True so that elemental abundance and the ionization equilibrium appropriate to the ion is included the the output spectrum.

The freeBoundEmiss is removed as it has become redundant

A new continuum method, freeBoundLossMao includes the radiative-recombination (free-bound) loss rate as calculated by Mao et al. (2017)

a bug in the inherited method base._IntensityRatio() was corrected.

9.9 Changes from 0.9.4 to 0.9.5

this is a bug-fix release.

a bug in the inherited method base._IntensityRatio() had a problem if lines were selected from different ions

9.10 Changes from 0.9.3 to 0.9.4

this is a bug-fix release.

changes in version 0.9.2 continued to give problems with ions that included autoionization rates

9.11 Changes from 0.9.3 to 0.9.3

this is a bug-fix release.

changes in version 0.9.2 gave problems with ions that included autoionization rates

9.12 Changes from 0.9.2 to 0.9.3

this is a bug-fix release.

changes in version 0.9.2 led to an error where ion.recombRate did not work. This has been fixed

9.13 Changes from 0.9.1 to 0.9.2

this is a bug-fix release.

changes in version 0.9.1 lead to an error where a bare ion has not recombination rate. This has been fixed

9.14 Changes from 0.9.0 to 0.9.1

this is a bug-fix release.

in cases when it is not possible to calculate the free-bound continuum for some ion, mspectrum did not handle this correctly and crashed

also, the ion zn_31 (Zn XXXI) is a bare ion and has no ionization potential (IP) and looking it up caused indexing errors.

9.15 Changes from 0.8.7 to 0.9.0

a new module model.maker has been added

```
import ChiantiPy.model as mdl
mymodel = mdl.maker(...)
```

a serious bug in ch.freeBound was fixed - the use of a single temperature was problematic

9.16 Changes from 0.8.6 to 0.8.7

continued code cleanup

9.17 Changes from 0.8.5 to 0.8.6

another bug-fix release

added argCheck method to make sure that sizes of temperature, density and emission measure were compatible

9.18 Changes from 0.8.4 to 0.8.5

9.19 This is a major bug-fix release.

Errors in calculating the proton rates were corrected.

All temperatures and densities are now numpy arrays

9.20 Changes from 0.8.3 to 0.8.4

9.21 This is a major bug-fix release.

Another significant bug was fixed in the important ion.populate method.

9.22 Changes from 0.7.1 to 0.8.3

9.23 This is a major bug-fix release.

a small but mighty bug was found in the important ion.populate method.

9.24 Version 0.8.x files are necessary to use with the new CHIANTI Version 9.0 database

Changes have been made to take into account the new way that CHIANTI is handling dielectronic recombination and autoionization

The release is also available on [PyPI](https://pypi.org/project/ChiantiPy/)

Documentation is available on [github.io](https://chianti-atomic.github.io/)

and on [ReadTheDocs](https://chiantipy.readthedocs.io/en/latest/?badge=latest)

9.25 changes from 0.7.1 to 0.8.0

ChiantiPy is now only compliant with Python 3. Development is currently with Python 3.6

The use of the PyQt4 and WxWidgets packages have been dropped and PyQt5 is now used

The documentation is now available on github.io and ReadTheDocs

9.26 changes from 0.7.0 to 0.7.1

version 0.7.0 included some changes in the ChiantiPy naming conventions, largely in the continuum class. These are being reverted to the original ChiantiPy naming conventions.

the ion.freeBoundxxx methods have been fixed and this also fixes the problem with the RadLoss class.

a pseudo-voigt filter has been added to tools.filters

the keyword argument wvlRange has been removed from the ion.emiss and ion.intensity methods

the keyword argument for the Emission Measure, em, has been removed from the ion.intensity and similar methods. It is now necessary to specify the emission when the object is instantiated.

a set of PyQt5 dialogs have been developed by **ktritz** and are now included

this is the last release that will use the PyQt4 widgets as an option.

the method **ioneqOne** is used by both the Ion and Continuum class. It has been moved to a single _IoneqOne.py file in the **base** directory

9.27 changes from 0.6.5 to 0.7.0

The primary change is that code development has been moved to Github.

Also, in order to be more compliant with other astrophysical packages on Github (Astropy and SunPy) the directory layout has been changed and renamed.

The core routines are now imported as

```
import ChiantiPy.core as ch
```

this give access to ch.ion, ch.spectrum, etc.

In terms of bug-fixes, the calculation of excitation-autoionization cross-sections and rates have been corrected in the eaCross() and eaRate() methods

Current development is with Python 3.4

9.28 changes from 0.6.0 to 0.6.5

matplotlib.pyplot is now imported for plotting

IPython version 4 / Jupyter is now listed as a prerequisite. However, v0.6.4 can be made compatible with IPython 2 or 3 with a simple edit.

An error in calculating the proton excitation rates was fixed.

The code has been edited to make it compatible with Python 3 and has been tested against Python 3.3

9.29 changes from 0.5.3 to 0.6.0

This is a major release.

First, ChiantiPy 0.6.0 is compatible with the most recently released CHIANTI database version 8.0. It also fixes some major bugs in the previous version. Documentation has been improved and a IPython notebook **QuickStart.ipynb**, that largely follows the 'Quick Start' documentation pages, has also been included.

There are two new multi-ion classes: **bunch** and **ipymspectrum**. **bunch** allows the user to calculate line intensities for a specified set of elements or individual ions as a function of temperature or density. One advantage of **bunch** is the ability to calculate the intensity ratio of lines of two different ions as a function of temperature or density.

ipymspectrum is much like the existing **spectrum** and **mspectrum** classes. **mspectrum** allows the use of the Python **multiprocessing** module to speed up spectral calculations. The **ipymspectrum** class uses the IPython **parallel** module so that multiprocessing spectral calculations can be performed in the IPython QtConsole and Notebook.

A new method **intensityList** has been developed to allow the user to list the most intense lines within a given wavelength range. This new methods, together with previously existing **intensityRatio** and **intensityRatioSave** are all now inherited by the **ion** classs and the multi-ion classes.

The **ion** and multi-ion classes now accept the keyword argument **abundanceName** that allow the user to specify the set of elemental abundances rather than just the default abundance file.

Additional we have replaced the FortranFormat module of Scientific Python by Konrad Hinsen with the **fortranformat** module of Brendan Arnold at http://bitbucket.org/brendanarnold/py-fortranformat. I have slightly modified fortranformat to make it Python 3 compliant.

For the future, I plan to make ChiantiPy compliant with both Python 2.7 and the current version of Python 3 (now 3.4), improve the documentation and move the project to github, in no particular order.

ChiantiPy is now released under a new license, the OSI approved ISCL license. From Wikipedia *The ISCL license is a permissive free software license written by the Internet Software Consortium (ISC). It is functionally equivalent to the simplified BSD and MIT/Expat licenses, . . .*

CHAPTER

TEN

REPORTING BUGS

All bugs are kept track of on the GitHub issue tracker. If you run into any unexpected behavior or have a question please send an email or add an issue directly to the issue tracker.

CHAPTER

ELEVEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

INDEX

D

E

F

G

I

K

L

M

O

P

Q

R

S

T

U

V

W

Z