acx2 Documentation

Release 2.4.1

Adam Foster

Aug 15, 2025

CONTENTS

The AtomDB CX model is a model of charge exchange during a collision between a recombining charged ion and a donor atom or ion. The electron is transferred from the donor to the recombining ion, forming a recombined ion often in an excited state. As this recombined ion relaxes to its ground state, it releases a cascade of photons with relative intensities charactersitic of CX recombination.

An original version of ACX was released in 2016, which used empirical formulae for CX emission of all ions of all the elements up to nickel. These formulae, crucially, did not include any velocity dependent effects, which are important for correctly calculating the n, l and S of the excited levels captured into. In addition, spectral information was hardwired and difficult to update, resulting in updates to AtomDB not often being reflected in the following charge exchange spectra.

We have now taken CX cross section data from the Kronos database ([1],[2],[3]), which covers many fully stripped and one electron recombining ions, and included it here. This has created a much improved dataset, which correctly captures the energy dependence of the process for these ions. For other ions not in the Kronos database, the model falls back on ACX1 behaviour.

Once Kronos or ACX1 have been used to calculate the correct capture cross sections for each n, l and/or S shell, the data is combined with the AtomDB database (www.atomdb.org) to calculate the cascade path to ground, and the subsequent emissivities and wavelengths. For ions with capture in to highly excited levels which AtomDB doesn't contain, AUTOSTRUCTURE calculations are preformed to get energy levels, wavelength and A-values for these transtitions. The result is a set of 3 files for each donor ion. The sigma file contains the cross section information for each ion. The line and cont[inuum] files contain the line emission and continuum emission from each shell capture in to, with a resolution appropriate for the model in question. For example, for ions with *nlS* resolved Kronos data, a spectrum is produced for each *n*, *l* and *S* capture and subsequent cascade. Thus there can be numerous spectra for each ion - there are 239 entries for Cl^{7+}reflecting each *n*, *l* and *S* which Kronos contains cross sections for. For ACX-level data, the spectra are calculated for each relevant *n* and the four *l* distributions, as outlined in the ACX documentation (even, statistical, Landau-Zener and separable).

For a given interaction velocity or energy, the model uses the center of mass energy to obtain the cross section for capture into each shell from Kronos. For each shell where the cross section is greater than zero, a spectrum is calculated from the *line* and *cont* files. These are then multiplied by the appropriate cross section and summed to give the spectrum for CX of a particular ion.

[1] Mullen, P. D., et al. ApJS 224, 31 (2016)
[2] Mullen, P. D., et al. ApJ 844, 7 (2017)
[3] Cumbee, R. S., et al. ApJ 852, 7 (2018)

CHAPTER

ONE

INSTALLATION

Standard python installation

Download/clone the code from GitHub.

```
pip install -e ACX2
```

Note

ACX2 is a python 3 only module. Depending on your system's setup, you may need to substitute `python3` for all references to `python`.

CHAPTER

TWO

USAGE

Each model in ACX2 can have an arbitrary set of donors. By default for the XSPEC model these are neutral H and He, but others may be selected. Additional input files will be required for these - please contact the project via the AtomDB or GitHub pages to make or discuss requests.

2.1 Data Files

Each model requires a set of data files to be installed with it. As these files are large they cannot be exported through GitHub. These files will be downloaded automatically to your $ATOMDB folder as required.

The files for each donor are:

- `sigma` files: the cross sections for capture into each n, l and S (depending on the ion) from the Kronos database
- `line` files: the line emission for capture into each n, l and S (depending on the ion) or each n and ACX1 l distribution for ions with no Kronos data
- `cont` files: same as line files, but including continuum emission. True continuum in CX is entirely 2-photon emission from H-, He- and Be-like ions.

2.2 Classes

The `acx2.py` file contains a range of classes which can be used to model different aspects of the charge exchange. The basic principal is that the fits files contain the emissivity for each ion, broken down to reflect the way that Kronos handles the data.

Kronos Resolution	Typical recombining ion	ACX2 handling
n, l, S resolved	hydrogenic bare C,N,O,Ne	Capture into each n, l, S
n resolved	bare	Capture into each n, ACX for l distribution
not included	all others	Capture into 2 n shells, ACX for l distribution

To handle this, the acx2 module contains 4 levels of classes:

- `ACXModel` : The overall ACX model. Can include multiple donor ACXDonorModel objects.
- `ACXDonorModel` : The ACX model for one donor. Contains spectra from each recombining ion.
- `CXIonSpectrum` : The spectrum for one recombining ion. Placeholder for `CXIonSpectrum_ACX1`, `CXIonSpectrum_N`, `CXIonSpectrum_NLS` classes, which handle the 3 cases is the table above. Contains CXShellSpectrum as required to get the data.
- `CXShellSpectrum` : The actual spectrum from a single each n, l, S shell (or n, ldist shell).

2.3 XSPEC

To use the model in XSPEC, one can ignore the class details above. Unfortunately, the code only works with the XSPEC python interface, pyxspec for now. Before loading the code, you will need to edit the `acx2_xspec.py` file to change the data file paths.

> **Note**
>
> You will need to edit the acx2_xspec.py file: #. It may need to be moved into your path (depending on the data) #. The data file locations are hardcoded to $ATOMDB, you will need to update them to reflect where you have installed the line, continuum and cross section files if you have put them somewhere else.

To load the ACX2 model into XSPEC, acx2_xspec module contains what you need. From a python3 shell:

```
# import the xspec python module
import xspec
# import  acx2 wrapper
import acx2_xspec
```

Once this is done, the data will load.

Five different models are loaded:

- acx2 : Emission from CX with the 14 main elements. Abundance is tied between all elements (so there is only 1 abundance keyword). Analogous to the apec model.
- vacx2 : Emission from CX with the 14 main elements. Abundance is free to vary between all the elements (though it starts frozen). Analagous to the vapec model.
- vvacx2 : Emission from 27 elements, H through Ni excluding Co. Abundance is free to vary between all the elements.
- acx2oneion : Emission from CX with one specific ion, specified with element and ion, where ion is the ion charge plus 1.
- vacxnei : As with vacx2, but with the initial ionization balance set from a non-equilibrium plasma

> **Note**
>
> Note that in the acx and vacx cases, unlike in the apec and vapec models, the effective abundance of the minor recombining elements is 0, not solar. This speeds up calculation time and does not significantly effect the resulting emission.

Once you have this, models can be used in pyxspec in the usual way, e.g.

```
m = xspec.Model('tbabs(pow+vacx2)')
```

2.3.1 Model parameters

Parameter	Definition
temperature	Plasma temperature (keV). Used for recombining particle ion fraction
collnpar	Collsion parameter (kev/u,km/s). Reduced energy or velocity of collision
collntype	Sets meaning of collnpar:
	1 - center of mass energy (kev/u)
	2 - center of mass velocity (km/s)
	3 - donor ion velocity (km/s)
	4 - recombining ion velocity (km/s)
acxmodel	ACX model to fall back on, from 1 to 8.
recombtype	single recombination (1) or all the way to neutral (2)
Hefrac	Number fraction of donor which is He (remainder is H).
abund	recombining elemental abundances. (given by individual element in vacx and vvacx)

Note

The units for collision velocity in XSPEC are km/s, not cm/s as in the underlying ACX models. This is to keep the numbers closer to 1, which XSPEC likes.

2.3.2 acxmodel definitions

These are based on analytical formulae. For ions with nlS resolved cross sections, these settings are ignored. For those with n resolved, the l distribution is implemented, but the n distribution is ignored (so models 1 and 5 give the same results). For those with no other information, capture is into n shells defined by:

$$n' = q\sqrt{\frac{I_H}{I_p}}\left(1+\frac{q-1}{\sqrt{2q}}\right)^{-1/2}$$

Where I_H is the Rydberg constant, I_p is the donor ionization potential, and q is the recombining ion charge.

For the acx1 models, the total cross section is always fixed to 3×10^{-15} cm:sup:*2*. Capture is split between the n shells, depending on the setting: if it is in one shell, then this is the one closest to n' (rounding as normal). If it is weighted, the split is (n'-n[round down]) into the n'[round up] shell, and vice versa. So if n' is 6.25, then 1/4 and 3/4 of the emission goes into the n=7 and 6 shells resepectively.

Value	n distribution	l distribution
1	one n shell	even distribution by l.
2	one n shell	statistical distribution by l.
3	one n shell	Landau-Zener distribution by l.
4	one n shell	Separable distribution by l.
5	weighted 2 shells	even distribution by l.
6	weighted 2 shells	statistical distribution by l.
7	weighted 2 shells	Landau-Zener distribution by l.
8	weighted 2 shells	Separable distribution by l.

2.3.3 Meaning of the l-shell distributions

Note that all of these weighting schemes refer to the l distribution. For example, the realtive weighting of total capture into levels with the 3s, 3p or 3d configurations. Within all the levels sharing a configuration, weighting is statistical (so 5 times more is captured into the $1s3p^3P_2$ state than the $1s3p^3P_0$).

Value	l distribution
Even	Weighted evenly between l shells
Statistical	Weighted by the statistical weight of each level .
Landau-Zener	Weighted by the function $W(l) = \frac{l(l+1)(2l+1)\times(n-1)!\times(n-2)!}{(n+l)!\times(n-l-1)!}$
Separable	Weighted by the function $W(l) = \frac{(2l+1)}{Z} \times \exp\left[\frac{-l\times(l+1)}{z}\right]$

2.3.4 Normalization of the model

Ths model deals with two emissivities, which can get confusing. The photon emissivity of a line $i \rightarrow j$ is defined as:

$$\epsilon_{ij} = N_i A_{ij}$$

That is, the number of emitted photons $cm^{-3}s^{-1}$ is the number density N_i of ions in state i, times the spontaneous transition probability A_{ij}.

We can ease the calculation of N_i by separating out the calculation:

$$N_i = \frac{N_i}{N_{z1}} \frac{N_{z1}}{N_Z} \frac{N_Z}{N_H^r} N_H^r$$

where N_{z1} is the ion abundance and N_Z is the element abundance. The N_{z1}/N_Z term is set by the temperature parameter, which is used to set the ion fraction. The N_Z/N_H^r is set for the recombining plasma by the abundance parameter, relative to the solar values of Anders and Grevesse 1989.

The ACX2 model solves the N_i/N_{z1} problem by setting up a radiative matrix, with levels populated by CX and then radiative decay to the ground state forming the rest of the matrix. The CX rate coefficient into level i is given by

$$\alpha_i^{CX}(cm^3s^{-1}) = \langle v_{com}\sigma_i(E)\rangle$$

and the rate per recombining ion per second is

$$\alpha_i^{CX}(s^{-1}) = \langle v_{com}\sigma_i(E)\rangle \, N_H^d$$

We solve the radiative matrix to obtain N_i/N_{z1}, without the donor densities included (as they are multipliers on all the diagonal matrix elements we are effectively just moving them outside the matrix). This leaves us with:

$$\epsilon_{ij} = \frac{N_i}{N_{z1}} \frac{N_{z1}}{N_Z} \frac{N_Z}{N_H} N_H^r A_{ij} N_H^d$$

ACX2 calculates the photon emissivity coefficient, $\varepsilon_{ij} = \frac{N_i}{N_{z1}} A_{ij}$, and multiplies in the elemental and ion abundances based on the abundance and temperatures specified. This leaves:

$$\epsilon_{ij} = \varepsilon_{ij} \frac{N_{z1}}{N_Z} \frac{N_Z}{N_H} N_H^r N_H^d$$

$$\epsilon_{ij} = (\text{ACX2output}) \, N_H^r N_H^d$$

To convert this to a flux from a source to our instrument we integrate over the emitting volume and account for radiation over 4π. We also, at this point, repeat the process for the He donor and add the results, accounting for the different donor ion fractions.

$$\Gamma_{ij}(cm^{-2}s^{-1}) = \frac{\int(\text{ACX2output}) N_H^r N_{(H+He)}^d dV}{4\pi D^2}$$

2.3.5 XSPEC Normalization of the model

The geometric norm represents the geometric parts of the flux calculation with a single number:

$$\text{norm}_{\text{geom}}(cm^{-5}) = \frac{\int N_H^r N_{(H+He)}^d dV}{4\pi D^2}$$

The XSPEC normalization is adjusted from the above in a few ways to make fitting more reliable. First, there is a factor of 10^{10} applied to bring the value closer to 1, which makes XSPEC fitting more reliable.

Secondly, for versions $\geq 1.1.0$, the normalization is divided by the center of mass velocity. This has been implemented to compensate for the increase in flux with an increase in velocity (since $\varepsilon_{ij} \propto \langle v_{com}\sigma_i(E)\rangle$), which resulted in the norm being anticorrelated with the collnpar. As this value would be different for every ion, the correction factor is based on a carbon-12 recombining ion and a hydrogen donor.

To recover the true emissivity of the plasma given an XSPEC fit result:

1. If using version $\geq 1.1.0$, calculate the correction factor, cf:
    1. If collntype == 1: cf = numpy.sqrt(4786031.3*collnpar/25.)
    2. If collntype == 2: cf = 1.0 * collnpar
    3. If collntype == 3: cf = 1.0 * collnpar/(1.0+12.0) = collnpar/13
    4. If collntype == 4: cf = 12.0 * collnpar/(1.0+12.0) = collnpar*12/13
2. Else, cf = 1
3. $\text{norm}_{\text{geom}} = \text{norm}_{\text{XSPEC}} * \text{cf} * 10^{-10}$

CHAPTER

THREE

VERSION HISTORY

1.0.0 March 15th 2019 Initial release

1.0.1 October 25th 2019 Fixed error in vacx2 XSPEC interface, which specified but did not implement fluorine leading to an off-by-one error for all higher-Z elements

1.0.2 February 27th 2020 Error in velocity unit conversion corrected, thanks to Gabrielle Betancourt-Martinez for reporting the bug. This will not have affected fits performed through XSPEC

1.0.3 July 9th 2020 Updated code for compatibility with changes in the PyAtomDB interface

1.1.0 November 16th 2022 Major changes to the normalization. It now has the center of mass velocity of carbon-12 divided out of it. This removed the velocity-normalization correlation which was otherwise present.

Added redshift to parameters.

Converted XSPEC interface collntype, acxmodel and recombtype into integer switches

1.1.1 December 1st 2022 Added extra option, 'calc_line_emissivity', which returns the emissivity of a specific transition due to CX. This can also be accessed during XSPEC sessions. Put more examples in the new "examples" directory

1.1.2 December 2nd 2022 Bugfix to 'calc_line_emissivity', updates to examples.

1.1.3 January 18th 2023 Bugfix to 'acx2_xspec' interface: vacx and vvacx had incorrect parameter indexes

2.1.0 January 8th 2024 Version number incremented to sync data and code release file numbers Major update to add velocity and thermal broadening. Data files rearranged to enable faster processing. A big thank you to McKenna Blake for working on this project.

2.1.1 May 28th 2024 Bugfix: acx2_xspec has mis-indexed the redshift, leading to incorrect energies. Thank you to Shuinai Zhang for identifying this mistake.

2.1.2 June 16th 2024 Renamed redshift to Redshift in XSPEC wrapper for consistency with other XSPEC models. Add in oneacx2 xspec model.

2.1.3 July 25th 2024 Added a third recombination option, where each ions' recombination is normalized by the total cross section. Fixed bug with calc_line_emissivity where normalization was incorrect. Thank you to Yuki Amano for spotting this bug.

2.1.4 August 8th 2024 Replaced deprecated scipy.integrate.cumptrapz with cumulative_trapezoid

2.2.0 February 13th 2025 Added a new function to list the lines in a wavelength region and then get the quantum numbers. See examples/test_acx_linelist.py

2.3.0 April 7th 2025 Added a new model, vacxnei, for non-equilibrium plasma. See examples/test_acx_linelist.py

2.4.0 July 16th 2025 Reworked code to increase model speed, should be ~ 10 times faster (after initialization)

2.4.1 August 15th 2025 Automatic installation of required datafiles should now work.

3.1 acx2 module class reference

class acx2.**ACXDonorModel**(*donor*, *donor_linefile=None*, *donor_contfile=None*, *donor_crosssectionfile=None*, *abundset='AG89'*, *elements=None*, *acxmodel=8*, *recombtype=1*, *collisiontype=1*)

A model of the ACX Donor

Parameters

- **donor** (*string*) – The donor symbol (e.g. "H")
- **donor_linefile** (*string*) – The file with the donor line emision
- **donor_contfile** (*string*) – The file with the donor continuum emision
- **donor_crosssectionfile** (*string*) – The cross section file for donor
- **abundset** (*optional, {"AG89"}*) – The abundance set. Only AG89 works currently
- **elements** (*optional, array-like int*) – The recombining element atomic numbers. Default is all.
- **acxmodel** (*optional, int, default=8*) – The acx model l, n distribution to fall back on.

linedata

The data in the donor_linefile

Type
HDUList

contdata

The data in the donor_contfile

Type
HDUList

crosssectiondata

The data in the donor_crosssectionfile

Type
HDUList

donorAbundance

The donor abundance. Defaults to 1.0. Intended for correctly mixing donors, e.g. in 10%He, 90%H donor plasma, set to 0.1 and 0.9 respectively, though no normalization is checked for.

Type
float

donormass

mass of the donor in a.m.u.

Type
float

abund

abundance of each element, relative to abundset. e.g. abund[6] = 2.0 means 2x solar C

Type
dict of float

`ionfrac_from_temperature`

was the ionfracance calculated from the temperature

Type
bool

`temperature`

the temperature in keV

Type
float

`ionfrac`

the ionization fraction, normalized to 1 for each element, e.g. ionfrac[2] = ndarray([0.01,0.1,0.89])

Type
dict of arrays of float

`calc_crosssection`(*collparam*)

Calculate the total and partial cross sections by n, l, S.

Parameters
collparam (*float*) – Collision energy, or velocity, as determined by set_collisiontype, in kev/u, cm/s or km/s

Return type
TBD

`calc_line_emissivity`(*Z*, *z1_in*, *up*, *lo*, *collvalue=None*)

Calculate the emissivity of a specific line.

Parameters

- **collvalue** (*float*) – The collision parameter (kev/u or cm/s, depending) to calculate the spectrum
- **Z** (*int*) – element charge
- **z1** (*int*) – recombining ion charge +1.
- **up** (*int*) – upper level of transition
- **lo** (*int*) – lower level of transition

Returns
ret – The spectrum, in ph cm^3 s-1 bin-1

Return type
array(float)

`calc_linelist`(*collparam*, *specrange*, *specunit='A'*, *redshift=0.0*)

Calculate the spectrum we want

Parameters

- **collparam** (*float*) – Collision energy, or velocity, as determined by set_collisiontype, in kev/u, cm/s or km/s
- **specrange** (*[float, float]*) – Minimum and maximum values for interval in which to search
- **specunit** (*{'Angstrom','keV'}*) – Units for specrange (default A)

Returns
linelist – Numpy array of lines and emissivity, custom dtype

Return type
numpy.array

`calc_spectrum`(*collparam*, *Tbroaden*, *vbroaden*, *ebroaden*)

Calculate the spectrum we want

Parameters
collparam (*float*) – Collision energy, or velocity, as determined by set_collisiontype, in kev/u, cm/s or km/s

Returns
self.emiss – The emissivity in photons cm^3 bin-1 s-1

Return type
array(float)

`find_crosssection_type`(*Z*, *z1*)

Find the cross section type, to assign correct CXIonSpectrum type

Parameters

- **Z** (*int*) – element charge
- **z1** (*int*) – recombining ion charge +1.

Returns

- **resolution** (*string*) – The coupling, currently N, NLS or ACX1 (returned in upper case)
- **ihdu** (*int*) – The HDU with the cross section data for the ion. Set to -1 for none.

`hashlib = <module 'hashlib' from '/home/afoster/.pyenv/versions/3.12.9/lib/python3.12/hashlib.py'>`

`initialize_CXIonspectrum`()

Called to ensure that all the ion spectra exist and are ready to use

`numpy = <module 'numpy' from '/home/afoster/.pyenv/versions/3.12.9/envs/py20250815/lib/python3.12/site-packages/numpy/__init__.py'>`

`os = <module 'os' (frozen)>`

`set_abund`(*abund*, *elements=None*)

Set the elemental abundance, also in each donor model

Parameters
abund (*array(float)*) – Abundances, relative to defulat

Return type
None

`set_abundset`(*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
updates self.abundset and self.abundsetvector.

Return type
none

`set_acxmodel`(*acxmodel*)

Set the ACX spectrum type

Parameters
acxmodel (*int*) – The acxmodel (between 1 and 8)

`set_collisionparam`(*collisionparam*)

Set the collision velocity or energy

Parameters
collisionparam (*float*) – The collision velocity or energy. Units and parameter type are set in ACXModel.set_collisiontype

Return type
None

`set_collisiontype`(*colltype*, *collunits='default'*)

Set the collision type and units

Parameters

- **colltype** (*int*) – Parameter for provided collision type Collision type - 1=energy/mass of center of mass Collision type - 2=velocity of center of mass Collision type - 3=velocity of donor Collision type - 4=velocity of receiver
- **collunits** (*string, optional*) – Units of collision paramter. Defaults to "kev/u" for colltype=1, "cm/s" for others

Return type
None

`set_donorabund`(*abund*)

Set the donor abundance

Parameters
abund (*float*) – The abundance of the donor.

`set_ebins`(*ebins*, *ebins_checksum=False*)

Set the energy bins for the spectrum being returned.

Parameters

- **ebins** (*array(float)*) – Energy bin edges (keV)
- **ebins_checksum** (*string, optional*) – The hex digest of the md5 sum of ebins. Used to check for changes.

`set_ionfrac`(*ionfrac*)

Recalculate the ionization balance based on equilibrium electron temperature

Parameters
ionfrac (*dict of arrays*) – ionization fraction, e.g. ionfrac[8]=numpy.array([0.0, 0.0, 0.0, 0.0, 0.1, 0.3, 0.4, 0.2, 0.0])

Return type
None

set_recombtype(*recombtype*)

Set the recombination type

Parameters
recombtype (*int*) – The type of recombination (1=single, 2=all the way to neutral, 3 = same as 2, but renomalized by total cross section)

set_temperature(*temperature*)

Recalculate the ionization balance based on equilibrium electron temperature

Parameters
temperature (*float*) – electron temperature in keV

Return type
None

class acx2.CXIonSpectrum(*Z*, *z1*, *ebins*, *crosssectiondata*, *linedata*, *contdata*, *cxdata*)

Class to store and prepare each ion's spectrum. Will have a subset of CXShellSpectrum for each shell. Can provide it a set of energy bins and get a spectrum back. This will store all the emissivity data.

expand_E_grid(*eedges*, *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
- **Econt_in_full** (*float(array)*) – The compressed continuum energies (keV)
- **cont_in_full** (*float(array)*) – The compressed continuum emissivities (ph cm3 s-1 keV-1)

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

Return type
float(array)

hashlib = <module 'hashlib' from '/home/afoster/.pyenv/versions/3.12.9/lib/python3.12/hashlib.py'>

numpy = <module 'numpy' from '/home/afoster/.pyenv/versions/3.12.9/envs/py20250815/lib/python3.12/site-packages/numpy/__init__.py'>

os = <module 'os' (frozen)>

set_acxmodel(*acxmodel*)

Set the ACX spectrum type

Parameters
acxmodel (*int*) – The acxmodel (between 1 and 8)

set_ebins(*ebins*, *ebins_checksum=False*)

Set the energy bins for the spectrum being returned.

Parameters

- **ebins** (*array(float)*) – Energy bin edges (keV)
- **ebins_checksum** (*string, optional*) – The hex digest of the md5 sum of ebins. Used to check for changes.

set_recombtype(*recombtype*)

Set the energy bins, also in each donor model

Parameters
ebins (*array(float)*) – The energy bins in keV for the spectrum

Return type
None

class acx2.**CXIonSpectrum_ACX1**(*Z*, *z1*, *crosssectionhdu*, *linedata*, *contdata*, *acxmodel=8*, *donor=False*, *receivermass=False*, *donormass=False*)

This is a class for ACX1 model data

ebins

The energy bin in keV

Type
array(float)

ebins_m5sum

hex digest of ebins

Type
md5hash

calc_crosssection(*collenergy*)

Calculate cross sections

calc_linelist(*collenergy*, *collvelocity*, *specrange*, *specunit='A'*, *redshift=0.0*)

Calculate the spectrum of the data

Parameters

- **ebins** (*array(float)*) – The energy bins (in keV) to calcualte the spectrum on
- **collenergy** (*flaot*) – The collision energy (keV/amu)
- **collvelocity** (*float*) – The velocity of the center of mass (cm/s)
- **specrange** (*[float, float]*) – Minimum and maximum values for interval in which to search
- **specunit** (*{'Angstrom','keV'}*) – Units for specrange (default A)

Returns
emissivity – Emissivity * velocity in photons cm4 s-2

Return type
array(float)

calc_spectrum(*collenergy*, *collvelocity*, *Tbroaden*, *vbroaden*, *ebroaden*)

Calculate the spectrum of the data

Parameters

- **ebins** (*array(float)*) – The energy bins (in keV) to calcualte the spectrum on
- **collenergy** (*flaot*) – The collision energy (keV/amu)
- **collvelocity** (*float*) – The velocity of the center of mass (cm/s)
- **linedata** (*hdulist*) – The line emissivity data
- **contdata** (*hdulist*) – The continuum emissivity data

- **acxmodel** (*int*) – n, l shell distribution, between 1 and 8:

Returns
emissivity – Emissivity * velocity in photons cm4 s-2

Return type
array(float)

hashlib = <module 'hashlib' from '/home/afoster/.pyenv/versions/3.12.9/lib/python3.12/hashlib.py'>

numpy = <module 'numpy' from '/home/afoster/.pyenv/versions/3.12.9/envs/py20250815/lib/python3.12/site-packages/numpy/__init__.py'>

os = <module 'os' (frozen)>

set_ebins(*ebins*, *ebins_checksum=False*)

Set the energy bins, also in each donor model

Parameters
ebins (*array(float)*) – The energy bins in keV for the spectrum

Return type
None

class acx2.**CXIonSpectrum_N**(*Z*, *z1*, *crosssectiondata*, *linedata*, *contdata*, *acxmodel=8*, *donor=False*, *receivermass=False*, *donormass=False*)

This is a class for n resolved Kronos data

calc_linelist(*collenergy*, *collvelocity*, *specrange*, *specunit='A'*)

Calculate the spectrum of the data

Parameters

- **ebins** (*array(float)*) – The energy bins (in keV) to calcualte the spectrum on
- **collenergy** (*flaot*) – The collision energy (keV/amu)
- **collvelocity** (*float*) – The velocity of the center of mass (cm/s)
- **specrange** (*[float, float]*) – Minimum and maximum values for interval in which to search
- **specunit** (*{'Angstrom','keV'}*) – Units for specrange (default A)

Returns
emissivity – Emissivity * velocity in photons cm4 s-2

Return type
array(float)

calc_spectrum(*collenergy*, *collvelocity*, *Tbroaden*, *vbroaden*, *ebroaden*)

Calculate the spectrum of the data

Parameters

- **ebins** (*array(float)*) – The energy bins (in keV) to calcualte the spectrum on
- **collenergy** (*flaot*) – The collision energy (keV/amu)
- **collvelocity** (*float*) – The velocity of the center of mass (cm/s)
- **linedata** (*hdulist*) – The line emissivity data
- **contdata** (*hdulist*) – The continuum emissivity data

- **acxmodel** (*int*) – n, l shell distribution, between 1 and 8

Returns
emissivity – Emissivity * velocity in photons cm4 s-2

Return type
array(float)

hashlib = <module 'hashlib' from '/home/afoster/.pyenv/versions/3.12.9/lib/python3.12/hashlib.py'>

numpy = <module 'numpy' from '/home/afoster/.pyenv/versions/3.12.9/envs/py20250815/lib/python3.12/site-packages/numpy/__init__.py'>

os = <module 'os' (frozen)>

set_ebins(*ebins*, *ebins_checksum=False*)

Set the energy bins, also in each donor model

Parameters
ebins (*array(float)*) – The energy bins in keV for the spectrum

Return type
None

class acx2.**CXIonSpectrum_NLS**(*Z*, *z1*, *crosssectiondata*, *linedata*, *contdata*, *donor=False*, *receivermass=False*, *donormass=False*)

This is a class for nls resolved Kronos data

calc_linelist(*collenergy*, *collvelocity*, *specrange*, *specunit='A'*)

Calculate the spectrum of the data

Parameters

- **ebins** (*array(float)*) – The energy bins (in keV) to calcualte the spectrum on
- **collenergy** (*flaot*) – The collision energy (keV/amu)
- **collvelocity** (*float*) – The velocity of the center of mass (cm/s)
- **specrange** (*[float, float]*) – Minimum and maximum values for interval in which to search
- **specunit** (*{'Angstrom','keV'}*) – Units for specrange (default A)

Returns
emissivity – Emissivity * velocity in photons cm4 s-2

Return type
array(float)

calc_spectrum(*collenergy*, *collvelocity*, *Tbroaden*, *vbroaden*, *ebroaden*)

Calculate the spectrum of the data

Parameters

- **ebins** (*array(float)*) – The energy bins (in keV) to calcualte the spectrum on
- **collenergy** (*flaot*) – The collision energy (keV/amu)
- **collvelocity** (*float*) – The velocity of the center of mass (cm/s)
- **linedata** (*hdulist*) – The line emissivity data
- **contdata** (*hdulist*) – The continuum emissivity data

- **acxmodel** (*int*) – n, l shell distribution, between 1 and 8

Returns
emissivity – Emissivity * velocity in photons cm4 s-2

Return type
array(float)

hashlib = <module 'hashlib' from '/home/afoster/.pyenv/versions/3.12.9/lib/python3.12/hashlib.py'>

numpy = <module 'numpy' from '/home/afoster/.pyenv/versions/3.12.9/envs/py20250815/lib/python3.12/site-packages/numpy/__init__.py'>

os = <module 'os' (frozen)>

set_ebins(*ebins*, *ebins_checksum=False*)

Set the energy bins, also in each donor model

Parameters
ebins (*array(float)*) – The energy bins in keV for the spectrum

Return type
None

class acx2.**CXShellSpectrum**(*Z*, *z1*, *n*, *l*, *linedata*, *contdata*, *s=False*)

Holds a single n, l, shell spectrum

expand_E_grid(*eedges*, *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
- **Econt_in_full** (*float(array)*) – The compressed continuum energies (keV)
- **cont_in_full** (*float(array)*) – The compressed continuum emissivities (ph cm3 s-1 keV-1)

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

Return type
float(array)

hashlib = <module 'hashlib' from '/home/afoster/.pyenv/versions/3.12.9/lib/python3.12/hashlib.py'>

numpy = <module 'numpy' from '/home/afoster/.pyenv/versions/3.12.9/envs/py20250815/lib/python3.12/site-packages/numpy/__init__.py'>

os = <module 'os' (frozen)>

scipy = <module 'scipy' from '/home/afoster/.pyenv/versions/3.12.9/envs/py20250815/lib/python3.12/site-packages/scipy/__init__.py'>

set_ebins(*ebins*, *ebins_checksum*)

Set the energy bins, also in each donor model

Parameters
ebins (*array(float)*) – The energy bins in keV for the spectrum

Return type
None

class acx2.**DummyCXShellSpectrum**(*Z*, *z1*, *n*, *l*, *s=False*)

Placeholder for a blank spectrum

calc_spectrum(*ebins*, *ebins_checksum*, *Tbroaden*, *vbroaden*, *ebroaden*)

Return zeros

Parameters

- **ebinshash** (*string*) – md5 sum of ebins
- **ebins** (*array(float)*) – array of floats

Returns
0.0

Return type
there is no data here, so return zero.

set_ebins(*ebins*, *ebins_checksum=False*)

Dummy

Parameters

- **ebinshash** (*string*) – md5 sum of ebins
- **ebins** (*array(float)*) – array of floats

Return type
None

acx2.**download_acx_emissivity_files**(*adbroot*, *userid*, *version*, *donor*)

Download the AtomDB CX model 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, including donors for the release, e.g. "2_1_0-h_he"

Return type
None

acx2.**get_full_line_info**(*linelist*, *filemap='$ATOMDB/filemap_acx'*, *atomdbroot='$ATOMDB'*)

Gets the data for the full linelist

acx2.**loginterp**(*newx*, *x*, *y*, *offset=1e-40*)

Interpolation helper function. Interpolates linearly on a log-log grid If newx < x[0], return x[0]. If newx > x[-1], extrapolate slope of last 2 points, unless y[-1] ==0 in which case return 0.

Parameters

- **newx** (*float*) – The new X parameters (energy, kev/amu)

- **x** (*array(float)*) – The x parameters (energy, kev/amu)
- **y** (*array(float)*) – The y parameters (cross section, cm2)
- **offset** (*float (optional)*) – An offset to apply before interpolation. Prevents log(0) issues.

Returns
newy – The interpolated cross section. Minimum of 0 in case of numerical issues.

Return type
float

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

O

S

T