

---

# **CRRLpy Documentation**

***Release 0.1***

**Pedro Salas**

October 20, 2015



## CONTENTS

<b>1</b>	<b>CRRLpy</b>	<b>3</b>
<b>2</b>	<b>crrlpy.crrls module</b>	<b>5</b>
<b>3</b>	<b>crrlpy.models.rrlmod module</b>	<b>9</b>
<b>4</b>	<b>Indices and tables</b>	<b>13</b>
	<b>Python Module Index</b>	<b>15</b>
	<b>Index</b>	<b>17</b>



Contents:



## **CRRLPY**

tools for processing CRRL spectra

The models are not shipped with the modules and scripts.





## 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 frequency 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.crpls.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.crpls.f2n` (*f, line, n\_max=1500*)  
Converts a given frequency to a principal quantum number *n* of a given transition and atomic specie.

`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.

`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.

`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 @param header - fits cube header @type header - pyfits header @param axis - axis to reconstruct @type axis - int @return - cube axis @rtype - 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*.

`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.crpls.get_min_sep` (*array*)  
Get the minimum element separation in an array.

`crrlpy.crpls.get_rms` (*data, axis=None*)

`crrlpy.crpls.is_number` (*s*)  
Checks whether a string is a number or not.

`crrlpy.crpls.line_width` (*dD, dL*)  
[http://en.wikipedia.org/wiki/Voigt\\_profile#The\\_width\\_of\\_the\\_Voigt\\_profile](http://en.wikipedia.org/wiki/Voigt_profile#The_width_of_the_Voigt_profile)

`crrlpy.crpls.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](http://en.wikipedia.org/wiki/Voigt_profile#The_width_of_the_Voigt_profile)

`crrlpy.crpls.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, spparams, 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.

## CRRLPY.MODELS.RRLMOD MODULE

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

### 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\_func** – 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.

`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  $\tau_\nu$ . The units are erg / (cm<sup>2</sup> 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(x, x0, y0, alpha1, alpha2)`

`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, trans, 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.

`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 Ann 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<sup>-3</sup>.

`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_itau_all(trans='C1alpha', n_min=5, n_max=1000, verbose=False, value='itau')`

Loads all the available models.

`crrlpy.models.rrlmod.load_itaue_all_hydrogen` (*trans*='alpha', *n\_max*=1000, *verbose*=False, *value*='itaue')

Loads all the available models.

`crrlpy.models.rrlmod.load_itaue_all_match` (*trans\_out*='alpha', *trans\_tin*='beta', *n\_max*=1000, *verbose*=False, *value*='itaue')

Loads all *trans\_out* models that can be found in *trans\_tin*.

`crrlpy.models.rrlmod.load_itaue_all_norad` (*trans*='alpha', *n\_max*=1000)

Loads all the available models.

`crrlpy.models.rrlmod.load_itaue_dict` (*dict*, *trans*, *n\_min*=5, *n\_max*=1000, *verbose*=False, *value*='itaue')

Loads the models defined by *dict*.

`crrlpy.models.rrlmod.load_itaue_nelim` (*temp*, *dens*, *trad*, *trans*, *n\_max*=1000, *verbose*=False, *value*='itaue')

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*='itaue')

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*)





## INDICES AND TABLES

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



## C

`crrlpy.crrls`, 5

`crrlpy.models.rrlmod`, 9



**A**

alphanum\_key() (in module crrlpy.crrls), 5  
 average() (in module crrlpy.crrls), 5

**B**

best\_match\_indx() (in module crrlpy.crrls), 5  
 best\_match\_indx2() (in module crrlpy.crrls), 5  
 best\_match\_value() (in module crrlpy.crrls), 5  
 blank\_lines() (in module crrlpy.crrls), 5  
 blank\_lines2() (in module crrlpy.crrls), 5  
 broken\_plaw() (in module crrlpy.models.rrlmod), 10

**C**

crrlpy.crrls (module), 5  
 crrlpy.models.rrlmod (module), 9

**D**

df2dv() (in module crrlpy.crrls), 5  
 dv2df() (in module crrlpy.crrls), 5  
 dv\_minus\_doppler() (in module crrlpy.crrls), 5  
 dv\_minus\_doppler2() (in module crrlpy.crrls), 5

**E**

eta() (in module crrlpy.models.rrlmod), 10

**F**

f2n() (in module crrlpy.crrls), 6  
 find\_lines\_in\_band() (in module crrlpy.crrls), 6  
 find\_lines\_sb() (in module crrlpy.crrls), 6  
 fit\_continuum() (in module crrlpy.crrls), 6  
 fit\_line() (in module crrlpy.crrls), 6  
 fit\_storage() (in module crrlpy.crrls), 6  
 freq2vel() (in module crrlpy.crrls), 6  
 FWHM2sigma() (in module crrlpy.crrls), 5

**G**

Gauss() (in module crrlpy.crrls), 5  
 gauss\_area() (in module crrlpy.crrls), 6  
 gauss\_area\_err() (in module crrlpy.crrls), 6  
 Gaussian() (in module crrlpy.crrls), 5  
 gaussian\_off() (in module crrlpy.crrls), 6

get\_axis() (in module crrlpy.crrls), 6  
 get\_line\_mask() (in module crrlpy.crrls), 6  
 get\_line\_mask2() (in module crrlpy.crrls), 6  
 get\_min\_sep() (in module crrlpy.crrls), 6  
 get\_rms() (in module crrlpy.crrls), 6

**I**

I\_Bnu() (in module crrlpy.models.rrlmod), 9  
 I\_broken\_plaw() (in module crrlpy.models.rrlmod), 9  
 I\_cont() (in module crrlpy.models.rrlmod), 9  
 I\_external() (in module crrlpy.models.rrlmod), 10  
 I\_total() (in module crrlpy.models.rrlmod), 10  
 is\_number() (in module crrlpy.crrls), 6  
 itau() (in module crrlpy.models.rrlmod), 10  
 itau\_h() (in module crrlpy.models.rrlmod), 10  
 itau\_norad() (in module crrlpy.models.rrlmod), 10

**J**

j\_line\_lte() (in module crrlpy.models.rrlmod), 10

**K**

kappa\_cont() (in module crrlpy.models.rrlmod), 10  
 kappa\_cont\_base() (in module crrlpy.models.rrlmod), 10  
 kappa\_line() (in module crrlpy.models.rrlmod), 10  
 kappa\_line\_lte() (in module crrlpy.models.rrlmod), 10

**L**

level\_pop\_lte() (in module crrlpy.models.rrlmod), 10  
 line\_width() (in module crrlpy.crrls), 6  
 line\_width\_err() (in module crrlpy.crrls), 6  
 linear() (in module crrlpy.crrls), 6  
 load\_betabn() (in module crrlpy.models.rrlmod), 10  
 load\_betabn\_h() (in module crrlpy.models.rrlmod), 10  
 load\_bn() (in module crrlpy.models.rrlmod), 10  
 load\_bn2() (in module crrlpy.models.rrlmod), 10  
 load\_itau\_all() (in module crrlpy.models.rrlmod), 10  
 load\_itau\_all\_hydrogen() (in module crrlpy.models.rrlmod), 10  
 load\_itau\_all\_match() (in module crrlpy.models.rrlmod), 11  
 load\_itau\_all\_norad() (in module crrlpy.models.rrlmod), 11

load\_ita\_dict() (in module `crrlpy.models.rrlmod`), 11  
load\_ita\_nelim() (in module `crrlpy.models.rrlmod`), 11  
load\_model() (in module `crrlpy.crrls`), 6  
load\_models() (in module `crrlpy.models.rrlmod`), 11  
load\_ref() (in module `crrlpy.crrls`), 7  
load\_ref2() (in module `crrlpy.crrls`), 7  
lookup\_freq() (in module `crrlpy.crrls`), 7  
lorentz\_width() (in module `crrlpy.crrls`), 7

## M

make\_betabn() (in module `crrlpy.models.rrlmod`), 11  
make\_betabn2() (in module `crrlpy.models.rrlmod`), 11  
mask\_outliers() (in module `crrlpy.crrls`), 7  
Mdn() (in module `crrlpy.models.rrlmod`), 10

## N

n2f() (in module `crrlpy.crrls`), 7  
natural\_sort() (in module `crrlpy.crrls`), 7

## P

plaw() (in module `crrlpy.models.rrlmod`), 11  
plot\_fit() (in module `crrlpy.crrls`), 7  
plot\_fit\_single() (in module `crrlpy.crrls`), 7  
plot\_model() (in module `crrlpy.crrls`), 7  
plot\_spec\_vel() (in module `crrlpy.crrls`), 7  
pressure\_broad() (in module `crrlpy.crrls`), 7  
pressure\_broad\_coefs() (in module `crrlpy.crrls`), 7  
pressure\_broad\_salgado() (in module `crrlpy.crrls`), 7

## R

radiation\_broad() (in module `crrlpy.crrls`), 7  
radiation\_broad\_salgado() (in module `crrlpy.crrls`), 7  
radiation\_broad\_salgado\_general() (in module `crrlpy.crrls`), 7  
remove\_baseline() (in module `crrlpy.crrls`), 7

## S

SavGol() (in module `crrlpy.crrls`), 5  
sigma2FWHM() (in module `crrlpy.crrls`), 7  
sigma2FWHM\_err() (in module `crrlpy.crrls`), 7  
stack\_interpol() (in module `crrlpy.crrls`), 7  
stack\_irregular() (in module `crrlpy.crrls`), 7  
str2val() (in module `crrlpy.models.rrlmod`), 11  
sum\_line() (in module `crrlpy.crrls`), 8  
sum\_storage() (in module `crrlpy.crrls`), 8

## T

tryint() (in module `crrlpy.crrls`), 8

## V

val2str() (in module `crrlpy.models.rrlmod`), 11  
valid\_ne() (in module `crrlpy.models.rrlmod`), 11  
vel2freq() (in module `crrlpy.crrls`), 8

Voigt() (in module `crrlpy.crrls`), 5  
voigt() (in module `crrlpy.crrls`), 8  
voigt\_area() (in module `crrlpy.crrls`), 8  
voigt\_area\_err() (in module `crrlpy.crrls`), 8  
voigt\_peak() (in module `crrlpy.crrls`), 8  
voigt\_peak2area() (in module `crrlpy.crrls`), 8  
voigt\_peak\_err() (in module `crrlpy.crrls`), 8

## W

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

## X

xi() (in module `crrlpy.models.rrlmod`), 11