PyAtomDB Documentation

Release 0.5.3

Adam Foster

February 12, 2019

Contents

CHAPTER 1

Introduction

PyAtomDB is a selection of utilities designed to interact with the AtomDB database . These utilities started life as routines scattered around my laptop, so some produce lots of unhelpful onscreen output.

There are several different modules currently. These are:

- *atomdb* : a series of codes for interacting with the AtomDB atomic database
- *atomic* : basic atomic data routines - e.g. converting element symbols to atomic number, etc.
- *const* : a series of physical constants
- *spectrum* : routines for generating spectra from the published AtomDB line and continuum emissivity files
- *util* : sumple utility codes (sorting etc) that pyatomdb relies on.
- *apec* : ultimately, the full apec code. For now, incomplete.

Expect bugs. Report those bugs! Make feature requests! Email the code authors or raise an issue at the github page

CHAPTER 2

Contents

Contents

2.1 PyAtomDB APEC module

This modules contains the APEC code. It calls many different subroutines from throughout the PyAtomDB module. Currently largely unwritten, as APEC code needs to be tidied up for transfer.

2.2 PyAtomDB Atomic module

This modules contains basic atomic parameters (i.e. atomic numbers, element symbols)

2.3 PyAtomDB AtomDB module

This modules is designed to interact with the main atomic database, extracting real values of coefficients and so on.

2.4 PyAtomDB Const module

A series of physical constants and constants relevant to running the APEC code.

2.5 PyAtomDB Spectrum module

This module contains codes for creating spectra from the AtomDB emissivity files

2.6 PyAtomDB Util module

This modules contains simple utility codes (sorting etc) that pyatomdb relies on.

2.7 PyAtomDB Example Scripts

These are examples of using the pyatomdb module in your projects. They can all be found in the examples subdirectory of the pyatomdb tarball.

Table of Contents

2.7.1 Initial installation

first_installation.py

```
import pyatomdb

"""
This script shows the commands you should run when you first download pyatomdb.

It is recommended that you choose the location you want to install the
AtomDB data files (not the same as the python module) and set your
ATOMDB environment variable to point to it.


Parameters
------
none

Returns
-------
none

"""

# call the setup routine
pyatomdb.util.initialize()

# this routine downloads a bunch of files and sets things up for you. It will
# take a few minutes, depending on your internet connection.

print "Install complete!"

#and that's it!
```

```
# If you want to switch versions of atomdb (in this case to 3.0.2) later, call:
# pyatomdb.util.switch_version('3.0.2')
```

2.7.2 Make Line List

List the strongest lines in a given temperature and wavelength region: `make_line_list.py`

```
import pyatomdb

"""
This code will produce a list of lines in a given wavelength range at a
given temperature. It also shows the use of an NEI version, where you
have to additionally specify the initial ionization temperature (or the
ionization fraction directly) and the elapsed Ne*t.

The results of the list_lines codes are numpy arrays which can be sorted any
way you wish. You can, of course, extract the lines easily at this point. There
is also a print_lines routine for a fixed format output.

Parameters
----------
none

Returns
-------
none

"""

# Adam Foster 2015-12-02
# version 0.1

#specify wavelength range, in Angstroms
wl = [8.0,9.0]

# electron temperature in K
Te = 1e7

# get equilibrium line list

res = pyatomdb.spectrum.list_lines(wl,Te=Te, teunit='K', minepsilon=1e-18)

# reprocess lines for printing
print "Unsorted line list:"
pyatomdb.spectrum.print_lines(res)

# re-sort lines, for a giggle
# for more information, look up numpy.sort: res is a numpy array.
# http://docs.scipy.org/doc/numpy/reference/generated/numpy.sort.html

res.sort(order=['Epsilon'])
print "sorted by Emissivity:"
pyatomdb.spectrum.print_lines(res)

# re-sort by element, ion then emissivity
res.sort(order=['Element','Ion','Epsilon'])
print "sorted by Element, Ion, Emissivity:"
```

```
pyatomdb.spectrum.print_lines(res)


# now do an NEI version. This is slow at the moment, but functional.
Te_init = 1e4
tau = 1e11
res_nei = pyatomdb.spectrum.list_nei_lines(wl,Te=Te, teunit='K', \
                                           minepsilon=1e-18,\
                                           Te_init=Te_init,\
                                           tau = tau)
print "NEI linelist (this takes a while):"
pyatomdb.spectrum.print_lines(res_nei)
```

2.7.3 Get PI Cross Sections

Extract the PI cross section data: `photoionization_data.py`

```
import pyatomdb, numpy,os, pylab
try:
  import astropy.io.fits as pyfits
except:
  import pyfits

# This is a sample routine that reads in the photoionization data
# It also demonstrates using get_data, which should download the data you
# need automagically from the AtomDB site.
#
# It also shows how to get the raw XSTAR PI cross sections.

# going to get PI cross section from iron 16+ to 17+ (Fe XVII-XVIII)
Z = 26
z1 = 17

# get the AtomDB level data
lvdata = pyatomdb.atomdb.get_data(Z, z1, 'LV')

# get the XSTAR PI data from AtomDB
pidata = pyatomdb.atomdb.get_data(Z, z1, 'PI')

# set up the figure
fig = pylab.figure()
fig.show()
ax = fig.add_subplot(111)


# to calculate the cross section (in cm^2) at a given single energy E (in keV)
# does not currently work with vector input, so have to call in a loop if you
# want multiple energies [I will fix this]

E = 10.

# get the ground level (the 0th entry in LV file) data
lvd = lvdata[1].data[0]

# This is the syntax for calculating the PI cross section of a given line
# This will work for non XSTAR data too.
sigma = pyatomdb.atomdb.sigma_photoion(E, Z, z1,lvd['phot_type'], lvd['phot_par'], \
```

```
                xstardata=pidata, xstarfinallev=1)


# To get the raw XSTAR cross sections (units: energy = keV, cross sections = Mb)
# for level 1 -> 1 (ground to ground)
pixsec = pyatomdb.atomdb.sort_pi_data(pidata, 1,1)
ax.loglog(pixsec['energy'], pixsec['pi_param']*1e-18, label='raw xstar data')

# label the plot
ax.set_title('Plotting raw XSTAR PI cross sections. Fe XVII gnd to Fe XVIII gnd')
ax.set_xlabel("Energy (keV)")
ax.set_ylabel("PI cross section (cm$^{2}$)")

pylab.draw()
zzz=raw_input('press enter to continue')
```

2.7.4 Make a Spectrum

Make a broadened and unbroadened spectrum: `make_spectrum.py`

```
import pyatomdb, numpy, pylab

# set up a grid of energy bins to model the spectrum on:
ebins=numpy.linspace(0.3,10,1000)

# define a broadening, in keV, for the lines
de = 0.01

# define the temperature at which to plot (keV)
te = 3.0

# find the index which is closest to this temperature
ite = pyatomdb.spectrum.get_index( te, teunits='keV', logscale=False)

# create both a broadened and an unbroadened spectrum
a = pyatomdb.spectrum.make_spectrum(ebins, ite,dummyfirst=True)
b = pyatomdb.spectrum.make_spectrum(ebins, ite, broadening=de, \
                                    broadenunits='kev',dummyfirst=True)
# The dummyfirst argument adds an extra 0 at teh beginning of the
# returned array so it is the same length as ebins. It allows
# accurate plotting using the "drawstyle='steps'" flag to plot.

# plot the results
fig = pylab.figure()
fig.show()
ax = fig.add_subplot(111)

ax.loglog(ebins, a, drawstyle='steps', label='Unbroadened')
ax.loglog(ebins, b, drawstyle='steps', label='sigma = %.2f'%(de))
ax.set_xlabel('Energy (keV)')
ax.set_ylabel('Emissivity (ph cm$^{3}$ s$^{-1}$ bin$^{-1}$)')
ax.legend(loc=0)
pylab.draw()
zzz = raw_input("Press enter to continue")

print "Listing lines between 1 and 2 A"
# now list the lines in a wavelength region
```

```
llist = pyatomdb.spectrum.list_lines([1,2.0], index=ite)
# print these to screen
pyatomdb.spectrum.print_lines(llist)
# print to screen, listing the energy, not the wavelength
print "Listing lines between 1 and 2 A, using keV."

pyatomdb.spectrum.print_lines(llist, specunits = 'keV')
```

2.7.5 Make a Spectrum version 2.0

Make a spectrum using the new Session class: `new_make_spectrum.py`. This is significantly faster if you need to make lots of spectra (fitting, interpolating between 2 temperatures etc). Note the example requires an RMF and ARF file - adjust to fit your available response.

```
import pyatomdb, pylab, numpy, time


rmf = '/export1/projects/atomdb_308/hitomi/resp_100041010sxs.rmf'
arf = '/export1/projects/atomdb_308/hitomi/arf_100041010sxs.arf'


# create a session object. This contains the apec files, response files,
# and any previously calculated spectrs so that simple multiplication
# can be used to get the results without recalculating everything from scratch

data = pyatomdb.spectrum.Session()

# If you want to specify custom energy bins:
ebins = numpy.linspace(1,2,1001)
data.set_specbins(ebins, specunits='A')

# alternative method: just load the response and use its binning. Note
# that this will always be in keV currently, because reasons.

data.set_response(rmf, arf=arf)
ebins = data.ebins_response

# now get the spectrum at 4keV. This calculates (and stores) the
# spectrum at each temperature nearby (~3.7, 4.3 keV)
# then linearly interpolates between the result
#
# vector is stored for each element at each temperature
# so if you change temperature/abundance, it's a simple multiplication and
# interpolation instead of a total recalculation

t0 = time.time()
s=data.return_spectra(4.0, teunit='keV')
t1 = time.time()
# let's change the abundance
data.set_abund([1,2,3,4,26],0.5)

# and see how fast this goes this time, changing temperature and abund
s2=data.return_spectra(4.1, teunit='keV')
t2 = time.time()

print "first spectrum took %g seconds" %(t1-t0)
print "second spectrum took %g seconds" %(t2-t1)
```

```
print "note how much faster the second one was as I didn't recalculate everything from scratch!"


#linedata = pyatomdb.pyfits.open('/export1/atomdb_latest/apec_v3.0.8_line.fits')


#spec = speclo*rlo + specup*rup

# some plotting of things

fig = pylab.figure()
fig.show()
ax = fig.add_subplot(111)
#ax2 = fig.add_subplot(212, sharex=ax)
s = numpy.append(0,s)
s += 1e-40

s2 = numpy.append(0,s2)
s2 += 1e-40
#spec = numpy.append(0,spec)
#spec += 1e-40

ax.plot(ebins, s, drawstyle='steps')
ax.plot(ebins, s2, drawstyle='steps')
#ax.plot(elo, spec, drawstyle='steps')


#ax2.plot(elo, s/spec, drawstyle='steps')


zzz=raw_input('Press enter to exit')
```

2.8 License

Pyatomdb is released under the Smithsonian License:

2.9 Usage

2.9.1 Examples

Note: there are example routines demonstrating use of these features in the examples directory of the package.

2.9.2 Installation

PyAtomDB can be installed from pypi, using the simple `pip install pyatomdb` command.

For PyAtomDB to be useful, it requires access to a range of AtomDB database files (these are all FITS files). The database has two broad types of files, emissivity files (APEC) and fundamental atomic data files (APED, the Astrophysical Plasma Emission Database).

The emissivity files are needed for things such as producing spectra. The APED files are underlying atomic data and are not strictly needed for creating a spectrum, but can be useful for getting later information out.

In order for PyAtomDB to work efficiently, you should choose a location to store all of these files (e.g. /home/username/atomdb). It is strongly recommended that you set the environment variable ATOMDB to point to this, i.e. for bash add the following line to your .bashrc file:

```
export ATOMDB=/home/username/atomdb
```

or for csh, add this to your .cshrc or .cshrc.login:

```
setenv ATOMDB /home/username/atomdb
```

If you run the following code within a python shell, PyAtomDB will download the files you need to get started:

```
import pyatomdb
pyatomdb.util.initialize()
```

This will prompt you for an install location (defaulting to *$ATOMDB*) and whether to download the emissivity files. It is suggested that you say yes. It will also ask if you mind sharing anonymous download information with us. We would appreciate it if you say yes, but it is not necessary for the functioning of the software.

2.9.3 Example: Making a Spectrum

These functions are in the `spectrum` module:

```
import pyatomdb, numpy, pylab

# set up a grid of energy bins to model the spectrum on:
ebins=numpy.linspace(0.3,10,1000)

# define a broadening, in keV, for the lines
de = 0.01

# define the temperature at which to plot (keV)
te = 3.0

# find the index which is closest to this temperature
ite = pyatomdb.spectrum.get_index( te, teunits='keV', logscale=False)

# create both a broadened and an unbroadened spectrum
a = pyatomdb.spectrum.make_spectrum(ebins, ite,dummyfirst=True)
```

```
b = pyatomdb.spectrum.make_spectrum(ebins, ite, broadening=de, \
                                    broadenunits='kev',dummyfirst=True)
# The dummyfirst argument adds an extra 0 at teh beginning of the
# returned array so it is the same length as ebins. It allows
# accurate plotting using the "drawstyle='steps'" flag to plot.

# plot the results
fig = pylab.figure()
fig.show()
ax = fig.add_subplot(111)

ax.loglog(ebins, a, drawstyle='steps', label='Unbroadened')
ax.loglog(ebins, b, drawstyle='steps', label='sigma = %.2f'%(de))
ax.set_xlabel('Energy (keV)')
ax.set_ylabel('Emissivity (ph cm$^{3}$ s$^{-1}$ bin$^{-1}$)')
ax.legend(loc=0)
pylab.draw()
zzz = raw_input("Press enter to continue")

print "Listing lines between 1 and 2 A"
# now list the lines in a wavelength region
llist = pyatomdb.spectrum.list_lines([1,2.0], index=ite)
# print these to screen
pyatomdb.spectrum.print_lines(llist)
# print to screen, listing the energy, not the wavelength
print "Listing lines between 1 and 2 A, using keV."

pyatomdb.spectrum.print_lines(llist, specunits = 'keV')
```

2.9.4 Interrogating the atomic database

The atomic database APED contains a range of data for a host of different ions. It contains a host of different files covering a range of different processes. The full database, when uncompressed is more than 10GB of data, so we are avoiding distributing it to all users. You can, however, get the individual data you need using the `get_data` routine:

```
mydata = pyatomdb.atomdb.get_data(Z, z1, ftype)
```

This will try to open the file locally if it exists, and if it does not it will then go to the AtomDB FTP server and download the data for element Z, ion z1, with ftype a 2-character string denoting the type of data to get:

- `IR`: ionization and recombination
- `LV`: energy levels
- `LA`: radiative transition data (lambda and A-values)
- `EC`: electron collision data
- `PC`: proton collision data
- `DR`: dielectronic recombination satellite line data
- `PI`: XSTAR photoionization data
- `AI`: autoionization data

So to open the energy levels for oxygen with 2 electrons (O 6+, or O VII):

```
lvdata = pyatomdb.atomdb.get_data(8,7,'LV')
```

Downloaded data files are stored in `$ATOMDB/APED/<elsymb>/<elsymb>_<ionnum>/`. You can delete them if you need to free up space, whenever a code needs the data it will reload them. There are many routines in the atomdb module which relate to extracting the data from the files, i.e. getting collisional excitation rates or line wavelengths. If you have trouble finding a routine to do what you want, please contact us and we'll be happy to write one if we can (this is how this module will grow - through user demand!)

CHAPTER 3

Indices and tables

- *genindex*
- *modindex*
- *search*