CRRLpy Documentation

Release 0.1

Pedro Salas

October 27, 2015

CONTENTS

Contents:

CHAPTER

ONE

CRRLPY

Tools for processing CRRL spectra.

The models are not shipped with the modules and scripts.

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

1.1 crrlpy.crrls module

crrlpy.crrls.**FWHM2sigma**(*fwhm*)
Converts a FWHM to the standard deviation of a Gaussian distribution.

crrlpy.crrls.**Gauss**(*y*, ***kwargs*)
Applies a Gaussian filter to y.

crrlpy.crrls.**Gaussian**(*x*, *sigma*, *center*, *amplitude*)
1-d Gaussian with no amplitude offset.

crrlpy.crrls.**SavGol**(*y*, ***kwargs*)

crrlpy.crrls.**Voigt**(*x*, *sigma*, *gamma*, *center*, *amplitude*)
The Voigt line shape in terms of its physical parameters x: independent variable sigma: HWHM of the Gaussian gamma: HWHM of the Lorentzian center: the line center amplitude: the line area

crrlpy.crrls.**Wiener**(*y*, ***kwargs*)

crrlpy.crrls.**alphanum_key**(*s*)

Turn a string into a list of string and number chunks. "z23a" -> ["z", 23, "a"]

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

crrlpy.crrls.**best_match_indx**(*value*, *array*, *tol*)
Searchs for the best match to a value inside an array given a tolerance. @param value - value to find inside the array @type value - float @param tol - tolerance for match @type tol - float @param array - list to search for the given value @type array - numpy.array @return - best match for val inside array @rtype - float

crrlpy.crrls.**best_match_indx2**(*value*, *array*)

crrlpy.crrls.**best_match_value**(*value*, *array*)

crrlpy.crrls.**blank_lines**(*freq*, *tau*, *reffreqs*, *v0*, *dv0*)

crrlpy.crrls.**blank_lines2**(*freq*, *tau*, *reffreqs*, *dv*)

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

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

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

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

crrlpy.crrls.**f2n**(*f*, *line*, *n_max=1500*)
Comverts a given frequency to a principal quantum number n of a given transition and atomic specie.

crrlpy.crrls.**find_lines_in_band**(*freq*, *species='CI'*, *transition='alpha'*, *z=0*, *verbose=False*)
Finds if there are any lines corresponding to transitions of the given species in the frequency range. The line transition frequencies are corrected for redshift.

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

crrlpy.crrls.**fit_continuum**(*x*, *y*, *degree*, *p0*)
Divide tb by given a model and starting parameters p0. Returns: tb/model - 1

crrlpy.crrls.**fit_line**(*sb*, *n*, *ref*, *vel*, *tau*, *rms*, *model*, *v0=None*, *verbose=True*)

crrlpy.crrls.**fit_storage**()
Returns a dictionary with the entries for the parameters to be fitted.

crrlpy.crrls.**freq2vel**(*f0*, *f*)
Convert a frequency axis to a velocity axis given a central frequency. Uses the radio definition of velocity.

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

crrlpy.crrls.**gauss_area_err**(*amplitude*, *amplitude_err*, *sigma*, *sigma_err*)

crrlpy.crrls.**gaussian_off**(*x*, *amplitude*, *center*, *sigma*, *c*)
1-d Gaussian with a constant amplitude offset.

crrlpy.crrls.**get_axis**(*header*, *axis*)
Constructs a cube axis @param header - fits cube header @type header - pyfits header @param axis - axis to reconstruct @type axis - int @return - cube axis @rtype - numpy array

crrlpy.crrls.**get_line_mask**(*freq*, *reffreq*, *v0*, *dv0*)
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.

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.

crrlpy.crrls.**get_min_sep**(*array*)
Get the minimum element separation in an array.

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

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

crrlpy.crrls.**line_width**(*dD*, *dL*)
http://en.wikipedia.org/wiki/Voigt_profile#The_width_of_the_Voigt_profile

crrlpy.crrls.**line_width_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

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

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

crrlpy.crrls.**load_ref**(*specie*, *trans*)
Loads the reference spectrum for the specified atomic specie and transition. Available species and transitions: CI alpha CI beta CI delta CI gamma CI13 alpha HeI alpha HeI beta HI alpha HI beta SI alpha SI beta

crrlpy.crrls.**load_ref2**(*transition*)
Loads the reference spectrum for the specified atomic specie and transition. Available transitions: CIalpha CIbeta CIdelta CIgamma CI13alpha HeIalpha HeIbeta HIalpha HIbeta SIalpha SIbeta

crrlpy.crrls.**lookup_freq**(*n*, *specie*, *trans*)
Returns the frequency of a given transition.

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).

crrlpy.crrls.**mask_outliers**(*data*, *m=2*)
Masks values larger than m times the data median.

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**(*l*)
Sort the given list in the way that humans expect. Sorting is done in place.

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)

crrlpy.crrls.**pressure_broad_coefs**(*Te*)

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)

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

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.**remove_baseline**(*freq*, *tb*, *model*, *p0*, *mask*)
Divide tb by given a model and starting parameters p0. Returns: tb/model - 1

crrlpy.crrls.**sigma2FWHM**(*sigma*)
Converts the sigma parameter of a Gaussian distribution to its FWHM.

crrlpy.crrls.**sigma2FWHM_err**(*dsigma*)
Converts the error on the sigma parameter of a Gaussian distribution to the error on the FWHM.

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.**sum_line**(*sb*, *n*, *ref*, *vel*, *tau*, *v0*, *tau0*, *dtau0*, *thr*, *rms*)
Integrate the spectrum near a given velocity v0. It stops when the channels are within a threshold from a reference level.

crrlpy.crrls.**sum_storage**()

crrlpy.crrls.**tryint**(*s*)

crrlpy.crrls.**vel2freq**(*f0*, *vel*)
Convert a velocity axis to a frequency axis given a central frequency. Uses the radio definition.

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_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_peak**(*A*, *alphaD*, *alphaL*)
Gives the peak of a Voigt profile given its Area and the HWHM of the Gaussian and Lorentz profiles.

crrlpy.crrls.**voigt_peak2area**(*peak*, *alphaD*, *alphaL*)
Converts the peak of a Voigt profile into the area under the profile given the HWHM of the profile.

crrlpy.crrls.**voigt_peak_err**(*peak*, *A*, *dA*, *alphaD*, *dalphaD*)
Gives the error on the peak of the Voigt profile.

1.2 crrlpy.models.rrlmod module

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** – Atomic specie to calculate for.
- **n** – 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 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.

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 is has parameters: Tr, nu0, alpha1 and alpha2.

Parameters

- **nu** – 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 erg / (cm2 Hz s sr). See astropy.analytic_functions.blackbody.blackbody_nu

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

Parameters

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

Gives the M(dn) factor for a given dn. ref. Menzel (1968)

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

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. The emission measure is unity. The output units are Hz.

Parameters

- **temp** – Electron temperature. Must be a string of the form '8d1'.
- **dens** – Electron density. Float
- **line** – Line to load models for.
- **n_min** – Minimum n value to include in the output. Int Default 1
- **n_max** – Maximum n value to include in the output. Int Default 1500, Maximum allowed value 9900
- **other** – String to search for different radiation fields and others.
- **verbose** – Verbose output? Bool
- **value** – ['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 $M(\Delta n)$. :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 with only 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*)

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 between levels ni and nf, ni>nf. This can only go up to n_max 1500 because of the tables used for the Einstein Anm coefficients.

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

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

crrlpy.models.rrlmod.**load_itau_all**(*trans='CIalpha'*, *n_min=5*, *n_max=1000*, *verbose=False*, *value='itau'*)
Loads all the available models.

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

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.

crrlpy.models.rrlmod.**load_itau_all_norad**(*trans='alpha'*, *n_max=1000*)
Loads all the available models.

crrlpy.models.rrlmod.**load_itau_dict**(*dict*, *trans*, *n_min=5*, *n_max=1000*, *verbose=False*, *value='itau'*)
Loads the models defined by dict.

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_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**(*temp*, *dens*, *trans*, *n_max=1000*, *other=''*)

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

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

crrlpy.models.rrlmod.**str2val**(*str*)

crrlpy.models.rrlmod.**val2str**(*val*)
Converts a float to the str format required for loading the CRRL models. E.g., a temperature of 70 K is 7d1.

crrlpy.models.rrlmod.**valid_ne**(*trans*)
Checks all the available models and lists the available ne values.

crrlpy.models.rrlmod.**xi**(*n*, *Te*, *Z*)

1.3 crrlpy.crrls.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** – Charge of the atom.
- **R_X** –
- **n** – Principal quantum number of the transition. $n + \Delta n \rightarrow n$.
- **dn** – 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** – Line to compute the frequencies for.
- **n_min** – Minimum n number to include in the list.
- **n_max** – Maximum n number to include in the list.
- **unitless** – If True the list will have no units. If not the list will be of astropy.units.Quantity objects.

Returns List with the frequency of the transitions for the line.

crrlpy.frec_calc.**set_dn**(*name*)
Sets the value of Delta n depending on the transition name.

Parameters **name** – 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** – 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.

crrlpy.frec_calc.**set_trans**(*dn*)
Sets a name depending on the difference between atomic levels.

Parameters **dn** – Separation between ni and nf, $\Delta n = n_i - n_f$.

Returns alpha, beta, gamma, delta or epsilon depending on `dn`.

Return type string

CHAPTER

TWO

INDICES AND TABLES

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

C

M

N

P

R

S

T

V

W

X