hera_stats Documentation

Release 0.1

HERA Collaboration

Mar 01, 2020

Contents:

The `hera_stats` module provides a collection of functions and container objects to help calculate various statistics on sets of delay spectra, and manage splits and other aspects of null tests.

You can find the code in the `hera_stats` GitHub repository.

CHAPTER 1

Jupyter notebook automation

The `hera_stats.automate` module contains functions to replace placeholder variables in template Jupyter notebooks (`jupyter_replace_tags`) and run them programmatically (`jupyter_run_notebook`).

`hera_stats.automate.`**`jupyter_replace_tags`**(*fname_tmpl*, *replace*, *outfile=None*, *overwrite=False*, *verbose=False*)

Search through a Jupyter notebook file for tagged lines, replace them with new values, and then save into a new Jupyter notebook.

Tags work in a simple way: If any line in the notebook has an in-line comment of the form '# @tag', and a key 'tag' exists in the 'replace' dict, the entire line will be replaced with 'tag = value', where 'value' is the value of 'replace[key]', which is assumed to be a string. Any expression that was on that line will be completely replaced.

Parameters

- **fname_tmpl** (*str*) – Filename of input notebook that contains tags to be replaced.
- **replace** (*dict*) – Dictionary of tag:value pairs. The values will all be inserted as Python strings, and so the code in the Jupyter notebook should be prepared to do a type conversion if necessary.
- **outfile** (*str, optional*) – Filename to save the tag-replaced notebook to. If not specified, the updated JSON dict will be returned from this function. Default: None.
- **overwrite** (*bool, optional*) – If outfile is not None, whether to overwrite a notebook file if one with the same filename already exists. Default: False.
- **verbose** (*bool, optional*) – If True, print out tags as they are found. Default: False.

Returns **new_tree** – If outfile=None, a dict containing the updated JSON data for the notebook is returned.

Return type JSON dict, optional

`hera_stats.automate.`**`jupyter_run_notebook`**(*tree=None*, *fname=None*, *outfile=None*, *rundir='.'*, *version=4*, *kernel='python3'*)

Run a Jupyter notebook programatically. The notebook to run can be passed as either a filename or a dict derived from JSON data.

If the notebook experiences an error, a CellExecutionError will be raised. The notebook will still be saved to disk even if it errors though.

Parameters

- **tree** (*dict, optional*) – Dict containing JSON tree representing a Jupyter notebook.
- **fname** (*str, optional*) – Filename of Jupyter notebook to load. Only one of 'tree' and 'fname' should be specified.
- **outfile** (*str, optional*) – File to store Jupyter notebook into after it has run. Default: None (no notebook file will be saved).
- **rundir** (*str, optional*) – Directory to run the script from. Default: '.' (current directory).
- **version** (*int, optional*) – Version of Jupyter notebooks to use.
- **kernel** (*str, optional*) – Name of Jupyter Python kernel to use. Default: 'python3'.

CHAPTER 2

Advanced power spectrum averaging functions

The `hera_stats.average` module contains several advanced averaging functions for power spectra. Currently, this only consists of cumulative averaging in time or baseline-pair (*average_spectra_cumul*) and differencing redundant groups with respect to their average (*redundant_diff*).

`hera_stats.average.`**`average_spectra_cumul`**(*uvp*, *blps*, *spw*, *polpair*, *mode='time'*, *min_samples=1*, *shuffle=False*, *time_avg=True*, *verbose=False*)

Cumulatively average a set of delay spectra as a function of time or blpair. The cumulative average can be performed sequentially, e.g. in time order, or in a random (shuffled) order.

Parameters

- **uvp** (*UVPSpec*) – Set of power spectra to be cumulatively averaged.
- **blps** (*list of ints or tuples*) – List of blpair ints or tuples. Only one redundant set of blpairs should be passed at once.
- **spw, polpair** (*int, str*) – Spectral window ID (integer) and polarization-pair (integer or str) of the power spectra to average.
- **mode** (*str, optional*) – Whether to cumulatively average in time or blpair. The other dimension will be averaged over non-cumulatively. (See the 'time_avg' kwarg below for other behaviors.) Possible options are 'time' or 'blpair'. Default: 'time'.
- **min_samples** (*int, optional*) – Minimum number of samples to allow in the cumulative average. Default: 1.
- **shuffle** (*bool, optional*) – Whether to randomly shuffle the order of the cumulative averaging, or to keep it in order. Default: False.
- **time_avg** (*bool, optional*) – Whether to average over times. This option is only used if mode='blpair'; otherwise it will be ignored. Default: True.
- **verbose** (*bool, optional*) – Whether to print status messages as the cumulative averaging progresses. Default: False.

Returns

- **ps** (*array_like*) – Cumulative averages of delay spectra, in a 2D array of shape (Nsamp, Ndelay), where Nsamp = Ntimes or Nblpairs.
- **dly** (*array_like*) – Power spectrum delay modes (in s).
- **n_samples** (*array_like*) – Number of times or blpairs that went into each average in the ps array.

hera_stats.average.**redundant_diff**(*uvd*, *bls*, *pol*, *return_mean=False*)

Calculate the difference between all baselines in a redundant group and the mean of the redundant group (as a fn. of frequency and time).

Empty (fully flagged) baselines are excluded from the average.

N.B. The averaging does not currently take into account data weights or number of samples.

Parameters

- **uvd** (*UVData*) – UVData object containing the data that will be averaged and differenced.
- **bls** (*list of tuple or int*) – List of baseline tuples or integers to be treated as a group. The mean will be calculated over all
- **pol** (*str*) – Which polarization to extract from the UVData file.
- **return_mean** (*bool, optional*) – If True, return the mean over the redundant group too. Default: False

Returns

- **bls** (*list of baseline tuple or int*) – List of baselines that were kept in the average.
- **diffs** (*list of array_like*) – List of arrays of differences between each baseline and the group mean.
- **mean** (*array_like, optional*) – Mean over data from all non-flagged baselines, as a function of freq. and time. Only retuend if *return_mean* is True.

hera_stats.average.**redundant_diff_summary**(*uvd*, *red_bls*, *pol*, *op_upper=<Mock name='mock.max' id='140286774406392'>*, *op_lower=None*, *verbose=False*)

Calculate summary stats on the differences of baselines with the mean of their redundant group. Returns an antenna-antenna matrix where each element is the value of the summary statistic for a given antenna pair.

See *hera_stats.plot.antenna_matrix* for an accompanying plotting function.

Parameters

- **uvd** (*UVData*) – Visibility data object.
- **red_bls** (*list of lists*) – List of redundant baseline group lists.
- **pol** (*str or int*) – String or integer specifying the polarization to use.
- **op_upper** (*func*) – Function that operates on the difference waterfall (visibility for baseline minus the mean over its redundant group) to return a single number, to be used as the summary statistic for that baseline.

  Functions must have the following signature:

  scalar = func(d, grp_mean)

  where *d* is the difference data (a 2D complex array) and *grp_mean* is the mean over the redundant group (also a 2D complex array).

  The values of the summary statistic are placed on the upper triangle of the output matrix.

- **op_lower** (*func, optional*) – Same as *op_upper*, but places values on the lower triangle of the output matrix.
- **verbose** (*bool, optional*) – Whether to print progress messages. Default: False.

Returns

- **unique_ants** (*array_like*) – Ordered array of unique antennas that were found in the redundant groups.
- **mat** (*array_like*) – 2D array of summary statistic values for every pair of antennas.

  NB. np.nan values are returned for antenna pairs that were not present in the data/redundant baseline list.

CHAPTER 3

Flagging algorithms and helper functions

The `hera_stats.flag` module contains flagging algorithms and utilities, including a way to randomly flag frequency channels (`apply_random_flags`), a convenience function to flag whole ranges of channels at once (`flag_channels`), and an implementation of a 'greedy' flagging algorithm (`construct_factorizable_mask`) that can construct factorizable (in time and frequency) masks that flag as small a total fraction of the data as possible.

`hera_stats.flag.`**`apply_random_flags`**(*uvd*, *flag_frac*, *seed=None*, *inplace=False*, *zero_flagged_data=False*)

Randomly flag a set of frequency channels. Flags are applied on top of any existing flags, and are applied to all baselines, times, and polarizations.

Parameters

- **uvd** (*UVData object*) – Input UVData object to be flagged.
- **flag_frac** (*float*) – Fraction of channels to flag. This is the fraction of channels to apply flags to; the actual fraction of flagged channels may be greater than this, depending on if there were already flagged channels in the input UVData object.
- **seed** (*int, optional*) – Random seed to use. Default: None.
- **inplace** (*bool, optional*) – Whether to apply the flags to the input UVData object in-place, or return a copy that includes the new flags. Default: False.
- **zero_flagged_data** (*bool, optional*) – Whether to set the flagged channels in the data_array to zero. This is useful for identifying functions that are ignoring the mask. All flagged data will be zeroed, not just the new flags added by this function.

Returns **uvd** – Returns UVData object with flags applied.

Return type UVData object

`hera_stats.flag.`**`construct_factorizable_mask`**(*uvd_list*, *spw_ranges*, *first='col'*, *greedy_threshold=0.3*, *n_threshold=1*, *retain_flags=True*, *unflag=False*, *greedy=True*, *inplace=False*)

Generates a factorizable mask using a 'greedy' flagging algorithm, run on a list of UVData objects. In this

context, factorizable means that the flag array can be written as F(freq, time) = f(freq) * g(time), i.e. entire rows or columns are flagged.

First, flags are added to the mask based on the minimum number of samples available for each data point. Next, depending on the *first* argument, either full columns or full rows that have flag fractions exceeding the *greedy_threshold* are flagged. Finally, any rows or columns with remaining flags are fully flagged. (Unflagging the entire array is also an option.)

Parameters

- **uvd_list** (*list*) – list of UVData objects to operate on
- **spw_ranges** (*list*) – list of tuples of the form (min_channel, max_channel) defining which spectral window (channel range) to flag. *min_channel* is inclusive, but *max_channel* is exclusive.
- **first** (*str, optional*) – Either 'col' or 'row', defines which axis is flagged first based on the *greedy_threshold*. Default: 'col'.
- **greedy_threshold** (*float, optional*) – The flag fraction beyond which a given row or column is flagged in the first stage of greedy flagging. Default: 0.3.
- **n_threshold** (*float, optional*) – The minimum number of samples needed for a pixel to remain unflagged. Default: 1.
- **retain_flags** (*bool, optional*) – If True, then data points that were originally flagged in the input data remain flagged, even if they meet the *n_threshold*. Default: True.
- **unflag** (*bool, optional*) – If True, the entire mask is unflagged. No other operations (e.g. greedy flagging) will be performed. Default: False.
- **greedy** (*bool, optional*) – If True, greedy flagging takes place. If False, only *n_threshold* flagging is performed (so the resulting mask will not necessarily be factorizable). Default: True.
- **inplace** (*bool, optional*) – Whether to return a new copy of the input UVData objects, or modify them in-place. Default: False (return copies).

Returns **uvdlist_new** – if inplace=False, a new list of UVData objects with updated flags

Return type list

`hera_stats.flag.`**`flag_channels`**(*uvd*, *spw_ranges*, *inplace=False*)

Flags a given range of channels entirely for a list of UVData objects

Parameters

- **uvd** (*UVData*) – UVData object to be flagged.
- **spw_ranges** (*list*) – list of tuples of the form (min_channel, max_channel) defining which channels to flag.
- **inplace** (*bool, optional*) – If True, then the input UVData objects' flag arrays are modified, and if False, new UVData objects identical to the inputs but with updated flags are created and returned (default is False).
- **Returns**
- **———-**
- **uvd_new** (*list*) – Flagged UVData object.

CHAPTER 4

Noise estimation

The hera_stats.noise module contains simple empirical noise estimation functions, including a function (estimate_noise_rms) to estimate the noise rms by differencing data in the time direction and fit a smooth polynomial model that interpolates over flagged channels.

hera_stats.noise.**estimate_noise_rms**(*uvd*, *bls*, *fit_poly=True*, *order=2*)

Estimate noise RMS per frequency channel by taking a simple standard deviation of differences along the time axis. The real and imaginary parts are treated independently. A smooth polynomial model fit can be returned if desired.

Parameters

- **uvd** (*UVData*) – UVData file containing data to fit.
- **bls** (*list of int/tuple*) – List of baselines/baseline keys to calculate the noise rms for.
- **fit_poly** (*bool, optional*) – Fit a polynomial to the standard deviations as a function of frequency. Default: True.
- **order** (*int, optional*) – If fit_poly is True, order of the polynomial to fit to the standard deviations. Default: 2.

Returns

- **noise_rms** (*array_like, complex*) – 1D array of noise RMS (in the same units as the input data) as a function of frequency. Shape is (Nbls, Nfreq).
- **rms_model** (*array_like, complex, optional*) – If poly_fit is True, return the smooth polynomial fit to the noise rms. Shape is (Nbls, Nfreq).

CHAPTER 5

Diagnostic plotting functions

The `hera_stats.plot` module contains a variety of diagnostic plotting functions. These include a function to generate long waterfalls plots of `nsamples` or `flags` across many files (`long_waterfall`), including summary statistics on the flag fraction for each time/frequency bin.

`hera_stats.plot.`**`antenna_matrix`**(*mat*, *ants*, *label=None*, *cmap='viridis'*)

Plot 2D square array, intended to be a matrix of antenna vs antenna values of some quantity.

Parameters

- **mat** (*array_like*) – 2D square array containing matrix values to be plotted.
- **ants** (*array_like*) – 1D array of unique antennas represented in the matrix, in the same ordering as the input matrix.
- **label** (*str, optional*) – Label for the plot (displayed as the colorbar label). Default: None.
- **cmap** (*str, optional*) – Matplotlib colormap name. Default: 'viridis'.

Returns **fig** – Matplotlib figure containing matshow.

Return type matplotlib.Figure

`hera_stats.plot.`**`long_waterfall`**(*uvd_list*, *bl*, *pol*, *title=None*, *cmap='gray'*, *starting_lst=[]*, *mode='nsamples'*, *operator='abs'*, *file_type='uvh5'*, *figsize=(20, 80)*)

Generates a waterfall plot of flags or nsamples with axis sums from an input array.

Parameters

- **uvd_list** (*list of UVData objects or list of str*) – List of UVData objects to be stacked and displayed. If a list of strings is specified, each UVData object will be loaded one at a time (reduces peak memory consumption).
- **bl** (*int or tuple*) – Baseline integer or antenna pair tuple of the baseline to plot.
- **pol** (*str or int*) – Polarization string or integer.
- **title** (*str, optional*) – Title of the plot. Default: none.
- **cmap** (*str, optional*) – Colormap parameter for the waterfall plot. Default: 'gray'.

- **starting_lst** (*list, optional*) – list of starting lst to display in the plot
- **mode** (*str, optional*) – Which array to plot from the UVData objects. Options: 'data', 'flags', 'nsamples'. Default: 'nsamples'.
- **operator** (*str, optional*) – If mode='data', the operator to apply when plotting the data. Can be 'real', 'imag', 'abs', 'phase'. Default: 'abs'.
- **file_type** (*str, optional*) – If *uvd_list* is passed as a list of strings, specifies the file type of the data files to assume when loading them. Default: 'uvh5'.
- **figsize** (*tuple, optional*) – The size of the figure, in inches. Default: (20, 80).

Returns

- **main_waterfall** (*matplotlib.axes*) – Matplotlib Axes instance of the main plot
- **freq_histogram** (*matplotlib.axes*) – Matplotlib Axes instance of the sum across times
- **time_histogram** (*matplotlib.axes*) – Matplotlib Axes instance of the sum across freqs
- **data** (*numpy.ndarray*) – A copy of the stacked_array output that is being displayed

hera_stats.plot.**redgrp_corrmat**(*corr*, *red_grp*, *cmap='RdBu'*, *figsize=(30.0, 20.0)*, *line_alpha=0.2*)

Plot the correlation matrix for a set of delay spectra in a redundant group. See *hera_stats.stats.redgrp_pspec_covariance()* for a function to calculate the correlation matrix.

The elements of the correlation matrix are assumed to be in the same order as the *red_grp* list. Furthermore, the *red_grp* list is assumed to be ordered by blpair integer. Blocks of elements that have the same first bl in common are marked in the matrix.

Parameters

- **corr** (*ndarray*) – Covariance or correlation matrix.
- **red_grp** (*list*) – List of baseline-pairs in the redundant group (one for each row/column of the *corr* matrix).
- **cmap** (*str, optional*) – Matplotlib colormap to use for the correlation matrix plot. Default: 'RdBu'.
- **vmin, vmax** (*float, optional*) – Minimum and maximum values of the
- **figsize** (*tuple, optional*) – Size of the figure, in inches. Default: (30, 20).
- **line_alpha** (*float, optional*) – Alpha value of the lines used to draw blocks in the correlation matrix. Default: 0.2.

Returns **fig** – Figure containing the correlation matrix.

Return type matplotlib.Figure

hera_stats.plot.**scatter_bandpowers**(*uvp*, *x_dly*, *y_dly*, *keys*, *operator='abs'*, *label=None*, *ax=None*)

Scatter plot of delay spectrum bandpowers from two different delay bins. The bandpowers can be taken from multiple spws / blpairs / polpairs at the same time (see the *keys* argument).

Example of *keys* argument: .. highlight:: python .. code-block:: python

red_grps, red_lens, red_angs = uvp.get_red_blpairs() keys = { 'spws': 0, 'blpairs': red_grps[0], 'polpairs': uvp.get_polpairs(), }

Parameters

- **uvp** (*UVPSpec*) – Input delay spectrum object.

- **x_dly, y_dly** (*int*) – Index of delay bins to use as the x and y values respectively.
- **keys** (*dict*) – Dictionary with keys 'spws', 'blpairs', 'polpairs', which can be lists or single values. This allows bandpowers from multiple spws / blpairs / polpairs to be plotted simultaneously.
- **operator** (*str, optional*) – If mode='data', the operator to apply when plotting the data. Can be 'real', 'imag', 'abs', 'phase'. Default: 'abs'. Default: 'abs'.
- **label** (*str, optional*) – Label to use for this set of data in the plot legend. Default: None.
- **ax** (*matplotlib.axes, optional*) – Default: None.

Returns **ax** – Matplotlib Axes instance.

Return type matplotlib.axes

CHAPTER 6

Shuffling data

The `hera_stats.shuffle` module contains functions to randomly shuffle data. This includes a function to construct new visibilities by shuffling samples from a set of visibilities within the same redundant baseline group (`shuffle_data_redgrp`).

`hera_stats.shuffle.`**`shuffle_data_redgrp`**(*uvd*, *redgrps*)

Construct a new set of visibilities by randomly shuffling samples between baselines in a redundant group.

Different random shuffles are performed at each frequency, time, and polarization. This creates a new set of visibilities that are made up of samples from a random combination of redundant baselines, but that do not mix or shuffle times, frequencies or polarizations.

The samples are shuffled _without_ replacement, so each sample is only ever used once.

Example: Original visibility (fixed time and pol, for freq. channels 1, 2, 3...):

bl_a = [a1, a2, a3, a4, ...] bl_b = [b1, b2, b3, b4, ...] bl_c = [c1, c2, c3, c4, ...]

Shuffled visibility (for example):

new_a = [b1, a2, c3, b4, ...] new_b = [c1, c2, a3, a4, ...] new_c = [a1, b2, b3, c4, ...]

Parameters

- **uvd** (*UVData object*) – Input visibilities.
- **redgrps** (*list of lists of bls*) – List of redundant baseline groups.

Returns **uvd_new** – Copy of *uvd* with baseline-shuffled visibilities.

Return type UVData object

CHAPTER 7

Splitting data

The `hera_stats.split` module contains a range of convenience functions for splitting data in various ways.

`hera_stats.split.`**`blps_by_antnum`**(*blps*, *split='norepeat'*)

Split a list of redundant groups of baseline-pairs into two, depending on whether there are repeated antennas in the baseline pair (*norepeat*), or whether there are auto baselines in the pair (*noautos*).

Parameters

- **blps** (*list of list of blpairs*) – List of redundant groups of baseline-pairs. The blps can be either tuples of tuples, or blpair integers.
- **split** (*str, optional*) – Type of split to perform on each baseline group. Available options are:
  - **'norepeat':** Split into one group where antennas are used at most once per blpair, and another where they are used more than once.
  - **'noautos':** Split into one group with auto-blpairs and one group with non-autos (but antennas can be used more than once per blpair).

Returns

blps_a, blps_b –

List of redundant groups of baseline-pairs.

- For 'norepeat', group A contains the blps *without* repeated antennas, while group B contains the blps *with* repeated antennas.
- For 'noautos', group A contains the blps *without* auto-baselines, while group B contains the blps *with* auto-baselines.

Return type list of list of blpairs

`hera_stats.split.`**`hour_angle`**(*uvp*, *bins_list*, *specify_bins=False*, *bls=None*)

Splits UVPSpec based on the galactic hour-angle at the time of measurement.

Parameters

- **uvp** (*list or UVPSpec*) – List or single hera_pspec.UVPSpec object, containing data to use.

- **bins_list** (*list*) – One entry for each bin layout, default is that it must be an integer, where min and max values for hourangle values will automatically be set as limits. If specify_bins is True, then the inpud must be a list of ndarrays.
- **specify_bins** (*boolean*) – If true, allows bins_list to be specified as a list of the bins themselves. Default: False
- **bls** (*list of tuples, optional*) – The baselines to use in in omitting antenna. If None, uses all baselines. Default: None.

Returns **uvpl** – The resulting data, one list per jackknife.

Return type list of UVPSpec pairs

hera_stats.split.**lst_blocks**(*uvp*, *blocks=2*, *lst_range=(0.0*, *6.28)*)

Split a UVPSpec object into multiple objects, each containing spectra within different contiguous LST ranges. There is no guarantee that each block will contain the same number of spectra or samples.

N.B. This function uses the *lst_avg_array* property of an input UVPSpec object to split the LSTs (and not the LSTs of the individual visibilities that went into creating each delay spectrum).

Parameters

- **uvp** (*UVPSpec*) – Object containing delay spectra.
- **blocks** (*int, optional*) – How many blocks to return. Default: 2.
- **lst_range** (*tuple, optional*) – Tuple containing the minimum and maximum LST to retain. This is the range that will be split up into blocks. Default: (0., 2*pi)

Returns

- **uvp_list** (*list of UVPSpec*) – List of UVPSpec objects, one for each LST range. Empty blocks will appear as None in the list.
- **lst_bins** (*array_like*) – Array of LST bin edges. This has dimension (blocks+1,).

hera_stats.split.**lst_stripes**(*uvp*, *stripes=2*, *width=1*, *lst_range=(0.0*, *6.28)*)

Split a UVPSpec object into multiple UVPSpec objects, each containing spectra within alternating stripes of LST.

N.B. Gaps in LST are ignored; this function stripes based on the ordered list of available LSTs in *uvp* only.

N.B. This function uses the *lst_avg_array* property of the input UVPSpec object to split the LSTs (and not the LSTs of the individual visibilities that went into creating each delay spectrum).

Parameters

- **uvp** (*UVPSpec*) – Object containing delay spectra.
- **stripes** (*int, optional*) – How many stripes to return. Default: 2.
- **width** (*int, optional*) – Width of each stripe, in number of LST bins. Default: 1.
- **lst_range** (*tuple, optional*) – Tuple containing the minimum and maximum LST to retain. This is the range that will be split up into blocks. Default: (0., 2*pi)

Returns **uvp_list** – List of UVPSpec objects, one for each LST range.

Return type list of UVPSpec

hera_stats.split.**omit_ants**(*uvp*, *ant_nums=None*, *bls=None*)

Splits UVPSpecs into groups, omitting one antenna from each.

uvp: UVPSpec or list Single UVPSpec or list of UVPSpecs to use in splitting.

ant_nums: list, optional A list containing integers, each entry will generate one UVPSpec which does not contain the antenna specified.

bls: list of tuples, optional The baselines to use in in omitting antenna. If None, uses all baselines. Default: None.

Returns **uvp_list** – A list containing one list of UVPSpecs, with one for every ant_num specified.

Return type list of UVPSpecs

CHAPTER 8

Statistical tests for jackknives

The `hera_stats.stats` module contains various statistical convenience functions to compare jackknifed power spectra and other data.

`hera_stats.stats.`**`redgrp_pspec_covariance`**(*uvp*, *red_grp*, *dly_idx*, *spw*, *polpair*, *mode='cov'*, *verbose=False*)

Calculate the covariance or correlation matrix for all pairs of delay spectra in a redundant group, for a single delay bin. The matrix is estimated by averaging over all LST samples.

Parameters

- **uvp** (*UVPSpec*) – Input UVPSpec object.
- **red_grp** (*list*) – List of redundant baseline pairs within a group.
- **dly_idx** (*int*) – Index of the delay bin to calculate the covariance matrix for.
- **spw** (*int*) – Index of spectral window to use.
- **polpair** (*int or str or tuple*) – Polarization pair.
- **mode** (*str, optional*) – Whether to calculate the covariance matrix ('cov') or correlation matrix ('corr'). Default: 'cov'.
- **verbose** (*bool, optional*) – Whether to print status messages. Default: false.

Returns **cov_real, cov_imag** – Real and imaginary covariance or correlation matrices, of shape (Nblps, Nblps).

Return type ndarrays

`hera_stats.stats.`**`uvp_zscore`**(*uvp*, *error_field='bs_std'*, *inplace=False*)

Calculate a zscore of a UVPSpec object using entry 'error_field' in its stats_array. This assumes that the UVPSpec object has been already mean subtracted using hera_pspec.uvpspec_utils.subtract_uvp().

The resultant zscore is stored in the stats_array as error_field + "_zscore".

Parameters

- **uvp** (*UVPSpec object*)

- **error_field** (*str, optional*) – Key of stats_array to use as z-score normalization.
- **inplace** (*bool, optional*) – If True, add zscores into input uvp, else make a copy of uvp and return with zscores.

Returns uvp : UVPSpec object

Return type if inplace

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

h

Index