
CRRLpy Documentation

Release 0.1

Pedro Salas

October 28, 2015

CONTENTS

1	CRRLpy	3
2	crrlpy.crrls module	5
3	crrlpy.models.rrlmod module	11
4	crrlpy.crrls.frec_calc module	17
5	Indices and tables	19
	Python Module Index	21
	Index	23

Contents:

CRRLPY

Tools for processing Carbon **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/>

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.

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_idx` (*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_idx2(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_idx2(3, a)
2
```

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

Searchs for the closest occurrence 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, dv0)`

Blanks the lines in a spectra.

Parameters **freq** (*array*) – Frequency axis of the spectra.

:

`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.

Parameters

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

Returns The equivalent velocity delta for the given frequency delta.

Return type *float in Hz*

`crrlpy.crpls.dv2df(f0, dv)`

Convert a velocity delta to a frequency delta given a central frequency.

Parameters

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

Returns The equivalent frequency delta for the given velocity delta.

Return type float in m s^{-1}

`crrlpy.crpls.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.crpls.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.crpls.f2n(f, line, n_max=1500)`

Converts a given frequency to a principal quantum number n for a given line.

`crrlpy.crpls.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.

Parameters

- **freq** –
- **species** –
- **z** – Redshift to apply to the rest frequencies.
- **verbose** (*bool*) – Verbose output?

Returns

Return type List of principal quantum numbers and list of reference frequencies.

`crrlpy.crpls.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.crpls.fit_continuum(x, y, degree, p0)`

Divide tb by given a model and starting parameters p0. Returns: tb/model - 1

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

`crrlpy.crpls.fit_storage()`

Returns a dictionary with the entries for the parameters to be fitted.

`crrlpy.crpls.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.crpls.gauss_area(amplitude, sigma)`

Returns the area under a Gaussian of a given amplitude and sigma.

`crrlpy.crpls.gauss_area_err(amplitude, amplitude_err, sigma, sigma_err)`

`crrlpy.crpls.gaussian_off(x, amplitude, center, sigma, c)`

1-d Gaussian with a constant amplitude offset.

`crrlpy.crpls.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.crpls.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.

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.
- **dv0** (*float, km/s*) – Velocity range to mask.

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

`crrlpy.crpls.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)`

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.

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

where V is the variance of the data.

`crrlpy.crrls.is_number(s)`

Checks whether 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.crpls.plot_spec_vel` (*out, x, y, fit, A, Aerr, x0, x0err, sx, sxerr*)

`crrlpy.crpls.pressure_broad` (*n, Te, ne*)
Pressure induced broadening in Hz. Shaver (1975)

`crrlpy.crpls.pressure_broad_coefs` (*Te*)

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

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

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

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

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

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

`crrlpy.crpls.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.crpls.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.crpls.sum_storage` ()

`crrlpy.crpls.tryint` (*s*)

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

`crrlpy.crpls.voigt` (*x, y*)

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

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

CRRLPY.MODELS.RRLMOD MODULE

`crrlpy.models.rrlmod.I_Bnu(specie, Z, n, Inu_func, *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** (*str*) – 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_func** (*function*) – Function to call and evaluate $I_{n+\Delta n, n}(\nu)$. It's first argument must be the frequency.
- ***args** – Arguments to *Inu_func*. The frequency must be left out. The frequency will be passed internally in units of MHz. Use the same unit when required. *Inu_func* must take the frequency as first parameter.

Returns

Return type array, Hz

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_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 $\text{erg cm}^{-2} \text{Hz}^{-1} \text{s}^{-1} \text{sr}^{-1}$.
`trypy.analytic_functions.blackbody.blackbody_nu`

See as-

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.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.broken_plaw (nu, nu0, T0, alpha1, alpha2)
```

Defines a broken power law.

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

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 evaluated at nu.

```
crrlpy.models.rrlmod.eta (freq, Te, ne, nion, Z, Tr, trans, n_max=1500)
```

Returns the correction factor for the Planck function.

`crllpy.models.rllmod.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.

`crllpy.models.rllmod.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.

`crllpy.models.rllmod.itau_norad(n, te, b, dn, mdn)`

Returns the optical depth with only the approximate solution to the radiative transfer problem.

`crllpy.models.rllmod.j_line_lte(n, ne, nion, Te, Z, trans)`

`crllpy.models.rllmod.kappa_cont(freq, Te, ne, nion, Z)`

Computes the absorption coefficient for the free-free process.

`crllpy.models.rllmod.kappa_cont_base(nu, Te, ne, nion, Z)`

`crllpy.models.rllmod.kappa_line(Te, ne, nion, Z, Tr, trans, n_max=1500)`

Computes the line absorption coefficient for CRRLs between levels :mat:'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='C1alpha'*, *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_itaui_all` (*trans='C1alpha'*, *n_min=5*, *n_max=1000*, *verbose=False*, *value='itau'*)

Loads all the available models for Carbon.

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

Loads all the available models for Hydrogen.

`crrlpy.models.rrlmod.load_itaui_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_itaui_all_norad` (*trans='alpha'*, *n_max=1000*)

Loads all the available models.

`crrlpy.models.rrlmod.load_itaui_dict` (*dict*, *trans*, *n_min=5*, *n_max=1000*, *verbose=False*, *value='itau'*)

Loads the models defined by *dict*.

`crrlpy.models.rrlmod.load_itaui_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 (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.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)  
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)
```


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** (*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** – 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** (*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 **:paramref:‘dn’**.

Return type *string*

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

C

`crrlpy.crrls`, 5

`crrlpy.frec_calc`, 17

`crrlpy.models.rrlmod`, 11

r

`rrlmod` (*Unix*), 11

A

alphanum_key() (in module crrlpy.crrls), 5
average() (in module crrlpy.crrls), 5

B

best_match_idx() (in module crrlpy.crrls), 5
best_match_idx2() (in module crrlpy.crrls), 6
best_match_value() (in module crrlpy.crrls), 6
blank_lines() (in module crrlpy.crrls), 6
blank_lines2() (in module crrlpy.crrls), 6
broken_plaw() (in module crrlpy.models.rrlmod), 12

C

crrlpy.crrls (module), 5
crrlpy.frec_calc (module), 17
crrlpy.models.rrlmod (module), 11

D

df2dv() (in module crrlpy.crrls), 6
dv2df() (in module crrlpy.crrls), 7
dv_minus_doppler() (in module crrlpy.crrls), 7
dv_minus_doppler2() (in module crrlpy.crrls), 7

E

eta() (in module crrlpy.models.rrlmod), 12

F

f2n() (in module crrlpy.crrls), 7
find_lines_in_band() (in module crrlpy.crrls), 7
find_lines_sb() (in module crrlpy.crrls), 7
fit_continuum() (in module crrlpy.crrls), 8
fit_line() (in module crrlpy.crrls), 8
fit_storage() (in module crrlpy.crrls), 8
freq2vel() (in module crrlpy.crrls), 8
FWHM2sigma() (in module crrlpy.crrls), 5

G

Gauss() (in module crrlpy.crrls), 5
gauss_area() (in module crrlpy.crrls), 8
gauss_area_err() (in module crrlpy.crrls), 8
Gaussian() (in module crrlpy.crrls), 5

gaussian_off() (in module crrlpy.crrls), 8
get_axis() (in module crrlpy.crrls), 8
get_line_mask() (in module crrlpy.crrls), 8
get_line_mask2() (in module crrlpy.crrls), 8
get_min_sep() (in module crrlpy.crrls), 8
get_rms() (in module crrlpy.crrls), 9

I

I_Bnu() (in module crrlpy.models.rrlmod), 11
I_broken_plaw() (in module crrlpy.models.rrlmod), 11
I_cont() (in module crrlpy.models.rrlmod), 11
I_external() (in module crrlpy.models.rrlmod), 12
I_total() (in module crrlpy.models.rrlmod), 12
is_number() (in module crrlpy.crrls), 9
itau() (in module crrlpy.models.rrlmod), 12
itau_h() (in module crrlpy.models.rrlmod), 13
itau_norad() (in module crrlpy.models.rrlmod), 13

J

j_line_lte() (in module crrlpy.models.rrlmod), 13

K

kappa_cont() (in module crrlpy.models.rrlmod), 13
kappa_cont_base() (in module crrlpy.models.rrlmod), 13
kappa_line() (in module crrlpy.models.rrlmod), 13
kappa_line_lte() (in module crrlpy.models.rrlmod), 13

L

level_pop_lte() (in module crrlpy.models.rrlmod), 14
line_freq() (in module crrlpy.frec_calc), 17
line_width() (in module crrlpy.crrls), 9
line_width_err() (in module crrlpy.crrls), 9
linear() (in module crrlpy.crrls), 9
load_betabn() (in module crrlpy.models.rrlmod), 14
load_betabn_h() (in module crrlpy.models.rrlmod), 14
load_bn() (in module crrlpy.models.rrlmod), 14
load_bn2() (in module crrlpy.models.rrlmod), 14
load_itau_all() (in module crrlpy.models.rrlmod), 14
load_itau_all_hydrogen() (in module crrlpy.models.rrlmod), 14
load_itau_all_match() (in module crrlpy.models.rrlmod), 14

load_itaun_all_norad() (in module crrlpy.models.rrlmod), 14
load_itaun_dict() (in module crrlpy.models.rrlmod), 14
load_itaun_nelim() (in module crrlpy.models.rrlmod), 14
load_model() (in module crrlpy.crrls), 9
load_models() (in module crrlpy.models.rrlmod), 14
load_ref() (in module crrlpy.crrls), 9
load_ref2() (in module crrlpy.crrls), 9
lookup_freq() (in module crrlpy.crrls), 9
lorentz_width() (in module crrlpy.crrls), 9

M

main() (in module crrlpy.frec_calc), 17
make_betabn() (in module crrlpy.models.rrlmod), 14
make_betabn2() (in module crrlpy.models.rrlmod), 15
make_line_list() (in module crrlpy.frec_calc), 17
mask_outliers() (in module crrlpy.crrls), 9
Mdn() (in module crrlpy.models.rrlmod), 12

N

n2f() (in module crrlpy.crrls), 9
natural_sort() (in module crrlpy.crrls), 9

P

plaw() (in module crrlpy.models.rrlmod), 15
plot_fit() (in module crrlpy.crrls), 9
plot_fit_single() (in module crrlpy.crrls), 9
plot_model() (in module crrlpy.crrls), 9
plot_spec_vel() (in module crrlpy.crrls), 9
pressure_broad() (in module crrlpy.crrls), 10
pressure_broad_coefs() (in module crrlpy.crrls), 10
pressure_broad_salgado() (in module crrlpy.crrls), 10

R

radiation_broad() (in module crrlpy.crrls), 10
radiation_broad_salgado() (in module crrlpy.crrls), 10
radiation_broad_salgado_general() (in module cr-
rlpy.crrls), 10
remove_baseline() (in module crrlpy.crrls), 10
rrlmod (module), 11

S

SavGol() (in module crrlpy.crrls), 5
set_dn() (in module crrlpy.frec_calc), 17
set_specie() (in module crrlpy.frec_calc), 18
set_trans() (in module crrlpy.frec_calc), 18
sigma2FWHM() (in module crrlpy.crrls), 10
sigma2FWHM_err() (in module crrlpy.crrls), 10
stack_interpol() (in module crrlpy.crrls), 10
stack_irregular() (in module crrlpy.crrls), 10
str2val() (in module crrlpy.models.rrlmod), 15
sum_line() (in module crrlpy.crrls), 10
sum_storage() (in module crrlpy.crrls), 10

T

tryint() (in module crrlpy.crrls), 10

V

val2str() (in module crrlpy.models.rrlmod), 15
valid_ne() (in module crrlpy.models.rrlmod), 15
vel2freq() (in module crrlpy.crrls), 10
Voigt() (in module crrlpy.crrls), 5
voigt() (in module crrlpy.crrls), 10
voigt_area() (in module crrlpy.crrls), 10
voigt_area_err() (in module crrlpy.crrls), 10
voigt_peak() (in module crrlpy.crrls), 10
voigt_peak2area() (in module crrlpy.crrls), 10
voigt_peak_err() (in module crrlpy.crrls), 10

W

Wiener() (in module crrlpy.crrls), 5

X

xi() (in module crrlpy.models.rrlmod), 15