CRRLpy Documentation

Release 0.1

Pedro Salas

March 16, 2018

CONTENTS

Contents:

CHAPTER

ONE

CRRLPY

Tools for processing **C**arbon **R**adio **R**ecombination **L**ine spectra.

The models are not shipped with the modules and scripts.

The documentation can be found in: http://astrofle.github.io/CRRLpy/

CHAPTER

TWO

CRRLPY.CRRLS MODULE

crrlpy.crrls.**alphanum_key**(*s*)
Turn a string into a list of string and number chunks.

Parameters **s** – String

Returns List with strings and integers.

Return type list

Example

```
>>> alphanum_key('z23a')
['z', 23, 'a']
```

crrlpy.crrls.**average**(*data*, *axis*, *n*)
Averages data along the given axis by combining n adjacent values.

Parameters

- **data** (*numpy array*) – Data to average along a given axis.
- **axis** (*int*) – Axis along which to average.
- **n** (*int*) – Factor by which to average.

Returns Data decimated by a factor n along the given axis.

Return type numpy array

crrlpy.crrls.**best_match_indx_tol**(*value*, *array*, *tol*)
Searchs for the best match to a value inside an array given a tolerance.

Parameters

- **value** (*float*) – Value to find inside the array.
- **tol** (*float*) – Tolerance for match.
- **array** (*numpy.array*) – List to search for the given value.

Returns Best match for val inside array.

Return type float

crrlpy.crrls.**best_match_value**(*value*, *array*)
Searchs for the closest ocurrence of value in array.

Parameters

- **value** (*float*) – Value to find inside the array.
- **array** (*list or numpy.array*) – List to search for the given value.

Returns Best match for the value inside array.

Return type float.

Example

```
>>> a = [1,2,3,4]
>>> best_match_value(3.5, a)
3
```

crrlpy.crrls.**blank_lines**(*freq*, *tau*, *reffreqs*, *v0*, *dv*)
Blanks the lines in a spectra.

Parameters

- **freq** (*array, MHz*) – Frequency axis of the spectra.
- **tau** (*array*) – Optical depth axis of the spectra.
- **reffreqs** (*list*) – List with the reference frequency of the lines. Should be the rest frequency.
- **v0** (*float*) – Velocity shift to apply to the lines defined by *reffreq*. (km/s)
- **dv** (*float*) – Velocity range to blank around the lines. (km/s)

crrlpy.crrls.**blank_lines2**(*freq*, *tau*, *reffreqs*, *dv*)
Blanks the lines in a spectra.

Parameters

- **freq** (*array*) – Frequency axis of the spectra. (MHz)
- **tau** (*array*) – Optical depth axis of the spectra.
- **reffreqs** (*list*) – List with the reference frequency of the lines. Should be the rest frequency.
- **dv** (*float*) – Velocity range to blank around the lines. (km/s)

crrlpy.crrls.**df2dv**(*f0*, *df*)
Convert a frequency delta to a velocity delta given a central frequency.

Parameters

- **f0** (*float*) – Rest frequency.
- **df** (*float*) – Frequency delta.

Returns The equivalent velocity delta for the given frequency delta.

Return type float in m s^{-1}

crrlpy.crrls.**dsigma2dfwtm**(*dsigma*)
Converts the σ parameter of a Gaussian distribution to its FWTM.

crrlpy.crrls.**dv2df**(*f0*, *dv*)
Convert a velocity delta to a frequency delta given a central frequency.

Parameters

- **f0** (*float*) – Rest frequency.
- **dv** (*float*) – Velocity delta in m/s.

Returns For the given velocity delta the equivalent frequency delta.

Return type float

crrlpy.crrls.**dv_minus_doppler**(*dV*, *ddV*, *dD*, *ddD*)
Returns the Lorentzian contribution to the line width assuming that the line has a Voigt profile.

Parameters

- **dV** (*float*) – Total line width
- **ddV** (*float*) – Uncertainty in the total line width.
- **dD** (*float*) – Doppler contribution to the line width.
- **ddD** – Uncertainty in the Doppler contribution to the line width.

Returns The Lorentz contribution to the total line width.

Return type float

crrlpy.crrls.**dv_minus_doppler2**(*dV*, *ddV*, *dD*, *ddD*)
Returns the Lorentzian contribution to the line width assuming that the line has a Voigt profile.

Parameters

- **dV** (*float*) – Total line width
- **ddV** (*float*) – Uncertainty in the total line width.
- **dD** (*float*) – Doppler contribution to the line width.
- **ddD** – Uncertainty in the Doppler contribution to the line width.

Returns The Lorentz contribution to the total line width.

Return type float

crrlpy.crrls.**f2n**(*f*, *line*, *n_max=1500*)
Converts a given frequency to a principal quantum number n for a given line.

Parameters

- **f** (*array*) – Frequency to convert. (MHz)
- **line** (*string*) – The equivalent n will be referenced to this line.
- **n_max** (*int*) – Maximum n number to include in the search. (optional, Default 1)

Returns Corresponding n for a given frequency and line. If the frequency is not an exact match, then it will return an empty array.

Return type array

crrlpy.crrls.**find_lines_sb**(*freq*, *line*, *z=0*, *verbose=False*)
Finds if there are any lines of a given type in the frequency range. The line frequencies are corrected for redshift.

Parameters **:param freq: Frequency axis in which to search for lines (MHz). It should not contain NaN or inf values.**

:type freq: array

:param line: Line type to search for.

:type line: string

:param z: Redshift to apply to the rest frequencies.

:type z: float

:param verbose: Verbose output?

:type verbose: bool

:returns: Lists with the princpipal quantum number and the reference frequency of the line. The frequencies are redshift corrected in MHz.

:rtype: array.

See also:

load_ref Describes the format of line and the available ones.

Examples

```
>>> from crrlpy import crrls
>>> freq = [10, 11]
>>> ns, rf = crrls.find_lines_sb(freq, 'RRL_CIalpha')
>>> ns
array([ 843.,  844.,  845.,  846.,  847.,  848.,  849.,  850.,  851.,
        852.,  853.,  854.,  855.,  856.,  857.,  858.,  859.,  860.,
        861.,  862.,  863.,  864.,  865.,  866.,  867.,  868.,  869.])
```

crrlpy.crrls.**fit_continuum**(*x*, *y*, *degree*, *p0*, *verbose=False*)

Fits a polynomial to the continuum.

Parameters **x** : x axis.

y : y axis.

crrlpy.crrls.**fit_model**(*x*, *y*, *model*, *p0*, *wy=None*, *mask=None*)

Fits a model to the data defined by *x* and *y*. It uses *p0* as starting values.

Parameters

- **x** (*array*) – Abscissa values of the data to be fit.
- **y** (*array*) – Ordinate values of the data to be fit.
- **model** (*callable*) – Model to be fit.
- **p0** (*dict*) – Dictionary with the starting values for the fit.
- **wy** (*array*) – Weights of the ordinate values. (Optional)
- **mask** (*array*) – Mask to apply to the *x* and *y* values. (Optional)

Returns An object containing the results of the fit.

Return type lmfit.model.ModelResult

crrlpy.crrls.**freq2vel**(*f0*, *f*)

Convert a frequency axis to a velocity axis given a central frequency. Uses the radio definition of velocity.

Parameters

- **f0** (*float*) – Rest frequency for the conversion. (Hz)
- **f** (*numpy array*) – Frequencies to be converted to velocity. (Hz)

Returns f converted to velocity given a rest frequency f_0.

Return type numpy array

crrlpy.crrls.**fwhm2sigma**(*fwhm*)

Converts a FWHM to the standard deviation, σ of a Gaussian distribution.

Parameters **fwhm** (*array*) – FWHM of the Gaussian.

Returns Equivalent standard deviation of a Gausian with a Full Width at Half Maximum *fwhm*.

Return type array

Example

```
>>> 1/fwhm2sigma(1)
2.3548200450309493
```

crrlpy.crrls.**gauss_area**(*amplitude*, *sigma*)
Returns the area under a Gaussian of a given amplitude and sigma.

Parameters

- **amplitude** – Amplitude of the Gaussian, A.
- **sigma** (*array*) – Standard deviation fo the Gaussian, σ.

Returns The area under a Gaussian of a given amplitude and standard deviation.

Return type array

crrlpy.crrls.**gauss_area2peak**(*area*, *sigma*)
Returns the maximum value of a Gaussian function given its amplitude and standard deviation “math:*sigma*.

crrlpy.crrls.**gauss_area2peak_err**(*amplitude*, *area*, *darea*, *sigma*, *dsigma*)
Returns the maximum value of a Gaussian function given its amplitude, area and standard deviation “math:*sigma*.

crrlpy.crrls.**gauss_area_err**(*amplitude*, *amplitude_err*, *sigma*, *sigma_err*)
Returns the error on the area of a Gaussian of a given *amplitude* and *sigma* with their corresponding errors. It assumes no correlation between *amplitude* and *sigma*.

Parameters

- **amplitude** (*array*) – Amplitude of the Gaussian.
- **amplitude_err** (*array*) – Error on the amplitude.
- **sigma** – Standard deviation of the Gaussian.
- **sigma_err** – Error on sigma.

Returns The error on the area.

Return type array

crrlpy.crrls.**gaussian**(*x*, *sigma*, *center*, *amplitude*)
Gaussian function in one dimension.

Parameters

- **x** (*array*) – x values for which to evaluate the Gaussian.
- **sigma** (*float*) – Standard deviation of the Gaussian.
- **center** (*float*) – Center of the Gaussian.
- **amplitude** (*float*) – Peak value of the Gaussian.

Returns Gaussian function of the given amplitude and standard deviation evaluated at x.

Return type array

crrlpy.crrls.**get_axis**(*header*, *axis*)
Constructs a cube axis.

Parameters

- **header** (*pyfits header*) – Fits cube header.
- **axis** (*int*) – Axis to reconstruct.

Returns cube axis

Return type numpy array

crrlpy.crrls.**get_line_mask**(*freq*, *reffreq*, *v0*, *dv*)

Return a mask with ranges where a line is expected in the given frequency range for a line with a given reference frequency at expected velocity v0 and line width dv0.

Parameters

- **freq** (*numpy array or list*) – Frequency axis where the line is located.
- **reffreq** (*float*) – Reference frequency for the line.
- **v0** (*float, km/s*) – Velocity of the line.
- **dv** (*float, km/s*) – Velocity range to mask.

Returns Mask centered at the line center and width *dv0* referenced to the input *freq*.

crrlpy.crrls.**get_line_mask2**(*freq*, *reffreq*, *dv*)

Return a mask with ranges where a line is expected in the given frequency range for a line with a given reference frequency and line width dv.

Parameters

- **freq** (*numpy array or list*) – Frequency axis where the line is located.
- **reffreq** (*float*) – Reference frequency for the line.
- **dv** (*float, km/s*) – Velocity range to mask.

Returns Mask centered at the line center and width *dv0* referenced to the input *freq*.

crrlpy.crrls.**get_rchi2**(*x_obs*, *x_mod*, *y_obs*, *y_mod*, *dy_obs*, *dof*)

Computes the reduced χ squared, $\chi^2_\nu = \chi^2/dof$.

Parameters

- **x_obs** (*array*) – Abscissa values of the observations.
- **x_mod** (*array*) – Abscissa values of the model.
- **y_obs** (*array*) – Ordinate values of the observations.
- **y_mod** (*array*) – Ordinate values of the model.
- **dy_obs** (*array*) – Error on the ordinate values of the observations.
- **dof** (*float*) – Degrees of freedom.

crrlpy.crrls.**get_rms**(*data*, *axis=None*)

Computes the rms of the given data.

Parameters

- **data** (*numpy array or list*) – Array with values where to compute the rms.
- **axis** (*int*) – Axis over which to compute the rms. Default: None

Returns The rms of data.

$$\mathrm{rms} = \sqrt{\langle \mathrm{data} \rangle^2 + V[\mathrm{data}]}$$

where V is the variance of the data.

crrlpy.crrls.**is_number**(*str*)
Checks wether a string is a number or not.

Parameters **str** (*string*) – String.

Returns True if *str* can be converted to a float.

Return type bool

Example

```
>>> is_number('10')
True
```

crrlpy.crrls.**lambda2vel**(*wav0*, *wav*)
Convert a wavelength axis to a velocity axis given a rest wavelength. Uses the optical definition of velocity.

Parameters

- **wav0** (*float*) – Rest frequency for the conversion.)
- **wav** (*numpy array*) – Frequencies to be converted to velocity.

Returns Velocity. (Default: m/s)

Return type numpy array

crrlpy.crrls.**linear**(*x*, *a*, *b*)
Linear model.

Parameters

- **x** (*array*) – x values where to evalueate the line.
- **a** (*float*) – Slope of the line.
- **b** (*float*) – y value for x equals 0.

Returns A line defined by $ax + b$.

Return type array

crrlpy.crrls.**load_model**(*prop*, *specie*, *temp*, *dens*, *other=None*)
Loads a model for the CRRL emission.

crrlpy.crrls.**load_ref**(*line*)
Loads the reference spectrum for the specified line.

Available lines:
RRL_CIalpha
RRL_CIbeta
RRL_CIdelta
RRL_CIgamma
RRL_CI13alpha
RRL_HeIalpha
RRL_HeIbeta
RRL_HIalpha
RRL_HIbeta
RRL_SIalpha
RRL_SIbeta

More lines can be added by including a list in the linelist directory.

Parameters **line** : string

Line for which the principal quantum number and reference frequencies are desired.

Returns **n** : array

Principal quantum numbers.

reference_frequencies : array

Reference frequencies of the lines inside the spectrum in MHz.

crrlpy.crrls.**lookup_freq**(*n*, *line*)

Returns the frequency of a line given the transition number n.

Parameters

- **n** (*int*) – Principal quantum number to look up for.
- **line** (*string*) – Line for which the frequency is desired.

Returns Frequency of line(n).

Return type float

crrlpy.crrls.**lorentz_width**(*n*, *ne*, *Te*, *Tr*, *W*, *dn=1*)

Gives the Lorentzian line width due to a combination of radiation and collisional broadening. The width is the FWHM in Hz. It uses the models of Salgado et al. (2015).

Parameters

- **n** (*array*) – Principal quantum number for which to evaluate the Lorentz widths.
- **ne** (*float*) – Electron density to use in the collisional broadening term.
- **Te** (*float*) – Electron temperature to use in the collisional broadening term.
- **Tr** (*float*) – Radiation field temperature.
- **W** (*float*) – Cloud covering factor used in the radiation broadening term.
- **dn** (*int*) – Separation between the levels of the transition. e.g., *dn=1* for CIalpha.

Returns The Lorentz width of a line due to radiation and collisional broadening.

Return type array

crrlpy.crrls.**mask_outliers**(*data*, *m=2*)

Masks values larger than m times the data median. This is similar to sigma clipping.

Parameters **data** (*array*) – Data to mask.

Returns An array of the same shape as data with True where the data should be flagged.

Return type array

Example

```
>>> data = [1,2,3,4,5,6]
>>> mask_outliers(data, m=1)
array([ True, False, False, False, False,  True], dtype=bool)
```

crrlpy.crrls.**n2f**(*n*, *line*, *n_min=1*, *n_max=1500*, *unitless=True*)

Converts a given principal quantum number n to the frequency of a given line.

crrlpy.crrls.**natural_sort**(*list*)

Sort the given list in the way that humans expect. Sorting is done in place.

Parameters **list** (*list*) – List to sort.

Example

```
>>> my_list = ['spec_3', 'spec_4', 'spec_1']
>>> natural_sort(my_list)
>>> my_list
['spec_1', 'spec_3', 'spec_4']
```

crrlpy.crrls.**ngaussian**(*x*, *sigma*, *center*)
Normalized Gaussian distribution.

crrlpy.crrls.**plot_fit**(*fig*, *x*, *y*, *fit*, *params*, *vparams*, *sparams*, *rms*, *x0*, *refs*, *refs_cb=None*, *refs_cd=None*, *refs_cg=None*)

crrlpy.crrls.**plot_fit_single**(*fig*, *x*, *y*, *fit*, *params*, *rms*, *x0*, *refs*, *refs_cb=None*, *refs_cd=None*, *refs_cg=None*)

crrlpy.crrls.**plot_model**(*x*, *y*, *xm*, *ym*, *out*)

crrlpy.crrls.**plot_spec_vel**(*out*, *x*, *y*, *fit*, *A*, *Aerr*, *x0*, *x0err*, *sx*, *sxerr*)

crrlpy.crrls.**pressure_broad**(*n*, *te*, *ne*)
Pressure induced broadening in Hz. Shaver (1975) Eq. (64a) for te <= 1000 K and Eq. (61) for te > 1000 K.

crrlpy.crrls.**pressure_broad_coefs**(*Te*)
Defines the values of the constants a and γ that go into the collisional broadening formula of Salgado et al. (2015).

Parameters **Te** (*float*) – Electron temperature.

Returns The values of a and γ.

Return type list

crrlpy.crrls.**pressure_broad_salgado**(*n*, *Te*, *ne*, *dn=1*)
Pressure induced broadening in Hz. This gives the FWHM of a Lorentzian line. Salgado et al. (2015)

Parameters

- **n** (*float or array*) – Principal quantum number for which to compute the line broadening.
- **Te** (*float*) – Electron temperature to use when computing the collisional line width.
- **ne** (*float*) – Electron density to use when computing the collisional line width.
- **dn** (*int*) – Difference between the upper and lower level for which the line width is computed. (default 1)

Returns The collisional broadening FWHM in Hz using Salgado et al. (2015) formulas.

Return type float or array

crrlpy.crrls.**radiation_broad**(*n*, *W*, *Tr*)
Radiation induced broadening in Hz. Shaver (1975)

crrlpy.crrls.**radiation_broad_salgado**(*n*, *W*, *Tr*)
Radiation induced broadening in Hz. This gives the FWHM of a Lorentzian line. Salgado et al. (2015)

crrlpy.crrls.**radiation_broad_salgado_general**(*n*, *W*, *Tr*, *nu0*, *alpha*)
Radiation induced broadening in Hz. This gives the FWHM of a Lorentzian line. The expression is valid for power law like radiation fields. Salgado et al. (2015)

crrlpy.crrls.**sigma2fwhm**(*sigma*)
Converts the σ parameter of a Gaussian distribution to its FWHM.

Parameters **sigma** (*float*) – σ value of the Gaussian distribution.

Returns The FWHM of a Gaussian with a standard deviation σ.

Return type float

crrlpy.crrls.**sigma2fwhm_err**(*dsigma*)
Converts the error on the sigma parameter of a Gaussian distribution to the error on the FWHM.

Parameters **dsigma** – Error on sigma of the Gaussian distribution.

Returns The error on the FWHM of a Gaussian with a standard deviation σ.

Return type float

crrlpy.crrls.**sigma2fwtm**(*sigma*)
Converts the σ parameter of a Gaussian distribution to its FWTM.

crrlpy.crrls.**stack_interpol**(*spectra*, *vmin*, *vmax*, *dv*, *show=True*, *rmsvec=False*)

crrlpy.crrls.**stack_irregular**(*lines*, *window=''*, ***kargs*)
Stacks spectra by adding them together and then convolving with a window to reduce the noise. Available window functions: Gaussian, Savitzky-Golay and Wiener.

crrlpy.crrls.**temp2tau**(*x*, *y*, *model*, *p0*, *wy=None*, *mask=None*)
Converts a temperature to optical depth. It will fit the continuum using model and then subtract it and divide by it.

Parameters

- **x** (*array*) – x values.
- **y** (*array*) – y values to be converted into optical depths.
- **model** (*callable*) – Model to fit to the continuum.
- **p0** – Starting values for the model to be fit to the continuum.
- **wy** (*array*) – Weights for the y values. (Optional)
- **mask** (*array*) – Mask to apply to the *x* and *y* values.

Returns y/model - 1.

Return type array

crrlpy.crrls.**tryint**(*str*)
Returns an integer if *str* can be represented as one.

Parameters **str** (*string*) – String to check.

Returns True is str can be cast to an int.

Return type int

crrlpy.crrls.**vel2freq**(*f0*, *vel*)
Convert a velocity axis to a frequency axis given a central frequency. Uses the radio definition, $\nu = f_0(1 - v/c)$.

Parameters

- **f0** (*float*) – Rest frequency in Hz.
- **vel** (*float or array*) – Velocity to convert in m/s.

Returns The frequency which is equivalent to vel.

Return type float or array

crrlpy.crrls.**voigt**(*x*, *sigma*, *gamma*, *center*, *amplitude*)
The Voigt line shape in terms of its physical parameters.

Parameters

- **x** – independent variable
- **sigma** – HWHM of the Gaussian
- **gamma** – HWHM of the Lorentzian
- **center** – the line center
- **amplitude** – the line area

crrlpy.crrls.**voigt_**(*x*, *y*)

crrlpy.crrls.**voigt_area**(*amp*, *fwhm*, *gamma*, *sigma*)
Returns the area under a Voigt profile. This approximation has an error of less than 0.5%

crrlpy.crrls.**voigt_area2**(*peak*, *fwhm*, *gamma*, *sigma*)
Area under the Voigt profile using the expression provided by Sorochenko & Smirnov (1990).

Parameters **peak** : `float`

Peak of the Voigt line profile.

fwhm : `float`

Full width at half maximum of the Voigt profile.

gamma : `float`

Full width at half maximum of the Lorentzian profile.

sigma : `float`

Full width at half maximum of the Doppler profile.

crrlpy.crrls.**voigt_area_err**(*area*, *amp*, *damp*, *fwhm*, *dfwhm*, *gamma*, *sigma*)
Returns the error of the area under a Voigt profile. Assumes that the parameter c has an error of 0.5%.

crrlpy.crrls.**voigt_fwhm**(*dD*, *dL*)
Computes the FWHM of a Voigt profile. http://en.wikipedia.org/wiki/Voigt_profile#The_width_of_the_Voigt_profile

$$FWHM_{\rm V} = 0.5346dL + \sqrt{0.2166dL^2 + dD^2}$$

Parameters

- **dD** (*array*) – FWHM of the Gaussian core.
- **dL** (*array*) – FWHM of the Lorentz wings.

Returns The FWHM of a Voigt profile.

Return type array

crrlpy.crrls.**voigt_fwhm_err**(*dD*, *dL*, *ddD*, *ddL*)
Computes the error in the FWHM of a Voigt profile. http://en.wikipedia.org/wiki/Voigt_profile#The_width_of_the_Voigt_profile

Parameters

- **dD** (*array*) – FWHM of the Gaussian core.
- **dL** (*array*) – FWHM of the Lorentz wings.
- **ddD** (*array*) – Error on the FWHM of the Gaussian.
- **ddL** (*array*) – Error on the FWHM of the Lorentzian.

Returns The FWHM of a Voigt profile.

Return type array

crrlpy.crrls.**voigt_peak**(*A*, *alphaD*, *alphaL*)

Gives the peak of a Voigt profile given its Area and the Half Width at Half Maximum of the Gaussian and Lorentz profiles.

Parameters

- **A** (*array*) – Area of the Voigt profile.
- **alphaD** (*array*) – HWHM of the Gaussian core.
- **alphaL** (*array*) – HWHM of the Lorentz wings.

Returns The peak of the Voigt profile.

Return type array

crrlpy.crrls.**voigt_peak2area**(*peak*, *alphaD*, *alphaL*)

Converts the peak of a Voigt profile into the area under the profile given the Half Width at Half Maximum of the profile components.

Parameters

- **peak** (*array*) – Peak of the Voigt profile.
- **alphaD** (*array*) – HWHM of the Gaussian core.
- **alphaL** (*array*) – HWHM of the Lorentz wings.

Returns The area under the Voigt profile.

Return type array

crrlpy.crrls.**voigt_peak_err**(*peak*, *A*, *dA*, *alphaD*, *dalphaD*)

Gives the error on the peak of the Voigt profile. It assumes no correlation between the parameters and that they are normally distributed.

Parameters

- **peak** (*array*) – Peak of the Voigt profile.
- **A** – Area under the Voigt profile.
- **dA** (*array*) – Error on the area *A*.
- **alphaD** (*array*) – HWHM of the Gaussian core.

CHAPTER

THREE

CRRLPY.SPEC MODULE

Module dedicated to the processing of RRL spectra.

class crrlpy.spec.**Spectrum**(*x*, *y*, *w=[]*, *spw=None*, *stack=None*)

Bases: object

Construction:

spec = Spectrum(x, y, w=None, spw=None, stack=None)

Parameters **x** : array_like

Spectrum x axis.
If x is a frequency axis, then its units should be MHz.

y : array_like

Spectrum y axis.

w : array_like, optional

Spectrum weight axis

spw : int

Spectrum spectral window.

Methods

apply_bandpass_corr(bandpass_x, bandpass_y)	Applies a bandpass correction to the y axis of Spectrum.
bandpass_corr(order[, offset])	Fits a polynomial to the unmasked elements of spec and uses it to correct the
find_good_lines(line, lines[, z, ...])	Find any good lines in the Spectrum.
find_lines(line[, z, verbose])	Finds if there are any lines of a given type in the frequency range.
make_line_mask(line[, z, df])	Creates a list of indexes to mask lines in the spectrum.
mask_edges(redge[, ledge])	Mask the edges of the Spectrum frequency and amplitude.
mask_ranges(ranges)	Masks the Spectrum inside the given ranges.
remove_model(line, model[, z, is_freq])	Subtracts the model from the y axis of the Spectrum.
save(filename)	Saves the spectrum.
split_lines(reffreqs)	Splits the spectrum to separate lines.

apply_bandpass_corr(*bandpass_x*, *bandpass_y*, *offset=1000.0*, *overwrite=False*)

Applies a bandpass correction to the y axis of Spectrum.

Parameters **bandpass** : numpy.ma.array

Bandpass to apply to Spectrum.

offset : `float`, optional

Offset to apply to the y axis data. Used to avoid division by 0.
Should be the same value used when deriving the bandpass solution.

overwrite : `bool`, optional

Should the bandpass applied overwrite the Spectrum.bp?

bandpass_corr(*order*, *offset=1000.0*)
Fits a polynomial to the unmasked elements of spec and uses it to correct the bandpass.

Parameters **order** : `int`

Order of the polynomial to be fit.

offset : `float`, optional

Offset to apply to the y axis data.
Used to avoid division by 0.

find_good_lines(*line*, *lines*, *z=0*, *separation=0*, *redge=0.05*, *ledge=None*)
Find any good lines in the Spectrum.

Parameters **line** : `str`

Search for good lines of this kind.

lines : `str`

Compare against this kind of lines.

z : `float`

Redshift correction to apply to the rest frequencies.

separation : `float`

Minimum separation between lines to be considered good.

redge : `float`

The line frequency should be this far

find_lines(*line*, *z=0*, *verbose=False*)
Finds if there are any lines of a given type in the frequency range. The line frequencies are corrected for redshift.

Parameters **line** : `string`

Line type to search for.

z : `float`

Redshift to apply to the rest frequencies.

verbose : `bool`

Verbose output?

Returns **n** : `numpy.array`

Principal quantum numbers.

reference_frequencies : `numpy.array`

Reference frequencies of the lines inside the spectrum in MHz. The frequencies are redshift corrected.

See also:

crrlpy.crrls.load_ref Describes the format of line and the available ones.

Examples

```
>>> from crrlpy.spec import Spectrum
>>> freq = [10, 11]
>>> temp = [1, 1]
>>> spec = Spectrum(freq, temp)
>>> ns, rf = spec.find_lines('RRL_CIalpha')
>>> ns
array([ 843.,  844.,  845.,  846.,  847.,  848.,  849.,  850.,  851.,
        852.,  853.,  854.,  855.,  856.,  857.,  858.,  859.,  860.,
        861.,  862.,  863.,  864.,  865.,  866.,  867.,  868.,  869.])
```

make_line_mask (*line*, *z=0*, *df=5*)
Creates a list of indexes to mask lines in the spectrum.

Parameters **line** : str

Line to mask.

See also:

crrlpy.crrls.load_ref Describes the format of line and the available ones.

mask_edges (*redge*, *ledge=None*)
Mask the edges of the Spectrum frequency and amplitude.

mask_ranges (*ranges*)
Masks the Spectrum inside the given ranges.

Parameters **ranges** : list of tuples

Indexes defining the ranges to be masked.

Examples

```
>>> x = np.arange(0,10)
>>> y = np.arange(0,10)
>>> spec = Spectrum(x,y)
>>> spec.y.compressed()
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> rngs = [[0,2], [7,9]]
>>> spec.mask_ranges(rngs)
>>> spec.y.compressed()
array([2, 3, 4, 5, 6, 9])
```

remove_model (*line*, *model*, *z=0*, *is_freq=False*)
Subtracts the model from the y axis of the Spectrum.

Parameters **line** : str

Line which the model represents.

model : array_like

Array with the model to be removed. Should have shape (2xN).

is_freq : bool

Is the model x axis in frequency units?

save (*filename*)
Saves the spectrum.

Parameters **filename** : str

Output filename.

split_lines (*reffreqs*)
Splits the spectrum to separate lines.

Parameters **lines** : array_like

List of lines to split.

class crrlpy.spec.**Stack** (*line*, *transitions*, *specs*, *vmin=-100*, *vmax=100*, *dv=0*)
Bases: object

Construction:

stack = Stack(line, transitions, specs)

Parameters **line** : str

Line contained in the spectrum.

transitions : array_like

Transitions inside the Stack.

specs : array of *Spectrum*

Array with Spectrum objects to be stacked.

Methods

compute_dv()	Determines the maximum velocity separation between the channels inside Stack.specs.
save(filename)	Saves the spectrum.
stack_interpol([rms, ch0, chf])	Stack by interpolating to a common grid.

compute_dv ()
Determines the maximum velocity separation between the channels inside Stack.specs.

save (*filename*)
Saves the spectrum.

Parameters **filename** : str

Output filename.

stack_interpol (*rms=True*, *ch0=0*, *chf=-1*)
Stack by interpolating to a common grid.

Parameters **rms** : bool

Compute a the rms for each substack?

ch0 : int

First channel used to compute the rms.

chf : int

Last channel used to compute the rms.

crrlpy.spec.**distribute_lines**(*lines*, *ngroups*)

Groups a list of lines into ngroups. This is used to make stacks.

Parameters **lines** : array_like

List of lines to group.

ngroups : **:ob:'int'**

Number of groups to make. If the number of lines is not divisible by the number of groups, then more lines are added to the first groups.

CHAPTER

FOUR

CRRLPY.MODELS.RRLMOD MODULE

Module with RRL models.

Models are not shipped with the code.

crrlpy.models.rrlmod.**G_CII**(*Tbg*)

crrlpy.models.rrlmod.**I_Bnu**(*specie*, *Z*, *n*, *Inu_funct*, **args*)

Calculates the product $B_{n+\Delta n,n}I_\nu$ to compute the line broadening due to a radiation field I_ν.

Parameters

- **specie** (*string*) – Atomic specie to calculate for.
- **n** (*int or list*) – Principal quantum number at which to evaluate $\frac{2}{\pi}\sum_{\Delta n} B_{n+\Delta n,n}I_{n+\Delta n,n}(\nu)$.
- **Inu_funct** (*function*) – Function to call and evaluate $I_{n+\Delta n,n}(\nu)$. It's first argument must be the frequency.
- **args** – Arguments to *Inu_funct*. The frequency must be left out. The frequency will be passed internally in units of MHz. Use the same unit when required. *Inu_funct* must take the frequency as first parameter.

Returns (Hz)

Return type array

Example

```
>>> I_Bnu('CI', 1.,  500, I_broken_plaw, 800, 26*u.MHz.to('Hz'), -1., -2.6)
array([ 6.65540582])
```

crrlpy.models.rrlmod.**I_CII**(*T158*, *R*, *NCII*)

Frequency integrated line intensity. Optically thin limit without radiative transfer.

Returns The frequency integrated line intensity in units of Jy Hz or g s-3

crrlpy.models.rrlmod.**I_CII_rt**(*wav*, *dnu*, *T158*)

Frequency integrated line intensity. Optically thin limit with radiative transfer.

Returns The frequency integrated line intensity in units of Jy Hz or g s-3

crrlpy.models.rrlmod.**I_broken_plaw**(*nu*, *Tr*, *nu0*, *alpha1*, *alpha2*)

Returns the blackbody function evaluated at nu. As temperature a broken power law is used. The power law shape has parameters: Tr, nu0, alpha1 and alpha2.

Parameters

- **nu** ((Hz) or astropy.units.Quantity) – Frequency. (Hz) or astropy.units.Quantity

- **Tr** – Temperature at nu0. (K) or astropy.units.Quantity
- **nu0** – Frequency at which the spectral index changes. (Hz) or astropy.units.Quantity
- **alpha1** – spectral index for $\nu < \nu_0$
- **alpha2** – spectral index for $\nu \geq \nu_0$

Returns Specific intensity in $\mathrm{erg\,cm^{-2}\,Hz^{-1}\,s^{-1}\,sr^{-1}}$. See astropy.analytic_functions.blackbody.blackbody_nu

Return type astropy.units.Quantity

crrlpy.models.rrlmod.**I_cont**(*nu*, *Te*, *tau*, *I0*, *unitless=False*)

Computes the specific intensity due to a blackbody at temperature T_e and optical depth τ. It considers that there is background radiation with I_0.

Parameters

- **nu** ((Hz) or astropy.units.Quantity) – Frequency.
- **Te** – Temperature of the source function. (K) or astropy.units.Quantity
- **tau** – Optical depth of the medium.
- **I0** – Specific intensity of the background radiation. Must have units of erg / (cm2 Hz s sr) or see *unitless*.
- **unitless** – If True the return

Returns The specific intensity of a ray of light after traveling in an LTE medium with source function $B_\nu(T_e)$ after crossing an optical depth τ_ν. The units are erg / (cm2 Hz s sr). See astropy.analytic_functions.blackbody.blackbody_nu

crrlpy.models.rrlmod.**I_external**(*nu*, *Tbkg*, *Tff*, *tau_ff*, *Tr*, *nu0=<Quantity 100000000.0 MHz>*, *alpha=-2.6*)

This method is equivalent to the IDL routine

Parameters **nu** – Frequency. (Hz) or astropy.units.Quantity

crrlpy.models.rrlmod.**I_total**(*nu*, *Te*, *tau*, *I0*, *eta*)

crrlpy.models.rrlmod.**K_CII**(*Tkin*)

crrlpy.models.rrlmod.**R_CII**(*ne*, *nh*, *gamma_e*, *gamma_h*)

Ratio between the fine structure level population of CII, and the level population in LTE. It ignores the effect of collisions with molecular hydrogen.

crrlpy.models.rrlmod.**R_CII_FS**(*ne*, *nh*, *gamma_e*, *gamma_h*)

crrlpy.models.rrlmod.**R_CII_mod**(*ne*, *nh*, *gamma_e*, *gamma_h*)

Ratio between the fine structure level population of CII, and the level population in LTE. It ignores the effect of collisions with molecular hydrogen.

crrlpy.models.rrlmod.**T_CII**(*Tex*, *tau*, *R*)

crrlpy.models.rrlmod.**T_CII_mod**(*Te*, *tau*, *R*)

crrlpy.models.rrlmod.**T_ex_CII**(*Te*, *R*)

crrlpy.models.rrlmod.**Ta_CII**(*X*, *G*, *K*)

crrlpy.models.rrlmod.**Ta_CII_thick_subthermal**(*X*, *G*, *K*)

crrlpy.models.rrlmod.**X_CII**(*Cul*, *beta*)

crrlpy.models.rrlmod.**alpha_CII**(*Te*, *R*)

Computes the value of $\alpha_{1/2}$. Sorochenko & Tsivilev (2000).

crrlpy.models.rrlmod.**alpha_CII_mod**(*Te*, *R*)
Computes the value of $\alpha_{1/2}$. Sorochenko & Tsivilev (2000).

crrlpy.models.rrlmod.**beta_CII**(*Te*, *R*)
Computes the value of β_{158}. Sorochenko & Tsivilev (2000).

crrlpy.models.rrlmod.**beta_CII_mod**(*Te*, *R*)
Computes the value of β_{158}. Sorochenko & Tsivilev (2000).

crrlpy.models.rrlmod.**bnbeta_approx**(*n*, *Te*, *ne*, *Tr*)
Approximates $b_n\beta_{n'n}$ for a particular set of conditions. Uses Equation (B1) of Salas et al. (2016).

Parameters

- **n** (*int*) – Principal quantum number
- **Te** (*float*) – Electron temperature in K.
- **ne** (*float*) – Electron density per cubic centimeters.
- **Tr** (*float*) – Temperature of the radiation field in K at 100 MHz.

Returns The value of $b_n\beta_{n'n}$ given an approximate expression.

Return type float

crrlpy.models.rrlmod.**bnbeta_approx_full**(*Te*, *ne*, *Tr*, *coefs*)
Approximates $b_n\beta_{n'n}$ given a set of coefficients. Uses Equations (5) and (B1)-(B5) of Salas et al. (2016).

crrlpy.models.rrlmod.**broken_plaw**(*nu*, *nu0*, *T0*, *alpha1*, *alpha2*)
Defines a broken power law.

$$T(\nu) = T_0 \left(\frac{\nu}{\nu_0}\right)^{\alpha_1} \text{ if } \nu < \nu_0$$
$$T(\nu) = T_0 \left(\frac{\nu}{\nu_0}\right)^{\alpha_2} \text{ if } \nu \geq \nu_0$$

Parameters

- **nu** – Frequency.
- **nu0** – Frequency at which the power law breaks.
- **T0** – Value of the power law at nu0.
- **alpha1** – Index of the power law for nu<nu0.
- **alpha2** – Index of the power law for nu>=nu0.

Returns Broken power law evalueated at nu.

crrlpy.models.rrlmod.**chi**(*n*, *Te*, *Z*)
Computes the χ_n value as defined by Salgado et al. (2015).

crrlpy.models.rrlmod.**eta**(*freq*, *Te*, *ne*, *nion*, *Z*, *Tr*, *trans*, *n_max=1500*)
Returns the correction factor for the Planck function.

crrlpy.models.rrlmod.**gamma_e_CII**(*Te*, *method='FS'*)
Computes the de-excitation rate of the CII atom due to collisions with electrons.

Parameters **Te** (*float*) – Electron temperature.

Returns The collisional de-excitation rate in units of cm-3 s-1.

Return type float

crrlpy.models.rrlmod.**gamma_h2_CII**(*Te*)

crrlpy.models.rrlmod.**gamma_h_CII**(*Te*, *method='FS'*)
Computes the de-excitation rate of the CII atom due to collisions with hydrogen atoms.

Parameters **Te** (*float*) – Electron temperature.

Returns The collisional de-excitation rate in units of cm-3 s-1.

Return type float

crrlpy.models.rrlmod.**itau**(*temp*, *dens*, *line*, *n_min=5*, *n_max=1000*, *other=''*, *verbose=False*, *value='itau'*)
Gives the integrated optical depth for a given temperature and density. It assumes that the background radiation field dominates the continuum emission. The emission measure is unity. The output units are Hz.

Parameters

- **temp** (*string*) – Electron temperature. Must be a string of the form '8d1'.
- **dens** (*float*) – Electron density.
- **line** (*string*) – Line to load models for.
- **n_min** (*int*) – Minimum n value to include in the output. Default 1
- **n_max** (*int*) – Maximum n value to include in the output. Default 1500, Maximum allowed value 9900
- **other** (*string*) – String to search for different radiation fields and others.
- **verbose** (*bool*) – Verbose output?
- **value** (*string*) – ['itau'|'bbnMdn'|'None'] Value to output. itau will output the integrated optical depth. bbnMdn will output the $\beta_{n,n'}b_n$ times the oscillator strenght $M(\Delta n)$. None will output the $\beta_{n,n'}b_n$ values.

Returns The principal quantum number and its asociated value.

crrlpy.models.rrlmod.**itau_h**(*temp*, *dens*, *trans*, *n_max=1000*, *other=''*, *verbose=False*, *value='itau'*)
Gives the integrated optical depth for a given temperature and density. The emission measure is unity. The output units are Hz.

crrlpy.models.rrlmod.**itau_norad**(*n*, *Te*, *b*, *dn*, *mdn_*)
Returns the optical depth using the approximate solution to the radiative transfer problem.

crrlpy.models.rrlmod.**j_line_lte**(*n*, *ne*, *nion*, *Te*, *Z*, *trans*)

crrlpy.models.rrlmod.**kappa_cont**(*freq*, *Te*, *ne*, *nion*, *Z*)
Computes the absorption coefficient for the free-free process.

crrlpy.models.rrlmod.**kappa_cont_base**(*nu*, *Te*, *ne*, *nion*, *Z*)

crrlpy.models.rrlmod.**kappa_line**(*Te*, *ne*, *nion*, *Z*, *Tr*, *trans*, *n_max=1500*)
Computes the line absorption coefficient for CRRLs between levels n_i and n_f, $n_i > n_f$. This can only go up to $n_{\max}$ 1500 because of the tables used for the Einstein Anm coefficients.

Parameters

- **Te** (*float*) – Electron temperature of the gas. (K)
- **ne** (*float*) – Electron density. (cm^{-3})
- **nion** (*float*) – Ion density. (cm^{-3})
- **Z** (*int*) – Electric charge of the atoms being considered.

- **Tr** (*float*) – Temperature of the radiation field felt by the gas. This specifies the temperature of the field at 100 MHz. (K)
- **trans** (*string*) – Transition for which to compute the absorption coefficient.
- **n_max** (*int<1500*) – Maximum principal quantum number to include in the output.

Returns

Return type array

crrlpy.models.rrlmod.**kappa_line_lte**(*nu*, *Te*, *ne*, *nion*, *Z*, *Tr*, *line*, *n_min=1*, *n_max=1500*)
Returns the line absorption coefficient under LTE conditions.

Parameters

- **nu** (*array*) – Frequency. (Hz)
- **Te** (*float*) – Electron temperature of the gas. (K)
- **ne** (*float*) – Electron density. (cm^{-3})
- **nion** (*float*) – Ion density. (cm^{-3})
- **Z** (*int*) – Electric charge of the atoms being considered.
- **Tr** (*float*) – Temperature of the radiation field felt by the gas. This specifies the temperature of the field at 100 MHz. (K)
- **trans** (*string*) – Transition for which to compute the absorption coefficient.
- **n_max** (*int<1500*) – Maximum principal quantum number to include in the output.

Returns

Return type array

crrlpy.models.rrlmod.**level_pop_lte**(*n*, *ne*, *nion*, *Te*, *Z*)
Returns the level population of level n. The return has units of cm^{-3}.

crrlpy.models.rrlmod.**load_betabn**(*temp*, *dens*, *other=''*, *trans='RRL_CIalpha'*, *verbose=False*)
Loads a model for the CRRL emission.

crrlpy.models.rrlmod.**load_betabn_h**(*temp*, *dens*, *other=''*, *trans='alpha'*, *verbose=False*)
Loads a model for the HRRL emission.

crrlpy.models.rrlmod.**load_bn**(*temp*, *dens*, *other=''*)
Loads the bn values from the CRRL models.

crrlpy.models.rrlmod.**load_bn2**(*Te*, *ne*, *Tr=''*, *n_min=5*, *n_max=1000*, *verbose=False*)
Loads the bn values from the CRRL models.

Parameters

- **Te** (*string*) – Electron temperature of the model.
- **ne** (*string*) – Electron density of the model.
- **Tr** (*string*) – Radiation field of the model.
- **verbose** (*bool*) – Verbose output?

Returns The b_n value for the given model conditions.

Return type array

crrlpy.models.rrlmod.**load_bn_all**(*n_min=5*, *n_max=1000*, *verbose=False*)

crrlpy.models.rrlmod.**load_bn_dict**(*dict*, *n_min=5*, *n_max=1000*, *verbose=False*)

Loads the b_n values defined by dict.

Parameters

- **dict** (*dict*) – Dictionary containing a list with values for Te, ne and Tr.
- **line** (*string*) – Which models should be loaded.
- **n_min** (*int*) – Minimum n number to include in the output.
- **n_max** (*int*) – Maximum n number to include in the output.
- **verbose** (*bool*) – Verbose output?

Returns List with the b_n values for the conditions defined by dict.

Return type numpy.array

Example

```
>>> from crrlpy.models import rrlmod
```

First define the range of parameters

```
>>> Te = np.array(['1d1', '2d1', '3d1', '4d1', '5d1'])
>>> ne = np.arange(0.01,0.105,0.01)
>>> Tr = np.array([800])
```

Put them in a dictionary

```
>>> models = {'Te':[t_ for t_ in Te for n_ in ne for tr_ in Tr], 'ne':[round(n
```

Load the models

```
>>> bn = rrlmod.load_bn_dict(models, n_min=200, n_max=500, verbose=False)
```

crrlpy.models.rrlmod.**load_itau_all**(*line='RRL_CIalpha'*, *n_min=5*, *n_max=1000*, *verbose=False*, *value='itau'*)

Loads all the available models for Carbon.

Parameters

- **line** (*string*) – Which models should be loaded.
- **n_min** (*int*) – Minimum n number to include in the output.
- **n_max** (*int*) – Maximum n number to include in the output.
- **verbose** (*bool*) – Verbose output?
- **value** (*string*) – ['itau'|'bbnMdn'|None] Which value should be in the output.

crrlpy.models.rrlmod.**load_itau_all_hydrogen**(*trans='alpha'*, *n_max=1000*, *verbose=False*, *value='itau'*)

Loads all the available models for Hydrogen.

crrlpy.models.rrlmod.**load_itau_all_match**(*trans_out='alpha'*, *trans_tin='beta'*, *n_max=1000*, *verbose=False*, *value='itau'*)

Loads all trans_out models that can be found in trans_tin. This is useful when analyzing line ratios.

crrlpy.models.rrlmod.**load_itau_all_norad**(*trans='alpha'*, *n_max=1000*)

Loads all the available models.

crrlpy.models.rrlmod.**load_itau_dict**(*dict*, *line*, *n_min=5*, *n_max=1000*, *verbose=False*, *value='itau'*)

Loads the models defined by dict.

Parameters

- **dict** (*dict*) – Dictionary containing a list with values for Te, ne and Tr.
- **line** (*string*) – Which models should be loaded.
- **n_min** (*int*) – Minimum n number to include in the output.
- **n_max** (*int*) – Maximum n number to include in the output.
- **verbose** (*bool*) – Verbose output?
- **value** (*string*) – ['itau'|'bbnMdn'|None] Which value should be in the output.

Example

```
>>> from crrlpy.models import rrlmod
```

First define the range of parameters

```
>>> Te = np.array(['1d1', '2d1', '3d1', '4d1', '5d1'])
>>> ne = np.arange(0.01,0.105,0.01)
>>> Tr = np.array([2000])
```

Put them in a dictionary

```
>>> models = {'Te':[t_ for t_ in Te for n_ in ne for tr_ in Tr],
...           'ne':[round(n_,3) for t_ in Te for n_ in ne for tr_ in Tr],
...           'Tr':['case_diffuse_{0}'.format(rrlmod.val2str(tr_))
...                 for t_ in Te for n_ in ne for tr_ in Tr]}
```

Load the models

```
>>> itau_mod = rrlmod.load_itau_dict(models, 'CIalpha', n_min=250, n_max=300,
```

crrlpy.models.rrlmod.**load_itau_nelim**(*temp*, *dens*, *trad*, *trans*, *n_max=1000*, *verbose=False*, *value='itau'*)

Loads models given a temperature, radiation field and an upper limit for the electron density.

crrlpy.models.rrlmod.**load_itau_numpy**(*filename*)

Loads all the models contained in filename.npy

Parameters **filename** : string

Filename with the models.

Returns

——-

crrlpy.models.rrlmod.**load_models**(*models*, *trans*, *n_max=1000*, *verbose=False*, *value='itau'*)

Loads the models in backwards compatible mode. It will sort the models by Te, ne and Tr.

crrlpy.models.rrlmod.**make_betabn**(*line*, *temp*, *dens*, *n_min=5*, *n_max=1000*, *other=''*)

crrlpy.models.rrlmod.**make_betabn2**(*line*, *temp*, *dens*, *n_min=5*, *n_max=1000*, *other=''*)

crrlpy.models.rrlmod.**mdn**(*dn*)

Gives the $M(\Delta n)$ factor for a given Δn. ref. Menzel (1968)

Parameters **dn** – Δn.

Returns $M(\Delta n)$

Return type float

Example

```
>>> mdn(1)
0.1908
>>> mdn(5)
0.001812
```

crrlpy.models.rrlmod.**models_dict**(*Te*, *ne*, *Tr*)
Creates a dict for loading models given arrays with ne, Te and Tr.

crrlpy.models.rrlmod.**plaw**(*x*, *x0*, *y0*, *alpha*)
Returns a power law.

$$y(x) = y_0 \left(\frac{x}{x_0} \right)^{\alpha}$$

Parameters

- **x** (*float or array like*) – x values for which to compute the power law.
- **x0** (*float*) – x value for which the power law has amplitude *y0*.
- **y0** (*float*) – Amplitude of the power law at *x0*.
- **alpha** (*float*) – Index of the power law.

Returns A power law of index *alpha* evaluated at *x*, with amplitude *y0* at *x0*.

Return type float or array

crrlpy.models.rrlmod.**str2val**(*str*)
Converts a string representing a number to a float. The string must follow the IDL convention for floats.

Parameters **str** (*string*) – String to convert.

Returns The equivalent number.

Return type float

Example

```
>>> str2val('2d2')
200.0
```

crrlpy.models.rrlmod.**tau_CII**(*Te*, *nc*, *L*, *dnu*, *alpha*, *beta*)
Computes the optical depth of the far infrared line of CII. Crawford et al. (1985).

Parameters

- **Te** (*float*) – Electron temperature.
- **nc** (*float*) – Ionized carbon number density.
- **L** (*float*) – Path lenght.
- **dnu** (*float*) – Line width FWHM in Hz.

Returns

Return type float

crrlpy.models.rrlmod.**tau_CII_PG**(*ncii*, *dv*, *Te*)
Optical depth of the far infrared line of [CII] following Goldsmith et al. (2012).

crrlpy.models.rrlmod.**val2str**(*val*)
Converts a float to the string format required for loading the CRRL models.

Parameters **val** (*float*) – Value to convert to a string.

Returns The value of val represented as a string in IDL double format.

Return type string

Example

```
>>> val2str(200)
'2d2'
```

crrlpy.models.rrlmod.**valid_ne**(*line*)

Checks all the available models and lists the available ne values.

CHAPTER

FIVE

CRRLPY.FREC_CALC MODULE

crrlpy.frec_calc.**line_freq**(*Z*, *R_X*, *n*, *dn*)

Uses the Rydberg formula to get the frequency of a transition to quantum number n for a given atom.

Parameters

- **Z** (*int*) – Charge of the atom.
- **R_X** (*float*) –
- **n** (*int*) – Principal quantum number of the transition. $n + \Delta n \rightarrow n$.
- **dn** (*int*) – Difference between the principal quantum number of the initial state and the final state. $\Delta n = n_f - n_i$.

Returns The frequency of the transition in MHz.

Return type float

crrlpy.frec_calc.**main**()

Main body of the program. Useful for calling as a script.

crrlpy.frec_calc.**make_line_list**(*line*, *n_min=1*, *n_max=1500*, *unitless=True*)

Creates a list of frequencies for the corresponding line. The frequencies are in MHz.

Parameters

- **line** (*string*) – Line to compute the frequencies for.
- **n_min** (*int*) – Minimum n number to include in the list.
- **n_max** (*int*) – Maximum n number to include in the list.
- **unitless** (*bool*) – If True the list will have no units. If not the list will be of astropy.units.Quantity objects.

Returns 3 lists with the line name, principal quantum number and frequency of the transitions.

Return type list

crrlpy.frec_calc.**set_dn**(*name*)

Sets the value of Delta n depending on the transition name.

Parameters **name** (*string*) – Name of the transition.

Returns Δn for the given transition.

Return type int

Example

```
>>> set_dn('CIalpha')
1
>>> set_dn('CIdelta')
4
```

crrlpy.frec_calc.**set_specie**(*specie*)

Sets atomic constants based on the atomic specie.

Parameters **specie** (*string*) – Atomic specie.

Returns Array with the atomic mass in a.m.u., ionization potential, abundance relative to HI, $V_X - V_H$ and the electric charge.

Example

```
>>> set_specie('CI')
[12.0, 11.4, 0.0003, 149.5, 1.0]
```

crrlpy.frec_calc.**set_trans**(*dn*)

Sets a name depending on the difference between atomic levels.

Parameters **dn** (*int*) – Separation between n_i and n_f, $\Delta n = n_i - n_f$.

Returns alpha, beta, gamma, delta or epsilon depending on Δn.

Return type string

Example

```
>>> set_trans(5)
'epsilon'
```

CHAPTER

SIX

CRRLPY.UTILS MODULE

Some utilities for python.

crrlpy.utils.**alphanum_key**(*s*)

Turn a string into a list of string and number chunks.

Parameters **s** – String

Returns List with strings and integers.

Return type list

Example

```
>>> alphanum_key('z23a')
['z', 23, 'a']
```

crrlpy.utils.**best_match_indx**(*value*, *array*)

Searchs for the index of the closest entry to value inside an array.

Parameters

- **value** (*float*) – Value to find inside the array.
- **array** (*list or numpy.array*) – List to search for the given value.

Returns Best match index for the value inside array.

Return type float

Example

```
>>> a = [1,2,3,4]
>>> best_match_indx(3, a)
2
```

crrlpy.utils.**factors**(*n*)

Decomposes a number into its factors. :param n: Number to decompose. :type n: int :return: List of values into which n can be decomposed. :rtype: list

crrlpy.utils.**flatten_list**(*list*)

Flattens a list of lists.

Based on:

http://stackoverflow.com/questions/457215/comprehension-for-flattening-a-sequence-of-sequences/5330178#5330178

crrlpy.utils.**get_max_sep**(*array*)

Get the maximum element separation in an array.

Parameters **array** : array

Array where the maximum separation is wanted.

Returns **max_sep** : float

The maximum separation between the elements in *array*.

Examples

```
>>> import numpy as np
>>> x = np.array([1,2,3,4,5,7])
>>> get_max_sep(x)
2
```

crrlpy.utils.**get_min_sep**(*array*)

Get the minimum element separation in an array.

Parameters **array** : array

Array where the minimum separation is wanted.

Returns **max_sep** : float

The minimum separation between the elements in *array*.

Examples

```
>>> import numpy as np
>>> x = np.array([1,2,3,4,5,7])
>>> get_min_sep(x)
1
```

crrlpy.utils.**myround**(*x*, *base=5*)

crrlpy.utils.**natural_sort**(*list*)

Sort the given list in the way that humans expect. Sorting is done in place.

Parameters **list** (*list*) – List to sort.

Example

```
>>> my_list = ['spec_3', 'spec_4', 'spec_1']
>>> natural_sort(my_list)
>>> my_list
['spec_1', 'spec_3', 'spec_4']
```

crrlpy.utils.**pow_notation**(*number*, *sig_fig=2*)

Converts a number to scientific notation keeping sig_fig signitifcant figures.

crrlpy.utils.**sci_notation**(*number*, *sig_fig=2*)

Converts a number to scientific notation keeping sig_fig signitifcant figures.

crrlpy.utils.**str2bool**(*str*)

Converts a string to a boolean value. The conversion is case insensitive.

Parameters **str** (*string*) – string to convert.

Returns True if str is one of: “yes”, “y”, “true”, “t” or “1”.

Return type bool

crrlpy.utils.**text_slope_match_line**(*text*, *x*, *y*, *line*, *dindx=1*)

crrlpy.utils.**tryint**(*str*)
Returns an integer if *str* can be represented as one.

Parameters **str** (*string*) – String to check.

Returns True is str can be cast to an int.

Return type int

crrlpy.utils.**update_text_slopes**()

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search

c

r

J

K

L

M

N

P

R

S

T

U

V

X