SACC Documentation

Release 0.1

A. Slosar

Nov 29, 2018

Contents:

SACC (Save All Correlations and Covariances, an utterly crappy acronym inspired by usually equally bad attempts by David A) is a format and reference library for general storage of 2-dimensional power spectra and correlation functions and their covariance matrices in the HDF5 format. It is very loosely inspired by Joe Zunz's 2point.

CHAPTER 1

Python API documentation

sacc contains one main class and 4 subclasses:

- *sacc.sacc.SACC*
- *sacc.binning.Binning*
- *sacc.tracer.Tracer*
- *sacc.meanvec.MeanVec*
- *sacc.precision.Precision*
- *sacc.window.Window*

class sacc.sacc.**SACC**(*tracers*, *binning*, *mean=None*, *precision=None*, *meta={}*)

SACC creator from its individual components.

SACC objects are the main container class for 2-point measurements.

Parameters

- **tracers** (Tracer) – list of *Tracer* objects used in the measurement and referenced in binning parameter
- **binning** (Binning) – *Binning* object describing what measurement each index in the mean vector and precision matrix contains.
- **mean** (MeanVec) – Vector representing the actual two-point measurement. If not a MeanVec object, will try to cast it into one.
- **precision** (Precision) – *Precision* object representing the covariance matrix or its inverse (a.k.a. the precision matrix).
- **meta** (*dict*) – dictionary containing additional metadata.

cullCross()

Cuts out all elements of the data vector (and covariance, etc.) that are not auto-correlations.

cullLminLmax(*lmin*, *lmax*)

Implement scale cuts on the data vector and the covariance.

Parameters **lmin, lmax** (*array_like*) – lists of minimum/maximum scales for each tracer. Any correlation will be cut to the most stringent scale cuts associated with the two tracers that go into it. CHECK THAT THIS IS TRUE.

get_exp_sample_set()
Return the set of exp_samples (essentially a label attached to each Tracer) contained in this SACC file.

Returns set with all the exp_sample labels.

ilrange(*t1i*, *t2i*)
Returns the indices of the data vector containing the cross-correlation between tracers t1i and t2i.

Parameters

- **t1i** (*int*) – index of the first tracer.
- **t2i** (*int*) – index of the second tracer.

Returns array of indices.

classmethod loadFromHDF(*filename*, *mean_filename=None*, *precision_filename=None*)
Create a SACC object from the contents of an HDF5 file.

Parameters

- **filename** (*str*) – path to input file. The file should contain at least tracers and binning.
- **mean_filename** (*str*) – path to file containing the data vector (set to None if *filename* already contains this or if you don't nead the data vector).
- **precision_filename** (*str*) – same as *mean_filename* for the precision/covariance matrix.

Returns *SACC* object.

lrange(*t1i*, *t2i*)
Returns the scales at which the cross-correlation between t1i and t2i are stored.

Parameters

- **t1i** (*int*) – index of the first tracer.
- **t2i** (*int*) – index of the second tracer.

Returns array of scales.

plot_vector(*subplot=None*, *plot_corr='all'*, *weightpow=2*, *set_logx=True*, *set_logy=True*, *show_axislabels=False*, *show_legend=True*, *prediction=None*, *clr='r'*, *lofsf=1.0*, *label=None*)
Plots the mean vector associated to the different tracers in the SACC file. The tracer correlations to plot can be selected by passing a list of pairs of values in plot_corr. It can also plot the autocorrelation by passing 'auto', the cross-correlation by passng 'cross', and both by passing 'all'. The correlations will be weighted by a factor of ell^{weightpow} (where ell is the nominal scale for each cross-correlation element).

TODO: finish describing all other parameters.

Parameters

- **plot_corr** (*str/array_like*) – select which correlations to plot. This can be done by passing a list of tracer index pairs, 'auto' to plot all auto-correlations, 'cross' to plot all cross-correlations or 'all' to plot everything.
- **weightpow** (*float*) – the correlations will be weighted by a factor of ell^{weightpow} (where ell is the nominal scale for each cross-correlation element).

printInfo()
Prints information about the contents of this SACC file

saveToHDF(*filename*, *save_mean=True*, *save_precision=True*, *mean_filename=None*, *precision_filename=None*)
Write this SACC object into an HDF5 file.

Parameters

- **filename** (*str*) – path to output file.
- **save_mean** (*boolean*) – whether to save the mean vector.
- **save_precision** (*boolean*) – whether to save the covariance matrix.
- **mean_filename** (*str*) – path to additional output file where the mean is to be saved.
- **precision_file** (*str*) – same as *mean_filename* for the precision/covariance matrix.

size()
Returns the size of the mean vector

Returns vector size.

sortTracers()
Return information about all the ingredients that define each element of the data vector.

Returns list of tuples, with one element per data vector element. Each tuple contains 5 elements: *t1i*, *t2i*, *typ*, *ells*, *ndx*. *t1i* and *t2i* are the indices of the tracers that go into this cross-correlation. *typ* is the type of correlation contained here. *ells* contains the scales at which this cross-correlation is stored. *ndx* contains the indices of the data vector containing this cross-correlation.

class sacc.tracer.**Tracer**(*name*, *type*, *z*, *Nz*, *exp_sample=None*, *Nz_sigma_logmean=None*, *Nz_sigma_logwidth=None*, *DNz=None*, *Mproxy_name=None*, *Mproxy_min=None*, *Mproxy_max=None*)
Tracer objects contain information about the maps contributing to any of the 2-point functions stored in a SACC file.

Parameters

- **name** (*str*) – Name for the tracer.
- **type** (*str*) – The type of tracer, i.e. whether it is a spin0 or spin2 observable. TODO:check this.
- **exp_sample** (*str*) – The experiment this tracer corresponds to. While *name* should be unique for each tracer, many tracers can have the same *exp_name*.
- **z,Nz** (*array_like*) – arrays describing the redshift distribution of this tracer
- **NZ_sigma_logmean** (*float*) – TODO
- **NZ_sigma_logwidth** (*float*) – TODO
- **DNz** (*float*) – TODO
- **Mproxy_name** (*str*) – name of the mass proxy if the tracer is clusters. Defaults to None.
- **Mproxy_min** (*float*) – minimum value of the mass proxy bin if the tracer is clusters. Defaults to None.
- **Mproxy_max** (*float*) – maximum value of the mass proxy bin if the tracer is clusters. Defaults to None.

addColumns(*columns*)
Adds extra columns describing a tracer (e.g. b(z) or some other function).

Parameters **columns** (*dict*) – dictionary where each value should be an array with as many elements as Tracer.Nz. These can be later accessed as *Tracer.extra_cols* or through *Tracer.extraColumn(column_name)*. This function can be called as many times as you want, and this dictionary will be updated with new columns.

extraColumn(*key*)
Returns a given extra column from this tracer. Literally the same as *Tracer.extra_cols[key]*.

Parameters **key** (*str*) – name of the extra column.

Returns array with the column.

is_CL()
Check if this tracer is a cluster stack.

Returns *True* or *False*.

is_WL()
Is the tracer a source for a lensing measurement. Returns a boolean. TODO: this doesn't make much sense.

classmethod loadFromHDF(*group*, *name*)
Create a Tracer object from an HDF file.

Parameters

- **group** (*h5py.Group*) – HDF5 group where this tracer is saved.
- **name** (*str*) – name of the tracer to load.

Returns *Tracer* object.

meanZ()
Returns mean redshift for this tracer.

Returns mean redshift.

saveToHDF(*group*)
Save the tracer to an HDF file.

TODO: change how cmb is implemented.

Parameters **group** (*h5py.Group*) – HDF5 group.

class sacc.binning.**Binning**(*typ*, *ls*, *T1*, *Q1*, *T2*, *Q2*, *windows=None*, *deltaLS=None*, *sunit=None*)
Binning objects contain information about the make up of each element of the data vector stored in a SACC file.

Parameters

- **typ** (*array_like*) – array of strings, with each string describing the type of correlation stored in the corresponding element of the data vector. TODO: check this.
- **ls** (*array_like*) – array of floats, with each value describing the token scale of the corresponding data vector element.
- **T1** (*array_like*) – array of ints, corresponding to the index of the first *Tracer* contributing to this data vector element.
- **Q1** (*array_like*) – TODO not sure what this is.
- **T2** (*array_like*) – array of ints, corresponding to the index of the second *Tracer* contributing to this data vector element.
- **Q2** (*array_like*) – TODO not sure what this is.

- **windows** (*array_like*) – array of *sacc.window.Window* objects, describing the contribution of different scales to this element of the data vector. If *None*, delta windows are assumed at the token scales.
- **deltaLS** (*array_like*) – TODO not sure what this is.
- **sunit** (*array_like*) – array of string, with each string describing the units of the corresponding data vector element.

cullBinning(*ndxlist*)
Reduce this Binning object to the indices in *ndxlist*.

Parameters **ndxlist** (*array_like*) – list of indices to preserve.

get_angle(*Q1*, *Q2*, *T1*, *T2*, *typ=None*)
Return an array with the scale values for a given cross-correlation.

Parameters

- **Q1** (*str*) – First quantity in pair, e.g. 'S' for shear, 'P' for position
- **Q2** (*str*) – Second quantity in pair.
- **T1** (*str*) – Index of the first tracer.
- **T2** (*str*) – Index of the second tracer.
- **typ** (*str*) – The type of pair to restrict to. If None use all pairs.

Returns array of scale values (length zero if no matches found).

get_bin_pairs(*Q1*, *Q2*, *typ=None*)
Return an array of the pairs of bin indices included in this data corresponding to the cross-correlation of quantities *Q1* and *Q2* (and additionally of type *typ*).

TODO: not sure at all why this is a useful function. Also, should rename to *get_tracer_pairs* to keep naming consistent.

Parameters

- **Q1** (*str*) – First quantity in pair, e.g. 'S' for shear, 'P' for position
- **Q2** (*str*) – Second quantity in pair.
- **typ** (*str*) – The type of pair to restrict to. If None use all pairs.

Returns nx2 array of tracer index pairs.

get_quantity_pairs(*typ=None*)
Return an array of the pairs of quantities included in this data

Parameters **typ** (*str*) – the correlation type to restrict to. If *None* use all pairs.

Returns array of unique pairs of quantities.

classmethod loadFromHDF(*group*)
Create a Binning object from an HDF file.

Parameters **group** (*h5py.Group*) – HDF5 group where this binning is saved.

Returns *Binning* object.

saveToHDF(*group*)
Save the binning to an HDF file.

Parameters **group** (*h5py.Group*) – HDF5 group.

saveToHDFFile(*filename*)
Save the binning to an HDF file.

Parameters **filename** (*str*) – path to output file.

size()
Returns the size of this Binning object.

Returns size of the binning object.

sacc.binning.**enc**(*x*)
If an object is a bytes instance or None, return it as-is Otherwise if it is a string object, return it as ascii bytes

Under python 2 bytes and string are the same, so this will always return the object as-is.

class sacc.meanvec.**MeanVec**(*values*)
MeanVec objects contain the data vector stored in a SACC file.

Parameters **values** (*array_like*) – data vector.

cullVector(*ndxlist*)
Reduce this MeanVec object to the indices in *ndxlist*.

Parameters **ndxlist** (*array_like*) – list of indices to preserve.

classmethod loadFromHDF(*group*)
Create a MeanVec object from an HDF file.

Parameters **group** (*h5py.Group*) – HDF5 group where this data vector is saved.

Returns *MeanVec* object.

saveToHDF(*group*)
Save the data vector to an HDF file.

Parameters **group** (*h5py.Group*) – HDF5 group.

saveToHDFFile(*filename*)
Save the data vector to an HDF file.

Parameters **filename** (*str*) – path to output file.

size()
Returns the size of the data vector

Returns data vector size.

class sacc.precision.**Precision**(*matrix=None*, *mode='dense'*, *is_covariance=True*, *binning=None*)
Precision objects contain the covariance (or inverse covariance a.k.a. precision) matrix for the data vector stored in a SACC file.

Parameters

- **matrix** (*array_like*) – covariance or precision matrix.
- **mode** (*str*) – how is the matrix passed? Options: *dense* (full matrix), *diagonal* (only diagonal passed, assume off-diagonal is zero), *ell_block_diagonal* (matrix is diagonal between different scales).
- **is_covariance** (*boolean*) – set to True (default) if passing the covariance matrix. Otherwise, *matrix* should contain the precision matrix.
- **binning** (Binning) – a Binning object describing the data vector of this covariance. Only needed if *mode=='ell_block_diagonal'*.

cullMatrix(*ndxlist*)
Reduce this Precision object to the indices in *ndxlist*.

Parameters **ndxlist** (*array_like*) – list of indices to preserve.

getCovarianceMatrix()
Get covariance matrix stored in this object.

Returns covariance matrix.

getPrecisionMatrix()
Get precision matrix stored in this object.

Returns precision matrix (inverse covariance).

classmethod loadFromHDF(*group*, *binning=None*)
Create a Precision object from an HDF file.

Parameters

- **group** (*h5py.Group*) – HDF5 group where this precision object is saved.
- **binning** (Binning) – a Binning object describing the data vector of this covariance. Only needed if *mode=='ell_block_diagonal'*.

Returns *Precision* object.

saveToHDF(*group*)
Save the precision object to an HDF file. If the covariance matrix exists then that one is saved, otherwise the precision is saved.

Parameters **group** (*h5py.Group*) – HDF5 group.

saveToHDFFile(*filename*)
Save the precision object to an HDF file.

Parameters **filename** (*str*) – path to output file.

class sacc.window.**Window**(*ls=None*, *w=None*)
Window objects contain information about the scales that contribute to a given data vector element.

Parameters

- **ls** (*array_like*) – array of scales.
- **w** (*array_like*) – array of weights for each scale.

classmethod loadFromHDF(*group*, *ndx*)
Create a Window object from an HDF file.

Parameters

- **group** (*h5py.Group*) – HDF5 group where this data vector is saved.
- **ndx** (*int*) – index of the data vector element that you want the window function of.

Returns *Window* object.

saveToHDF(*group*, *ndx*)
Save the windowr to an HDF file.

Parameters

- **group** (*h5py.Group*) – HDF5 group.
- **ndx** (*int*) – index of the data vector element that this window corresponds to.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index