PyAtomDB Documentation

Release 0.8.1

Adam Foster

Mar 11, 2020
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>2 Contents</td>
<td>3</td>
</tr>
<tr>
<td>3 Indices and tables</td>
<td>79</td>
</tr>
<tr>
<td>Python Module Index</td>
<td>81</td>
</tr>
<tr>
<td>Index</td>
<td>83</td>
</tr>
</tbody>
</table>
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**: simple 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](https://github.com)
2.1 PyAtomDB APEC module

This module 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.

The apec module contains routines crucial for the APEC code. This also includes some interfaces to external C libraries (or will, eventually).

Version 0.1 - initial release Adam Foster September 16th 2015

pyatomdb.apec.calc_brems_gaunt(E, T, z1, brems_type, datacache=False, settings=False)

calculate the bremsstrahulung free-free gaunt factor

Parameters

- E [float] Energy (in keV) to calculate gaunt factor
- T [float] Temperature (in K) of plasma
- z1 [int] Ion charge +1 of ion (e.g. 6 for C VI)

settings [dict] See description in atomdb.get_data
datacache [dict] Used for caching the data. See description in atomdb.get_data

Returns
gaunt_ff [float] The gaunt factor for the free-free process.

pyatomdb.apec.calc_cascade_population (matrixA, matrixB)

pyatomdb.apec.calc_ee_brems (E, T, N)
calculate the electron-electron bremsstrahlung.

Parameters

E [array (float)] energy grid (keV)
T [float] Electron temperature (keV)
N [float] electron density (cm^-3)

Returns

array(float) ee_brems in photons cm^s s^-1 keV^-1 at each point E. This should be multiplied by the bin width to get flux per bin.

References

Need to check this!

pyatomdb.apec.calc_full_ionbal (Te, tau=False, init_pop=False, Te_init=False, Zlist=False, teunit='K', extrapol=True, cie=True, settings=False)
Calculate the ionization balance for all the elements in Zlist.

One of init_pop or Te_init should be set. If neither is set, assume all elements start from neutral.

Parameters

Te [float] electron temperature in keV or K (default K)
tau [float] N_e * t for the non-equilibrium ionization (default False, i.e. off)
init_pop [dict of float arrays, indexed by Z] initial populations. E.g. init_pop[6]=[0.1,0.2,0.3,0.2,0.2,0.0,0.0]
Te_init [float] initial ionization balance temperature, same units as Te
Zlist [int array] array of nuclear charges to include in calculation (e.g. [8,26] for oxygen and iron)
teunit ["K", "keV"] units of temperatures (default K)
extrapol [bool] Extrapolate rates to values outside their given range. (default False)
cie [bool] If true, collisional ionization equilibrium calculation (tau, init_pop, Te_init all ignored)

Returns

final_pop [dict of float arrays, indexed by Z] final populations. E.g. final_pop[6]=[0.1,0.2,0.3,0.2,0.2,0.0,0.0]
pyatomdb.apec.calc_ioniz_popn(levpop, Z, z1, z1_drv, T, Ne, settings=False, datacache=False, do_xi=False)

Calculate the level population due to ionization into the ion

Parameters

levpop: array(float) The level population of the parent ion. Should already have abundance and ion fraction built in.

Z: int
z1: int
z1_drv: int
T: float
Ne: float
settings: dict
datacache: dict
do_xi: bool Include collisional ionization

Returns

levpop_out: array(float) The level populations of the Z,z1 ion

pyatomdb.apec.calc_recomb_popn(levpop, Z, z1, z1_drv, T, dens, drlevrates, rrlevrates, settings=False, datacache=False, dronly=False, rronly=False)

Calculate the level population of a recombined ion

Parameters

levpop: array(float) Level populations, already taking into account elemental abundance and ion fraction of z1_drv

Z: int
z1: int
z1_drv: int
T: electron temperature (K)
dens: electron density (cm^-3)
drlevrates: array(float) Rates into each level from DR calculations
rrlevrates: array(float) Rates into each level from RR calculations

Returns

array(float) Level population

pyatomdb.apec.calc_satellite(Z, z1, T, datacache=False, settings=False)

Calculate DR satellite lines

Parameters

Z: int The nuclear charge of the element
z1: int Recombined Ion charge +1 of ion (e.g. 5 for C VI -> C V)
t: float The electron temperature (K)
settings: dictionary The settings read from the apec.par file by parse_par_file

Returns
array(linelist) List of DR lines
array(levlistin) Rates into each lower level, driven by DR

`pyatomdb.apec.calc_total_coco(cocodata, settings)`
Calculate the total emission in erg cm^3 s^-1

`pyatomdb.apec.compress_continuum(xin, yin, tolerance, minval=0.0)`
Compress the continuum into linear interpolatable grids

**Parameters**
- `xin` [array(float)] The bin edges (keV)
- `yin` [array(float)] The continuum in photons (or ergs) cm^3 s^-1 bin^-1. Should be 1 element shorter than xin
- `tolerance` [float] The tolerance of the final result (if 0.01, the result will always be within 1% of the original value)

**Returns**
- `xout` [array (float)] The energy points of the compressed energy grid (keV)
- `yout` [array (float)] The continuum, in photons(or ergs) cm^3 s^-1 keV^-1

`pyatomdb.apec.continuum_append(a, b)`
Join two continuum arrays together, expanding arrays as necessary

**Parameters**
- `a`: numpy.array(dtype=continuum) The first array
- `b`: numpy.array(dtype=continuum) The second array

**Returns**
- `c`: numpy.array(dtype=continuum) The two arrays combined, with continuum arrays resized as required.

`pyatomdb.apec.create_chdu_cie(cocodata)`
`pyatomdb.apec.create_cparamhdu_cie(cocodata)`
`pyatomdb.apec.create_lhdu_cie(linedata)`
`pyatomdb.apec.create_lhdu_nei(linedata)`
`pyatomdb.apec.create_lparamhdu_cie(linedata)`
`pyatomdb.apec.do_brems(Z, z1, T, abund, brems_type, eedges)`
Calculate the bremsstrahlung emission in units of photon cm^3 s^-1 bin^-1

**Parameters**
- `Z` [int] nuclear charge for which result is required
- `z1` [int] ion charge +1
- `T` [float] temperature (Kelvin)
- `abund` [float] elemental abundance (should be between 1.0 and 0.0)
  3 = RELATIVISTIC = Relativistic: 1998ApJ...507..530N
  4 = BREMS_NONE = no bremsstrahlung
**edges**  [array(float)] The energy bin edges for the spectrum (keV)

**Returns**

array(float)  bremstrahlung emission in units of photon cm^3 s^-1 bin^-1

```python
pyatomdb.apec.do_lines(Z, z1, lev_pop, N_e, datacache=False, settings=False, z1_drv_in=-1)
```

Convert level populations into line lists

**Parameters**

- **Z**: int  The nuclear charge of the element
- **z1**: [int]  Ion charge +1 of ion (e.g. 6 for C VI)
- **lev_pop**: [array(float)]  The level population for the ion. Should already have elemental abundance and ion fraction multiplied in.
- **N_e**: [float]  Electron Density (cm^-3)
- **datacache**: [dict]  Used for caching the data. See description in atomdb.get_data
- **settings**: [dict]  See description in atomdb.get_data
- **z1_drv_in**: [int]  the driving ion for this calculation, if not z1 (defaults to z1)

**Returns**

- **linelist**: numpy.dtype(linetype)  The list of lines and their emissivities. see generate_datatypes
- **twophot**: array(float)  The two-photon continuum on the grid specified by the settings If settings['TwoPhoton'] is False, then returns a grid of zeros.

```python
pyatomdb.apec.extract_gauntff(Z, gamma2, gaunt_U, gaunt_Z, gaunt_Ng, gaunt_g2, gaunt_gf)
```

Extract the appropriate Gaunt free-free factor from the relativistic data tables of Nozawa, Itoh, & Kohyama, 1998 ApJ, 507,530

**Parameters**

- **Z**: int  Z for which result is required
- **gamma2**: [array(float)]  gamma^2 in units of Z^2 Rydbergs/kT
- **gaunt_U**: [array(float)]  u=E/kT
- **gaunt_Z**: [array(int)]  nuclear charge
- **gaunt_Ng**: [array(int)]  number of gamma^2 factors
- **gaunt_g2**: [array(float)]  gamma^2 factors
- **gaunt_gf**: [array(float)]  ff factors

**Returns**

array(float)  Gaunt factors.

**References**


```python
pyatomdb.apec.gather_rates(Z, z1, te, dens, datacache=False, settings=False, do_la=True, do_ai=True, do_ec=True, do_pc=True, do_ir=True)
```

fetch the rates for all the levels of Z, z1

**Parameters**  

2.1. PyAtomDB APEC module
Z: int  The nuclear charge of the element
z1  [int] ion charge +1
te  [float] temperature (Kelvin)
dens: float  electron density (cm^-3)
settings [dict] See description in atomdb.get_data
datacache [dict] Used for caching the data. See description in atomdb.get_data

Returns
up: numpy.array(float)  Initial level of each transition
lo: numpy.array(float)  Final level of each transition
rate: numpy.array(float)  Rate for each transition (in s^-1)

pyatomdb.apec.generate_apec_headerblurb (settings, linehdulist, cocohdulist)
Generate all the headers for an apec run, and apply them to the HDUlist.

Parameters
settings: dict  The output of read_apec_parfile
hdulist [list or array of fits HDUs]  The hdus to have headings added.

Returns
None

pyatomdb.apec.generate_cie_outputs (settings, Z, linelist, contlist, pseudolist)
Convert a linelist and continuum values into an equilibrium AtomDB fits output

Parameters
settings: dictionary  The settings read from the apec.par file by parse_par_file
Z: int  The nuclear charge of the element
linelist: numpy.array(dtype=linelisttype)  The list of lines, separated by ion
contlist: dict  Dictionary with the different continuum contributions from each ion. Each is an array of ph cm^3 s^-1 bin^-1
pseudolist: dict  Dictionary with the different pseudocontinuum contributions from each ion. Each is an array of ph cm^3 s^-1 bin^-1

Returns
None

pyatomdb.apec.generate_datatypes (dtype, npseudo=0, ncontinuum=0)
returns the various data types needed by apec

Parameters
dtype [string]  One of “linetype”, “cielinetype”, “continuum”
npseudo [int (default=0)]  Number of pseudocontinuum points for “continuum” type
ncontinuum [int (default=0)]  Number of continuum points for “continuum” type

Returns
numpy.dtype  The data dtype in question
pyatomdb.apec.generate_nei_outputs (settings, Z, linelist, contlist, pseudolist, ionfrac_nei)
Convert a linelist and continuum values into a non-equilibrium AtomDB fits output

Parameters

settings: dictionary  The settings read from the apec.par file by parse_par_file
Z: int  The nuclear charge of the element
linelist: numpy.array(dtype=linelisttype)  The list of lines, separated by ion
contlist: dict  Dictionary with the different continuum contributions from each ion. Each is an
array of ph cm^3 s^-1 bin^-1
pseudolist: dict  Dictionary with the different pseudocontinuum contributions from each ion.
Each is an array of ph cm^3 s^-1 bin^-1

Returns
None

pyatomdb.apec.kurucz (uin, gam)
Correction factors to Kellogg bremsstrahlung calculation by Bob Kurucz

Parameters
uin [array(float)] energy grid, units of E/kT (both in keV)
gam [array(float)] Z**2/T, in units of Rydbergs

Returns
array(float)  gaunt factors at high gam (> 0.1)

pyatomdb.apec.make_vector (linear, minval, step, nstep)
Create a vector from the given inputs

Parameters
linear: boolean  Whether the array should be linear or log spaced
minval: float  initial value of the array. In dex if linear==False
step: float  step between points on the array. In dex if linear==False
nstep: int  number of steps

Returns
array(float)  array of values spaced out use the above parameters

pyatomdb.apec.make_vector_nbins (linear, minval, maxval, nstep)
Create a vector from the given inputs

Parameters
linear: boolean  Whether the array should be linear or log spaced
minval: float  initial value of the array. In dex if linear==False
maxval: float  maximum value of the array. In dex if linear==False
nstep: int  number of steps

Returns
array(float)  array of values spaced out use the above parameters
pyatomdb.apec.parse_par_file(fname)
Parse the apec.par input file for controlling APEC

Parameters
fname [string] file name

Returns
dict The settings in “key:value” pairs.

pyatomdb.apec.run_apec(fname)
Run the entire APEC code using the data in the parameter file fname

Parameters
fname [string] file name

Returns
None

pyatomdb.apec.run_apec_element(settings, te, dens, Z)
Run the APEC code using the settings provided for one element

Parameters
settings: dictionary The settings read from the apec.par file by parse_par_file
te: float The electron temperature (K)
dens: float The electron density (cm^-3)
Z: int The nuclear charge of the element

Returns
None

pyatomdb.apec.run_apec_ion(settings, te, dens, Z, z1, ionfrac, abund)
Run the APEC code using the settings provided for an individual ion.

Parameters
settings: dictionary The settings read from the apec.par file by parse_par_file
te: float The electron temperature (K)
dens: float The electron density (cm^-3)
Z: int The nuclear charge of the element
z1: int The ion charge +1 of the ion
ionfrac: float The fractional abundance of this ion (between 0 and 1)
abund: float The elemental abundance of the element (normalized to H)

Returns
linelist [numpy array] List of line details and emissivities
continuum [array] Continuum emission in photons bin-1 s-1. This is a 3-item dict, with “rrc”, “twophot”, “brems” entries for each continuum source
pseudocont [array] Pseudo Continuum emission in photons bin-1 s-1

pyatomdb.apec.run_wrap_run_apec(fname, Z, iTe, iDens)
After running the APEC code ion by ion, use this to combine into FITS files.
Parameters

fname  [string] file name of par file
Z:  int  The atomic numbers
iTe:  int  The temperature index
iDens:  int  The density index

Returns

None

pyatomdb.apec.solve_ionbal (ionrate, recreate, init_pop=False, tau=False)
solve_ionbal: given a set of ionization and recombination rates, find the equilibrium ionization balance. If init_pop and tau are set, do an non-equilibrium calculation starting from init_pop and evolving for n_e * t = tau (cm^-3 s)

Parameters

ionrate  [float array] the ionization rates, starting with neutral ionizing to +1
recreate  [float array] the recombination rates, starting with singly ionized recombining to neutral
init_pop  [float array] initial population of ions for non-equilibrium calculations. Will be renormalised to 1.
tau  [float] N_e * t for the non-equilibrium ionization

Returns

final_pop  [float array] final populations.

Notes

Note that init_pop & final_pop will have 1 more element than ionrate and recreate.

pyatomdb.apec.solve_ionbal_eigen (Z, Te, init_pop=False, tau=False, Te_init=False, teunit='K',
filename=False, datacache=False, debug=False)
Solve the ionization balance for a range of ions using the eigenvector approach and files as distributed in XSPEC.

Parameters

Z  [int] atomic number of element
Te  [float or array] electron temperature(s), default in K
init_pop  [float array] initial population of ions for non-equilibrium calculations. Will be renormalised to 1.
tau  [float or array] N_e * t for the non-equilibrium ionization, in cm^3 s^-1.
Te_init  [float] initial ionization balance temperature, same units as Te
teunit  [[‘K’, ‘keV’]] units of temperatures (default K)
filename  [string] Can optionally point directly to the file in question, i.e. to look at older data look at $HEADAS/../spectral/modelData/eigenELSYMB_v3.0.fits. If not set, download from AtomDB FTP site.
datacache  [dict] Used for caching the data. See description in atomdb.get_data

Returns

final_pop  [float array] final populations.
pyatomdb.apec.solve_level_pop (init, final, rates, settings)
Solve the level population

Parameters
- **init** [array(int)] The initial level for each transition
- **final** [array(int)] The initial level for each transition
- **rates** [array(float)] The rate for each transition
- **settings**: dictionary The settings read from the apec.par file by parse_par_file

Returns
- array(float) The level population

pyatomdb.apec.wrap_ion_directly (fname, ind, Z, z1)

pyatomdb.apec.wrap_run_apec (fname, readpickle=False, writepickle=False)
After running the APEC code ion by ion, use this to combine into FITS files.

Parameters
- **fname** [string] file name
- **readpickle** [bool] Load apec results by element from pickle files, instead of regenerating

Returns
- None

pyatomdb.apec.wrap_run_apec_element (settings, te, dens, Z, ite, idens, writepickle=False, readpickle=False)
Combine wrap_run_apec_ion results for an element

Parameters
- **settings**: dictionary The settings read from the apec.par file by parse_par_file
- **te**: float The electron temperature (K)
- **dens**: float The electron density (cm^-3)
- **Z**: int The nuclear charge of the element
- **ite**: int The temperature index
- **idens**: int The density index
- **writepickle**: bool Dump data into a pickle file. Useful for rapidly combining data after runs.
- **readpickle**: bool Read data from a pickle file. Useful for rapidly combining data after runs. Usually the result of a previous call using writepickle=True

Returns
- None

### 2.2 PyAtomDB Atomic module

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

atomic.py contains routines related to basic atomic data, e.g. converting integer nuclear charge to element symbols, etc.
pyatomdb.atomic.Z_to_mass(Z, raw=False)

Converts element symbol to atomic mass, e.g. “C” -> 12.0107

Isotope fractions based on those found in earth’s crust samples, your astrophysical object may vary.

**Parameters**

- **Z** [int] nuclear charge, e.g 6 for C
- **raw** [bool] if true, ignore Z, and return the entire mass list as an array with a 0 at the beginning

**Returns**

- **float** mass in a.m.u. for the element. (e.g. 12.0107 for C)

**References**

Atomic masses are taken from: Pure Appl. Chem. 81 NO 11, 2131-2156 (2009) Masses for Technetium, Promethium, Polonium, Astatine, Radon, Francium, Radium & Actinium are estimates. If you need these you probably aren’t doing astronomy...

pyatomdb.atomic.Ztoelname(Z)

**Returns** element name of element with nuclear charge Z.

**Parameters**

- **Z** [int] nuclear charge of element (e.g. 6 for carbon)

**Returns**

- **str** element name (e.g. “Carbon” for carbon)

pyatomdb.atomic.Ztoelsymb(Z)

**Returns** element symbol of element with nuclear charge Z.

**Parameters**

- **Z** - nuclear charge of element (e.g. 6 for carbon)

**Returns**

- element symbol (e.g. “C” for carbon)

Version 0.1 28 July 2009

Adam Foster

pyatomdb.atomic.config_to_occup(cfgstr, nel=-1, shlmax=-1, noccup=[-1])

pyatomdb.atomic.elsymb_to_Z(elsymb)

Converts element symbol to nuclear charge, e.g. “C” -> 6

**Parameters**

- **elsymb** [str] Element symbol, e.g. “C”. Case insensitive.

**Returns**

- **int** Z for the ion. (e.g. 6 for C)

pyatomdb.atomic.elsymb_to_z0(elsymb)

Converts element symbol to nuclear charge, e.g. “C” -> 6 (wrapper to elsymb_to_Z, retained for consistency)
Parameters

elsymb [str] Element symbol, e.g. “C”. Case insensitive.

Returns

int Z for the ion. (e.g. 6 for C)

pyatomdb.atomic.get_maxn(cfgstr)
pyatomdb.atomic.get_parity(cfgstr)
pyatomdb.atomic.int2roman(number)
pyatomdb.atomic.int_to_roman(input)
   Convert an integer to Roman numerals.
pyatomdb.atomic.occup_to_cfg(occlist)
pyatomdb.atomic.occup_to_config(occup)
pyatomdb.atomic.parse_config(cfgstr)
pyatomdb.atomic.parse_eissner(cfgstr, nel=0)
pyatomdb.atomic.roman_to_int(input)
   Convert a roman numeral to an integer.
pyatomdb.atomic.shorten_config(cfgstr, nel=0)
   Shorten the configuration as required

Parameters

cfgstr [string] configuration string. Should be simplified already e.g. ‘1s2 2s2 3p1’

Returns

cfgshrt [string] shortened configuration, e.g. ‘3p1’

pyatomdb.atomic.spectroscopic_name(Z, z1)
   Converts Z,z1 to spectroscopic name, e.g. 6,5 to “C V”

Parameters

Z [int] nuclear charge (e.g. 6 for C)

z1 [int] ion charge +1 (e.g. 5 for C4+)

Returns

str spectroscopic symbol for ion (e.g. “C V” for C+4)

pyatomdb.atomic.spectroscopictoz0(name)
   Converts spectroscopic name to Z, z1, e.g. “C V” to 6,5

Parameters

name [str] Ion name, e.g. “C V”

Returns

int, int Z, z1 for the ion. (e.g. 6,5 for C V)

pyatomdb.atomic.z0_to_mass(z0)
   Converts element symbol to atomic mass, e.g. “C” -> 12.0107
   (wrapper to Z_to_mass, retained for consistency)

Isotope fractions based on those found in earth’s crust samples, your astrophysical object may vary.
Parameters

\( z_0 \) [int] nuclear charge, e.g 6 for C

Returns

float mass in a.m.u. for the element. (e.g. 12.0107 for C)

References

Atomic masses are taken from: Pure Appl. Chem. 81 NO 11, 2131-2156 (2009) Masses for Technetium, Promethium, Polonium, Astatine, Radon, Francium, Radium & Actinium are estimates. If you need these you probably aren’t doing astronomy...

\texttt{pyatomdb.atomic.z0toelname(z0)}

Returns element name of element with nuclear charge \( z_0 \). (wrapper to Ztoelname for compatibility purposes)

Parameters

\( z_0 \) [int] nuclear charge of element (e.g. 6 for carbon)

Returns

str element name (e.g. “Carbon” for carbon)

\texttt{pyatomdb.atomic.z0toelsymb(z0)}

Returns element symbol of element with nuclear charge \( z_0 \). (wrapper to Ztoelsymb for compatibility purposes)

Parameters

\( z_0 \) [int] nuclear charge of element (e.g. 6 for carbon)

Returns

str element symbol (e.g. “C” for carbon)

2.3 PyAtomDB AtomDB module

This module is designed to interact with the main atomic database, extracting real values of coefficients and so on. The atomdb module contains several routines for interfacing with the AtomDB database to extract useful physical quantities, line lists, write new fits files and more. It is currently a dump of everything I’ve done with AtomDB. This should all be considered unstable and possibly susceptible to being wrong. It will be fixed, including moving many routines out of this library, as time goes on.

Version 0.1 - initial release Adam Foster July 17th 2015
Version 0.2 - added PI reading routines and get_data online enhancements. Adam Foster August 17th 2015
Version 0.3 - added RRC generation routines Adam Foster August 28th 2015

\texttt{pyatomdb.atomdb.A_twoph(A, E0, E)}

Convert the A value into energy distribution for 2-photon transitions

Parameters

A [float] Einstein A for transition
E0 [float] Energy in keV of transition
E [array(float)] Energies of each bin to output continuum at (keV)

Returns
array(float) Distribution of transition rate amongst bins E (s^-1)

References

From Nussbaumer & Schmutz, 1984, A+A, 138,495 Z is the element, and E is the energy of the bin, in keV y is unitless, and is equal to nu/nu0 = lambdat/lambda, where lambda0 = 1215.7 A for hydrogen—the base wavelength of the 2s->1s transition. This fit is accurate to better than 0.6% for 0.01 < y < 0.99

The A_norm is the A value for neutral hydrogen for this transition. For other transitions, we renormalize to the appropriate A value.

This routine is used for BOTH hydrogenic and He-like two-photon distributions. This is justified using the result of Derevianko & Johnson, 1997, Phys Rev A, 56, 1288 who show in Figures 5 and 2 of that paper that the difference is everywhere less than 10% between these two for Z=6-28 – it is about 5% or so.

pyatomdb.atomdb.B_hyd(s, l, m, eta)
pyatomdb.atomdb.G_hyd(l, m, eta, rho)
pyatomdb.atomdb.addline(xbins, yvals, wv, amp, dx)
pyatomdb.atomdb.addline2(xbins, wv, amp, dx)

Calculate the collisional ionization rates using the Dere 2007 method

Parameters

Te [float or array(float)] Electron temperature (K)
ionpot [float] Ionization potential (eV)
Tscal [array(float)] scaled temperatures
Upsscal [array(float)] scaled upsilons

Returns

float or array(float) Ionization rate in cm^3 s^-1

References

2007A&A...466..771D

pyatomdb.atomdb.calc_ionrec_ci(cidat, Te, extrap=False, ionpot=False)
pyatomdb.atomdb.calc_ionrec_dr(cidat, Te, extrap=False)
pyatomdb.atomdb.calc_ionrec_ea(cidat, Te, extrap=False)
pyatomdb.atomdb.calc_ionrec_rr(cidat, Te, extrap=False)
pyatomdb.atomdb.calc_kato(coll_type, par, Z, Te)
pyatomdb.atomdb.calc_maxwell_rates(coll_type, min_T, max_T, Tarr, om, dE, T, Z, degl, degu, quiet=False, levdat=False, ladat=False, lollev=False, uplev=False, force_extrap=False, did_extrap=False, datacache=False)
pyatomdb.atomdb.calc_rad_rec_cont(Z, z1, z1_drv, T, ebins, abund=1.0, ion_pop=1.0, settings=False, datacache=False)

Calculate the radiative recombination continuum for an ion at temperature T
Parameters

- **Z** [int] nuclear charge
- **z1** [int] recombined ion charge+1
- **z1_drv** [int] recombining ion charge+1
- **T** [float] temperature (K)
- **ebins** [array(float)] energy bins (in keV) on which to calculate the spectrum
- **abund** [float] elemental abundance, relative to hydrogen
- **ion_pop** [float] the ion’s population fraction of that element (i.e. sum of all ion_pop for an element = 1)

Returns

- **array(float)** RRC in photons cm^3 s^-1 bin^-1, in an array of length(ebins)-1
- **array(float)** Recombination rates into the excited levels, in s^-1

```python
calc_rcc(Z, z1, eedges, Te, lev, xstardat=False, xstarlevfinal=1, settings=False, datacache=False, returntotal=False)
```
Calculate the radiative recombination continuum for a given ion

Parameters

- **Z** [int] Atomic number
- **z1** [int] recombined ion charge
- **eedges** [array(float)] the bin edges for the spectrum to be calculated on (keV)
- **Te** [float] The electron temperature (K)
- **lev** [int] The level of the ion for the recombination to be calculated into
- **xstardat** [dict or HDUList] The xstar PI data. This can be an already sorted dictionary, as returned by sort_xstar_data, or the raw results of opening the PI file
- **xstarlevfinal** [int] If you need to identify the recombining level, you can do so here. Should normally be 1.
- **settings** [dict] See description in read_data
- **datacache** [dict] See description in read_data
- **returntotal** [bool] If true, return the total recombination rate as well

Returns

- **array(float)** The rrc in photons cm^3 s^-1 keV^-1
- **optional float** If returntotal is set, also return total RRC calculated by separate integral from the ionization edge to infinity.

```python
calc_sampson_h(om, Z, Te)
calc_sampson_p(om, Z, Te)
calc_sampson_s(om, Z, Te)
calc_spline_atomdb(xa, ya, y2a, n, x)
calc_two_phot(wavelength, einstein_a, lev_pop, ebins)
```
Calculate two photon spectrum

Parameters
wavelength [float] Wavelength of the transition (Angstroms)

einstein_a [float] The Einstein_A parameter for the transition

lev_pop [float] The level population for the upper level

ebins [array(float)] The bin edges for the spectrum (in keV)

Returns

array(float) The flux in photons cm^{-3} s^{-1} bin^{-1} array is one element shorter than ebins.

pyatomdb.atomdb.ci_younger (Te, c)
Calculates Collisional Ionization Rates from Younger formula

Parameters

Te [array(float)] Temperatures in Kelvin

c [the ionrec_par from the transition in the AtomDB IR file]

Returns

array(float) returns ionization rate in cm^{3} s^{-1}

pyatomdb.atomdb.dr_badnell (Te, c)
Convert data from Badnell constants into a DR Rate

Parameters

Te [float or array(float)] Electron temperature[s] in K

c [array] Constants from DR rates. Stored as alternating pairs in AtomDB, so c1,e1,c2,e2,c3,e3 etc in the IONREC_PAR column

Returns

float DR rate in cm^{3} s^{-1}

References

See http://amdpp.phys.strath.ac.uk/tamoc/DATA/DR/

pyatomdb.atomdb.dr_mazzotta (Te, c)

pyatomdb.atomdb.ea_mazzotta (Te, c, par_type)

Te is an array in Kelvin c is the ionrec_par par_type is the number denoting the type of the parameter returns excitation-autoionization rate in cm^{3} s^{-1}

pyatomdb.atomdb.ea_mazzotta_iron (T_eV, c)

pyatomdb.atomdb.extract_n (conf_str)

pyatomdb.atomdb.f1_fcn (x)

pyatomdb.atomdb.f2_fcn (x)

pyatomdb.atomdb.get_abundance (abundfile=False, abundset='AG89', element=[-1], datacache=False, settings=False)

Get the elemental abundances, relative to H (H=1.0)

Parameters

abundfile [string] special abundance file, if not using the default from filemap

abundset [string] Abundance set. Available:
• GA88: Grevesse, N, and Anders, E.1988, Cosmic abundances of matter, ed. C. J. Waddington, AIP Conference, Minneapolis, MN

Default is AG89

**element** [list of int] Elements to find abundance for. If not specified, return all.

**datacache** [dict] See get_data

**datacache** [settings] See get_data

**Returns**

**dict** abundances in dictionary, i.e :

```python
{1: 1.0,
  2: 0.097723722095581111,
  3: 1.4454397707459272e-11,
  4: 1.4125375446227541e-11,
  5: 3.9810717055349735e-10,
  6: 0.00036307805477010178,...
```

pyatomdb.atomdb.get_bt_approx(*om, Tin, Tout, uplev, levlol, ladat*)

pyatomdb.atomdb.get_burgess_tully_extrap(*bttype, lolev, uplev, Aval, Tarr, om, TTarg*)

pyatomdb.atomdb.get_burgess_tully_transition_type(*lolev, uplev, Aval*)

pyatomdb.atomdb.get_data(*Z, zl, ftype, datacache=False, settings=False, indexzero=False, offline=False*)

Read AtomDB data of type ftype for ion rmJ of element Z.

If settings are set, the filemap can be overwritten (see below), otherwise $ATOMDB/filemap will be used to locate the file. If indexzero is set, all levels will have 1 subtracted from them (AtomDB indexes lines from 1, but python and C index arrays from 0, so this can be useful)

**Parameters**

**Z** [int] Element nuclear charge

**rmJ** [int] Ion charge +1 (e.g. 5 for C^{4+}, a.k.a. C V)

**ftype** [string]

*type of data to read. Currently available*

• ‘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
• ‘ALL’ - reads all of the above. Does not return anything. Used for bulk downloading.

Or, for non-ion-specific data (abundances and bremsstrahlung coefficients) * ‘ABUND’ - abundance tables * ‘HBREMS’ - Hummer bremsstrahlung coefficients * ‘RBREMS’ - relativistic bremsstrahlung coefficients * ‘IONBAL’ - ionization balance tables * ‘EIGEN’ - eigenvalue files

filemap [string] The filemap to use, if you do not want to use the default one.

settings [dict] This will let you override some standard inputs for get_data:

  • settings['filemap']: the filemap to use if you do not want to use the default $ATOMDB/filemap
  • settings['atomdbroot']: If you have files in non-standard locations you can replace $ATOMDB with this value

datacache [dict] This variable will hold the results of the read in a dictionary. It will also be checked to see if the requested data has already been cached here before re-reading from the disk. If you have not yet read in any data but want to start caching, provide it as an empty dictionary i.e. mydatacache={}

2 parts of the data are stored here:

  • Settings['data'] will store a copy of the data you read in. This means that if your code ends up calling for the same file multiple times, rather than re-reading from the disk, it will just point to this data already in memory. To clear the read files, just reset the data dictionary (e.g. settings['data'] ={} )
  • settings['datasums'] stores the datasum when read in. Can be used later to check files are the same.

Both data and datasums store the data in identical trees, e.g.: settings['data'][Z][z1][ftype] will have the data.

indexzero: bool If True, subtract 1 from all level indexes as python indexes from 0, while AtomDB indexes from 1.

offline: bool If True, do not search online to download data files - just return as if data does not exist

Returns

HDUlist the opened pyfits hdulist if successful. False if file doesn’t exist

pyatomdb.atomdb.get_emissivity (linefile, elem, ion, upper, lower, kT=-1, hdu=-1, kTu-nits='keV')

pyatomdb.atomdb.get_filemap_file (ftype, Z, zl, fmapfile='$ATOMDB/filemap', atomdb-root='$ATOMDB', quiet=False, misc=False)

Find the correct file from the database for atomic data of type ftype for ion with nuclear charge Z and ion-charge+1 = zl

Parameters

ftype [str]

  • ‘ir’ = ionization & recombination data
  • ‘lv’ = energy levels
  • ‘la’ = wavelength and transition probabilities (lambda & a-values)
• ‘ec’ = electron collision rates
• ‘pc’ = proton collision rates
• ‘dr’ = dielectronic recombination satellite line information
• ‘ai’ = autoionization rate data
• ‘pi’ = XSTAR photoionization data
• ‘em’ = emission feature data (currently unused)

\[ Z \] [int] Element atomic number (=6 for C+4)
\[ z1 \] [int] Ion charge +1 (=5 for C+4)

\texttt{fmapfile} [str] Specific filemap to use. Otherwise defaults to \texttt{atomdbroot+/filemap’}

\texttt{atomdbroot} [str] Location of ATOMDB database. Defaults to ATOMDB environment variable. all \$ATOMDB in the filemap will be expanded to this value

\texttt{quiet} [bool] If true, suppress warnings about files not being present for certain ions

\texttt{misc} [bool] If requesting “misc” data, i.e. the Bremsstrahlung inputs, use this. This is for non ion-specific data, therefore \[ Z, z1 \] are ignored. Types are: 10 or ‘abund’: elemental abundances 11 or ‘hbrems’: Hummer bremsstrahlung gaunt factor coefficients 13 or ‘rbrems’: Relativistic bremsstrahlung gaunt factor coefficients

\textbf{Returns}

\texttt{str} The filename for the relevant file, with all \$ATOMDB expanded. If no file exists, returns zero length string.

\texttt{pyatomdb.atomdb.get_ion_lines} (\texttt{linefile}, \[ Z \], \[ z1 \], \texttt{fullinfo=False})
\texttt{pyatomdb.atomdb.get_ionbal} (\texttt{ionbalfile}, \texttt{element}, \texttt{ion=-1})
\texttt{pyatomdb.atomdb.get_ionfrac} (\texttt{ionbalfile}, \[ Z \], \texttt{te}, \[ z1 \]=-1)

Reads the ionization fraction of a given ion at a given Te from an ionbalfile Assumes ionization equilibrium

\textbf{Parameters}

\texttt{ionbalfile} [str] location of ionization balance file
\[ Z \] [int] atomic number of element (e.g. 6 for carbon)
\texttt{te} [float] electron temperature (in K)
\[ z1 \] [int] if provided, \( z+1 \) of ion (e.g. 5 for O V). If omitted, returns ionization fraction for all ions of element

\textbf{Returns}

\textit{ionization fraction of ion or, if not specified, of all ions at Te}

\texttt{pyatomdb.atomdb.get_ionpot} (\[ Z \], \[ z1 \], \texttt{settings=False}, \texttt{datacache=False})

Get the ionization potential of an ion in eV

\textbf{Parameters}

\[ Z \] [int] The atomic number of the element
\[ z1 \] [int] The ion charge + 1 of the ion
\texttt{settings} [dict] See description in get\_data
\texttt{datacache} [dict] Used for caching the data. See description in get\_data

\textbf{Returns}
float  The ionization potential of the ion in eV.

```python
pyatomdb.atomdb.get_ionrec_rate(Te_in, irdat_in=False, Te_unit='K', lvdatp1_in=False, ionpot=False, separate=False, Z=-1, z1=-1, settings=False, datacache=False, extrap=True)
```

Get the ionization and recombination rates at temperature(s) Te from ionization and recombination rate data file irdat.

**Parameters**

- **Te_in** [float or arr(float)] electron temperature in K (default), eV, or keV
- **irdat_in** [HDUList] ionization and recombination rate data
- **lvdat_in** [HDUList] level data for ion with lower charge (i.e. ionizing ion or recombined ion)
- **Te_unit** [‘K’, ‘keV’, ‘eV’] temperature unit
- **lvdatp1_in** [HDUList] level data for the ion with higher charge (i.e ionized or recombining ion)
- **ionpot** [float] ionization potential of ion (eV).
- **separate** [bool] if set, return DR, RR, EA and CI rates seperately. (DR = dielectronic recombination, RR = radiative recombination, EA = excitation autoionization, CI = collisional ionization) Note that EA & CI are not stored separately in all cases, so may return zeros for EA as the data is incorporated into CI rates.
- **Z** [int] Element charge to get rates for (ignores “irdat_in”)
- **z1** [int] Ion charge +1 to get rates for (ignores “irdat_in”) e.g. Z=6,z1=4 for C IV (C 3+)
- **settings** [dict] See description in read_data
- **datacache** [dict] See description in read_data
- **extrap** [bool] Extrapolate rates to Te ranges which are off the provided scale

**Returns**

- **float, float**: (ionization rate coeff., recombination rate coeff.) in cm^3 s^-1 unless separate is set, in which case:
- **float, float, float, float**: (CI, EA, RR, DR rate coeff)s in cm^3 s^-1 Note that these assume low density & to get the real rates you need to multiply by N_e N_ion.

```python
pyatomdb.atomdb.get_level_details(level, Z=-1, z1=-1, filename="", filemap="", atomdbroot="")
```

Function returns the details in the level file for the specified level. LV file can be specified by filename, or by filemap, Z, z1

```python
pyatomdb.atomdb.get_line_emissivity(Z, z1, upind, loind, linefile='$ATOMDB/apec_line.fits', ion_drv=False, elem_drv=False, use_nei=False, use_nei_raw=False)
```

Get the emissivity of a line as fn of temperature from APEC line file

**Parameters**

- **Z** [int] Atomic number of element of line
- **z1** [int] Ion charge +1 of ion
- **upind** [int] Upper level of transition
- **loind** [int] Lower level of transition
- **linefile** [str] line emissivity file. defaults to $ATOMDB/apec_line.fits
ion_drv [int] if set, return only the contribution from driving ion ion_drv. This is useful for non-equilibrium plasma calculations, and requires an nei_line file to be specified in linefile

elem_drv [int] same as ion_drv, but specified driving element. Currently this setting is pointless, as all transitions have the same driving element as element.

use_nei [bool] This can be useful when trying to get line emissivities which fall below the 1e-20 cut off. Applying this flag, the NEI file will be used by default and an ionization balance applied. This should give the same results as normal for strong emissivities, but go to a lower emissivity before being set to zero. Use with caution...

use_nei_raw [bool] Return the emissivities by driving ion. This changes the epsilon returned to no longer be a single array, but a dict where e.g. epsilon[5] is an array of the spectrum with driving ion 5 (e.g. C V or similar)

Returns

dict dictionary with the following data in it:

[‘kT’] [array(float)] the electron temperatures, in keV

[‘dens’] [array(float)] the electron densities, in cm^-3

[‘time’] [array(float)] the time (for old-style NEI files only, typically all zeros in current files)

[‘epsilon’] [array(float)] the emissivity in ph cm^3 s^-1

pyatomdb.atomdb.get_lorentz_levpop (Z, z1, up, lo, Te, Ne, version, linelabel)
calculate the level population for a particular ion

pyatomdb.atomdb.get_maxwell_rate (Te, colldata=False, index=-1, lvdata=False, Te_unit='K', lvdatap1=False, ionpot=False, force_extrap=False, silent=True, finallev=False, initlev=False, Z=-1, z1=-1, dtype=False, exconly=False, datacache=False, settings=False, ladat=False)

Get the maxwellian rate for a transition from a file, typically for ionization, recombination or excitation.

Parameters

Te [float] electron temperature(s), in K by default

colldata [HDUList] If provided, the HDUList for the collisional data

index [int] The line in the HDUList to do the calculation for. Indexed from 0.

lvdata [HDUList] the hdulist for the energy level file (as returned by pyfits.open('file'))


lvdatap1 [HDUList] The level data for the recombining or ionized data.

ionpot [float] The ionization potential in eV (required for some calculations, if not provided, it will be looked up)

force_extrap [bool] Force extrapolation to occur for rates outside the nominal range of the input data

silent [bool] Turn off notifications

finallev [int] Instead of specifying the index, can use upperlev, lowerlev instead.

initlev [int] Instead of specifying the index, can use upperlev, lowerlev instead

Z [int] Instead of providing colldata, can provide Z & z1. Z is the atomic number of the element.

z1 [int] Instead of providing colldata, can provide Z & z1. z1 is the ion charge +1 for the initial ion
**dtype** [str] data type. One of:
- ‘EC’ : electron impact excitation
- ‘PC’ : proton impact excitation
- ‘CI’ : collisional ionization
- ‘EA’ : excitation-autoionization
- ‘XI’ : excluded ionization
- ‘XR’ : excluded recombination
- ‘RR’ : radiative recombination
- ‘DR’ : dielectronic recombination

**exconly** [bool] For collisional excitation, return only the excitation rate, not the de-excitation rate.

**settings** [dict] See description in read_data

**datacache** [dict] See description in read_data

**Returns**

- **float or array(float)** Maxwellian rate coefficient, in units of cm^3 s^-1 For collisional excitation (proton or electron) returns excitation, deexcitation rates

**Examples**

```python
Te = numpy.logspace(4,9,20)

(1) Get excitation rates for row 12 of an Fe XVII file colldata = pyatomdb.atomdb.get_data(26,17,'EC') exc, dex = get_maxwell_rate(Te, colldata=colldata, index=12)

(2) Get excitation rates for row 12 of an Fe XVII file exc, dex = get_maxwell_rate(Te, Z=26,z1=17, index=12)

(3) Get excitation rates for transitions from level 1 to 15 of FE XVII exc, dex = get_maxwell_rate(Te, Z=26, z1=17, dtype='EC', finallev=15, initlev=1)

pyatomdb.atomdb.get_oscillator_strength(Z, z1, upperlev, lowerlev, datacache=False)
Get the oscillator strength \( f_{ij} \) of a transition

**Parameters**

- **Z** [int] The atomic number of the element
- **z1** [int] The ion charge + 1 of the ion
- **upperlev** [int] The upper level, indexed from 1
- **lowerlev** [int] The lower level, indexed from 1
- **datacache** [dict] Used for caching the data. See description in get_data

**Returns**

- **float** The oscillator strength. Returns 0 if transition not found. If transition is not found but the inverse transition is present the oscillator strength is calculated for this instead.

pyatomdb.atomdb.interp_rate(\( Te, npar, Te_grid, ionrec_par \))
pyatomdb.atomdb.interpol_huntd(x, y, z)
pyatomdb.atomdb.interpolate_ionrec_rate(cidat, \( Te, force_extrap=False \))
pyatomdb.atomdb.lorentz_cie(version)
Calculate the CSD of equilibrium plasmas at 1e6, 6e6K and 4keV.

Parameters

version [string] The version string

Returns

None

pyatomdb.atomdb.lorentz_levpop(version)
Calculate the level populating processes for each line in the stronglines Files. This will require a significant rerun of APEC. Hmmmmm

Processes to be tracked: electron excitation, electron de-excitation, proton excitation and deexcitation, cascade into the level, radiative out, recombination (incl. cascade) in, DR (incl cascade) in, and inner-shell ionization in (why only inner shell?)

pyatomdb.atomdb.lorentz_neicont(version)
Full spectrum of a gas ionizing from 1e4K to 2.321e7K (=2keV) at a fluence ($n_e$ * t, or $ au$) of $10^{10}$ cm$^{-3}$ s

Parameters

version [string] The version string

Returns

None

pyatomdb.atomdb.lorentz_neicsd(version)
Charge state distribution of a gas ionizing from 1e4K to 2.321e7K (=2keV) at a fluence ($n_e$ * t, or $ au$) of $10^{10}$ cm$^{-3}$ s

Parameters

version [string] The version string

Returns

None

pyatomdb.atomdb.lorentz_neilines(version)
100 strongest lines with wavelength < 1000Å for a 1cm$^3$ plasma (1) starting at 1e4K, going to 2.321e7K at a fluence ($n_e$ * t, or $ au$) of $10^{10}$ cm$^{-3}$ s (2) starting at 3.5keV, going to 1.5keV at a fluence ($n_e$ * t, or $ au$) of $10^{10}$ cm$^{-3}$ s

Parameters

version [string] The version string

Returns

None

pyatomdb.atomdb.lorentz_power(version)
Calculate the power emitted from 13.6eV to 13.6keV in a 1m$^3$ slab of plasma with n_e=1e6m$^{-3}$.
pyatomdb.atomdb.lorentz_stronglines(version)
Calculate the 100 strongest lines below 1000A from a 1m3 slab of plasma with n_e = 1e6m-3, at 3 different
temperatures: 10^6K, 6e6K, 4.642e7K

Parameters

version [string] The version string

Returns

None

pyatomdb.atomdb.make_level_descriptor(lv)

pyatomdb.atomdb.make_lorentz(version=False, do_all=True, cie=False, power=False, strong-
lines=False, neicsd=False, neiines=False, neoncont=False, lev-
op=False)
This makes all the Lorentz data comparison files from the Astrophysical Collisional Plasma Test Suite, version
0.4.0

Parameters

version [string (optional)] e.g. “3.0.7” to run the suite for v3.0.7. Otherwise uses latest version.

Returns

none

pyatomdb.atomdb.prep_spline_atomdb(x, y, n)

pyatomdb.atomdb.read_filemap(filemap='$ATOMDB/filemap', atomdbroot='$ATOMDB')
Reads the AtomDB filemap file in to memory. By default, tries to read $ATOMDB/filemap, replacing all
instances of $ATOMDB in the filemap file with the value of the environment variable $ATOMDB

Parameters

filemap: str the filemap file to read
atomdbroot: str location of files, if not $ATOMDB.

pyatomdb.atomdb.rr_badnell(Te, c)

Convert data from Badnell constants into a RR Rate

Parameters

Te [float or array(float)] Electron temperature[s] in K

c [array] Constants from DR rates. Stored as alternating pairs in AtomDB, so c1,e1,c2,e2,c3,e3
etc in the IONREC_PAR column

Returns

float RR rate in cm^3 s-1

References

See http://amdpp.phys.strath.ac.uk/tamoc/DATA/RR/

pyatomdb.atomdb.rr_shull(Te, c)

pyatomdb.atomdb.rr_verner(Te, c)

pyatomdb.atomdb.rrc_ph_value(E, Z, z1, rrc_ph_factor, IonE, kT, levdat, xstdata=False, xstarfi-
nallev=False)
Returns RRC in photons cm3 s-1 keV-1
Parameters

E:
Z: int  Atomic number of element (i.e. 8 for Oxygen)
z1: int  Ion charge +1 e.g. 5 for C+4, a.k.a. C V
rrc_ph_factor: float  Conversion factor for RRC.
IonE: float  Ionization potential of ion
kT: float  Temperature (keV)
levdat: lvdat line  Line from the lvdat file
xstardata [dict, str or HDUList] if the data is XSTAR data (pi_type=3), supply the xstardata. This can be a dictionary with 2 arrays, one “Energy”, one “sigma”, the file name, or the entire PI file (already loaded):

```python
# load level data
lvdata = atomdb.get_data(26, 24, 'LV', settings)

# load XSTAR PI data if it exists
pidata = atomdb.get_data(26, 24, 'PI', settings)

# get pi xsection at energy E for the ground state to ground state
sigma_photoion(E,
    lvdata[1].data['pi_type'][0],
    lvdata[1].data['pi_param'][0],
    xstardata=pidata,
    xstarfinallev=1)
```

xstarfinallev: the level to ionize in to. Defaults to 1.

Returns

float  The RRC in photons cm³ s⁻¹ keV⁻¹ at energy(ies) E.

`pyatomdb.atomdb.sigma_hydrogenic(Z, N, L, Ein)`
Calculate the PI cross sections of type hydrogenic.

Parameters

N [int] n shell
L [int] l quantum number
Z [int] nuclear charge
Ein [array(float)] energy grid for PI cross sections (in keV)

Returns

array(float)  Photoionization cross section (in cm²)

`pyatomdb.atomdb.sigma_photoion(E, Z, z1, pi_type, pi_coeffts, xstardata=False, xstarfinallev=1)`
Returns the photoionization cross section at E, given an input of sig_coeffts.

Parameters

E: float or array of floats  Energy/ies to find PI cross section at (keV)
Z: int  Atomic number of element (i.e. 8 for Oxygen)
pi_type [int] the “PI_TYPE” from the energy level file for this level, can be:
pi_coeffts [array(float)] the “PI_PARAM” array for this level from the LV file

xstardata [dict, str or HDUList] if the data is XSTAR data (pi_type=3), supply the xstardata. This can be a dictionary with 2 arrays, one “Energy”, one “sigma”, the file name, or the entire PI file (already loaded):

```python
# load level data
lvdata = atomdb.get_data(26, 24, 'LV', settings)

# load XSTAR PI data if it exists
pidata = atomdb.get_data(26, 24, 'PI', settings)

# get pi xsection at energy E for the ground state to ground state
sigma_photoion(E,
    lvdata[1].data['pi_type'][0],
    lvdata[1].data['pi_param'][0],
    xstardata=pidata,
    xstarfinallev=1)
```

### xstarfinallev: the level to ionize in to. Defaults to 1.

Returns

array(float) pi cross section in cm^2 at energy E.

pyatomdb.atomdb.sort_pi_data(pidat, lev_init, lev_final)

Given the pidat (returned by opening the PI data file, i.e. pyfits.open(‘XX_YY_PI.fits’), and the initial and final levels, return the PI cross section data.

Parameters

pidat [hdulist] The photoionization data for the ion
lev_init [int] The initial level
lev_final [int] The final level

Returns

dict: which contains the following information:
    pi['ion_init'] - the initial ion charge +1
    pi['lev_init'] - the initial level pi['ion_final'] - the final ion charge+1 (should be ion_init+1)
    pi['lev_final'] - the final level pi['pi_type'] - the type. (best to ignore)
    pi['g_ratio'] - the ratio of the statistical weight of the intitial and final levels
    pi['energy'] - the array of energies (keV)
    pi['pi_param'] - the array of pi cross sections in Mbarn.

pyatomdb.atomdb.write_filemap(d, filemap, atomdbroot="")

Write filemap to file

Parameters

d [dict] Dictionary with filemap data in it. Structure defined as return value from read_filemap.
filemap [str] Name of filemap file to read. If zero length, use “$ATOMDB/filemap”
atomdbroot [str] Replace any $ATOMDB in the file names with this. If not provided, use “ATOMDB” environment variable instead

Returns

none
2.4 PyAtomDB Const module

A series of physical constants and constants relevant to running the APEC code.
This contains a list of constants, both physical and apec code related.
Version 0.1 - initial release Adam Foster July 17th 2015

2.5 PyAtomDB Spectrum module

This module contains codes for creating spectra from the AtomDB emissivity files
This module contains methods for creating spectra from the AtomDB files. Some are more primitive than others...

class pyatomdb.spectrum.CIESession(linefile='$ATOMDB/apec_line.fits', cocofile='$ATOMDB/apec_coco.fits', elements=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30], abundset='AG89')

Bases: object

Load and generate a collisional ionization equilibrium spectrum

Parameters

- linefile [string or HDUList, optional] The line emissivity data file (either name or already open)
- cocofile [string or HDUList, optional] The continuum emissivity data file (either name or already open)
- elements [arraylike(int), optional] The atomic number of elements to include (default all)
- abundset [string] The abundance set to use. Default AG89.

Examples

Create a session instance:

```python
>>> s=CIESession()
```

Set up the responses, in this case a dummy response from 0.1 to 10 keV, with area 1cm^2 in each bin

```python
>>> ebins = numpy.linspace(0.1,10,1000)
>>> s.set_response(ebins, raw=True)
```

(Alternatively, for a real response file, s.set_response(rmffile, arf=arffile)

Turn on thermal broadening

```python
>>> s.set_broadening(True)
Will thermally broaden lines with emissivity > 1.000000e-18 ph cm3 s-1
```

Return spectrum at 1.0keV

```python
>>> spec = s.return_spectrum(1.0)
```

spec is in photons cm^5 s^-1 bin^-1; ebins are the bin edges (so spec is 1 element shorter than ebins)

Attributes
datacache [dict] Any Atomdb FITS files which have to be opened are stored here
spectra [CIESpectra] Object storing the actual spectral data
elements [list(int)] Nuclear charge of elements to include.
default_abundset [string] The abundance set used for the original emissivity file calculation
abundset [string] The abundance set to be used for the returned spectrum
abundsetvector [array_like(float)] The relative abundance between default_abundset and
abundset for each element
response_set [bool] Have we loaded a response (or set a dummy response)
dolines [bool] Calculate line emission
docont [bool] Calculate continuum emission
dopseudo [bool] Calculate pseudocontinuum emission
broaden_limit [float] Apply broadening to lines with epsilon > this value (ph cm3 s-1)
thermal_broadening [bool] Apply thermal broadening to lines (default = False)
velocity_broadening [float] Apply velocity broadening with this velocity (km/s). If <=0, do
not apply.

apply_response (self, spectrum)
Apply a response to a spectrum

Parameters

spectrum [array(float)] The spectrum, in counts/bin/second, to have the response applied
to. Must be binned on the same grid as the rmf.

Returns

array(float) spectrum folded through the response

return_line_emissivity (self, Te, Z, z1, up, lo, specunit='A', teunit='keV', apply_aeff=False,
apply_abund=True, log_interp=True)
Get line emissivity as function of Te.

Parameters

Te [float or array(float)] Temperature(s) in keV or K
Z [int] nuclear charge of element
z1 [int] ion charge +1 of ion
up [int] upper level for transition
lo [int] lower level for transition
specunit [['Angstrom', 'keV']] Units for wavelength or energy (a returned value)
teunit [['keV', 'K']] Units of Te (kev or K, default keV)
apply_aeff [bool] If true, apply the effective area to the line emissivity in the linelist to
modify their intensities.
apply_abund [bool] If true, apply the abundance set in the session to the result.
log_interp [bool] Interpolate between temperature on a log-log scale (default). Otherwise
linear
Returns

ret [dict] Dictionary containing: Te, tau, teunit: as input wavelength : line wavelength (A) energy : line energy (keV) epsilon : emissivity in ph cm^3 s-1 (or ph cm^5 s^-1 if apply_aeff=True)

first index is temperature, second is tau.

return_linelist (self, Te, specrange, specunit='A', teunit='keV', apply_aeff=False, nearest=False, apply_binwidth=False)

Get the list of line emissivities vs wavelengths

Parameters

Te [float] Temperature in keV or K
specrange [[float, float]] Minimum and maximum values for interval in which to search
specunit [{'Angstrom','keV'}] Units for specrange
teunit [{'keV', 'K'}] Units of te (kev or K, default keV)
apply_aeff [bool] If true, apply the effective area to the lines in the linelist to modify their intensities.

Returns

linelist [array(dtype)] The list of lines with lambda (A), energy (keV), epsilon (ph cm3 s-1), epsilon_aeff (ph cm5 s-1) ion (string) and upper & lower levels.

return_spectrum (self, te, teunit='keV', nearest=False, get_nearest_t=False, log_interp=True)

Get the spectrum at an exact temperature. Interpolates between 2 neighbouring spectra

Finds HDU with kT closest to desired kT in given line or coco file.

Opens the line or coco file, and looks for the header unit with temperature closest to te. Use result as index input to make_spectrum

Parameters

te [float] Temperature in keV or K
teunit [{'keV', 'K'}] Units of te (kev or K, default keV)
raw [bool] If set, return the spectrum without response applied. Default False.
nearest [bool] If set, return the spectrum from the nearest tabulated temperature in the file, without interpolation
get_nearest_t [bool] If set, and nearest set, return the nearest tabulated temperature as well as the spectrum.

Returns

spectrum [array(float)] The spectrum in photons cm^5 s^-1 bin^-1, with the response, or photons cm^3 s^-1 bin^-1 if raw is set.

nearest_T [float, optional] If nearest is set, return the actual temperature this corresponds to. Units are same as teunit

set_abund (self, elements, abund)

Set the elemental abundance, relative to the abundset. Defaults to 1.0 for everything

Parameters

elements [int or array_like(int)] The elements to change the abundance of
**abund** [float or array_like(float)] The new abundances. If only 1 value, set all *elements* to this abundance Otherwise, should be of same length as elements.

**Returns**

None

**Examples**

Set the abundance of iron to 0.5

```python
>>> myspec.set_abund(26, 0.5)
```

Set the abundance of iron and nickel to 0.1 and 0.2 respectively

```python
>>> myspec.set_abund([26, 28], [0.1, 0.2])
```

Set the abundance of oxygen, neon, magnesium and iron to 0.1

```python
>>> myspec.set_abund([8, 10, 12, 26], 0.1)
```

**set_abundset**(self, abundstring)

Set the abundance set.

**Parameters**

abundstring [string] The abundance string (e.g. “AG89”, “uniform”. Case insensitive. See atomdb.get_abundance for list of possible abundances

**Returns**

None updates self.abundset and self.abundsetvector.

**set_apec_files**(self, linefile='$ATOMDB/apec_line.fits', cocofile='$ATOMDB/apec_coco.fits')

Set the apec line and coco files, and load up their data

**Parameters**

linefile [str or HDUList] The filename of the line emissivity data, or the opened file.

cocofile [str or HDUList] The filename of the continuum emissivity data, or the opened file.

**Returns**

None

**Notes**

Updates self.linefile, self.linedata, self.cocofile and self.cocodata

**set_broadening**(self, thermal_broadening=True, broaden_limit=False, velocity_broadening=0.0, velocity_broadening_units='km/s')

Turn on or off thermal broadening, and the emissivity limit for host lines

**Parameters**

thermal_broadening [bool] If true, turn on broadening. If False, turn it off.

broaden_limit [float] The emissivity limit for lines to be broadened. If False, this value will not be updated.

velocity_broadening [float] velocity broadening to apply. If <=0, not applied
velocity_broadening_units [string] Units of velocity_broadening. ‘km/s’ is default and only value so far.

Notes

Updates attributes thermal_broadening, broaden_limit, velocity_broadening, velocity_broadening_units

set_response (self, rmf, arf=False, raw=False)
Set the response. rmf, arf can either be the filenames or the opened files (latter is faster if called repeatedly)
Amends the following items:

self.rmf [string] The rmf file name
self.rmf [string] The response matrix
self.arffile [string] The arf file name
self.arf [string] The arf data
self.specbins [array(float)] The spectral bins on which to calculate the spectrum (keV or A)
self.specbin_units [string ['A','keV']] Units of specbins
self.ebins [array(float)] The spectral bins on which to calculate the spectrum (keV).
self.ebins_out [array(float)] The spectral bins on which to return the spectrum (keV). Can be different from specbins depending on the spectrum
self.response_set [bool] A response has been loaded
self.specbins_set [bool] The spectral bins are set
self.ebins_checksum [string] The md5checksum of the specbins

Parameters

   rmf: string or HDUlist or array The response matrix file or energy bins (see raw k/w)
   arf: string or HDUList The ancillary response file
   raw [bool] If true, the rmf variable contains the energy bin edges (keV) for all the bins, and each bin has a perfect response. This is effectively a dummy response

Returns

   none

class PyAtomDB.spectrum.CIESpectrum (linedata, cocodata)
Bases: object

A class holding the emissivity data for CIE emission, and returning spectra

Parameters

   linefile [string or HDUList, optional] The line emissivity data file (either name or already open)
   cocofile [string or HDUList, optional] The continuum emissivity data file (either name or already open)
   elements [arraylike(int), optional] The atomic number of elements to include (default all)
   abundset [string] The abundance set to use. Default AG89.

Attributes
**session**  [CIESession] The parent CIESession

**SessionType**  [string] “CIE”

**spectra**  [dict of ElementSpectra] a dictionary containing the emissivity data for each HDU, subdivided by element (spectra[12][18] is an ElementSpectrum object containing the argon data for the 12th HDU)

**kTlist**  [array] The temperatures for each emissivity HDU, in keV

**logkTlist**  [array] log of kTlist

**get_nearest_Tindex**  (*self*,  *Te*,  *teunit*='keV',  *nearest*=False,  *log_interp*=True)
Return the nearest temperature index in the emissivity file, or, alternatively, the array of fractions to sum

**Parameters**

- **Te**  [float] Temperature (keV by default)
- **teunit**  [string] Units of kT
- **nearest**  [bool] If true, return only nearest. Otherwise, return nearest 2 and fractions

**Returns**

- **ikT**  [list[int]] Index of temperature in HDU file (from 0, not 2)
- **f**  [list[float]] fractional weight to apply to each ikT. Should sum to 1.

**return_line_emissivity**  (*self*,  *Te*,  *Z*,  *z1*,  *up*,  *lo*,  *specunit*='A',  *teunit*='keV',  *abundance*=1.0,  *log_interp*=True)
Return the emissivity of a line at kT, tau. Assumes ionization from neutral for now

**Parameters**

- **Te**  [float] Temperature in keV or K
- **Z**  [int] nuclear charge of element
- **z1**  [int] ion charge +1 of ion
- **up**  [int] upper level for transition
- **lo**  [int] lower level for transition
- **specunit**  [{‘Angstrom’,’keV’}] Units for wavelength or energy (a returned value)
- **teunit**  [{‘keV’, ‘K’}] Units of Telist (kev or K, default keV)
- **abundance**  [float] Abundance to multiply the emissivity by

**Returns**

- **Emissivity**  [float] Emissivity in photons cm^3 s^-1
- **spec**  [float] Wavelength or Energy of line, depending on specunit

Return the linelist of the element

**Parameters**

- **Te**  [float] Electron temperature (default, keV)
- **teunit**  [string] Units of kT (keV by default, K also allowed)
- **nearest**  [bool] If True, return spectrum for the nearest temperature index. If False, use the weighted average of the (log of) the 2 nearest indexes. default is False.
specrange: [[float, float]] Minimum and maximum values for interval in which to search

specunit: [{'Angstrom', 'keV'}] Units for specrange (default A)

return_spectrum (self, Te, teunit='keV', nearest=False, elements=False, abundances=False, loginterp=True, broaden_object=False)

Return the spectrum of the element on the energy bins in self.session.specbins

Parameters

Te [float] Electron temperature (default, keV)

teunit [string] Units of kT (keV by default, K also allowed)

nearest [bool] If True, return spectrum for the nearest temperature index. If False, use the weighted average of the (log of) the 2 nearest indexes. default is False.

Returns

spec [array(float)] The element’s emissivity spectrum, in photons cm^-3 s^-1 bin^-1

class pyatomdb.spectrum.ContinuumData (cocoentry, docont=True, dopseudo=True)

Bases: object

A class holding the continuum data for an element in one HDU

Parameters

cocoentry [array(cocodatatype)] A single row from the continuum data in an AtomDB file.

parentElementSpectrum [ElementSpectrum] Parent ElementSpectrum object

Attributes

ECont [array(float)] The continuum energies (keV)

EPseudo [array(float)] The pseudocontinuum energies (keV)

Cont [array(float)] The continuum emissivities (ph cm^-3 s^-1 keV^-1)

Pseudo [array(float)] The pseudocontinuum energies (ph cm^-3 s^-1 keV^-1)

parentElementSpectrum [ElementSpectrum] Parent ElementSpectrum object

spectrum_calculated [bool] True if spectrum has already been calculated, otherwise false

ebins_checksum [string] md5sum of the ebins the spectrum was last calculated on. Used to identify if new calculations are required or can just return the previous value.

docont [bool] Calculate the continuum emission

dopseudo [bool] Calculate the pseudocontinuum emission

return_spec (self, ededges, ebins_checksum=False)

class pyatomdb.spectrum.ElementSpectrum (linedata, cocodata, Z, z1_drv=0)

Bases: object

A class holding the emissivity data for an element in one HDU

Parameters

linedata [array(linedatatype)] array of line wavelengths and emissivities, from AtomDB files.

Should already be filtered to only be from one element.

cocodata [array(cocodatatype)] array of continuum wavelengths and emissivities, from AtomDB files Should already be filtered to only be from one element.

Z [int] The atomic number of the element
**z1_drv** [int] The charge + 1 for the ion. 0 = whole element.

**parent** [CIESpectrum] Parent CIESpectrum object

**Attributes**

- **lines** [LineData] A LineData object containing all the line information
- **continuum** [ContinuumData] A ContinuumData object containing all the continuum information
- **parent** [CIESpectrum] Parent CIESpectrum object
- **session** [CIESession] Parent Session of parent CIESpectrum object

**return_linelist** *(self, specrange, specunit='A', teunit='keV')*

Return the list of lines in specrange

**Parameters**

- **specrange** [array] spectral range to look for lines in
- **specunits** [string] units of spectral range (A or keV)
- **teunit** [string] units of Te (keV, eV, K)

**Returns**

- **linelist** [array] list of lines and epsilons

**return_linematch** *(self, Z, z1, up, lo, z1_drv=0)*

Return the line(s) which match the transition

**Parameters**

- **Z** [int] nuclear charge
- **z1** [int] ion charge + 1
- **up** [int] upper level of transition
- **lo** [int] lower level of transition
- **z1_drv** [int] if provided, also filter on driving ion charge (NEI only)

**Returns**

- **linelist** [array] list of lines and epsilons

**return_spectrum** *(self, edgess, Te, ebins_checksum=False, thermal_broadening=False, broaden_limit=False, velocity_broadening=0.0, teunit='keV', broaden_object=False)*

Calculate the spectrum

**Parameters**

- **edgess** [array] bin edges (keV)
- **Te** [float] temperature (keV by default)
- **ebins_checksum** [string] the md5 checksum of edgess
- **thermal_broadening** [bool] true to apply thermal broadening
- **broaden_limit** [float] only broaden lines stronger than this.
- **velocity_broadening** [float] velocity broadening to apply, km/s. Set <=0 for none (default)
**teunit** [string] Temperature unit (K, keV, eV)

**Returns**

**spectrum** [array(float)] The spectrum in ph cm^3 s^-1 bin^-1

---

**class** pyatomdb.spectrum.Gaussian_CDF

**Bases:** object

For fast interpolation, pre-calculate the CDF and interpolate it when broadening lines

**Parameters**

None

**Examples**

Create a CDF instance:

```python
>>> s=Gaussian_CDF()
```

Broaden a line on ebins grid, with centroid and width.

```python
>>> cdf = a.broaden(centroid, width, ebins)
```

Convert to flux in each bin

```python
>>> flux= cdf[1:]-cdf[:-1]
```

**broaden**(self, centroid, width, ebins)

Broaden a line, return CDF

**Parameters**

**centroid** [float] The line energy (keV)

**width** [float] The sigma of the normal distribution, in keV

**ebins** [array(float)] Energy grid to return CDF on

**Returns**

**CDF** [array(float)] cumulative flux distribution of linen at each bin edge.

---

**class** pyatomdb.spectrum.LineData(linelist)

**Bases:** object

A class holding the line data for an element in one HDU

**Parameters**

**linelist** [array(linedatatype)] array of line wavelengths and emissivities, from AtomDB files.

**Attributes**

**lines** [array(linedatatype)] List of lines, wavelength and emissivities

**lineenergies** [array(float)] list of line energies

**spectrum_calculated** [bool] True if spectrum has already been calculated, otherwise false

**T** [float] Temperature (K) last spectrum was calculated at (for broadening)

**v** [float] Velocity (km/s) last spectrum was calculated at (for broadening)
ebins_checksum  [string] md5sum of the ebins the spectrum was last calculated on. Used to identify if new calculations are required or can just return the previous value.

return_spec  (self, eedges, T, ebins_checksum=False, thermal_broadening=False, velocity_broadening=0.0, broaden_limit=1e-18, broaden_object=False)
return the line emission spectrum at temperature T

Parameters
  eedges  [array] energy bin edges, keV
  T  [float] temperature in Kelvin
  ebins_checksum  [string] the md5 checksum of eedges
  thermal_broadening  [bool] true to apply thermal broadening
  velocity_broadening  [float] velocity broadening to apply, km/s. Set <=0 for none (default)
  broaden_limit  [float] only broaden lines stronger than this.

Returns
  spectrum  [array(float)] Emissivity on eedges spectral bins of the lines, in ph cm^3 s^-1 bin^-1

class pyatomdb.spectrum.NEISession (linefile='$ATOMDB/apec_nei_line.fits', coconutfile='$ATOMDB/apec_nei_comp.fits', elements=False, abundset='AG89')
Bases: pyatomdb.spectrum.CIESession
Load and generate a collisional ionization equilibrium spectrum

Parameters
  linefile  [string or HDUList, optional] The line emissivity data file (either name or already open)
  coconutfile  [string or HDUList, optional] The continuum emissivity data file (either name or already open)
  elements  [arraylike(int), optional] The atomic number of elements to include (default all)
  abundset  [string] The abundance set to use. Default AG89.

Examples

Create a session instance:

```python
>>> s=CIESession()
```

Set up the responses, in this case a dummy response from 0.1 to 10 keV

```python
>>> ebins = numpy.linspace(0.1,10,1000)
>>> s.set_response(ebins, raw=True)
```

Turn on thermal broadening

```python
>>> s.set_broadening(True)
Will thermally broaden lines with emissivity > 1.000000e-18 ph cm3 s^-1
```

Return spectrum at 1.0keV
spec = s.return_spectrum(1.0)

spec is in photons cm^{-3} s^{-1} bin^{-1}; ebins are the bin edges (so spec is 1 element shorter than ebins)

Attributes

datacache  [dict] Any Atomdb FITS files which have to be opened are stored here
spectra  [NEISpectra] Object storing the actual spectral data
elements  [list(int)] Nuclear charge of elements to include.
default_abundset  [string] The abundance set used for the original emissivity file calculation
abundset  [string] The abundance set to be used for the returned spectrum
abundsetvector  [array_like(float)] The relative abundance between default_abundset and abundset for each element
response_set  [bool] Have we loaded a response (or set a dummy response)
dolines  [bool] Calculate line emission
docont  [bool] Calculate continuum emission
dopseudo  [bool] Calculate pseudocontinuum emission
broaden_limit  [float] Apply broadening to lines with epsilon > this value (ph cm3 s^{-1})
thermal_broadening  [bool] Apply thermal broadening to lines (default = False)
velocity_broadening  [float] Apply velocity broadening with this velocity (km/s). If <=0, do not apply.

apply_response  (self, spectrum)
Apply a response to a spectrum

Parameters

spectrum  [array(float)] The spectrum, in counts/bin/second, to have the response applied to. Must be binned on the same grid as the rmf.

Returns

array(float) spectrum folded through the response

return_line_emissivity  (self, Telist, taulist, Z, z1, up, lo, specunit='A', teunit='keV', apply_aeff=False, apply_abund=True, log_interp=True, init_pop='ionizing')
Get line emissivity as function of Te, tau. Assumes ionization from neutral.

Parameters

Telist  [float or array(float)] Temperature(s) in keV or K
taulist  [float] ionization timescale(s), ne * t (cm^{-3} s).
Z  [int] nuclear charge of element
z1  [int] ion charge +1 of ion
up  [int] upper level for transition
lo  [int] lower level for transition
specunit  [{'Angstrom','keV'}] Units for wavelength or energy (a returned value)
teunit | {'keV', 'K'} | Units of Telist (keV or K, default keV)
apply_aeff | bool | If true, apply the effective area to the line emissivity in the linelist to modify their intensities.
apply_abund | bool | If true, apply the abundance set in the session to the result.
log_interp | bool | Interpolate between temperature on a log-log scale (default). Otherwise linear.

Returns
ret | dict | Dictionary containing: Te, tau, teunit: as input wavelength : line wavelength (A) energy : line energy (keV) epsilon : emissivity in ph cm^3 s-1 (or ph cm^5 s^-1 if apply_aeff=True)
        first index is temperature, second is tau.

return_linelist(self, Te, tau, specrange, specunit='A', teunit='keV', apply_aeff=False, develop=False)
Get the list of line emissivities vs wavelengths.

Parameters
Te | float | Temperature in keV or K
tau | float | ionization timescale, ne * t (cm^-3 s).
specrange | [float, float] | Minimum and maximum values for interval in which to search
specunit | [Angstrom', 'keV'] | Units for specrange
teunit | [keV, K] | Units of te (keV or K, default keV)
apply_aeff | bool | If true, apply the effective area to the lines in the linelist to modify their intensities.

Returns
linelist | array(dtype) | The list of lines with lambda (A), energy (keV), epsilon (ph cm^3 s^-1), epsilon_aeff (ph cm^5 s^-1) ion (string) and upper & lower levels.

return_spectrum(self, Te, tau, init_pop=False, Te_init=False, teunit='keV', nearest=False, get_nearest_t=False, log_interp=True, freeze_ion_pop=False)
Get the spectrum at an exact temperature. Interpolates between 2 neighbouring spectra.
Finds HDU with kT closest to desired kT in given line or coco file.
Opens the line or coco file, and looks for the header unit with temperature closest to te. Use result as index input to make_spectrum.

Parameters
Te | float | Temperature in keV or K
teunit | [keV, K] | Units of te (keV or K, default keV)
raw | bool | If set, return the spectrum without response applied. Default False.
nearest | bool | If set, return the spectrum from the nearest tabulated temperature in the file, without interpolation.
get_nearest_t | bool | If set, and nearest set, return the nearest tabulated temperature as well as the spectrum.

Returns
**spectrum** [array(float)] The spectrum in photons cm\(^5\) s\(^-1\) bin\(^-1\), with the response, or photons cm\(^3\) s\(^-1\) bin\(^-1\) if raw is set.

**nearest_T** [float, optional] If nearest is set, return the actual temperature this corresponds to. Units are same as teunit.

**set_abund** (self, elements, abund)
Set the elemental abundance, relative to the abundset. Defaults to 1.0 for everything

- **Parameters**
  - elements [int or array_like(int)] The elements to change the abundance of
  - abund [float or array_like(float)] The new abundances. If only 1 value, set all elements to this abundance. Otherwise, should be of same length as elements.

- **Returns**
  - None

**Examples**

Set the abundance of iron to 0.5

```python
>>> myspec.set_abund(26, 0.5)
```

Set the abundance of iron and nickel to 0.1 and 0.2 respectively

```python
>>> myspec.set_abund([26, 28], [0.1,0.2])
```

Set the abundance of oxygen, neon, magnesium and iron to 0.1

```python
>>> myspec.set_abund([8,10,12,26], 0.1)
```

**set_abundset** (self, abundstring)
Set the abundance set.

- **Parameters**
  - abundstring [string] The abundance string (e.g. “AG89”, “uniform”. Case insensitive. See atomdb.get_abundance for list of possible abundances

- **Returns**
  - none updates self.abundset and self.abundsetvector.

**set_apec_files** (self, linefile='$ATOMDB/apec_nei_line.fits', coco='ATOMDB/apec_nei_comp.fits')
Set the apec line and coco files, and load up their data

- **Parameters**
  - linefile [str or HDUList] The filename of the line emissivity data, or the opened file.
  - coco [str or HDUList] The filename of the continuum emissivity data, or the opened file.

- **Returns**
  - None
Notes

Updates self.linefile, self.linedata, self.cocofile and self.cocodata

set_broadening (self, thermal_broadening, broaden_limit=False, velocity_broadening=0.0, velocity_broadening_units='km/s')

Turn on or off thermal broadening, and the emissivity limit for those lines

Parameters

thermal_broadening [bool] If true, turn on broadening. If False, turn it off.

broaden_limit [float] The emissivity limit for lines to be broadened. If False, this value will not be updated.

velocity_broadening [float] velocity broadening to apply. If <=0, not applied

velocity_broadening_units [string] Units of velocity_broadening. ‘km/s’ is default and only value so far.

Notes

Updates attributes thermal_broadening, broaden_limit, velocity_broadening, velocity_broadening_units

set_response (self, rmf, arf=False, raw=False)

Set the response. rmf, arf can either be the filenames or the opened files (latter is faster if called repeatedly)

Amends the following items:

self.rmf [string] The rmf file name

self.rmf [string] The response matrix

self.arffile [string] The arf file name

self.arf [string] The arf data

self.specbins [array(float)] The spectral bins on which to calculate the spectrum (keV or Å)

self.specbin_units [string ['Å', 'keV']] Units of specbins

self.ebins [array(float)] The spectral bins on which to calculate the spectrum (keV).

self.ebins_out [array(float)] The spectral bins on which to return the spectrum (keV). Can be different from specbins depending on the spectrum

self.response_set [bool] A response has been loaded

self.specbins_set [bool] The spectral bins are set

self.ebins_checksum [string] The md5checksum of the specbins

Parameters

rmf: string or HDUList or array The response matrix file or energy bins (see raw k/w)

arf: string or HDUList The ancillary response file

raw [bool] If true, the rmf variable contains the energy bin edges (keV) for all the bins, and each bin has a perfect response. This is effectively a dummy response

Returns

none
class pyatomdb.spectrum.NEISpectrum(linedata, cocodata)
Bases: pyatomdb.spectrum.CIESpectrum

A class holding the emissivity data for NEI emission, and returning spectra

Parameters

linefile [string or HDUList, optional] The line emissivity data file (either name or already open)
cocofile [string or HDUList, optional] The continuum emissivity data file (either name or already open)
elements [arraylike(int), optional] The atomic number of elements to include (default all)
abundset [string] The abundance set to use. Default AG89.

Attributes

session [CIESession] The parent CIESession
SessionType [string] “CIE”
spectra [dict of ElementSpectra] a dictionary containing the emissivity data for each HDU, subdivided by element (spectra[12][18] is an ElementSpectrum object containing the argon data for the 12th HDU)
kTlist [array] The temperatures for each emissivity HDU, in keV
logkTlist [array] log of kTlist

get_nearest_Tindex(self, Te, teunit=’keV’, nearest=False, log_interp=True)

Return the nearest temperature index in the emissivity file, or, alternatively, the array of fractions to sum

Parameters

Te [float] Temperature (keV by default)
teunit [string] Units of kT
nearest [bool] If true, return only nearest. Otherwise, return nearest 2 and fractions

Returns

ikT [list[int]] Index of temperature in HDU file (from 0, not 2)
f [list[float]] fractional weight to apply to each ikT. Should sum to 1.

return_line_emissivity(self, Te, tau, Z, zl, up, lo, specunit='A', teunit='keV', abundance=1.0, log_interp=True, init_pop='ionizing')

Return the emissivity of a line at kT, tau. Assumes ionization from neutral for now

Parameters

Te [float] Temperature in keV or K
tau [float] ionization timescale, ne * t (cm^-3 s).
Z [int] nuclear charge of element
zl [int] ion charge +1 of ion
up [int] upper level for transition
lo [int] lower level for transition
specunit [(‘Angstrom’,’keV’)] Units for wavelength or energy (a returned value)
teunit [(‘keV’, ‘K’)] Units of Telist (kev or K, default keV)
abundance [float] Abundance to multiply the emissivity by
**init_pop** [string or float]

If string: if ‘ionizing’ : all ionizing from neutral (so [1,0,0,0,...]) if ‘recombining’: all recombining from ionized (so[...0,0,1]) if array of length (Z+1) : the actual fractional populations if single float : the temperature (same units as Te)

**Returns**

**Emissivity** [float] Emissivity in photons cm^3 s^-1

**spec** [float] Wavelength or Energy of line, depending on specunit

**return_linelist** (self, Te, tau, Te_init=False, teunit='keV', nearest=False, specrange=False, specunit='A', elements=False, abundances=False, init_pop='ionizing', log_interp=True)

Return the linelist of the element

**Parameters**

**Te** [float] Electron temperature (default, keV)

**teunit** [string] Units of kT (keV by default, K also allowed)

**nearest** [bool] If True, return spectrum for the nearest temperature index. If False, use the weighted average of the (log of) the 2 nearest indexes. default is False.

**specrange** [[float, float]] Minimum and maximum values for interval in which to search

**specunit** [{‘Ansgtrom’,’keV’}] Units for specrange (default A)

**return_spectrum** (self, Te, tau, init_pop=False, Te_init=False, teunit='keV', nearest=False, elements=False, abundances=False, log_interp=True, broaden_object=False, freeze_ion_pop=False)

Return the spectrum of the element on the energy bins in self.session.specbins

**Parameters**

**Te** [float] Electron temperature (default, keV)

**tau** [float] ionization timescale, ne * t (cm^-3 s).

**teunit** [string] Units of kT (keV by default, K also allowed)

**nearest** [bool] If True, return spectrum for the nearest temperature index. If False, use the weighted average of the (log of) the 2 nearest indexes. default is False.

**Returns**

**spec** [array(float)] The element’s emissivity spectrum, in photons cm^3 s^-1 bin^-1

**pyatomdb.spectrum.add_lines** (Z, abund, lldat, ebins, zl=False, zl_drv=False, broadening=False, broadenunits='A')

Add lines to spectrum, applying gaussian broadening.

Add the lines in list lldat, with atomic number Z, to a spectrum delineated by ebins (these are the edges, in keV). Apply broadening to the spectrum if broadening != False, with units of broadenunits (so can do constant wavelength or energy broadening)

**Parameters**

**Z** [int] Element of interest (e.g. 6 for carbon)

**abund** [float] Abundance of element, relative to AG89 data.

**lldat** [dtype linelist] The linelist to add. Usually the hdu from the apec_line.fits file, often with some filters pre-applied.
z1  [int] Ion charge +1 of ion to return
z1_drv  [int] Driving Ion charge +1 of ion to return
broadening  [float] Apply spectral broadening if > 0. Units of Å or keV
broadenunits  [[‘Å’, ‘keV’]] The units of broadening, Angstroms or keV

Returns
array of float  broadened emissivity spectrum, in photons cm^3 s^-1 bin^-1. Array has len(ebins)-1 values.

pyatomdb.spectrum.apply_response(spectrum, rmf, arf=False)
Apply a response to a spectrum

Parameters
spectrum  [array(float)] The spectrum, in counts/bin/second, to have the response applied to. Must be binned on the same grid as the rmf.
rmf  [string or pyfits.hdu.hdulist.HDUList] The filename of the rmf or the opened rmf file
arf  [string or pyfits.hdu.hdulist.HDUList] The filename of the arf or the opened arf file

Returns
——–
array(float)  energy grid (keV) for returned spectrum
array(float)  spectrum folded through the response

pyatomdb.spectrum.broaden_continuum(bins, spectrum, binunits='keV', broadening=False, broadenunits='keV')
Apply a broadening to the continuum

Parameters
bins  [array(float)] The bin edges for the spectrum to be calculated on, in units of keV or Angstroms. Must be monotonically increasing. Spectrum will return len(bins)-1 values.
spectrum  [array(float)] The emissivities in each bin in the unbroadened spectrum
binunits  [[‘keV’, ‘Å’]] The energy units for bins. “keV” or “Å”. Default keV.
broadening  [float] Broaden the continuum by gaussians of this width (if False, no broadening is applied)
broadenunits  [[‘keV’, ‘Å’]] Units for broadening (kev or A)

Returns
——–
array(float)  spectrum broadened by gaussians of width broadening

pyatomdb.spectrum.convert_spec(spec, specunit, specunitout)
Convert spectral ranges from specunit to specunitout

Parameters
spec  [array] The units to return
specunit  [string] The input spectral unit (‘keV’, ‘Å’)
specunitout  [string] The output spectral unit (‘keV’, ‘Å’)

Returns
```

specout  [array] spec, converted to specunitout

pyatomdb.spectrum.convert_temp(Te, teunit, teunitout)
    Convert temperature (Te) from units teunit to teunitout

Parameters
    Te  [float] The temperature
    teunit  [string] units of Te
    teunitout  [string] output temperature units

pyatomdb.spectrum.expand_E_grid(eedges, n, Econt_in_full, cont_in_full)
    Code to expand the compressed continuum onto a series of bins.

Parameters
    eedges  [float(array)] The bin edges for the spectrum to be calculated on, in units of keV
    n  [int] The number of good data points in the continuum array
    Econt_in_full: float(array) The compressed continuum energies
    cont_in_full: float(array) The compressed continuum emissivities

Returns
    float(array) len(bins)-1 array of continuum emission, in units of photons cm^3 s^-1 bin^-1

pyatomdb.spectrum.get_effective_area(rmf, arf=False)
    Get the effective area of a response file

Parameters
    rmf  [string or pyfits.hdu.hdulist.HDUList] The filename of the rmf or the opened rmf file
    arf  [string or pyfits.hdu.hdulist.HDUList] The filename of the arf or the opened arf file

Returns
    array(float) energy grid (keV) for returned response
    array(float) effective area for the returned response

pyatomdb.spectrum.get_index(te, filename='$ATOMDB/apec_line.fits',
    teunits='keV',
    logscale=False)
    Finds HDU with kT closest ro desired kT in given line or coco file.
    Opens the line or coco file, and looks for the header unit with temperature closest to te. Use result as index input to make_spectrum

Parameters
    te  [float] Temperature in keV or K
    teunits  [{'keV', 'K'}] Units of te (kev or K, default keV)
    logscale  [bool] Search on a log scale for nearest temperature if set.
    filename  [str or hdulist] line or continuum file, already opened or filename.

Returns
    int Index in HDU file with nearest temperature to te.

pyatomdb.spectrum.get_nei_line_emissivity(Z, zl, up, lo)
    Return the line emissivity for a single line, separated out by the ion driving it
```
Parameters

Z [int] nuclear charge
z1 [int] ion charge +1
up [int] the upper level
lo [int] the lower level

Returns

emiss [dict] a dictionary containing an array, one for each ion_drv, with the emissivity in it. E.g. emiss[6] is a 51 element array, with the emissivity due to z1=6 as a fn of temperature. Also emiss[‘Te’] is the temperture in keV.

pyatomdb.spectrum.get_response_ebins(rmf)
Get the energy bins from the rmf file

Parameters

rmf [string or pyfits.hdu.hdulist.HDUList] The filename of the rmf or the opened rmf file

Returns

specbins_in [array(float)] input energy bins used. nbins+1 length, with the last item being the final bin. This is the array on which the input spectrum should be calculated.

specbins_out [array(float)] output energy bins used. nbins+1 length, with the last item being the final bin. This is the array on which the output spectrum will be returned. Often (but not always) the same as specbins_in.

pyatomdb.spectrum.list_lines(specrange, lldat=False, index=False, linefile=False, units='angstroms', Te=False, teunit='K', minepsilon=1e-20)
Gets list of the lines in a given spectral range

Parameters

specrange [[float,float]] spectral range [min,max] to return lines on
lldat [see notes] line data
index [int] index in lldat, see notes
linefile [see notes] line data file, see notes
units [{‘A’, ‘keV’}] units of specrange (default A)
Te [float] electron temperature (used if index not set to select appropriate data HDU from line file)
teunit [{‘K’, ‘keV’, ‘eV’}] units of Te
minepsilon [float] minimum epsilon for lines to be returned, in ph cm^3 s^-1

Returns

Notes

The actual line list can be defined in one of several ways:

`specrange = [10,100]`

1. `lldat` as an actual list of lines:

   ```python
   a = pyfits.open('apec_line.fits')
   llist = a[30].data
   l = list_lines(specrange, lldat=llist)
   ```

2. `lldat` as a numpy array of lines:

   ```python
   a = pyfits.open('apec_line.fits')
   llist = numpy.array(a[30].data)
   l = list_lines(specrange, lldat=llist)
   ```

3. `lldat` is a BinTableHDU from pyfits:

   ```python
   a = pyfits.open('apec_line.fits')
   llist = numpy.array(a[30])
   l = list_lines(specrange, lldat=llist)
   ```

4. `lldat` is a HDUList from pyfits. In this case index must also be set:

   ```python
   a = pyfits.open('apec_line.fits')
   index = 30
   l = list_lines(specrange, lldat=a, index=index)
   ```

5. `lldat` NOT set, `linefile` contains `apec_line.fits` file location, `index` identifies the HDU:

   ```python
   linefile = 'mydir/apec_v2.0.2_line.fits'
   index = 30
   l = list_lines(specrange, linefile=linefile, index=index)
   ```

6. `lldat` NOT set & `linefile` NOT set, `linefile` is set to `$ATOMDB/apec_line.fits`. `index` identifies the HDU:

   ```python
   index = 30
   l = list_lines(specrange, index=index)
   ```

   ```python
   pyatomdb.spectrum.list_nei_lines(specrange, Te, tau, Te_init=False, lldat=False, linefile=False, units='angstroms', teunit='K', minepsilon=1e-20, data-cache=False)
   ```

   Gets list of the lines in a given spectral range for a given NEI plasma

   For speed purposes, this takes the nearest temperature tabulated in the `linefile`, and applies the exact ionization balance as calculated to this. This is not perfect, but should be good enough.

   Note that the output from this can be passed directly to `print_lines`

Parameters

- **specrange**  [[float,float]] spectral range [min,max] to return lines on
- **Te**  [float] electron temperature
- **tau**  [float] electron density * time (cm^-3 s)
- **Te_init**  [float] initial ionization balance temperature
- **lldat**  [see notes] line data
linefile  [see notes] line data file, see notes
units  [{'\text{A}', 'keV'}] units of specrange (default A)
teunit  [{'K', 'keV'}] units of temperatures (default K)
minepsilon  [float] minimum emissivity (ph cm^3 s^{-1}) for inclusion in linelist

Returns

linelist  [dtype=(('Lambda', '>f4'), ('Lambda_Err', '>f4'), ('Epsilon', '>f4'), ('Epsilon_Err', '
>f4'), ('Element', '>i4'), ('Elem_drv', '>i4'), ('Ion', '>i4'), ('Ion_drv', '>i4'), ('Upper-Lev', '
>i4'), ('LowerLev', '>i4'))] A line list filtered by the various elements.

Notes

The actual line list can be defined in one of several ways:
specrange = [10, 100]
1. lldat as an actual list of lines:
   ```
a = pyfits.open('apec_nei_line.fits')
llist = a[30].data
l = list_nei_lines(specrange, lldat=llist)
   ```
2. lldat as a numpy array of lines:
   ```
a = pyfits.open('apec_nei_line.fits')
llist = numpy.array(a[30].data)
llist = list_nei_lines(specrange, lldat=llist)
   ```
3. lldat is a BinTableHDU from pyfits:
   ```
a = pyfits.open('apec_nei_line.fits')
llist = numpy.array(a[30])
llist = list_nei_lines(specrange, lldat=llist)
   ```
4. lldat is a HDUList from pyfits. In this case index must also be set:
   ```
a = pyfits.open('apec_nei_line.fits')
index = 30
l = list_nei_lines(specrange, lldat=a, index=index)
   ```
5. lldat NOT set, linefile contains apec_line.fits file location, index identifies the HDU:
   ```
linefile = 'mydir/apec_v3.0.2_nei_line.fits'
index = 30
l = list_nei_lines(specrange, linefile=linefile, index=index)
   ```
6. lldat NOT set & linefile NOT set, linefile is set to $ATOMDB/apec_line.fits. index identifies the HDU:
   ```
index = 30
l = list_nei_lines(specrange, Te, tau)
   ```

pyatomdb.spectrum.list_nei_lines_accurate(specrange, Te, tau, Te_init=False, lldat=False, 
linefile=False, units='angstroms', teunit='K', minepsilon=1e-20, datacache=False)

Gets list of the lines in a given spectral range for a given NEI plasma
For speed purposes, this takes the nearest temperature tabulated in the linefile, and applies the exact ionization balance as calculated to this. This is not perfect, but should be good enough.

Note that the output from this can be passed directly to print_lines

**Parameters**

- **specrange** $[[\text{float, float}]]$ spectral range $[\text{min, max}]$ to return lines on
- **Te** $[\text{float}]$ electron temperature
- **tau** $[\text{float}]$ electron density * time $(\text{cm}^{-3}\text{ s})$
- **Te_init** $[\text{float}]$ initial ionization balance temperature
- **lldat** $[\text{see notes}]$ line data
- **linefile** $[\text{see notes}]$ line data file, see notes
- **units** $[\{\text{‘A’, ‘keV’}\}]$ units of specrange (default A)
- **teunit** $[\{\text{‘K’, ‘keV’}\}]$ units of temperatures (default K)

**Returns**


**Notes**

The actual line list can be defined in one of several ways:

**specrange** = [10, 100]

1. lldat as an actual list of lines:

   ```python
   a = pyfits.open('apec_nei_line.fits')
   llist = a[30].data
   l = list_nei_lines(specrange, lldat=llist)
   ```

2. lldat as a numpy array of lines:

   ```python
   a = pyfits.open('apec_nei_line.fits')
   llist = numpy.array(a[30].data)
   l = list_nei_lines(specrange, lldat=llist)
   ```

3. lldat is a BinTableHDU from pyfits:

   ```python
   a = pyfits.open('apec_nei_line.fits')
   llist = numpy.array(a[30])
   l = list_nei_lines(specrange, lldat=llist)
   ```

4. lldat is a HDUList from pyfits. In this case index must also be set:

   ```python
   a = pyfits.open('apec_nei_line.fits')
   index = 30
   l = list_nei_lines(specrange, lldat=a, index=index)
   ```

5. lldat NOT set, linefile contains apec_line.fits file location, index identifies the HDU:
linefile = 'mydir/apec_v3.0.2_nei_line.fits'
index = 30
l = list_nei_lines(specrange, linefile=linefile, index=index)

6. Ildat NOT set & linefile NOT set, linefile is set to $ATOMDB/apec_line.fits. index identifies the HDU:

index = 30
l = list_nei_lines(specrange, Te, tau)

pyatomdb.spectrum.make_ion_index_continuum(bins, element, index=False, 
cocofile='$ATOMDB/apec_coco.fits', bin-
units='keV', fluxunits='ph', no_coco=False, 
no_pseudo=False, ion=0, broadening=False, 
broadunits='keV')

Creates the continuum for a given ion.

Parameters

- **bins** [array(float)] The bin edges for the spectrum to be calculated on, in units of keV or 
  Angstroms. Must be monotonically increasing. Spectrum will return len(bins)-1 values.
- **element** [int] Atomic number of element to make spectrum of (e.g. 6 for carbon)
- **binunits** [{‘keV’ , ‘A’}] The energy units for bins. “keV” or “A”. Default keV.
- **fluxunits** [{‘ph’ , ‘erg’}] Whether to return the emissivity in photons (‘ph’) or ergs (‘erg’).
  Defaults to photons
- **no_coco** [bool] If true, do not include the compressed continuum
- **no_pseudo** [bool] If true, do not include the pseudo continuum (weak lines)
- **ion** [int] Ion to calculate, e.g. 4 for C IV. By default, 0 (whole element).
- **index** [int] The index to generate the spectrum from. Note that the AtomDB files the emission 
  starts in hdu number 2. So for the first block, you set index=2. Only required if cocofile is a 
  filename or an HDULIST
- **cocofile** [HDUList, HDU or str] The continuum file, either already open (HDULIST) or file-
  name. alternatively, provide the HDU itself, and then do not need to define the index
- **broadening** [float] Broaden the continuum by gaussians of this width (if False, no broadening 
  is applied)
- **broadunits**: {'keV', 'A'} Units for broadening (kev or A)

Returns

- **array(float)** len(bins)-1 array of continuum emission, in units of photons cm^3 s^-1 bin^-1 
  (fluxunits = ‘ph’) or ergs cm^3 s^-1 bin^-1 (fluxunits = ‘erg’)

pyatomdb.spectrum.make_ion_spectrum(bins, index, Z, z1, 
line-
file='$ATOMDB/apec_nei_line.fits', 
cocofile='$ATOMDB/apec_nei_comp.fits', binunits='keV', 
broadening=False, broadunits='keV', abund=False, 
dummyfirst=False, nei=True, dolines=True, do-
cont=True, dopseudo=True)

make_spectrum is the most generic “make me a spectrum” routine.

It returns the emissivity in counts cm^3 s^-1 bin^-1.

Parameters
bins [array(float)] The bin edges for the spectrum to be calculated on, in units of keV or Angstroms. Must be monotonically increasing. Spectrum will return len(bins)-1 values.

index [int] The index to plot the spectrum from. Note that the AtomDB files the emission starts in hdu number 2. So for the first block, you set index=2

Z [int] Element of spectrum (e.g. 6 for carbon)

z1 [int] Ion charge +1 for the spectrum (e.g. 3 for C III)

linefile [str or HDUList] The file containing all the line emission. Defaults to “$ATOMDB/apec_line.fits”. Can also pass in the opened file, i.e. “linefile = pyatomdb.pyfits.open(‘apec_nei_line.fits’)”

cocofile [str or HDUList] The file containing all the continuum emission. Defaults to “$ATOMDB/apec_coco.fits”. Can also pass in the opened file, i.e. “cocofile = pyatomdb.pyfits.open(‘apec_nei_comp.fits’)”

binunits [{'keV', 'A'}] The energy units for bins. “keV” or “A”. Default keV.

broadening [float] Line broadening to be applied

broadenunits [{'keV', 'A'}] Units of line broadening “keV” or “A”. Default keV.

elements [iterable of int] Elements to include, listed by atomic number. If not set, include all.

abund [iterable of float, length same as elements.] If set, and array of length (elements) with the abundances of each element relative to the Andres and Grevesse values. Otherwise, assumed to be 1.0 for all elements

dummyfirst [bool] If true, add a “0” to the beginning of the return array so it is of the same length as bins (can be useful for plotting results)

nei [bool] If set, return the spectrum from the driving ion being Z, rmJ. If not set, return the spectrum for the collisional ionization equilibrium BUT note that the continuum will be wrong, as it is provided for each element as a whole.

dolines [bool] Include lines in the spectrum

docont [bool] Include the continuum in the spectrum

dopseudo [bool] Include the pseudocontinuum in the spectrum.

Returns

array of floats Emissivity in counts cm^3 s^-1 bin^-1.

pyatomdb.spectrum.make_spectrum(bins, index, linefile='$ATOMDB/apec_line.fits', cocofile='$ATOMDB/apec_coco.fits', binunits='keV', broadening=False, broadenunits='keV', elements=False, abund=False, dummyfirst=False, dolines=True, docont=True, dopseudo=True)

make_spectrum is the most generic “make me a spectrum” routine. It returns the emissivity in counts cm^3 s^-1 bin^-1.

Parameters

bins [array(float)] The bin edges for the spectrum to be calculated on, in units of keV or Angstroms. Must be monotonically increasing. Spectrum will return len(bins)-1 values.

index [int] The index to plot the spectrum from. Note that the AtomDB files the emission starts in hdu number 2. So for the first block, you set index=2

linefile [str] The file containing all the line emission. Defaults to “$ATOMDB/apec_line.fits”
cocofile [str] The file containing all the continuum emission. Defaults to “$ATOMDB/apec_coco.fits”

binunits [{‘keV’, ‘A’}] The energy units for bins. “keV” or “A”. Default keV.

broadening [float] Line broadening to be applied

broadenunits [{‘keV’, ‘A’}] Units of line broadening “keV” or “A”. Default keV.

elements [iterable of int] Elements to include, listed by atomic number. if not set, include all.

abund [iterable of float, length same as elements.] If set, and array of length (elements) with the abundances of each element relative to the Andres and Grevesse values. Otherwise, assumed to be 1.0 for all elements

dummyfirst [bool] If true, add a “0” to the beginning of the return array so it is of the same length as bins (can be useful for plotting results)

dolines [bool] Include lines in the spectrum

docont [bool] Include the continuum in the spectrum
dopseudo [bool] Include the pseudocontinuum in the spectrum.

Returns

cocofile: array of floats Emissivity in counts cm^3 s^-1 bin^-1.

pyatomdb.spectrum.print_lines (llist, specunits='A', do_cfg=False)

Prints lines in a linelist to screen

This routine is very primitive as things stand. Plenty of room for refinement.

Parameters

llist: dtype(linelist) list of lines to print. Typically returned by list_lines.

specunits: {'A', 'keV'} units to list the line positions by (A or keV, default A)

do_cfg: bool Show full configuration information for each level

Returns

Nothing, though prints data to standard out.

2.6 PyAtomDB Util module

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

util.py contains a range of miscellaneous helper codes that assist in running other AtomDB codes but are not in any way part of a physical calculation.

Version -.1 - initial release Adam Foster July 17th 2015

exception pyatomdb.util.NotImplementedError

Bases: ValueError

args

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception pyatomdb.util.ReadyError

Bases: ValueError
exception pyatomdb.util.UnitsError

Bases: ValueError

args

with_traceback()  
Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

pyatomdb.util.check_version()

Checks if there is a more recent version of the database to install.

Parameters

None.

Returns

None

pyatomdb.util.download_atomdb_emissivity_files(adbroot, userid, version)

Download the AtomDB equilibrium emissivity files for AtomDB

This code will go to the AtomDB FTP site and download the necessary files. It will then unpack them into a directory adbroot. It will not overwrite existing files with the same md5sum (to avoid pointless updates) but it will not know this until it has downloaded and unzipped the main file.

Parameters

adbroot [string] The location to install the data. Typically should match $ATOMDB

userid [string] An 8 digit ID number. Usually passed as a string, but integer is also fine (provided it is all numbers)

version [string] The version string for the release, e.g. “3.0.2”

Returns

None

pyatomdb.util.download_atomdb_nei_emissivity_files(adbroot, userid, version)

Download the AtomDB non-equilibrium emissivity files for AtomDB

This code will go to the AtomDB FTP site and download the necessary files. It will then unpack them into a directory adbroot. It will not overwrite existing files with the same md5sum (to avoid pointless updates) but it will not know this until it has downloaded and unzipped the main file.

Parameters

adbroot [string] The location to install the data. Typically should match $ATOMDB

userid [string] An 8 digit ID number. Usually passed as a string, but integer is also fine (provided it is all numbers)

version [string] The version string for the release, e.g. “3.0.2”

Returns

None

pyatomdb.util.figcoords(lowxpix, lowypix, highxpix, highypix, lowxval, lowyval, highxval, highyval, xpix, ypix, logx=False, logy=False)
pyatomdb.util.generate_equilibrium_ionbal_files(filename, settings=False)

Generate the eigen files that XSPEC uses to calculate the ionizatoin balances

Parameters

filename [string] file to write

settings [dict] This will let you override some standard inputs for get_data:

• settings['filemap']: the filemap to use if you do not want to use the default
  $ATOMDB/filemap

• settings['atomdbroot']: If you have files in non-standard locations you can replace
  $ATOMDB with this value

Returns

none

pyatomdb.util.generate_isis_files(version=", outfile='atomdb_VERSION_lineid.tar.bz2'

Generate the atomic data necessary solely for identifying lines in AtomDB. Useful in ISIS, for example.

Parameters

version [string] version number to generate line ID tarball for. Defaults to version in
  $ATOMDB/VERSION

outfile [string] the file to be generated. Defaults to atomdb_VERSION_lineid.tar.bz2

Returns

——-

none

pyatomdb.util.generate_web_fitsfiles(version=", outdir=

Split the linelist files into many small files and make an index for them

Parameters

version [string] version number to generate this for. Defaults to version in
  $ATOMDB/VERSION

outdir [string] Output files will be placed in this directory. Defaults to ‘webonly’

Returns

——-

none

pyatomdb.util.generate_xspec_ionbal_files (Z, filesuffix, settings=False)

Generate the eigen files that XSPEC uses to calculate the ionizatoin balances

Parameters

Z [int] atomic number of element

filesuffix [string] the filename will be eigenELSYMB_filesuffix.fits

settings [dict] This will let you override some standard inputs for get_data:

• settings['filemap']: the filemap to use if you do not want to use the default
  $ATOMDB/filemap

• settings['atomdbroot']: If you have files in non-standard locations you can replace
  $ATOMDB with this value
PyAtomDB Documentation, Release 0.8.1

Returns
none

pyatomdb.util.initialize()
Initialize your AtomDB Setup

This code will let you select where to install AtomDB, get the latest version of the filemap, and download the emissivity files needed for various functions to work.

Parameters
None.

Returns
None

pyatomdb.util.keyword_check(keyword)
Returns False is the keyword is in fact false, otherwise returns True

Parameters
keyword: any The keyword value

Returns
bool True if the keyword is set to not False, otherwise False

pyatomdb.util.load_user_prefs(adbroot='$ATOMDB')
Loads user preference data from $ATOMDB/userdata

Parameters
adbroot [string] The AtomDB root directory. Defaults to environment variable $ATOMDB.

Returns
dictionary keyword/setting pairs e.g. settings['USERID'] = "12345678"

pyatomdb.util.make_linelist(linefile, outfile)
Create atomdb linelist file from line.fits file

Parameters
linefile [string] The filename of the line file
outfile [string] The output filename of the string

Returns
none

pyatomdb.util.make_release_filetree(filemapfile_in, filemapfile_out, replace_source, destination, versionname)
Take an existing filemap, copy the files to the atomdbftp folder as required.

Parameters
filemapfile_in [string] The existing filemap file for the new release
filemapfile_out [string] The filename for the produced filemap
replace_source [string] All new files are in this directory.
destination [string] The folder to store the files in
versionname [string] The version string for the new files (e.g. 3_0_4)

Returns
None

Notes

This code searches for any files which don’t have $ATOMDB in the filename and assumes they are new. It updates the file name to be $ATOMDB/ename/ename_ion/ename_ion_FTYPE_versionname.fits

Versionname will have its last number stripped and replaced with “a”. So 3.0.4.2 becomes 3.0.4.a. This reflects that 4-number versions are for revisions of a file under development, while 3 number + letter are for released data.

And then copies it to the destination folder, compressing it with gzip.

pyatomdb.util.make_release_tarballs(ciefileroot, neifileroot, filemap, versionname, releasenotes, parfile, neiparfile, makelinelist=False)

Create tarball for exmissivity files for a new release.

Parameters

ciefileroot [string] The path to the CIE line and coco files, with the _line.fits and _coco.fits omitted.

neifileroot [string] The path to the NEI line and coco files, with the _line.fits and _comp.fits omitted.

filemap [string] The filemap file

versionname [string] The version string for the new files (e.g. 3.0.4).

releasenotes [string] The file name for the release notes.

parfile [string] The parameter file used to create the data

neiparfile [string] The parameter file used to create the NEI data

makelinelist [bool] Remake the line list from the line file. If not specified, assumes linelist file already exists.

Returns

None

pyatomdb.util.make_vec(d)

Create vector version of d, return True or false depending on whether input was vector or not

Parameters

  d: any scalar or vector  The input

Returns

  vecd [array of floats] d as a vector (same as input if already an iterable type)

  isvec [bool] True if d was a vector, otherwise False.

pyatomdb.util.md5Checksum(filePath)

Calculate the md5 checksum of a file

Parameters

  filepath [str] the file to calculate the md5sum of

Returns

  string the hexadecimal string md5 hash of the file
References

Taken from http://joelverhagen.com/blog/2011/02/md5-hash-of-file-in-python/

pyatomdb.util.mkdir_p(path)
Create a directory. If it already exists, do nothing.

Parameters

path [string] The directory to make

Returns

none

pyatomdb.util.question(question, default, multichoice=[])
Ask question with default answer provided. Return answer

Parameters

question [str] Question to ask
default [str] Default answer to question
multichoice [str] if set, answer must be one of these choices

Returns

str The answer.

pyatomdb.util.record_upload(fname)
Transmits record of a file transfer to AtomDB

This simply transmits the USERID, filename, and time to AtomDB. If USERID=0, then the user has chosen not

to share this information and this is skipped

Parameters

fname [string] The file name being downloaded.

Returns

None

pyatomdb.util.switch_version(version)
Changes the AtomDB version. Note this will overwrite several links on your hard disk, and will NOT be repaired

upon quitting python.

The files affect are the VERSION file and the soft links $ATOMDB/apec_line.fits, $ATOMDB/apec_coco.fits,

$ATOMDB/filemap and $ATOMDB/apec_linelist.fits

Parameters

version: string The version of AtomDB to switch to. Should be of the form “2.0.2”

Returns

None

pyatomdb.util.unique(s)
Return a list of the elements in s, but without duplicates.

For example, unique((1,2,3,1,2,3)) is some permutation of [1,2,3], unique(“abcabc”) some permutation of [“a”,

“b”, “c”], and unique(([1, 2], [2, 3], [1, 2])) some permutation of [[2, 3], [1, 2]].

For best speed, all sequence elements should be hashable. Then unique() will usually work in linear time.
If not possible, the sequence elements should enjoy a total ordering, and if list(s).sort() doesn’t raise TypeError it’s assumed that they do enjoy a total ordering. Then unique() will usually work in \(O(N\log_2(N))\) time.

If that’s not possible either, the sequence elements must support equality-testing. Then unique() will usually work in quadratic time.

**Parameters**

s  [list type object] List to remove the duplicates from

**Returns**

list type object  ...with all the duplicates removed

**References**

Taken from Python Cookbook, written by Tim Peters. [http://code.activestate.com/recipes/52560/](http://code.activestate.com/recipes/52560/)

```python
pyatomdb.util.write_ai_file (fname, dat, clobber=False)
```

Write the data in list dat to fname

**Parameters**

fname  [string] The file to write
dat  [list] The data to write. Should be a list with the following keywords:

- Z : int: nuclear charge
- z1 : int: ion charge + 1
- comments : iterable of strings: comments to append to the file
- data : numpy.array : stores all the individual level data, with the following types:
  - ion_init : int : Initial ion state of transition
  - ion_final : int : Final ion state of transition
  - level_init : int : Initial level of transition
  - level_final : int : Final level of transition
  - auto_rate : float : Autoionization rate \((s^{-1})\)
  - auto_err : float : Error in autoionization rate \((s^{-1})\)
  - auto_ref : string(20) : Autoionization rate reference (bibcode)

clobber  [bool] Overwrite existing file if it exists.

**Returns**

none

```python
pyatomdb.util.write_develop_data (data, filemapfile, Z, z1, ftype, folder, froot)
```

```python
pyatomdb.util.write_dr_file (fname, dat, clobber=False)
```

Write the data in list dat to fname

**Parameters**

fname  [string]
dat  [list]

- Z : int : nuclear charge
• z1 : int: ion charge + 1
• comments : iterable of strings: comments to append to the file
• data : numpy.array: stores all the individual level data, with the following types
  – upper_lev : int : upper level of transition
  – lower_lev : int : lower level of transition
  – wavelen : float : Wavelength of transition (A)
  – wave_obs : float : Observed wavelength of transition (A)
  – wave_err : float : Error in wavelength (A)
  – dr_type : int : DR data type. 1=Jaconelli, 2 = Safranova
  – e_excite : float : transition excitation energy (keV)
  – eexc_err : float : error in transition excitation energy (keV)
  – satelint : float : intensity factor (s-1)
  – satinterr : float : error in intensity factor (s-1)
  – params : float(10) : parameters
  – drrate_ref : string(20) : DR rate reference (usually bibcode)
  – wave_ref : string(20) : wavelength reference (bibcode)
  – wv_obs_ref : string(20) : observed wavelength reference (bibcode)

clobber [bool] Overwrite existing file if it exists.

Returns
none

pyatomdb.util.write_ec_file (fname, dat, clobber=False)
Write the data in list dat to fname

Parameters

fname [string] The file to write
dat [list] The data to write. Should be a list with the following keywords:
  • Z : int : nuclear charge
  • z1 : int : ion charge + 1
  • comments : iterable of strings: comments to append to the file
  • data : numpy.array : stores all the individual level data, with the following types:
    – lower_lev : int : Lower level of transition
    – upper_lev : int : Upper level of transition
    – coeff_type : int : Coefficient type
    – min_temp : float : Minimum temperature in range (K)
    – max_temp : float : Maximum temperature in range (K)
    – temperature : float(20) : List of temperatures (K)
    – effcollstrpar : float(20) : Effective collision strength parameters
PyAtomDB Documentation, Release 0.8.1

- inf_limit : float (OPTIONAL - if type 1.2.0) : High temperature limit point, if provided.
- reference : string(20) : Collisional excitation reference (bibcode)

**clobber** [bool] Overwrite existing file if it exists.

**Returns**

none

```python
def write_ionbal_file(Te, dens, ionpop, filename, Te_linear=False, dens_linear=False):
    Create ionization balance file
```

**Parameters**

- **Te** [array(float)] temperatures (in K)
- **dens** [array(float)] electron densities (in cm^-3)
- **ionpop** [dict of arrays] one entry for each element: ionpop[2] = numpy.array(nion,nte, ndens)
- **filename** [str] filename to write to
- **Te_linear** [bool] if true, temperature grid is linear
- **dens_linear** [bool] if true, density grid is linear

```python
def write_ir_file(fname, dat, clobber=False):
    Write the data in list dat to fname
```

**Parameters**

- **fname** [string] The file to write
- **dat** [list] The data to write. Should be a list with the following keywords:
  - **Z** : int : nuclear charge
  - **z1** : int : ion charge + 1
  - **comments** : iterable of strings : comments to append to the file
  - **ionpot** : float : ionization potential (eV)
  - **ip_dere** : float : ionization potential (eV) (from dere, optional)
  - **data** : numpy.array : stores all the individual level data, with the following types:
    - **element** : int : Nuclear Charge
    - **ion_init** : int : Initial ion stage
    - **ion_final** : int : Final ion stage
    - **level_init** : int : Initial level
    - **level_final** : int : Final level
    - **tr_type** : string(2) : Transition type:
      - CI = collisional excitation
      - EA = excitation autoionization
      - RR = radiative recombination
      - DR = dielectronic recombination
      - XI = ionization, excluded from total rate calculation
      - XR = recombination, excluded from total rate calculation
      (XR and XI are used to populate level directly)
– tr_index: int : index within the file
– par_type: int : parameter type, i.e. how the data is stored
– min_temp: float : Minimum temperature in range (K)
– max_temp: float : Maximum temperature in range (K)
– temperature: float(20) : List of temperatures (K)
– ionrec_par: float(20) : Ionization and recombination rate parameters
– wavelen: float : Wavelength of emitted lines (Å) [not used]
– wave_obs: float : Observed wavelength of emitted lines (Å) [not used]
– wave_err: float : Error in these wavelengths (Å) [not used]
– br_ratio: float : Branching ratio of this line [not used]
– br_rat_err: float : Error in branching ratio [not used]
– label: string(20) : Label for the transition
– rate_ref: string(20) : Rate reference (bibcode)
– wave_ref: string(20) : Wavelength reference (bibcode)
– wv_obs_ref: string(20) : Observed wavelength reference (bibcode)
– br_rat_ref: string(20) : Branching ratio reference (bibcode)

`clobber` [bool] Overwrite existing file if it exists.

Returns

`None`

`pyatomdb.util.write_la_file(fname, dat, clobber=False)`

Write the data in list `dat` to `fname`

Parameters

`fname` [string] The file to write

`dat` [list] The data to write. Should be a list with the following keywords:

• `Z` : int : nuclear charge
• `z1` : int : ion charge + 1
• `comments` : iterable of strings : comments to append to the file
• `data` : numpy.array: stores all the individual level data, with the following types:
  – `upper_lev` : int : Upper level of transition
  – `lower_lev` : int : Lower level of transition
  – `wavelen` : float : Wavelength of transition (Å)
  – `wave_err` : float : Error in wavelength (Å)
  – `einstein_a` : float : Einstein A coefficient (s⁻¹)
  – `ein_a_err` : float : Error in A coefficient (s⁻¹)
  – `wave_ref` : string(20) : wavelength reference (bibcode)
  – `ein_a_ref` : string(20) : A-value reference (bibcode)
clobber [bool] Overwrite existing file if it exists.

Returns
none

pyatomdb.util.write_lv_file (fname, dat, clobber=False)
Write the data in list dat to fname

Parameters

fname [string]
dat [list]

• Z : int : nuclear charge
• z1 : int: ion charge + 1
• comments : iterable of strings: comments to append to the file
• data : numpy.array: stores all the individual level data, with the following types
  – elec_config : string (40 char max) : Electron configuration strings
  – energy : float: Level energy (eV)
  – e_error : float : Energy level error (eV)
  – n_quan : int : N quantum number
  – l_quan : int : L quantum number
  – s_quan : float : S quantum number
  – lev_deg : int : level degeneracy
  – phot_type : int : photoionization data type:
    1. none
    0. hydrogenic
    1. Clark
    2. Verner
    3. XSTAR data
  – phot_par : float(20) : photoionization parameters (see specific PI type for definition)
  – Aaut_tot : float (optional) : the total autoionization rate out of the level (s^-1)
  – Arad_tot : float (optional) : the total radiative rate out of the level (s^-1)
  – energy_ref : string(20) : energy reference (usually bibcode)
  – phot_ref : string(20) : photoionization reference (bibcode)
  – Aaut_ref : string(20) : total autoionization rate reference (bibcode)
  – Arad_ref : string(20) : total radiative decay rate reference (bibcode)

clobber [bool] Overwrite existing file if it exists.

Returns
none
pyatomdb.util.write_user_prefs(prefs, adbroot='$ATOMDB')
Write user preference data to $ATOMDB/userdata. This will overwrite the entire file.
Therefore you should use “load_user_prefs”, then add in additional keywords, the call write_user_prefs.

Parameters
 prefs: dictionary  keyword/setting pairs e.g. settings[‘USERID’] = “12345678”
 adbroot [string] The AtomDB root directory. Defaults to environment variable $ATOMDB.

Returns
 None

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

- PyAtomDB Example Scripts
  - Initial installation
  - Examples with Spectra
  - Make Line List
  - Get PI Cross Sections
  - Make Cooling Curve

2.7.1 Initial installation

first_installation.py

```python
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
```

(continues on next page)
# 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 Examples with Spectra

The spectrum.py module contains routines for taking the spectrum generated from the apec model and extracting line
emissivities or continuum emission.

**CIESession Class**

The heart of the spectral analysis is the spectrum.py class. This reads in the results of an apec run (by default,$ATOMDB/apec_line.fits and $ATOMDB/apec_coco.fits) and allows the user to obtain spectra at a range of tempera-
tures accounting for instrument responses, thermal and velocity broadening, abundance changes and other issues.

**Making a Spectrum**

```python
import pyatomdb, numpy, pylab

# declare the Collisional Ionization Equilibrium session
sess = pyatomdb.spectrum.CIESession()

# set the response (here, a dummy response)
ebins = numpy.linspace(0.6,12.4,1000)
sess.set_response(ebins, raw=True)

# alternatively, could set to real RMF, ARF files using:
# sess.set_response(rmf, arf=arf)
# where rmf and arf are the file names

# Turn on (or off) thermal broadening, and if desired velocity broadening
# in this case turn on thermal broadening, and set velocity broadening to 500 km/s

sess.set_broadening(True, velocity_broadening=500.0)

# return spectrum at given temperature
kT = 0.8 # temperature in keV
spec = sess.return_spectrum(kT)
```

(continues on next page)
# note returned spectrum has units of photons cm^{-5} s^{-1} bin^{-1}, and has 1 less
# value than the energy bin grid. Prepend a 0 to it for plotting purposes
fig = pylab.figure()
fig.show()
ax = fig.add_subplot(111)
ax.plot(sess.ebins_out, numpy.append(0, spec), drawstyle='steps', label='dummy → response')

# label the figure
ax.set_xlabel('Energy (keV)')
ax.set_ylabel('Intensity (ph cm^{-3} s^{-1} bin^{-1})')

# now do the same thing but with an instrument response
sess.set_response('../tests/testdata/aciss_heg1_cy19.grmf', arf='../tests/testdata/aciss_heg1_cy19.garf')

# return the spectrum
spec = sess.return_spectrum(kT)
ax.plot(sess.ebins_out, numpy.append(0, spec), drawstyle='steps', label='HEG')

# change abundance set
sess.set_abundset('Feldman')
spec = sess.return_spectrum(kT)
ax.plot(sess.ebins_out, numpy.append(0, spec), drawstyle='steps', label='HEG+Feldman')
ax.legend(loc=0)
pylab.draw()

zzz=input("Press enter to exit")

# Showing Line Details

import pyatomdb, numpy, pylab

# declare the Collisional Ionization Equilibrium session
sess = pyatomdb.spectrum.CIESession()

kT = 0.8 # temperature in keV
waverange = [21,27]

# find the lines in a certain wavelength range
llist = sess.return_linelist(kT, waverange)

# filter out to get only lines within 10% of the strongest
llist = llist[ llist['Epsilon'] > max(llist['Epsilon'])]*0.1]

# print the results:
# column names
print(llist.dtype.names)

# data
print(llist)

# now let's see how one of the lines varies as a function of temperature. Let's look
# at the O VII resonance line (7 -> 1)

# list of temperatures to look at
kTlist = numpy.linspace(0.05,3.0,20)

# get the emissivity as a function of temperature (returned in a dict)
ldata = sess.return_line_emissivity(kTlist, 8, 7, 7, 1)

# print out the data
for i in range(len(kTlist)):
    print("%.3f keV %e ph cm^3 s^-1"%(ldata['Te'][i], ldata['epsilon'][i]))

**NEISession Class**

Derived from the CIESession class, this handles non-equilibrium spectra. As such, a few extra parameters should be set. The ionization timescale (tau) and the initial ionization fraction should be specified. This can either be as an initial temperature or an exact specified input distribution of ion populations.

**Making a spectrum**

```python
import pyatomdb, numpy, pylab

# declare the Non-Equilibrium Ionization session
sess = pyatomdb.spectrum.NEISession()

# set the response (here, a dummy response)
ebins = numpy.linspace(0.6,12.4,1000)
sess.set_response(ebins, raw=True)

# alternatively, could set to real RMF, ARF files using:
# sess.set_response(rmf, arf=arf)
# where rmf and arf are the file names

# Turn on (or off) thermal broadening, and if desired velocity broadening
# in this case turn on thermal broadening, and set velocity broadening to 500 km/s
sess.set_broadening(True, velocity_broadening=500.0)

# return spectrum at given temperature
kT = 0.8 # temperature in keV
tau = 1e11 # n_e*t cm^-3 s
kT_init=0.01 # initial temperature (for initial ion distribution)

spec = sess.return_spectrum(kT, tau, Te_init=kT_init)
```

(continues on next page)
# note returned spectrum has units of photons cm^5 s^-1 bin^-1, and has 1 less value than the energy bin grid. Prepend a 0 to it for plotting purposes
fig = pylab.figure()
fig.show()
ax = fig.add_subplot(111)
ax.plot(sess.ebins_out, numpy.append(0, spec), drawstyle='steps', label='dummy response')

# label the figure
ax.set_xlabel('Energy (keV)')
ax.set_ylabel('Intensity (ph cm$^3$ s$^{-1}$ bin$^{-1}$)')

# now do the same thing but with an instrument response
sess.set_response('../tests/testdata/aciss_heg1_cy19.grmf', arf='../tests/testdata/aciss_heg1_cy19.garf')

# return the spectrum
spec = sess.return_spectrum(kT, tau, Te_init=kT_init)
ax.plot(sess.ebins_out, numpy.append(0, spec), drawstyle='steps', label='HEG Ioniz')

# same again, but for a recombining spectrum - set initial temperature > kT
kT_init=10.0
spec = sess.return_spectrum(kT, tau, Te_init=kT_init)

# plot the results
ax.plot(sess.ebins_out, numpy.append(0, spec), drawstyle='steps', label='HEG Recomb')
ax.legend(loc=0)
pylab.draw()

zzz=input("Press enter to exit")

Showing Line Details

```python
import pyatomdb, numpy, pylab

# declare the Non-Equilibrium Ionization session
sess = pyatomdb.spectrum.NEISession()

kT = 0.8 # temperature in keV
tau = 1e11 # n_e*t cm^-3 s
kT_init=0.01 # initial temperature (for initial ion distribution)

waverange = [21,27]

# find the lines in a certain wavelength range
llist = sess.return_linelist(kT,tau, waverange)

# filter out to get only lines within 10% of the strongest
llist = llist[ llist['Epsilon'] > max(llist['Epsilon'])*0.1]```
# print the results:
# column names
print(llist.dtype.names)

# data
print(llist)

# now let's see how one of the lines varies as a function of temperature. Let's look
# at the O VII resonance line (7 -> 1)

# list of temperatures to look at
kTlist = numpy.linspace(0.05,3.0,20)

# get the emissivity as a function of temperature (returned in a dict)
ldata = sess.return_line_emissivity(kTlist, 8, 7, 7, 1)

# print out the data
for i in range(len(kTlist)):
    print("%.3f keV %e ph cm^3 s^-1"%(ldata['Te'][i], ldata['epsilon'][i]))

2.7.3 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.
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)

## 2.7.4 Get PI Cross Sections

Extract the PI cross section data: photoionization_data.py

```python
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=input('press enter to continue')

## 2.7.5 Make Cooling Curve

Make a cooling curve, total emissivity in keV cm$^{-3}$ s$^{-1}$, for each element in a specified spectral range (e.g. 2 to 10 keV).

```python
import pyatomdb, numpy, os
```

(continues on next page)
This code is an example of generating a cooling curve: the total power radiated in keV cm\(^3\) s\(^{-1}\) by each element at each temperature. It will generate a text file with the emission per element at each temperature from 1e4 to 1e9K.

This is similar to the atomdb.lorentz_power function, but with a few normalizations removed to run a little quicker.

Note that the Anders and Grevesse (1989) abundances are built in to this. These can be looked up using atomdb.get_abundance(abundset='AG89'), or the 'angr' column of the table at https://heasarc.nasa.gov/xanadu/xspec/xspec11/manual/node33.html

Adjustable parameters (energy range, element choice) are in the block marked #### ADJUST THINGS HERE

Usage: python3 calc_power.py

```python

def calc_power_oneelem_oneT(Z, Elo, Ehi, ihdu, linefile="$ATOMDB/apec_line.fits", \cocofile="$ATOMDB/apec_coco.fits"):
    
    Calculate the radiated power between 2 different energies

    INPUTS
    -----
    Z : int
        The element atomic number
    Elo : float
        The lower energy bound
    Ehi : float
        The upper energy bound
    ihdu : int
        The HDU to use, from 2 to 52. Each is a different temperature from
        10^4 to 10^9K in log space.
    linefile : string/HDUlist
        If a string, filename of the line emission. If HDU list, that file, already open
    cocofile : string/HDUlist
        If a string, filename of the continuum emission. If HDU list, that file, already open

    # make energy bins
    ebins = numpy.linspace(Elo, Ehi, 10000)
    en = (ebins[1:]+ebins[:-1])/2

    #
    spec = pyatomdb.spectrum.make_spectrum(ebins, ihdu, linefile=linefile, \cocofile=cocofile, \elements=[Z])

    # now you have a spectrum in photons. Convert to keV
```
\[
E = \text{spec}\cdot en \quad \# \text{energy in keV cm}^3 \text{ s}^{-1}
\]

```
return sum(E)
```

```python
def calc_power_oneT(Zlist, Elo, Ehi, ihdu, linefile="$ATOMDB/apec_line.fits",
   cocofile="$ATOMDB/apec_coco.fits"):
    
    """
    Zlist : [int]
    List of element nuclear charges
    Elo : float
    The lower energy bound
    Ehi : float
    The upper energy bound
    ihdu : int
    The HDU to use, from 2 to 52. Each is a different temperature from
    10^4 to 10^9K in log space.
    linefile : string/HDUlist
    If a string, filename of the line emission. If HDU list, that file, already open
    cocofile : string/HDUlist
    If a string, filename of the continuum emission. If HDU list, that file, already open
    """
    E={}
    for Z in Zlist:
        E[Z] = calc_power_oneelem_oneT(Z, Elo, Ehi, ihdu, linefile = linefile, cocofile =
            cocofile)
    return E
```

```python
def calc_power(Zlist, Elo, Ehi, hdulist=range(2,53), linefile="$ATOMDB/apec_line.fits",
   cocofile="$ATOMDB/apec_coco.fits"):
    
    """
    Zlist : [int]
    List of element nuclear charges
    Elo : float
    The lower energy bound
    Ehi : float
    The upper energy bound
    hdulist : [int]
    The HDUs to calculate the emission on, from 2 to 52. Each is a different temperature from
    10^4 to 10^9K in log space. If not given, will do for all 51 temperatures.
    linefile : string/HDUlist
    If a string, filename of the line emission. If HDU list, that file, already open
    cocofile : string/HDUlist
    If a string, filename of the continuum emission. If HDU list, that file, already open
    """
    res = {}
    res['power'] = {}
    res['temperature'] = []
    for i, ihdu in enumerate(hdulist):
        T = 10**(4+(0.1*(ihdu-2)))
        # Get temperature (there are more sophisticated ways to do this, this should just work for what you need)
res['temperature'].append(T)
res['power'][i] = calc_power_oneT(Zlist, Elo, Ehi, ihdu, linefile = linefile,
                                 cocofile = cocofile)

return res

if __name__=='__main__':

    #### ADJUST THINGS HERE

    # Elements to include
    Zlist = range(1,31) #< all the elements
    Zlist = [1,2,6,7,8,10,12,13,14,16,18,20,26,28] #< just a few

    # specify energy range you want to integrate over (min = 0.001keV, max=100keV)
    Elo = 2 #keV
    Ehi = 10 #

    # specify output file name (default output.txt)
    outfile = 'output.txt'

    #pre-open the emissivity files (not required, but saves a lot of disk access time)
    linedata = pyatomdb.pyfits.open(os.path.expandvars('$ATOMDB/apec_line.fits'))
    cocodata = pyatomdb.pyfits.open(os.path.expandvars('$ATOMDB/apec_coco.fits'))

    #### END ADJUST THINGS HERE

    # crunch the numbers
    k = calc_power(Zlist, Elo, Ehi, linefile = linedata, cocofile = cocodata)

    # output generation
    o = open(outfile, 'w')

    # header row
    s = '# Temperature log10(K)'
    for i in range(len(Zlist)):
        s += ' %12i'%Zlist[i]
    o.write(s + '
')

    # for each temperature
    for i in range(len(k['temperature'])):
        s = '%22e'%(numpy.log10(k['temperature'][i]))
        for Z in Zlist:
            s+='%12e'%(k['power'][i][Z])
        o.write(s + '
')

    # notes
    o.write('# Total Emissivity in keV cm^3 s^-1 for each element with AG89 abundances,
            between %e and %e keV\n"%(Elo, Ehi))
    o.write('# To get cooling power, multiply by Ne NH")
    o.close
2.8 License

Pyatomdb is released under the Smithsonian License:

Copyright 2015-16 Smithsonian Institution. Permission is granted to use, copy, modify, and distribute this software and its documentation for educational, research and non-profit purposes, without fee and without a signed licensing agreement, provided that this notice, including the following two paragraphs, appear in all copies, modifications and distributions. For commercial licensing, contact the Office of the Chief Information Officer, Smithsonian Institution, 380 Herndon Parkway, MRC 1010, Herndon, VA. 20170, 202-633-5256.

This software and accompanying documentation is supplied “as is” without warranty of any kind. The copyright holder and the Smithsonian Institution: (1) expressly disclaim any warranties, express or implied, including but not limited to any implied warranties of merchantability, fitness for a particular purpose, title or non-infringement; (2) do not assume any legal liability or responsibility for the accuracy, completeness, or usefulness of the software; (3) do not represent that use of the software would not infringe privately owned rights; (4) do not warrant that the software is error-free or will be maintained, supported, updated or enhanced; (5) will not be liable for any indirect, incidental, consequential special or punitive damages of any kind or nature, including but not limited to lost profits or loss of data, on any basis arising from contract, tort or otherwise, even if any of the parties has been warned of the possibility of such loss or damage.

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:

```python
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
dep = 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 the 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()
as = fig.add_subplot(111)
as.loglog(ebins, a, drawstyle='steps', label='Unbroadened')
as.loglog(ebins, b, drawstyle='steps', label='sigma = %.2f' % (de))
as.set_xlabel('Energy (keV)')
as.set_ylabel('Emissivity (ph cm$^{-3}$ s$^{-1}$ bin$^{-1}$)')
as.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 \texttt{get\_data} routine:

\begin{verbatim}
mydata = pyatomdb.atomdb.get_data(Z, z1, ftype)
\end{verbatim}

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:

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

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

\begin{verbatim}
Lvdata = pyatomdb.atomdb.get_data(8,7,'LV')
\end{verbatim}

Downloaded data files are stored in \texttt{$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 \texttt{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
Python Module Index

p
pyatomdb.apec, 3
pyatomdb.atomicdb, 15
pyatomdb.atomic, 12
pyatomdb.const, 29
pyatomdb.spectrum, 29
pyatomdb.util, 53
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>download_atomdb_emissivity_files()</td>
<td>pyatomdb.util</td>
<td>54</td>
</tr>
<tr>
<td>download_atomdb_nei_emissivity_files()</td>
<td>pyatomdb.util</td>
<td>54</td>
</tr>
<tr>
<td>dr_badnell()</td>
<td>pyatomdb.atomdb</td>
<td>18</td>
</tr>
<tr>
<td>dr_mazzotta()</td>
<td>pyatomdb.atomdb</td>
<td>18</td>
</tr>
<tr>
<td>ea_mazzotta()</td>
<td>pyatomdb.atomdb</td>
<td>18</td>
</tr>
<tr>
<td>ea_mazzotta_iron()</td>
<td>pyatomdb.atomdb</td>
<td>18</td>
</tr>
<tr>
<td>ElementSpectrum</td>
<td>pyatomdb.spectrum</td>
<td>35</td>
</tr>
<tr>
<td>elsymb_to_Z()</td>
<td>pyatomdb.atomic</td>
<td>13</td>
</tr>
<tr>
<td>elsymb_to_z0()</td>
<td>pyatomdb.atomic</td>
<td>13</td>
</tr>
<tr>
<td>expand_E_grid()</td>
<td>pyatomdb.spectrum</td>
<td>46</td>
</tr>
<tr>
<td>extract_gauntff()</td>
<td>pyatomdb.apec</td>
<td>7</td>
</tr>
<tr>
<td>extract_n()</td>
<td>pyatomdb.atomdb</td>
<td>18</td>
</tr>
<tr>
<td>f1_fcn()</td>
<td>pyatomdb.atomdb</td>
<td>18</td>
</tr>
<tr>
<td>f2_fcn()</td>
<td>pyatomdb.atomdb</td>
<td>18</td>
</tr>
<tr>
<td>gaussian_CDF</td>
<td>pyatomdb.spectrum</td>
<td>37</td>
</tr>
<tr>
<td>generate_apec_headerblurb()</td>
<td>pyatomdb.apec</td>
<td>8</td>
</tr>
<tr>
<td>generate_datatypes()</td>
<td>pyatomdb.apec</td>
<td>8</td>
</tr>
<tr>
<td>generate_cie_outputs()</td>
<td>pyatomdb.apec</td>
<td>8</td>
</tr>
<tr>
<td>generate_nei_outputs()</td>
<td>pyatomdb.apec</td>
<td>48</td>
</tr>
<tr>
<td>generate_web_fitsfiles()</td>
<td>pyatomdb.apec</td>
<td>8</td>
</tr>
<tr>
<td>generate_xspec_ionbal_files()</td>
<td>pyatomdb.apec</td>
<td>55</td>
</tr>
<tr>
<td>get_abundance()</td>
<td>pyatomdb.atomdb</td>
<td>18</td>
</tr>
<tr>
<td>get_bt_approx()</td>
<td>pyatomdb.atomdb</td>
<td>19</td>
</tr>
<tr>
<td>get_burgess_tully_extrap()</td>
<td>pyatomdb.atomdb</td>
<td>19</td>
</tr>
<tr>
<td>get_burgess_tully_transition_type()</td>
<td>pyatomdb.atomdb</td>
<td>19</td>
</tr>
<tr>
<td>get_data()</td>
<td>pyatomdb.atomdb</td>
<td>19</td>
</tr>
<tr>
<td>get_effective_area()</td>
<td>pyatomdb.spectrum</td>
<td>46</td>
</tr>
<tr>
<td>get_emissivity()</td>
<td>pyatomdb.atomdb</td>
<td>20</td>
</tr>
<tr>
<td>get_filemap_file()</td>
<td>pyatomdb.atomdb</td>
<td>20</td>
</tr>
<tr>
<td>get_index()</td>
<td>pyatomdb.spectrum</td>
<td>46</td>
</tr>
<tr>
<td>get_ion_lines()</td>
<td>pyatomdb.atomdb</td>
<td>21</td>
</tr>
<tr>
<td>get_ionbal()</td>
<td>pyatomdb.atomdb</td>
<td>21</td>
</tr>
<tr>
<td>get_ionfrac()</td>
<td>pyatomdb.atomdb</td>
<td>21</td>
</tr>
<tr>
<td>get_ionpot()</td>
<td>pyatomdb.atomdb</td>
<td>21</td>
</tr>
<tr>
<td>get_ionrec_rate()</td>
<td>pyatomdb.atomdb</td>
<td>22</td>
</tr>
<tr>
<td>get_level_details()</td>
<td>pyatomdb.atomdb</td>
<td>22</td>
</tr>
<tr>
<td>get_line_emissivity()</td>
<td>pyatomdb.atomdb</td>
<td>22</td>
</tr>
<tr>
<td>get_lorentz_levpop()</td>
<td>pyatomdb.atomdb</td>
<td>23</td>
</tr>
<tr>
<td>get_maxn()</td>
<td>pyatomdb.atomic</td>
<td>14</td>
</tr>
<tr>
<td>get_maxwell_rate()</td>
<td>pyatomdb.atomdb</td>
<td>23</td>
</tr>
<tr>
<td>get_nearest_Tindex()</td>
<td>pyatomdb.spectrum.CIESpectrum</td>
<td>34</td>
</tr>
<tr>
<td>get_nearest_Tindex()</td>
<td>pyatomdb.spectrum.NEISpectrum</td>
<td>43</td>
</tr>
<tr>
<td>get_nei_line_emissivity()</td>
<td>pyatomdb.spectrum</td>
<td>46</td>
</tr>
<tr>
<td>get_oscillator_strength()</td>
<td>pyatomdb.atomic</td>
<td>24</td>
</tr>
<tr>
<td>get_parity()</td>
<td>pyatomdb.atomic</td>
<td>14</td>
</tr>
<tr>
<td>get_response_ebins()</td>
<td>pyatomdb.spectrum</td>
<td>47</td>
</tr>
<tr>
<td>initialize()</td>
<td>pyatomdb.util</td>
<td>56</td>
</tr>
<tr>
<td>int2roman()</td>
<td>pyatomdb.atomic</td>
<td>14</td>
</tr>
<tr>
<td>int_to_roman()</td>
<td>pyatomdb.atomic</td>
<td>14</td>
</tr>
<tr>
<td>interp_rate()</td>
<td>pyatomdb.atomdb</td>
<td>24</td>
</tr>
<tr>
<td>interpulse_huntd()</td>
<td>pyatomdb.atomdb</td>
<td>24</td>
</tr>
<tr>
<td>interpolate_ionrec_rate()</td>
<td>pyatomdb.atomdb</td>
<td>24</td>
</tr>
<tr>
<td>keyword_check()</td>
<td>pyatomdb.util</td>
<td>56</td>
</tr>
<tr>
<td>kurucz()</td>
<td>pyatomdb.apec</td>
<td>9</td>
</tr>
<tr>
<td>LineData</td>
<td>pyatomdb.spectrum</td>
<td>37</td>
</tr>
<tr>
<td>list_lines()</td>
<td>pyatomdb.spectrum</td>
<td>47</td>
</tr>
<tr>
<td>list_nei_lines()</td>
<td>pyatomdb.spectrum</td>
<td>48</td>
</tr>
</tbody>
</table>
list_nei_lines_accurate() (in module pyatomdb.spectrum), 49
load_user_prefs() (in module pyatomdb.util), 56
lorentz_cie() (in module pyatomdb.atomdb), 24
lorentz_levpop() (in module pyatomdb.atomdb), 25
lorentz_neicont() (in module pyatomdb.atomdb), 25
lorentz_neicsd() (in module pyatomdb.atomdb), 25
lorentz_neilines() (in module pyatomdb.atomdb), 25
lorentz_power() (in module pyatomdb.atomdb), 25
lorentz_stronglines() (in module pyatomdb.atomdb), 25
make_ion_index_continuum() (in module pyatomdb.spectrum), 51
make_ion_spectrum() (in module pyatomdb.spectrum), 51
make_level_descriptor() (in module pyatomdb.atomdb), 26
make_linelist() (in module pyatomdb.util), 56
make_lorentz() (in module pyatomdb.atomdb), 26
make_release_filetree() (in module pyatomdb.util), 56
make_release_tarballs() (in module pyatomdb.util), 57
make_spectrum() (in module pyatomdb.spectrum), 52
make_vec() (in module pyatomdb.util), 57
make_vector() (in module pyatomdb.apec), 9
make_vector_nbins() (in module pyatomdb.apec), 9
md5Checksum() (in module pyatomdb.util), 57
mkdir_p() (in module pyatomdb.util), 58

NEISession (class in pyatomdb.spectrum), 38
NEISpectrum (class in pyatomdb.spectrum), 42
NotImplementedError, 53

occup_to_cfg() (in module pyatomdb.atomic), 14
occup_to_config() (in module pyatomdb.atomic), 14

parse_config() (in module pyatomdb.atomic), 14
parse_eissner() (in module pyatomdb.atomic), 14
parse_par_file() (in module pyatomdb.apec), 9
prep_spline_atomdb() (in module pyatomdb.atomdb), 26
print_lines() (in module pyatomdb.spectrum), 53
pyatomdb.apec (module), 3
pyatomdb.atomdb (module), 15
pyatomdb.atomic (module), 12
pyatomdb.const (module), 29
pyatomdb.spectrum (module), 29
pyatomdb.util (module), 53

read_filemap() (in module pyatomdb.atomdb), 26
ReadyError, 53
record_upload() (in module pyatomdb.util), 58
return_line_emissivity() (pyatomdb.spectrum.CIESession method), 30
return_line_emissivity() (pyatomdb.spectrum.CIESpectrum method), 34
return_line_emissivity() (pyatomdb.spectrum.NEISession method), 39
return_line_emissivity() (pyatomdb.spectrum.NEISpectrum method), 43
return_linelist() (pyatomdb.spectrum.CIESession method), 31
return_linelist() (pyatomdb.spectrum.CIESpectrum method), 34
return_linelist() (pyatomdb.spectrum.ElementSpectrum method), 36
return_linelist() (pyatomdb.spectrum.NEISession method), 40
return_linelist() (pyatomdb.spectrum.NEISpectrum method), 44
return_linematch() (pyatomdb.spectrum.ElementSpectrum method), 36
return_spec() (pyatomdb.spectrum.ContinuumData method), 35
return_spec() (pyatomdb.spectrum.LineData method), 38
return_spectrum() (pyatomdb.spectrum.CIESession method), 31
return_spectrum() (pyatomdb.spectrum.CIESpectrum method), 35
return_spectrum() (pyatomdb.spectrum.ElementSpectrum method), 36

Index 85
return_spectrum() (pyatomdb.spectrum.NEISession method), 40
return_spectrum() (pyatomdb.spectrum.NEISpectrum method), 44
roman_to_int() (in module pyatomdb.atomic), 14
rr_badnell() (in module pyatomdb.atomic), 26
rr_shull() (in module pyatomdb.atomic), 26
rr_ph_value() (in module pyatomdb.atomic), 26
run_apec() (in module pyatomdb.apec), 10
run_apec_element() (in module pyatomdb.apec), 10
run_apec_ion() (in module pyatomdb.apec), 10
run_wrap_run_apec() (in module pyatomdb.apec), 10
set_abund() (pyatomdb.spectrum.CIESession method), 31
set_abund() (pyatomdb.spectrum.NEISession method), 41
set_abundset() (pyatomdb.spectrum.CIESession method), 32
set_abundset() (pyatomdb.spectrum.NEISession method), 41
set_apec_files() (pyatomdb.spectrum.CIESession method), 32
set_apec_files() (pyatomdb.spectrum.NEISession method), 41
set_broadening() (pyatomdb.spectrum.CIESession method), 32
set_broadening() (pyatomdb.spectrum.NEISession method), 42
set_response() (pyatomdb.spectrum.CIESession method), 33
set_response() (pyatomdb.spectrum.NEISession method), 42
shorten_config() (in module pyatomdb.atomic), 14
sigma_hydrogenic() (in module pyatomdb.atomic), 14
sigma_photoion() (in module pyatomdb.atomic), 27
solve_ionbal() (in module pyatomdb.apec), 11
solve_ionbal_eigen() (in module pyatomdb.apec), 11
solve_level_pop() (in module pyatomdb.apec), 11
sort_pi_data() (in module pyatomdb.atomic), 28
spectroscopic_name() (in module pyatomdb.atomic), 14
spectroscopictoz0() (in module pyatomdb.atomic), 14
switch_version() (in module pyatomdb.util), 58
unique() (in module pyatomdb.util), 58
UnitsError, 54
with_traceback() (pyatomdb.util.NotImplementedError method), 53
with_traceback() (pyatomdb.util.ReadyError method), 54
with_traceback() (pyatomdb.util.UnitsError method), 54
wrap_ion_directly() (in module pyatomdb.apec), 12
wrap_run_apec() (in module pyatomdb.apec), 12
wrap_run_apec_element() (in module pyatomdb.apec), 12
write_ai_file() (in module pyatomdb.util), 59
write_develop_data() (in module pyatomdb.util), 59
write_dr_file() (in module pyatomdb.util), 59
write_ec_file() (in module pyatomdb.util), 60
write_filemap() (in module pyatomdb.atomic), 28
write_ionbal_file() (in module pyatomdb.util), 61
write_ir_file() (in module pyatomdb.util), 61
write_la_file() (in module pyatomdb.util), 62
write_lv_file() (in module pyatomdb.util), 63
write_user_prefs() (in module pyatomdb.util), 63
z0_to_mass() (in module pyatomdb.atomic), 14
ztoelname() (in module pyatomdb.atomic), 15
ztoelsymb() (in module pyatomdb.atomic), 15
Z_to_mass() (in module pyatomdb.atomic), 13
Ztoelname() (in module pyatomdb.atomic), 13
Ztoelsymb() (in module pyatomdb.atomic), 13

U
unique() (in module pyatomdb.util), 58
UnitsError, 54

W
with_traceback() (pyatomdb.util.NotImplementedError method), 53
with_traceback() (pyatomdb.util.ReadyError method), 54
with_traceback() (pyatomdb.util.UnitsError method), 54
wrap_ion_directly() (in module pyatomdb.apec), 12
wrap_run_apec() (in module pyatomdb.apec), 12
wrap_run_apec_element() (in module pyatomdb.apec), 12
write_ai_file() (in module pyatomdb.util), 59
write_develop_data() (in module pyatomdb.util), 59
write_dr_file() (in module pyatomdb.util), 59
write_ec_file() (in module pyatomdb.util), 60
write_filemap() (in module pyatomdb.atomic), 28
write_ionbal_file() (in module pyatomdb.util), 61
write_ir_file() (in module pyatomdb.util), 61
write_la_file() (in module pyatomdb.util), 62
write_lv_file() (in module pyatomdb.util), 63
write_user_prefs() (in module pyatomdb.util), 63
z0_to_mass() (in module pyatomdb.atomic), 14
ztoelname() (in module pyatomdb.atomic), 15
ztoelsymb() (in module pyatomdb.atomic), 15
Z_to_mass() (in module pyatomdb.atomic), 13
Ztoelname() (in module pyatomdb.atomic), 13
Ztoelsymb() (in module pyatomdb.atomic), 13

Z
z0_to_mass() (in module pyatomdb.atomic), 14
ztoelname() (in module pyatomdb.atomic), 15
ztoelsymb() (in module pyatomdb.atomic), 15
Z_to_mass() (in module pyatomdb.atomic), 13
Ztoelname() (in module pyatomdb.atomic), 13
Ztoelsymb() (in module pyatomdb.atomic), 13