
RascalC Documentation

Oliver Philcox, Ross O'Connell, Daniel Eisenstein, Alex Wiegand

Oct 09, 2019

Contents:

1	Overview	1
1.1	Package Installation	1
1.2	Getting Started	2
1.3	Usage Tutorial	5
1.4	Pre-Processing	10
1.5	Computing Jackknife Weights	12
1.6	Computing Random Counts and Survey Correction Functions	13
1.7	Correlation Functions	16
1.8	Covariance Matrix Estimation	19
1.9	Post-Processing & Reconstruction	24

RascalC is a code to quickly estimate covariance matrices from two- or three-point galaxy correlation functions, written in C++ and Python. Given an input set of random particle locations and a two-point correlation function (or input set of galaxy positions), RascalC produces an estimate of the associated covariance for a given binning strategy, with non-Gaussianities approximated by a ‘shot-noise-rescaling’ parameter. For the 2PCF, the rescaling parameter can be calibrated by dividing the particles into jackknife regions and comparing sample to theoretical jackknife covariance. RascalC can also be used to compute Legendre-binned covariances and cross-covariances between different two-point correlation functions.

The main estimators are described in [O’Connell et al. 2016](#), [O’Connell & Eisenstein 2018](#), [Philcox et al. 2019](#) and [Philcox & Eisenstein \(in prep.\)](#) with the third and fourth papers discussing the new algorithms and C++ implementation.

The source code is publicly available on [Github](#) and builds upon the Python package [Rascal](#). For general usage, a comprehensive tutorial is provided.

1.1 Package Installation

To install RascalC, simply clone the Github repository and compile the C++ code (see [Dependencies](#) below). This is done as follows:

```
git clone https://github.com/oliverphilcox/RascalC.git
cd RascalC
make
```

NB: RascalC can be run in various modes by adding compiler flags in the Makefile. See [Getting Started](#) and [Covariance Matrix Estimation](#) for more information.

Once RascalC is installed, see the [Getting Started](#) and [Usage Tutorial](#) sections.

1.1.1 Dependencies

RascalC requires the following packages:

- **C compiler**: Tested with gcc 5.4.0
- **Gnu Scientific Library (GSL)**: Any recent version
- **Corrfunc**: 2.0 or later
- *(Optional but encouraged)* **OpenMP**: Any recent version (required for parallelization)

Corrfunc can be installed using `pip install corrfunc` and is used for efficient pair counting.

For the Python pre- and post-processing we require:

- **Python**: 2.7 or later, 3.4 or later
- **Numpy**: 1.10 or later
- *(Optional)* **Healpy**: any recent version. (Necessary if using HealPix jackknife regions)

These can be simply installed with `pip` or `conda`.

1.2 Getting Started

RascalC computes covariance matrix estimates from a given correlation function and set of random particles. Here, we give a broad overview of the procedure and the relevant *File Inputs*. A *Usage Tutorial* is provided to demonstrate basic use of the code pipeline.

1.2.1 Computation Modes

The RascalC code may be run a variety of modes, which compute the covariances of different statistics. These are enabled via compilation flags, set in the Makefile (see *Covariance Matrix Estimation* for usage). The various modes are outlined below:

1. **DEFAULT** (No compiler flags): Compute the full-survey covariance of the anisotropic two-point correlation function (2PCF) in (r, μ) bins.
2. **JACKKNIFE** (`-DJACKKNIFE` flag): Compute the full-survey and jackknife covariance matrices of the anisotropic 2PCF in (r, μ) bins. The theoretical jackknife matrix can be compared to the sample jackknife matrix to compute the shot-noise rescaling parameter α .
3. **LEGENDRE** (`-DLEGENDRE` flag): Compute the full-survey covariance of (even) Legendre multipoles of the 2PCF. *NB*: We do not provide functionality to compute the jackknife covariance in Legendre multipole bins, since this has a more complex functional form. However, the shot-noise rescaling parameter α can be found from (r, μ) jackknife covariance matrix fitting and applied to the output full-survey Legendre-binned matrix.
4. **3PCF** (`-DTHREE_PCF` flag): Compute the full-survey covariance of (odd and even) Legendre multipoles of the isotropic three-point correlation function (3PCF).

1.2.2 Outline of Pipeline

In order to compute the covariance matrices there are several steps:

1. **Pre-Processing (Optional)**: We provide a suite of utility functions to convert input files to the correct forms used by RascalC. This includes conversion from (Ra,Dec,redshift) to (x,y,z) coordinate space, creation of binfiles and assignment of HealPix jackknife regions to particles. Alternatively, this step can be skipped if the input files are already of the correct format.

2. **Computing Jackknife Weights (Only required in JACKKNIFE mode):** Before the main C++ code is run, we compute the weights for each jackknife region, by computing jackknife-specific RR pair counts using `Corrfunc`. This is run via a Python script.
- 3a. **Computing Random Counts and Survey Correction Functions (Required in DEFAULT, LEGENDRE and 3PCF modes):** The main C++ code requires either the RR counts (DEFAULT mode) or the survey-geometry correction function Φ (LEGENDRE and 3PCF modes). We provide Python and C++ scripts to compute these, with the correction functions requiring a set of RR or RRR counts to be computed first.
- 3b. **Correlation Functions (Optional):** This provides functions to compute the overall survey correlation functions for one or two fields using `Corrfunc` (which may also be defined by the user). In addition, we provide routines to compute the jackknife correlation functions $\xi^J(r, \mu)$, which are later used to calibrate the shot-noise rescaling parameter(s) in the JACKKNIFE mode.
4. **Covariance Matrix Estimation:** The main C++ code computing the 2PCF or 3PCF individual covariance matrix terms using Monte Carlo integration. For multiple input correlation functions in the 2PCF modes, this computes all relevant terms for the six non-trivial cross-covariance matrices. The covariances are saved as `.txt` files which can be reconstructed in Python.
5. **Post-Processing & Reconstruction:** If jackknife covariances have been computed, this Python script computes the shot-noise rescaling parameter(s) and reconstructs output covariance matrices from the jackknife correlation function estimates produced in *Correlation Functions*. Else, we provide scripts to reconstruct the *Covariance Matrix Estimation* output. In both cases, a single `.npz` file is created including the output covariance and precision matrices as well as the effective number of mocks N_{eff} .

1.2.3 File Inputs

The required input files and formats are described below. Note that several of these can be computed using the *Pre-Processing* codes.

- **Random Particle File(s):**
 - This lists the locations and weights of random particles which describe a survey geometry.
 - This must specify the $\{x, y, z, w, j\}$ coordinates for each particle, where $\{x, y, z\}$ are Cartesian coordinates (in comoving Mpc/h units), w are particle weights and j are integers referencing which jackknife the particle is in.
 - $\{RA, Dec, redshift\}$ coordinates can be converted to $\{x, y, z\}$ positions using the *Coordinate Conversion* script.
 - HealPix jackknives can be added using the *Adding Jackknives* script.
 - *Format:* An ASCII file with each particle defined on a new row, and tab-separated columns indicating the $\{x, y, z, w, j\}$ coordinates.
- **Galaxy Position File(s):**
 - This lists the locations and weights of galaxies in a specific survey, in the same manner as the random particles.
 - This is only required to compute the correlation functions in the *Correlation Functions* scripts.
 - *Format:* See above.
- **Covariance Matrix Binning File:**
 - This specifies the radial binning in the output covariance matrix.
 - For each bin we specify the minimum and maximum radii in comoving Mpc/h units.
 - Linear, logarithmic and hybrid binning files can be created using the *Create Binning Files* scripts.

- *Format*: An ASCII file with each bin occupying a separate line, with tab-separated columns specifying (r_{\min}, r_{\max}) for each bin.
- **Correlation Function Binning File:**
 - File specifying the radial binning used in the input correlation function.
 - The lowest bin must extend to zero for this, and the highest bin should be at least as large as the maximum covariance matrix bin.
 - Currently must be the same for all input correlation functions, for the multiple field case.
 - *Format*: See above.
- **(Usually created internally) Correlation Function(s):**
 - This specifies the input correlation function estimates to be used by RascalC.
 - For two sets of tracer particles, we require three correlation functions; two auto-correlations and a cross-correlation.
 - These can be user input or created with Corrfunc using the *Full Survey Correlation Functions* codes.
 - Estimates of $\xi(r, \mu)$ must be given for a grid of values of (r, μ) , which must extend close to zero for r with the bins as specified in the correlation function binning file.
 - *Format*: An ASCII file with space separated values. Line 1 lists the radial coordinates of the bin centers and line 2 lists the angular coordinates. Successive lines list the correlation function estimates $\xi(r, \mu)$, with the column indicating the μ bin center and the row indicating the r bin center.
- **(Only required in JACKKNIFE mode and usually created internally) Jackknife Correlation Functions:**
 - This specifies the input correlation function estimates for each *unrestricted* jackknife, $\xi_A^J(r, \mu)$.
 - For two sets of tracer particles, we require three correlation functions; two auto-correlations and a cross-correlation.
 - This is conventionally created with Corrfunc using the *Jackknife Matrix Correlation Functions (for the JACKKNIFE mode only)* codes, but may be user input if desired.
 - The radial and angular binning should match that desired for the output covariance matrix.
 - If this is supplied separately, the user must ensure that the pair count terms are normalized by the ratio of summed galaxy and random particle weights across the **entire** survey, not just those in the relevant jackknife region. This is for later convenience when estimating the jackknife covariance matrix model.
 - *Format*: An ASCII file with space separated values. Lines 1 and 2 list the radial and angular bin centers (as for the full correlation function). Each succeeding line gives the entire correlation function estimate for a given jackknife. The rows indicate the jackknife and the columns specify the collapsed bin, using the indexing $\text{bin}_{\text{collapsed}} = \text{bin}_{\text{radial}} \times n_{\mu} + \text{bin}_{\text{angular}}$ for a total of n_{μ} angular bins (unlike for the full correlation function).
- **(Required in JACKKNIFE mode and usually created internally) Jackknife Weights and Random Particle Counts:**
 - These specify the weights of each jackknife region for each bin and the random particle counts both for each jackknife, and for the entire survey.
 - These should be created using the *Computing Jackknife Weights* script.
 - They are saved in .dat files with the name `jackknife_weights_n{N}_m{M}_j{J}_{INDEX}.dat`, `jackknife_pair_counts_n{N}_m{M}_j{J}_{INDEX}.dat` and `binned_pair_counts_n{N}_m{M}_j{J}_{INDEX}.dat` where N and M specify the number of radial and angular bins respectively and J gives the number of non-empty jackknife

regions. INDEX specifies which fields are being used (e.g. 12 specifies the cross-weights between fields 1 and 2).

- **(Required in *DEFAULT* mode and usually created internally) Random Particle Counts:**

- These specify random particle counts for the entire survey, which are needed to normalize the (r, μ) binned covariances.
- These should be created using the RR count script described in [Computing Random Counts and Survey Correction Functions](#) (and *not* normalized by the summed squared weights).
- They are saved in .dat files with the name `binned_pair_counts_n{N}_m{M}_{INDEX}.dat` where N and M specify the number of radial and angular bins respectively. INDEX specifies which fields are being used (e.g. 12 specifies the cross-weights between fields 1 and 2).

- **(Required in *LEGENDRE* and *3PCF* modes and usually created internally) Survey Correction Function Parameters:**

- These give the necessary parameters for the main C++ code to reconstruct the survey-correction function, $\Phi(r_a, \mu)$ (2PCF) or $\Phi(r_a, r_b, \chi)$ (3PCF).
- For multiple input fields, we will have three output bin correction factors of the same format.
- These should be created using the survey-correction functions described in [Computing Random Counts and Survey Correction Functions](#), and require the RR or RRR counts to be computed (also described in [Computing Random Counts and Survey Correction Functions](#)).
- They are saved as ASCII files with the names `BinCorrectionFactor_n{N}_m{M}.txt` or `BinCorrectionFactor3PCF_n{N}_m{M}.txt` and specify polynomial fitting parameters (2PCF) or the first seven multipoles of Φ^{-1} (3PCF), which are found to well describe the fit. These have one row per radial bin (or pair of bins for the 3PCF), and must be constructed using the same radial binning as for the output covariance matrix.

1.3 Usage Tutorial

We present a basic example of the use of the RascalC code for a single field, computing the 2PCF in (r, μ) bins, and calibrating for the non-Gaussianity parameter using jackknives. Multiple field cases proceed similarly. Detailed documentation for all functions is given in associated pages, as overviewed in the [Getting Started](#) pages. Note that we here run the main code in the JACKKNIFE mode.

Here, we compute the covariance matrix for a single QPM mock dataset. We'll work in the directory in which RascalC is installed for simplicity.

1.3.1 1) Pre-Processing

Inputs:

- `mock_galaxy_DR12_CMASS_N_QPM_0001.txt`: Galaxy positions and weights for the QPM mock in (Ra,Dec,redshift,weight) format.
- `mock_random_DR12_CMASS_N_50x1.txt`: 50x Random positions and weights for the CMASS survey in (Ra,Dec,redshift,weight).

First, we'll convert these into Cartesian (x,y,z,weight) coordinates, using $\Omega_m = 0.29$, $\Omega_k = 0$, $w_\Lambda = -1$ (to be consistent with the QPM mock data creation):

```
python python/convert_to_xyz.py mock_galaxy_DR12_CMASS_N_QPM_0001.txt qpm_galaxies.
↪xyz 0.29 0. -1
python python/convert_to_xyz.py mock_random_DR12_CMASS_N_50x1.txt qpm_randoms_50x.xyz_
↪0.29 0. -1
```

(See *Coordinate Conversion*).

These are saved as `qpm_galaxies.xyz` and `qpm_randoms_50x.xyz` in (x,y,z,weight) format.

Now let's add some jackknives to these files. We'll use HEALPIX NSIDE=8 jackknife regions here:

```
python python/create_jackknives.py qpm_galaxies.xyz qpm_galaxies.xyzwj 8
python python/create_jackknives.py qpm_randoms_50x.xyz qpm_randoms_50x.xyzwj 8
```

(See *Adding Jackknives*).

These are saved as `qpm_galaxies.xyzwj` and `qpm_randoms_50x.xyzwj` in (x,y,z,weight,jackknife-id) format, and we're using 169 non-empty jackknives here.

We've got 50x the number of random particles as galaxies here which seems a little big. Let's reduce this to 10x (noting that there are 642051 galaxies in the galaxy file):

```
python python/take_subset_of_particles.py qpm_randoms_50x.xyzwj qpm_randoms_10x.xyzwj_
↪6420510
```

(See *Take Subset of Particles*).

Great! Now we have a random particle file with 10x randoms, and all files are in the correct format.

Let's create the radial binning files next. We'll create two binning files; one for the correlation functions and one for the covariance matrix.

For the covariance matrix, we'll use a linear binning file with $\Delta r = 5$ for $r \in [20, 200]$ and for the correlation function we'll use a linear binning file with $\Delta r = 1$ for $r \in [0, 200]$. **NB:** The correlation function binning file must extend down to $r = 0$:

```
python python/write_binning_file_linear.py 36 20 200 radial_binning_cov.csv
python python/write_binning_file_linear.py 200 0 200 radial_binning_corr.csv
```

(See *Create Binning Files*).

Here we're using 36 radial bins for the covariance matrix. Let's have a look at the covariance binning file:

```
20.00000000    25.00000000
25.00000000    30.00000000
30.00000000    35.00000000
35.00000000    40.00000000
40.00000000    45.00000000
45.00000000    50.00000000
50.00000000    55.00000000
55.00000000    60.00000000
....
```

This all looks as expected.

1.3.2 2) Jackknife Weights

We're now ready to compute the jackknife weights w_{aA} for this set of random particles. This determines how much weight we assign to each jackknife region later in the analysis, via the RR pair counts in each bin and jackknife.

Here we'll use 12 angular bins with $\mu \in [0, 1]$ and recall that this dataset is non-periodic (so μ is measured from the line-of-sight, as opposed to the z -axis). We'll run 10-threaded for speed and save in the `weights/` directory.:

```
python python/jackknife_weights.py qpm_randoms_10x.xyzwj radial_binning_cov.csv 1. 12_
↪10 0 weights/
```

(See *Computing Jackknife Weights*).

This computes pair counts for each pair of random particles in the survey (using Corrfunc), so may take a while...

The outputs are saved as `weights/jackknife_weights_n36_m12_j169_11.dat`, `weights/jackknife_pair_counts_n36_m12_j169_11.dat` and `weights/binned_pair_counts_n36_m12_j169_11.dat` containing the weights w_{aA} , bin-counts RR_{aA} and summed bin counts RR_a respectively.

1.3.3 3) Correlation Functions

Using the galaxy and random particle files, we can obtain estimates of the correlation function. Firstly, we'll compute an estimate of $\xi(r, \mu)$ to be used to compute the theoretical covariance matrices. In addition, we'll use 120 μ bins in $[0, 1]$ and set the code to run for aperiodic input data. This must use the *correlation function* radial binning file, giving us a fine estimate of the correlation function.:

```
python python/xi_estimator.py qpm_galaxies.xyzwj qpm_randoms_50x.xyzwj qpm_randoms_
↪10x.xyzwj radial_binning_corr.csv 1. 120 10 0 xi/
```

(See *Full Survey Correlation Functions*).

This uses Corrfunc to perform pair counting and computes ξ_a for each bin, a , via the Landy-Szalay estimator. Here we're using 10x randoms to compute the RR pair counts and 50x randoms to compute the DR pair counts. The output is saved as `xi/xi_n200_m120_11.dat` in the format specified in *File Inputs*. We'll use this full correlation function to compute the theoretical covariance matrix later on. In addition, at the end of the code, we're told that the number of galaxies is 642051; this is an important quantity that we'll need later on.

Now let's compute the jackknife correlation function estimates for each bin, ξ_{aA}^J . These are the individual correlation functions obtained from each unrestricted jackknife, and we can use them to create a data jackknife covariance matrix which we can compare to theory. This is run in a similar way to before, but we must now use the *covariance matrix* radial binning file, since we use these to directly compute a covariance. Here, we'll use 10x randoms for RR counts and 50x randoms for DR counts, but we can skip some of the work by loading in the jackknife pair counts computed by the *Computing Jackknife Weights* script (in the same binning as here), which avoids recomputing RR counts. (The input 10x random file isn't loaded in this case).:

```
python python/xi_estimator_jack.py qpm_galaxies.xyzwj qpm_randoms_50x.xyzwj qpm_
↪randoms_10x.xyzwj radial_binning_cov.csv 1. 12 10 0 xi_jack/ weights/jackknife_pair_
↪counts_n36_m12_j169_11.dat
```

(See *Jackknife Matrix Correlation Functions (for the JACKKNIFE mode only)*).

NB: This may take a little while to compute, depending on the number of randoms and galaxies used. The output jackknife correlation functions are saved as `xi_jack/xi_jack_n36_m12_j169_11.dat` in the format specified in *File Inputs*. These will be automatically read later on.

1.3.4 4) Computing the Covariance Matrix

(See *Covariance Matrix Estimation*).

Now that all of the inputs have been computed, we can run the main C++ code to compute the theoretical covariance matrix terms.

There's two ways to run the code here; firstly we could edit parameters in the `modules/parameters.h` file, to tell the code where to find the relevant inputs. Here are the important lines

```
....

//----- ESSENTIAL PARAMETERS -----

// The name of the input random particle files (first set)
char *fname = NULL;
const char default_fname[500] = "/mnt/store1/oliverphilcox/Mock1QPM2/qpm_randoms_10x.
↳xyzwj";

// Name of the radial binning .csv file
char *radial_bin_file = NULL;
const char default_radial_bin_file[500] = "/mnt/store1/oliverphilcox/Mock1QPM2/radial_
↳binning_cov.csv";

// The name of the correlation function file for the first set of particles
char *corname = NULL;
const char default_corname[500] = "/mnt/store1/oliverphilcox/Mock1QPM2/xi/xi_n200_
↳m120_11.dat";

// Name of the correlation function radial binning .csv file
char *radial_bin_file_cf = NULL;
const char default_radial_bin_file_cf[500] = "/mnt/store1/oliverphilcox/Mock1QPM2/
↳radial_binning_corr.csv";

// Number of galaxies in first dataset
Float nofznorm=642051;

// Output directory
char *out_file = NULL;
const char default_out_file[500] = "/mnt/store1/oliverphilcox/Mock1QPM2/";

// The number of mu bins in the correlation function
int mbin_cf = 120;

// The number of threads to run on
int nthread=10;

// The grid size, which should be tuned to match boxsize and rmax.
// This uses the maximum width of the cuboidal box.
int nside = 251;

// Whether or not we are using a periodic box
bool perbox = false;

//----- (r,mu) PARAMETERS -----

// The number of mu bins
int mbin = 12;

// Name of the RR bin file
char *RR_bin_file = NULL; // RR_{aA}^{11} file
const char default_RR_bin_file[500] = "/mnt/store1/oliverphilcox/Mock1QPM2/weights/
↳binned_pair_counts_n36_m12_j169_11.dat";
```

(continues on next page)

(continued from previous page)

```
//----- JACKKNIFE PARAMETERS -----

// Name of the jackknife weight file
char *jk_weight_file = NULL; // w_{aA}^{11} weights
const char default_jk_weight_file[500] = "/mnt/store1/oliverphilcox/Mock1QPM2/weights/
↪jackknife_weights_n36_m12_j169_11.dat";

....

//----- PRECISION PARAMETERS -----

// Maximum number of iterations to compute the C_ab integrals over
int max_loops=10;

// Number of random cells to draw at each stage
int N2 = 20; // number of j cells per i cell
int N3 = 40; // number of k cells per j cell
int N4 = 80; // number of l cells per k cell

....
```

Here we're using 10 loops (to get 10 independent estimates of the covariance matrix), and setting N2-N4 such that we'll get good precision in a few hours of runtime. Note that the `nofznorm` parameter is set to the summed galaxy weights we found before. Now, we'll compile the code::

```
bash clean
make
```

The first line simply cleans the pre-existing `./cov` file, if present and the second compiles `grid_covariance.cpp` using the Makefile (using the `g++` compiler by default). We have edited the Makefile to add the `-DJACKKNIFE` flag to ensure we compute jackknife covariances here. If we were using periodic data we'd need to set the `-DPERIODIC` flag in the Makefile before running this step. Similarly, we could remove the `-DOPENMP` flag to run single threaded. The code is then run with the default parameters;

```
./cov -def
```

Alternatively, we could simply pass these arguments on the command line (after the code is compiled). (**NB:** We can get a summary of the inputs by simply running `./cov` with no parameters)

```
./cov -in qpm_randoms_10x.xyzwj -binfile radial_binning_cov.csv -cor xi/xi_n200_m120_
↪11.dat -binfile_cf radial_binning_corr.csv -norm 1.07636096e+05 -jackknife weights/
↪jackknife_pair_counts_n36_m12_j169_11.dat -RRbin weights/binned_pair_counts_n36_m12_
↪j169_11.dat -output ./ -mbin 12 -mbin_cf 120 -nside 251 -maxloops 10 -N2 20 -N3 40 -
↪N4 80
```

It's often just easier to edit the `modules/parameter.h` file, but the latter approach allows us to change parameters without recompiling the code.

This runs in around 5 hours on 10 cores here, giving output matrix components saved in the `CovMatricesFull` and `CovMatricesJack` directories as `.txt` files. We'll now reconstruct these.

1.3.5 5) Post-Processing

Although the C++ code computes all the relevant parts of the covariance matrices, it doesn't perform any reconstruction, since this is much more easily performed in Python. Post-processing is used to compute the optimal value of

the shot-noise rescaling parameter α (by comparing the data-derived and theoretical covariance matrices), as well as construct the output covariance and precision matrices.

For a single field analysis, this is run as follows, specifying the jackknife correlation functions, output covariance term directory and weights. Since we used $N_{\text{loops}} = 10$ above, we'll set this as the number of subsamples here:

```
python python/post_process.py xi_jack/xi_jack_n36_m12_j169_11.dat weights/ ./ 12 10 ./
```

(See post-processing-single).

The output is a single compressed Python .npz file which contains the following analysis products:

- Optimal shot-noise rescaling parameter α^*
- Full theory covariance matrix $C_{ab}(\alpha^*)$
- Jackknife theory covariance matrix $C_{ab}^J(\alpha^*)$
- Jackknife data covariance matrix $C_{ab}^{J,\text{data}}$
- Full (quadratic bias corrected) precision matrix $\Psi_{ab}(\alpha^*)$
- Jackknife (quadratic bias corrected) precision matrix $\Psi_{ab}^J(\alpha^*)$
- Full quadratic bias \tilde{D}_{ab} matrix
- Effective number of mocks N_{eff}
- Individual full covariance matrix estimates $C_{ab}^{(i)}(\alpha^*)$

This completes the analysis!

1.4 Pre-Processing

We provide a suite of Python scripts to create input files for the RascalC code. These are found in the `python/` directory.

1.4.1 Coordinate Conversion

This converts a set of input particles (either random particle or galaxy positions) in the form $\{\text{RA}, \text{Dec}, \text{redshift}, \text{weight}\}$ to the Cartesian form $\{x, y, z, \text{weight}\}$, given some cosmological parameters (using a Λ CDM coordinate converter by Daniel Eisenstein). The output coordinates are in comoving Mpc/h units, and are saved as an ASCII file for later use.

Usage:

```
python python/convert_to_xyz.py {INFILE} {OUTFILE} [{OMEGA_M} {OMEGA_K} {W_DARK_
↪ENERGY}]
```

Parameters:

- **{INFILE}**: Input data file containing $\{\text{RA}, \text{Dec}, \text{redshift}, \text{weight}\}$ coordinates for each particle. This should be in the form of an input ASCII datafile, with each particle on a separate row.
- **{OUTFILE}**: Output `.txt`, `.dat` or `.csv` filename.
- *Optional* **{OMEGA_M}**: Current matter density, Ω_m (default 0.31)
- *Optional* **{OMEGA_K}**: Current curvature density. Ω_k (default 0)
- *Optional* **{W_DARK_ENERGY}**: Dark energy equation of state parameter, w_Λ (default -1)

1.4.2 Adding Jackknives

This function assigns each particle (either random particles or galaxy positions) to a jackknife region, j , by assigning a HEALPix pixel number to each datapoint, with a given value of NSIDE. Data is saved as an ASCII file with $\{x,y,z,w,j\}$ columns. *NB:* This is only required for the JACKKNIFE mode, and the code will simply ignore the jackknife numbers (if present) if run in other modes.

Usage:

```
python python/create_jackknives.py {INFILE} {OUTFILE} {HEALPIX_NSIDE}
```

Parameters:

- **{INFILE}**: Input data ASCII file of (random/galaxy) Cartesian particle positions with space-separated columns $\{x,y,z,w\}$, such as that created by the [Coordinate Conversion](#) script. This can be in .txt, .dat or .csv format.
- **{OUTFILE}**: Output .txt, .dat or .csv filename.
- **{HEALPIX_NSIDE}**: HealPix NSIDE parameter which controls the number of pixels used to divide up the sky. For $\text{NSIDE} = n$, a total of $12n^2$ pixels are used.

1.4.3 Take Subset of Particles

A utility function to reduce the number of particles in an input ASCII file to a given number. This is primarily used to select a random subsample of the random particle file to speed computation.

Usage:

```
python python/take_subset_of_particles.py {INFILE} {OUTFILE} {N_PARTICLES}
```

Parameters:

- **{INFILE}**: Input data ASCII file with particle positions, in $\{x,y,z,w\}$, $\{x,y,z,w,j\}$ or $\{RA,Dec,redshift,w\}$ format.
- **{OUTFILE}**: Outfile .txt, .dat or .csv filename.
- **{N_PARTICLES}**: Desired number of particles in output file. A random sample of length N_PARTICLES is selected from the input file.

1.4.4 Create Binning Files

A utility function to create radial binning files used by RascalC. We provide three scripts for different binning regimes. This file may be alternatively specified by the user, in the format described in [File Inputs](#).

- *Linear*: Bins are linearly spaced bins in (r_{\min}, r_{\max}) .
- *Logarithmic*: Bins are evenly spaced in logarithmic space (base e) in (r_{\min}, r_{\max}) .
- *Hybrid*: Bins are logarithmically spaced in $(r_{\min}, r_{\text{cutoff}})$, then linearly spaced in $(r_{\text{cutoff}}, r_{\max})$.

Usage:

```
python python/write_binning_file_linear.py {N_BINS} {MIN_R} {MAX_R} {OUTPUT_FILE}
python python/write_binning_file_logarithmic.py {N_BINS} {MIN_R} {MAX_R} {OUTPUT_FILE}
python python/wrtie_binning_file_hybrid.py {N_LOG_BINS} {N_LIN_BINS} {MIN_R} {CUTOFF_
↪R} {MAX_R} {OUTPUT_FILE}
```

Parameters:

- {N_BINS}: Total number of linear or logarithmic bins.
- {MIN_R}: Minimum bin radius, r_{\min} .
- {MAX_R}: Maximum bin radius, r_{\max} .
- {N_LOG_BINS}: Number of logarithmic bins for hybrid binning.
- {N_LIN_BINS}: Number of linear bins for hybrid binning.
- {CUTOFF_R}: Radius at which to switch from logarithmic to linear binning, r_{cutoff} (for hybrid binning).
- {OUTPUT_FILE}: Output .txt, .csv or .dat filename.

1.5 Computing Jackknife Weights

NB: This is *only* required when running the RascalC code in the JACKKNIFE mode.

Here, we compute the weights assigned to each jackknife region for each bin. This is done using the [Corrfunc](#) code of Sinha & Garrison to compute the weights $w_{aA}^{XY} = RR_{aA}^{XY} / \sum_B RR_{aB}^{XY}$ for bin a , jackknife A and fields X and Y .

Two codes are supplied; one using a single set of tracer particles and the other with two input sets, for computation of cross-covariance matrices. These are in the `python/` directory. This must be run before the main C++ code.

1.5.1 Usage

For a single field analysis:

```
python python/jackknife_weights.py {RANDOM_PARTICLE_FILE} {BIN_FILE} {MU_MAX} {N_MU_
↪BINS} {NTHREADS} {PERIODIC} {OUTPUT_DIR}
```

For an analysis using two distinct fields:

```
python python/jackknife_weights_cross.py {RANDOM_PARTICLE_FILE_1} {RANDOM_PARTICLE_
↪FILE_2} {BIN_FILE} {MU_MAX} {N_MU_BINS} {NTHREADS} {PERIODIC} {OUTPUT_DIR}
```

NB: The two field script computes all three combinations of weights between the two random fields, thus has a runtime ~ 3 times that of `jackknife_weights.py`. Running these together in one script ensures that we have the same number of jackknives for all fields. Also, the two fields must be distinct, else there are issues with double counting.

Input Parameters

- {RANDOM_PARTICLE_FILE}, {RANDOM_PARTICLE_FILE_1}, {RANDOM_PARTICLE_FILE_2}: Input ASCII file containing random particle positions and jackknife numbers in {x,y,z,weight,jackknife_ID} format, such as that created with the [Pre-Processing](#) scripts. This should be in .csv, .txt or .dat format with space-separated columns.
- {BIN_FILE}: ASCII file specifying the radial bins, as described in [File Inputs](#). This can be user-defined or created by the [Create Binning Files](#) scripts.
- {MU_MAX}: Maximum $\mu = \cos \theta$ used in the angular binning.
- {N_MU_BINS}: Number of angular bins used in the range $[0, \mu]$.
- {NTHREADS}: Number of CPU threads to use for pair counting parallelization.
- {PERIODIC}: Whether the input dataset has periodic boundary conditions (0 = non-periodic, 1 = periodic). See note below.

- {OUTPUT_DIR}: Directory in which to house the jackknife weights and pair counts. This will be created if not in existence.

Notes:

- This is a very CPU intensive computation since we must compute pair counts between every pair of random particles up to the maximum particle separation. The process can be expedited using multiple CPU cores or a reduced number of random particles (e.g. via the *Take Subset of Particles* script).

Note on Periodicity

The code can be run for datasets created with either periodic or non-periodic boundary conditions. Periodic boundary conditions are often found in cosmological simulations. If periodic, the pair-separation angle θ (used in $\mu = \cos \theta$) is measured from the z axis, else it is measured from the radial direction. If periodic data is used, the C++ code **must** be compiled with the -DPERIODIC flag.

1.5.2 Output files

This code creates ASCII files containing the jackknife weights for each bin, the RR pair counts for each bin in each jackknife and the summed RR pair counts in each bin. The output files have the format `jackknife_weights_n{N}_m{M}_j{J}_{INDEX}.dat`, `jackknife_pair_counts_n{N}_m{M}_j{J}_{INDEX}.dat` and `binned_pair_counts_n{N}_m{M}_j{J}_{INDEX}.dat` respectively. N and M specify the number of radial and angular bins respectively and J gives the number of non-empty jackknives. INDEX specifies which fields are being used i.e. INDEX = 12 implies the w_{aA}^{12} , RR_{aA}^{12} and RR_a^{12} quantities.

The binned pair counts is a list of weighted pair counts for each bin, summed over all jackknife regions, in the form $RR_a^{j,XY} = \sum_B RR_{aB}^{XY}$, with each bin on a separate row. The jackknife pair counts and jackknife weights files list the quantities RR_{aA}^{XY} and w_{aA}^{XY} for each bin and jackknife region respectively. We note that the RR_{aA}^{XY} quantities (and only these) are normalized by the whole-survey product of summed weights $(\sum_i w_i)^2$ for later convenience.

The j -th row contains the (tab-separated) quantities for each bin using the j -th jackknife. The first value in each row is the jackknife number, and the bins are ordered using the collapsed binning $\text{bin}_{\text{collapsed}} = \text{bin}_{\text{radial}} \times n_\mu + \text{bin}_{\text{angular}}$ for a total of n_μ angular bins.

1.6 Computing Random Counts and Survey Correction Functions

Here, we describe the scripts provided to compute RR and RRR random counts, in addition to the *survey correction functions*, defined as the ratio of ideal to true RR (RRR) counts for the 2PCF (3PCF). The RR counts are computed using the *Corrfunc* code of Sinha & Garrison. We additionally provide functionality to compute multi-field RR and survey-correction functions for the 2PCF. Most of the scripts below are in the `python/` directory, and must be run before the main C++ code.

Note on Periodicity

The codes below can be run for datasets created with either periodic or non-periodic boundary conditions. Periodic boundary conditions are often found in cosmological simulations. If periodic, the pair-separation angle θ (used in $\mu = \cos \theta$) is measured from the z axis, else it is measured from the radial direction. If periodic data is used, the C++ code **must** be compiled with the -DPERIODIC flag.

1.6.1 Estimating RR pair counts

This is required to normalize the covariances in DEFAULT mode, and to compute the survey-correction-function in LEGENDRE mode (for aperiodic surveys only). In JACKKNIFE mode, the scripts in *Computing Jackknife Weights* should be used instead.

Usage

For a single field analysis:

```
python python/RR_counts.py {RANDOM_PARTICLE_FILE} {BIN_FILE} {MU_MAX} {N_MU_BINS}
↪ {NTHREADS} {PERIODIC} {OUTPUT_DIR} {NORMED}
```

For an analysis using two distinct fields:

```
python python/RR_counts_multi.py {RANDOM_PARTICLE_FILE_1} {RANDOM_PARTICLE_FILE_2}
↪ {BIN_FILE} {MU_MAX} {N_MU_BINS} {NTHREADS} {PERIODIC} {OUTPUT_DIR} {NORMED}
```

Input Parameters

- {RANDOM_PARTICLE_FILE}, {RANDOM_PARTICLE_FILE_1}, {RANDOM_PARTICLE_FILE_2}: Input ASCII file containing random particle positions and jackknife numbers in {x,y,z,weight,[jackknife_ID]} format, such as that created with the *Pre-Processing* scripts. This should be in .csv, .txt or .dat format with space-separated columns. The 'jackknife_id' column will not be used if included.
- {BIN_FILE}: ASCII file specifying the radial bins, as described in *File Inputs*. This can be user-defined or created by the *Create Binning Files* scripts.
- {MU_MAX}: Maximum $\mu = \cos \theta$ used in the angular binning.
- {N_MU_BINS}: Number of angular bins used in the range $[0, \mu]$. *NB*: This should be large (preferably ~ 100) if the RR counts are being used for LEGENDRE mode survey correction computation. In DEFAULT mode, this should match the output number of angular bins required.
- {NTHREADS}: Number of CPU threads to use for pair counting parallelization.
- {PERIODIC}: Whether the input dataset has periodic boundary conditions (0 = non-periodic, 1 = periodic). See note below.
- {OUTPUT_DIR}: Directory in which to house the RR counts. This will be created if not in existence.
- {NORMED}: Whether to normalize the output RR counts by the summed squared random weights (0 = no normalization, 1 = normalization). This should be done in LEGENDRE mode (where the RR counts are used to compute the survey correction function) but *not* in the DEFAULT mode.

Notes:

- Output files will be saved as `RR_counts_n{N}_m{M}_{INDEX}.txt` for N radial and M angular bins. INDEX specifies which fields are used (e.g. INDEX=12 for the field 1 x field 2 pair count). This is saved as a simple list of $N \times M$ values using the indexing $\text{bin}_{\text{collapsed}} = \text{bin}_{\text{radial}} * M + \text{bin}_{\text{angular}}$.
- This is a very CPU intensive computation since we must compute pair counts between every pair of random particles up to the maximum particle separation. The process can be expedited using multiple CPU cores or a reduced number of random particles (e.g. via the *Take Subset of Particles* script).
- For two sets of input particles, three sets of weights must be computed for the three possible pairs of two distinct fields, hence the computation time increases by a factor of three.

1.6.2 Computing 2PCF Survey Correction Functions

This script computes the survey correction function $\Phi(r_a, \mu)$ in each bin a and fits it to a smooth model. The output parameters can be fed into the main C++ script. For convenience, we divide the output Φ by $V(nw)^2$, for survey volume V . For the periodic case, the survey correction function is simply unity everywhere, so it is simple to compute. If aperiodic, we require the survey RR pair counts as an input, either from the *Estimating RR pair counts* scripts or elsewhere.

Usage

For a single field analysis:

```
python python/compute_correction_function.py {GALAXY_FILE} {BIN_FILE} {OUTPUT_DIR}
↪ {PERIODIC} [{RR_COUNTS}]
```

For an analysis using two distinct fields:

```
python python/compute_correction_function_multi.py {GALAXY_FILE} {GALAXY_FILE_2} {BIN_
↪ FILE} {OUTPUT_DIR} {PERIODIC} [{RR_COUNTS_11} {RR_COUNTS_12} {RR_COUNTS_22}]
```

Input Parameters:

See the *Estimating RR pair counts* parameters above. Additionally;

- `{RR_COUNTS}`, `{RR_COUNTS_11}`, `{RR_COUNTS_12}` `{RR_COUNTS_22}` (*Only required if dataset is periodic*): RR counts computed by the *Estimating RR pair counts* script, or externally. These should be normalized by the sum of the squared random weights (by using the NORMED flag above).

Notes:

- **NB:** For aperiodic data, this assumes that the weights are FKP weights, such that they can be used to find the random number density at each galaxy position. This is not assumed for periodic data (where number density is constant everywhere).
- Output files are saved as `BinCorrectionFactor_n{N}_m{M}_{INDEX}.txt` (aperiodic) or `BinCorrectionFactor_n{N}_periodic_{INDEX}.txt` (periodic) for N radial (and M angular) bins, with INDEX specifying which fields are used. If run for two distinct fields, three correction factors are output, for each non-trivial combination of bins.
- File format is a list of N rows (for N radial bins) with 7 columns specifying the fitting parameters (as specified in Philcox & Eisenstein 2019). This is automatically read by the main C++ code.
- For a periodic data-set, we output a set of parameters which will lead to the survey correction function being reconstructed as unity everywhere.

1.6.3 Estimating RRR triple counts

We provide a simple code to compute the RRR triple counts, required to find the 3PCF survey correction function for aperiodic survey geometries. (This is not required for periodic surveys). This is a stochastic estimator (similar to the main C++ covariance matrix code) and does *not* exhaustively count all triples of particles. For heavy usage, an exhaustive code such as the $\mathcal{O}(N^2)$ algorithm of Slepian & Eisenstein (2015) would be more appropriate.

This is a C++ code and is run analogous to the covariance code in *Covariance Matrix Estimation*. To compile and run use the following:

```
cd triple_counts
bash clean
make
./triple [OPTIONS]
```

In the associated Makefile we can specify the `-DPERIODIC` and `-DOPENMP` flags for periodic and OpenMP parallelized compilation respectively. The command line options (and the majority of the code) are the same as for the main code (see descriptions in *Covariance Matrix Estimation*), and we must specify at least the input random file, (r, μ) binning strategy and accuracy parameters. **NB:** We must set the minimum μ to be -1 here, as there is no $\mu \leftrightarrow -\mu$ symmetry unlike for the 2PCF. In addition, the number of angular bins should be relatively large ($\gtrsim 20$) to reduce bias in later conversion of the correction function into Legendre multipole space.

The code outputs a single text file in the specified output directory, named `RRR_counts_n{N}_m{M}_full.txt` which contains the RRR counts, normalized by the summed cubed random particle weights. Each estimate is on a separate line, with the indexing $\text{bin}_{\text{collapsed}} = (\text{bin}_{\text{radial},1} \times N + \text{bin}_{\text{radial},2}) \times M + \text{bin}_{\text{angular}}$. These can be read-in by the survey correction function.

1.6.4 Computing 3PCF Survey Correction Functions

This script computes the 3-point survey correction function $\Phi(r_a, r_b, \chi)$ in each radial bin pair a, b , and outputs the multipoles of Φ^{-1} which are reconstructed by the main C++ code in runtime. For convenience, we divide the output Φ by $6V(nw)^3$, for survey volume V . For the periodic case, the survey correction function is simply unity everywhere, so it is simple to compute. If aperiodic, we require the (normalized) survey RRR pair counts as an input, either from the *Estimating RRR triple counts* scripts or elsewhere.

NB: We do not provide functionality for the multi-field 3PCF covariances here.

Usage

```
python python/compute_3pcf_correction_function.py {GALAXY_FILE} {BIN_FILE}
↪ {OUTPUT_DIR} {PERIODIC} [{RRR_COUNTS}]
```

Input Parameters:

See the *Estimating RR pair counts* parameters above. Additionally;

- `{RRR_COUNTS}` (*Only required if dataset is periodic*): RRR counts computed by the *Estimating RRR triple counts* script, or externally. These should be normalized by the sum of the cubed random weights (done natively by the above code).

Notes:

- **NB:** For aperiodic data, this assumes that the weights are FKP weights, such that they can be used to find the random number density at each galaxy position. This is not assumed for periodic data (where number density is constant everywhere).
- Output files are saved as `BinCorrectionFactor3PCF_n{N}_m{M}.txt` (aperiodic) or `BinCorrectionFactor3PCF_n{N}_periodic.txt` (periodic) for N radial (and M angular) bins.
- File format is a list of N^2 rows (for N radial bins) with 7 columns specifying the first seven multipoles of the (normalized) inverse Φ multipoles. This is automatically read by the main C++ code.
- For a periodic data-set, we output a set of parameters which will lead to the survey correction function being reconstructed as unity everywhere.

1.7 Correlation Functions

The scripts described below are wrappers of the *Corrfunc* code (Sinha & Garrison 2017), used to create full-survey and jackknife correlation functions. The former are used in the computation of the Gaussian covariance matrices, and the latter allow for determination of the shot-noise rescaling parameter, if the JACKKNIFE mode is used. If the correlation function is required to be computed in a different manner, user-input correlation functions can simply replace the output of these codes, with the file-types described in *File Inputs*.

1.7.1 Full Survey Correlation Functions

To compute the covariance matrix estimates \hat{C}_{ab} we require some estimate of the correlation function. Here, we compute the full-survey correlation function with a specified binning using Corrfunc. We provide routines for both 1- and 2-field scenarios (computing the fields $\{\xi^{11}(r, \mu), \xi^{12}(r, \mu), \xi^{22}(r, \mu)\}$ in the latter case). This uses both the galaxies and random particle files, and requires a correlation function binning file, such as created by [Create Binning Files](#). The estimations are computed via the Landy-Szalay estimator using $\xi_a^{XY} = (\widehat{DD}_a^{XY} - \widehat{DR}_a^{XY} - \widehat{DR}_a^{YX} + \widehat{RR}_a^{XY}) / (\widehat{RR}_a^{XY})$ for bin a , fields X, Y with DD/DR/RR specifying data-data / data-random / random-random pair counts. The hats indicate that the quantities are normalized by the product of the summed weights in the two fields (i.e. $\sum_{i \in X} w_i^X \sum_{j \in Y} w_j^Y$). Note that the binned correlation function estimates $\hat{\xi}_a^{XY}$ cannot simply be placed at the bin-centers and interpolated to give a smooth $\xi^{XY}(r, \mu)$ estimate; here we use an iterative approach to convert these estimates into interpolation points for the smooth $\xi^{XY}(r, \mu)$.

The scripts take two sets of random particle files; one to compute DR counts and one to compute RR counts. This allows for a larger number of randoms to be used for RR counts, as is often useful.

Periodicity: This script can be run for periodic or aperiodic input data; this corresponds to measuring μ from the z or radial axis respectively. If the data is periodic (e.g. from a cosmological box simulation) the `-DPERIODIC` flag should be set on compilation of the full C++ code in [Covariance Matrix Estimation](#).

Usage

For a single field analysis:

```
python python/xi_estimator.py {GALAXY_FILE} {RANDOM_FILE} {RADIAL_BIN_FILE_DR}
↪ {RADIAL_BIN_FILE_RR} {MU_MAX} {N_MU_BINS} {NTHREADS} {PERIODIC} {OUTPUT_DIR} [{RR_
↪ counts}]
```

For an analysis using two distinct fields:

```
python python/xi_estimator_cross.py {GALAXY_FILE_1} {GALAXY_FILE_2} {RANDOM_FILE_1_DR}
↪ {RANDOM_FILE_1_RR} {RANDOM_FILE_2_DR} {RANDOM_FILE_2_RR} {RADIAL_BIN_FILE} {MU_MAX}
↪ {N_MU_BINS} {NTHREADS} {PERIODIC} {OUTPUT_DIR} [{RR_counts_11} {RR_counts_12} {RR_
↪ counts_22}]
```

NB: The two field script computes all three distinct (cross-)correlations between the two fields, thus has a runtime ~ 3 times that of `xi_estimator.py`. The two fields should be distinct to avoid issues with double counting.

Input Parameters

- `{GALAXY_FILE}`, `{GALAXY_FILE_1}`, `{GALAXY_FILE_2}`: Input ASCII file containing galaxy positions and weights in `{x,y,z,weight,jackknife_ID}` format such as that created with the [Pre-Processing](#) scripts. (Jackknives are not used in this script and may be omitted). This should be in `.csv`, `.txt` or `.dat` format with space-separated columns.
- `{RANDOM_FILE_DR}`, `{RANDOM_FILE_1_DR}`, `{RANDOM_FILE_2_DR}`: Input ASCII file containing random particle positions and weights to be used for DR pair counting (with filetype as for the galaxy files).
- `{RANDOM_FILE_RR}`, `{RANDOM_FILE_1_RR}`, `{RANDOM_FILE_2_RR}`: Input ASCII file containing random particle positions and weights to be used for RR pair counting (with filetype as for the galaxy files). **NB:** If pre-computed RR pair counts are specified only the length of the RR random file is used by the code (for normalization).
- `{RADIAL_BIN_FILE}`: ASCII file specifying the radial bins for $\xi(r, \mu)$, as described in [File Inputs](#). This can be user-defined or created by the [Create Binning Files](#) scripts. **NB:** This bin-file specifies the bins for the *correlation function*, which may be distinct from the *covariance-matrix* bins. In particular, the lowest bin should extend to $r = 0$.
- `{MU_MAX}`: Maximum $\mu = \cos \theta$ used in the angular binning.

- {N_MU_BINS}: Number of angular bins used in the range $[0, \mu_{\max}]$.
- {NTHREADS}: Number of CPU threads to use for pair counting parallelization.
- {PERIODIC}: Whether the input dataset has periodic boundary conditions (0 = non-periodic, 1 = periodic). See note below.
- {OUTPUT_DIR}: Directory in which to house the correlation functions. This will be created if not in existence.
- *Optional* {RR_counts}, {RR_counts_XY}: Pre-computed RR pair counts (for the single field and between fields X and Y respectively). These should be in the format described in [File Inputs](#), and must use the same number of radial and angular bins as specified above. If not specified, these are recomputed by the code. Since the full correlation function typically uses a different binning to the output covariance matrix, we typically cannot use the pair counts computed in [Computing Jackknife Weights](#) and must recompute them. In addition, these should be normalized by the squared sum of weights $(\sum_i w_i)^2$ where i runs across all random particles in the dataset.

Output Files

ASCII files are created specifying the correlation function in the file-format given in [File Inputs](#). The filename has the format `xi_n{N}_m{M}_{INDEX}.dat`, where N and M specify the number of radial and angular bins respectively. INDEX specifies the correlation function type, where 11 = field 1 auto-correlation, 22 = field 2 auto-correlation, 12 = cross-correlation of fields 1 and 2. The first and second lines of the `.dat` file list the radial and angular bin centers, then each subsequent line lists the $\xi(r, \mu)$ estimate, with the column specifying the μ bin and the row specifying the r bin.

NB: The code also prints the number of galaxies in each dataset to the terminal, N_{gal} . This quantity is important for later normalization of the C++ code.

1.7.2 Jackknife Matrix Correlation Functions (*for the JACKKNIFE mode only*)

For later comparison of the jackknife covariance matrix estimate with the data, we require the jackknife covariance matrix, which is derived from the correlation function estimates in each unrestricted jackknife. The scripts below are provided to compute these using Corrfunc. For jackknife J and fields $\{X, Y\}$, we compute the pair counts FG_a^{XY} in bin a (where $F, G \in [D, R]$ for data and random fields D and R), from a cross-pair counts between particles in jackknife A of F^X and the entire of field G^Y . These are added to the pair counts from the cross of particles in jackknife A of field G^Y with the entire of field F^X if the fields are distinct. This allows us to compute all n_{jack} correlation functions $\xi_A^{XY}(r, \mu)$ via the Landy-Szalay estimator $\xi_{aA}^{XY} = (\widehat{DD}_{aA}^{XY} - \widehat{DR}_{aA}^{XY} - \widehat{DR}_{aA}^{YX} + \widehat{RR}_{aA}^{XY}) / \widehat{RR}_{aA}^{XY}$ for bin a . As before, the code takes two random particle fields of each type, allowing different sized random fields to be used for DR and RR pair counting. For convenience the quantities are normalized by the summed weights across the **entire** set of particles, not just those specific to the given jackknife. The jackknife correlation functions are thus not quite true estimates of ξ_a , since they neglect differences in the ratio of galaxies and random particles between galaxies.

NB: The binning file used here should be the same as that used for the *covariance matrix* **not** the full correlation function, to allow comparison with the C_{ab}^J estimate.

Usage

For a single field analysis:

```
python python/xi_estimator_jack.py {GALAXY_FILE} {RANDOM_FILE_DR} {RANDOM_FILE_RR}
↪ {RADIAL_BIN_FILE} {MU_MAX} {N_MU_BINS} {NTHREADS} {PERIODIC} {OUTPUT_DIR} [{RR_
↪ jackknife_counts}]
```

For an analysis using two distinct fields:

```
python python/xi_estimator_jack_cross.py {GALAXY_FILE_1} {GALAXY_FILE_2} {RANDOM_FILE_
↪ 1_DR} {RANDOM_FILE_1_RR} {RANDOM_FILE_2_DR} {RANDOM_FILE_2_RR} {RADIAL_BIN_FILE}
↪ {MU_MAX} {N_MU_BINS} {NTHREADS} {PERIODIC} {OUTPUT_DIR} [{RR_jackknife_counts_11}
↪ {RR_jackknife_counts_12} {RR_jackknife_counts_22}]
```

(continues on next page)

(continued from previous page)

This computes estimates of the auto- and cross-correlations for all unrestricted jackknife regions. Since there are three distinct correlations for each, the run-time is increased by a factor of 3.

Following computation of ξ_{aA}^J we can estimate the single-survey jackknife covariance matrix via $C_{ab,data}^J = \sum_A w_{aA} w_{bA} (\xi_{aA}^J - \bar{\xi}_a^J)(\xi_{bA}^J - \bar{\xi}_b^J) / (1 - \sum_B w_{aB} w_{bB})$. This is done internally in the post-processing-single code.

Input Parameters

See the input parameters for the *Full Survey Correlation Functions* script. In addition, the {RR_jackknife_counts_XY} quantities are the RR_{aA}^{XY} pair counts which can be specified to avoid recomputation. These have been previously output by the *Computing Jackknife Weights* code as `jackknife_pair_counts_n{N}_m{M}_j{J}_{INDEX}.dat` (using the correct covariance-matrix binning) hence can be used here for a significant speed boost. The RR_{aA}^{XY} pair counts must be normalized by the squared full-survey summed weights $(\sum_i w_i)^2$ - this is done automatically in the preceding script.

Output Files

This script creates ASCII files for each output correlation function, of the form `xi_jack_n{N}_m{M}_{INDEX}.dat` for N radial bins, M angular bins and INDEX specifying the correlation function type (11 = autocorrelation of field 1 (default), 12 = cross-correlation of fields 1 and 2, 22 = autocorrelation of field 2). **NB:** These have a different file format to the non-jackknife correlation functions. The first and second lines of the `.dat` file list the radial and angular bin centers, but each succeeding line gives the entire correlation function estimate for a given jackknife. The rows indicate the jackknife and the columns specify the collapsed bin, using the indexing $\text{bin}_{\text{collapsed}} = \text{bin}_{\text{radial}} \times n_\mu + \text{bin}_{\text{angular}}$ for a total of n_μ angular bins.

These files are read automatically by the post-processing-multi code.

1.8 Covariance Matrix Estimation

1.8.1 Overview

This is the main section of RascalC, where 2PCF or 3PCF covariance matrix estimates are computed via Monte Carlo integration from a given set of input particles. For the 2PCF, depending on the number of input fields the code will compute either components for a single covariance matrix or all required components for 6 cross-covariance matrices (i.e. for multi-tracer analysis).

Prerequisites: - In JACKKNIFE mode, the jackknife weights and binned pair counts must be computed via the *Computing Jackknife Weights* script before the C++ code is run. - In DEFAULT mode, the RR pair counts must be computed via the *Computing Random Counts and Survey Correction Functions* script before the C++ code is run. - In LEGENDRE and 3PCF modes, the survey correction functions Φ must be computed via the *Computing Random Counts and Survey Correction Functions* script before the C++ code is run.

Particle Grid and Cells

In the code, the particles are divided up into many cells for efficient computation, with each containing ~ 10 random particles. A cuboidal grid of cells is constructed aligned with the (x, y, z) axes, which fully encompasses all random particles. The cellsize is set by the N_{side} parameter, which gives the number of (cubic) cells along the largest dimension along the box. This must be an odd integer, and the code will automatically exit if N_{side} is too small or too large (since this gives inefficient performance).

Covariance Matrix Precision

The precision of covariance matrix estimators can be user-controlled in the RascalC code. To understand these parameters we must briefly outline the selection of random quads of particles (denoted $\{i, j, k, l\}$) used in the Monte Carlo integration. The particle selection algorithm has the following form:

1. Loop over all i -particles N_{loops} times. Each loop is independent and can be run on separate cores.
2. For each i particle, we pick N_2 j -particles at random, according to some selection rule. Here we compute the 2-point contribution to the covariance matrix, if we are computing a 2PCF covariance.
3. For each j particle, we pick N_3 k -particles at random, according to some selection rule. Here we compute the 3-point contribution to the covariance matrix.
4. For each k particle, we pick N_4 l -particles at random, according to some selection rule. Here we compute the 4-point contribution to the covariance matrix.
5. (3PCF mode only): For each l particle, we pick N_5 m -particles at random, according to some selection rule. Here we compute the 5-point contribution to the 3PCF covariance matrix.
6. (3PCF mode only): For each m particle, we pick N_6 n -particles at random, according to some selection rule. Here we compute the 6-point contribution to the 3PCF covariance matrix.

By setting the parameters $(N_{\text{loops}}, N_2, N_3, N_4, [N_5, N_6])$ we can control the precision of each matrix component. Standard values of $N_2 \sim N_3 \sim N_4 [\sim N_5 \sim N_6] \sim 10$ normally work well. Each loop of the code produces an independent estimate of the full covariance matrix, which can be used to create accurate inverse matrices and effective number of mock calculations. The covariance converges relatively fast, so setting N_{loops} to a few times the number of cores should work well. Values of $N_{\text{loops}} \gtrsim 100$ should be avoided to stop file sizes and reconstruction times becoming large.

1.8.2 Usage

The code is used as follows, with the command line options specified below:

```
bash clean
make
./cov [OPTIONS]
```

The first line removes any pre-existing C++ file before it is recompiled in line 2 to produce the `./cov` file. The Makefile may need to be altered depending on the particular computational configuration used. The default Makefile is for a standard Unix installation, with the `Makefile_mac` file giving a sample Makefile for a Mac installation. This uses the following optional flags in the Makefile;

- `-DOPENMP`: (Recommended) Run code in parallel with OpenMP, using the OpenMP installation specified by the `-lgomp` and `-fopenmp` flags.
- `-DPERIODIC`: Use periodic boundary conditions (appropriate for a cubic simulation box, but not mock surveys).
- `-DJACKKNIFE`: Compute both full-survey and jackknife 2PCF covariance matrix terms in (r, μ) binning, allowing for shot-noise-rescaling calibration from the survey itself.
- `-DLEGENDRE`: Compute the full-survey covariance matrix terms for (even) Legendre multipoles of the 2PCF.
- `-DTHREE_PCF`: Compute the full-survey covariance matrix terms for (even and odd) Legendre multipoles of the isotropic 3PCF.

NB: For a summary of input command line parameters, simply run `./cov` with no arguments.

Options

Input parameters for the RascalC code may be specified by passing options on the command line or by setting variables in the Parameters class in the `modules/parameters.h`. When using two sets of random particles, the code will exit automatically if all the required files are not input. We list the major code options below, in the form `-option` (*variable*) for command line option `option` and Parameters class variable *variable*.

Essential Parameters:

- `-def`: Run the code with the default options for all parameters (as specified in the `modules/parameters.h` file).
- `-in` (*fname*): Input ASCII random particle file for the first set of tracer particles. This must be in `{x,y,z,w,j}` format, as described in [File Inputs](#).
- `-binfile` (*radial_bin_file*): Radial binning ASCII file (see [File Inputs](#)) specifying upper and lower bounds of each radial bin.
- `-cor` (*corname*): Input correlation function estimate for the first set of particles in ASCII format, as specified in [File Inputs](#). This can be user defined or created by [Full Survey Correlation Functions](#).
- `-binfile_cf` (*radial_bin_file_cf*): Radial binning ASCII file for the correlation function (see [File Inputs](#)) specifying upper and lower bounds of each radial bin.
- `-norm` (*nofznorm*): Number of galaxies in the first set of tracer particles. This is used to rescale the random particle covariances.
- `-output` (*out_file*): Output directory in which to store covariance matrix estimates. This directory will be created if not already present. **Beware:** the code can produce a large volume of output (~ 1 GB for a standard run with one field and ~ 1000 bins).
- `-mbin_cf` (*mbin_cf*): Number of μ bins used for the correlation function.
- `-nside` (*nside*): Number of cubic cells to use along the longest dimension of the grid encompassing the random particles, i.e. N_{side} . See [Particle Grid and Cells](#) note for usage.
- `-nthread` (*nthread*): Number of parallel processing threads used if code is compiled with OpenMPI.
- `-perbox` (*perbox*): Whether or not we are using a periodic box.

DEFAULT and JACKKNIFE mode Binning Parameters: - `-mbin` (*mbin*): Number of μ bins used. This must match that used to create the jackknife weights. - `-RRbin` (*RR_bin_file*): Location of the `binned_pair_counts_n{N}_m{M}_j{J}_11.dat` ASCII file containing the summed pair counts in each bin (RR_{aA}^{11}), created by the `jackknife_weights` scripts.

JACKKNIFE mode Parameters:

- `-jackknife` (*jk_weight_file*): Location of the `jackknife_weights_n{N}_m{M}_j{J}_11.dat` file containing the jackknife weights for each bin (w_{aA}^{11}), as created by the `jackknife_weights` scripts.

LEGENDRE and 3PCF mode Parameters:

- `max_l` (*max_l*): Maximum Legendre moment to compute. This must be even in the LEGENDRE mode.
- `phi_file` (*phi_file*): Location of the file containing the survey correction function parameters, as created by the [Computing Random Counts and Survey Correction Functions](#) script.

Precision Parameters

- `-maxloops` (*max_loops*): This is the number of matrix subsamples to compute. See [Covariance Matrix Precision](#) note for usage guidelines. (Default: 10)
- `-N2, -N3, -N4` (*N2, N3, N4*): The parameters controlling how many random particles to select at each stage. See [Covariance Matrix Precision](#) note above. (Default: 10)

- `-N5, -N5 (N5, N6)`: As above, but for the 3PCF mode only. (Default: 10)

General Multi Field Parameters:

- `-in2 (fname2)`: Input ASCII random particle file for the second set of tracer particles.
- `(nofznorm2)`: Total number of galaxies in the second set of tracer particles.
- `-cor12 (corname12)`: Input cross correlation function file between the two sets of random particles, as created by [Full Survey Correlation Functions](#).
- `-cor2 (corname2)`: Input autocorrelation function for the second set of particles, either user-defined or created by [Full Survey Correlation Functions](#).
- `-norm2 (nofznorm2)`: Number of galaxies in the second set of tracer particles. This is used to rescale the random particle covariances.

DEFAULT and JACKKNIFE mode Multi Field Parameters:

- `-jackknife12 (jk_weight_file12)`: Location of the `jackknife_weights_n{N}_m{M}_j{J}_12.dat` file containing the jackknife weights for each bin for the combination of random particle sets 1 and 2 (w_{aA}^{12}), as created by the `jackknife_weights` scripts.
- `-jackknife2 (jk_weight_file2)`: Location of the `jackknife_weights_n{N}_m{M}_j{J}_22.dat` file containing the jackknife weights for each bin for the second set of random particles (w_{aA}^{22}), as created by the `jackknife_weights` scripts.
- `-RRbin12 (RR_bin_file12)`: Location of the `binned_pair_counts_n{N}_m{M}_j{J}_12.dat` ASCII file containing the summed jackknife pair counts in each bin for the combination of random particle sets 1 and 2 (RR_{aA}^{12}), created by the `jackknife_weights` scripts.
- `-RRbin2 (RR_bin_file2)`: Location of the `binned_pair_counts_n{N}_m{M}_j{J}_22.dat` ASCII file containing the summed jackknife pair counts in each bin for the combination of random particle sets 1 and 2 (RR_{aA}^{22}), created by the `jackknife_weights` scripts.

LEGENDRE mode Multi Field Parameters:

- `phi_file12 (phi_file12)`: Location of the file containing the survey correction function parameters for the for the second field, as created by the [Computing Random Counts and Survey Correction Functions](#) script.
- `phi_file2 (phi_file2)`: Location of the file containing the survey correction function parameters for the for the combination of fields 1 and 2, as created by the [Computing Random Counts and Survey Correction Functions](#) script.

Optional Parameters

- `-mumin (mumin)`: Minimum μ binning to use in the analysis. (Default: 0, or -1 in 3PCF mode)
- `-mumax (mumax)`: Maximum μ binning to use in the analysis. (Default: 1)
- `-cf_loops (cf_loops)`: Number of iterations over which to refine the correlation functions.
- `(perbox)`: Boolean controlling whether we are using a periodic box. (Default: False)
- `-boxsize (boxsize)`: If creating particles randomly, this is the periodic size of the computational domain. If particles are read from file, this is set dynamically. (Default: 400)
- `-rescale (rescale)`: Factor by which to dilate the input positions. Zero or negative values cause this to be set to the boxsize. (Default: 1)
- `-xicut (xicutoff)`: The radius beyond which the correlation functions $\xi(r, \mu)$ are set to zero. (Default: 400)
- `-nmax (nmax)`: The maximum number of particles to read in from the random particle files. (Default: 1e12)
- `-save (savename)`: If `savename` is set, the cell selection probability grid is stored as `savename`. This must end in `.bin`. (Default: NULL)

- `-load (loadname)`: If set, load a cell selection probability grid computed in a previous run of RascalC. (Default: NULL)
- `-invert (qinvert)`: If this flag is passed to RascalC, all input particle weights are multiplied by -1. (Default: 0)
- `-balance (qbalance)`: If this flag is passed to RascalC, all negative weights are rescaled such that the total particle weight is 0. (Default: 0)
- `-np (np, make_random)`: If `make_random = 1`, this overrides any input random particle file and creates `np` randomly drawn particles in the cubic box. **NB**: The command line argument automatically sets `make_random = 1`. Currently creating particles at random is only supported for a single set of tracer particles.
- `-rs (rstart)`: If inverting particle weights, this sets the index from which to start weight inversion. (Default: 0)

1.8.3 Code Output

In the specified output directory, RascalC creates the directories `3PCFCovMatricesAll/` (3PCF mode), `CovMatricesAll/` (DEFAULT, LEGENDRE and JACKKNIFE modes) and `CovMatricesJack/` (JACKKNIFE mode) containing the relevant output matrix estimates. These contain multiple estimates of the each part of the total matrix and should be reconstructed using the [Post-Processing & Reconstruction](#) scripts.

The full output files take the following form (for N radial bins, M angular bins, maximum Legendre bin L and J non-zero jackknife regions, with `FIELDS` specifying the utilized tracer fields):

3PCF or LEGENDRE mode:

- `c{X}_n{N}_l{L}_{FIELDS}_{I}.txt`: I -th estimate of the X-point covariance matrix estimates, i.e. X^C . The summed covariance matrix terms have the suffix ‘full’.
- `binct_c{X}_n{N}_l{L}_{FIELDS}_{I}.txt`: Total used counts per bin for the X-point covariance matrix.
- `total_counts_n{N}_l{L}_{FIELDS}_{I}`: Total number of sets of particles attempted for the summed integral.

DEFAULT or JACKKNIFE mode:

- `c{X}_n{N}_m{M}_j{J}_{FIELDS}_{I}.txt`: I -th estimate of the X-point covariance matrix estimates, i.e. $C_{X,ab}$. The summed covariance matrix has the suffix ‘full’.
- `RR_n{N}_m{M}_{FIELDS}_{I}.txt`: I -th estimate of the (non-jackknife) RR_{ab}^{XY} pair counts which can be compared with `Corrfunc`.
- `binct_c{X}_n{N}_m{M}_{FIELDS}.txt`: Total used counts per bin for the X-point covariance matrix.
- `total_counts_n{N}_m{M}_{FIELDS}.txt`: Total number of pairs, triples and quads attempted for the summed integral.

JACKKNIFE mode only:

- `RR{P}_n{N}_m{M}_{FIELDS}.txt`: Estimate of RR_{ab} pair count for particles in random-subset P ($P \in [1, 2]$). This is used to compute the disconnected jackknife matrix term.
- `EE{P}_n{N}_m{M}_{FIELDS}.txt`: Estimate of EE_{ab} ξ -weighted pair count for particles in random-subset P . This is also used for the disconnected jackknife matrix term.

Each file is an ASCII format file containing the relevant matrices with the collapsed bin indices $\text{bin}_{\text{collapsed}} = \text{bin}_{\text{radial}} \times n_{\mu} + \text{bin}_{\text{angular}}$ (2PCF) or $\text{bin}_{\text{collapsed}} = (\text{bin}_{\text{radial},1} \times n_r + \text{bin}_{\text{radial},2}) \times n_{\mu} + \text{bin}_{\text{angular}}$ (3PCF) for a total of n_{μ} angular (or Legendre) bins and n_r radial bins.

1.9 Post-Processing & Reconstruction

These scripts post-process the single- or multi-field integrals computed by the C++ code described in [Covariance Matrix Estimation](#). For the DEFAULT, LEGENDRE and 3PCF modes, the output covariance matrix products (as well as precision matrices and effective number of mocks) are computed and saved to a .npz file, as described below. This is done for some input shot-noise rescaling parameter α (or one for each field, for the multi-field case).

In JACKKNIFE mode, the shot-noise rescaling parameter(s), α_i , are computed from data derived covariance matrices (from individual jackknife correlation functions computed in the [Jackknife Matrix Correlation Functions \(for the JACKKNIFE mode only\)](#) script). As above, a variety of analysis products are output as an .npz file.

NB: Before full computation of precision matrices and shot-noise rescaling, we check that the matrices have converged to a sufficient extent to allow convergence. As described in the RascalC paper, we require $\min \text{eig}(C_4) \geq -\min \text{eig}(C_2)$ to allow inversion. If this condition is met (due to too small sampling time, i.e. too low $\{N_i\}$ and/or N_{loops} values), the code exits. For multi-field cases, we require the (11,11) and (22,22) autocovariance matrices to be sufficiently converged.

General Input Parameters

- `{COVARIANCE_DIR}`: Directory containing the covariance matrix .txt files output by the [Covariance Matrix Estimation](#) C++ code. This directory should contain the subdirectories `CovMatricesAll` and `CovMatricesJack` containing the relevant analysis products.
- `{N_R_BINS}`: Number of radial bins used in the analysis.
- `{N_MU_BINS}`: Number of angular (μ) bins used in the analysis.
- `{MAX_L}`: Maximum Legendre multipole used in the analysis.
- `{N_SUBSAMPLES}`: Number of individual matrix subsamples computed in the C++ code. This is the N_{loops} parameter used in the main-code code. Individual matrix estimates are used to remove quadratic bias from the precision matrix estimates and compute the effective number of mocks N_{eff} .
- `{OUTPUT_DIR}`: Directory in which to save the analysis products. This will be created if not present.
- `{SHOT_NOISE_RESCALING}`, `{SHOT_NOISE_RESCALING_1}`, `{SHOT_NOISE_RESCALING_2}` (*DEFAULT, LEGENDRE and 3PCF modes*): Shot-noise rescaling parameter to be applied to the output covariance and precision matrices. This also affects the effective number of mocks, N_{eff} . If unspecified, this is set to unity.

Output

The single field scripts create a single compressed Python file `Rescaled_Covariance_Matrices_{MODE}_n{N}_m{M}_{j}{J}.npz` as an output in the given output directory. All 2PCF (3PCF) matrices follow the collapsed bin indexing $\text{bin}_{\text{collapsed}} = \text{bin}_{\text{radial}} \times n_{\mu} + \text{bin}_{\text{angular}}$ ($\text{bin}_{\text{collapsed}} = (\text{bin}_{\text{radial},1} \times n_r + \text{bin}_{\text{radial},2}) \times n_{\mu} + \text{bin}_{\text{angular}}$) for a total of n_{μ} angular bins and have dimension $n_{\text{bins}} \times n_{\text{bins}}$ for a total of n_{bins} bins. Precision matrices are computed using the quadratic bias elimination method of [O’Connell & Eisenstein 2018](#). All matrices are output using the optimal shot-noise rescaling parameter. This file may be large (up to 1GB) depending on the values of n_{bins} and N_{loops} used.

The output file has the following entries:

- `shot_noise_rescaling` (Float): Optimal value of the shot-noise rescaling parameter, α^* , from the \mathcal{L}_1 maximization.
- `full_theory_covariance` (np.ndarray): Theoretical full covariance matrix estimate $\hat{C}_{ab}(\alpha^*)$.
- `full_theory_precision` (np.ndarray): Associated precision matrix to the theoretical full covariance matrix estimate, $\Psi_{ab}(\alpha^*)$.
- `individual_theory_covariances` (list): List of individual (and independent) full theoretical covariance matrix estimates. These are used to compute \tilde{D}_{ab} and comprise $N_{\text{SUBSAMPLES}}$ estimates.

- `full_theory_D_matrix` (np.ndarray): Quadratic bias correction \tilde{D}_{ab} matrix for the full theoretical covariance matrix, as described in O’Connell & Eisenstein 2018.
- `N_eff` (Float): Effective number of mocks in the output full covariance matrix, N_{eff} , computed from \tilde{D}_{ab} .

For multi-field cases, we also create a single compressed Python file for the output analysis products, now labelled `Rescaled_Multi_Field_Covariance_Matrices_{MODE}_n{N}_m{M}_{_j{J}}.npz`, which contains output matrices for all combinations of the two fields. This could be a large file. This file has the same columns as the single field case, but now `shot_noise_rescaling` becomes a length-2 array (α_1^*, α_2^*). All other products are arrays of matrices (shape $2 \times 2 \times 2 \times 2 \times n_{\text{bins}} \times n_{\text{bins}}$) which are specified by 4 input parameters, corresponding to the desired X, Y, Z, W fields in $C^{XY,ZW}$. This uses Pythonic indexing from 0 to label the input fields. For example, we can access the $\Psi_{ab}^{11,21}$ precision matrix by loading the relevant column and specifying the index [0,0,1,0] e.g. to load this matrix we simply use:

```
>>> dat=np.load("Rescaled_Multi_Field_Covariance_Matrices_Jackknife_n36_m12_j169.npz
↳") # load the full data file
>>> full_precision = dat['full_theory_precision'] # load the precision matrix
>>> psi_1121 = full_precision[0,0,1,0] # specify the (11,21) component
```

1.9.1 DEFAULT, LEGENDRE and 3PCF mode reconstruction

Here we reconstruct the output covariance matrices and associated products, given an input shot-noise rescaling parameter. In 3PCF mode, we do not include the first six-point term, ${}^6_A C$, as noted in Philcox & Eisenstein (in prep.) since this is expected to be small for a large survey, yet difficult to accurately measure.

NB: In 3PCF mode, we require a long integration time for matrix convergence with even a moderate number of bins. If the matrix is not well converged (and invertible) the script will report a precision matrix and effective number of mocks of zero. In many analyses (e.g. Philcox & Eisenstein (in prep.)) the 3PCF covariance is compressed via some metric which improves the convergence. Thus, whilst the full matrix may not be invertible, the compressed version often will be. -

Usage

For a single field:

```
python python/post_process_default.py {COVARIANCE_DIR} {N_R_BINS} {N_MU_BINS} {N_
↳SUBSAMPLES} {OUTPUT_DIR} [{SHOT_NOISE_RESCALING}]
python python/post_process_legendre.py {COVARIANCE_DIR} {N_R_BINS} {MAX_L} {N_
↳SUBSAMPLES} {OUTPUT_DIR} [{SHOT_NOISE_RESCALING}]
python python/post_process_3pcf.py {COVARIANCE_DIR} {N_R_BINS} {MAX_L} {N_SUBSAMPLES}
↳{OUTPUT_DIR} [{SHOT_NOISE_RESCALING}]
```

For multiple fields:

```
python python/post_process_default_multi.py {COVARIANCE_DIR} {N_R_BINS} {N_MU_BINS}
↳{N_SUBSAMPLES} {OUTPUT_DIR} [{SHOT_NOISE_RESCALING_1} {SHOT_NOISE_RESCALING_2}]
python python/post_process_legendre_multi.py {COVARIANCE_DIR} {N_R_BINS} {MAX_L} {N_
↳SUBSAMPLES} {OUTPUT_DIR} [{SHOT_NOISE_RESCALING_1} {SHOT_NOISE_RESCALING_2}]
```

1.9.2 JACKKNIFE mode reconstruction

NB: This can only be run if the C++ code was run in JACKKNIFE mode for the 2PCF.

This script differs from the above in that we now compute the shot-noise rescaling parameters by comparing the theoretical jackknife covariance matrix $\hat{C}_{ab}^J(\alpha)$ with that computed from the data itself, using individual unrestricted jackknife estimates $\hat{\xi}_{aA}^J$. We define the data jackknife covariance matrix as $C_{ab}^{\text{data}} =$

$\sum_A w_{aA} w_{bA} \left(\hat{\xi}_{aA}^J - \bar{\xi}_a \right) \left(\hat{\xi}_{bA}^J - \bar{\xi}_b \right) / (1 - \sum_B w_{aB} w_{bB})$, where $\bar{\xi}_a$ is the mean correlation function in bin a . We compute α via minimizing the likelihood function $-\log \mathcal{L}_1(\alpha) = \text{trace}(\Psi^J(\alpha) C^{\text{data}}) - \log \det \Psi^J(\alpha) + \text{const.}$ using the (bias-corrected) precision matrix $\Psi^J(\alpha)$. When run for multiple input fields, the (11,11) and (22,22) covariance matrices are used to constrain α_1 and α_2 respectively.

Usage

For a single field:

```
python python/post-process_jackknife.py {XI_JACKKNIFE_FILE} {WEIGHTS_DIR} {COVARIANCE_
↪DIR} {N_MU_BINS} {N_SUBSAMPLES} {OUTPUT_DIR}
```

For multiple fields:

```
python python/post-process_jackknife_multi.py {XI_JACKKNIFE_FILE_11} {XI_JACKKNIFE_
↪FILE_12} {XI_JACKKNIFE_FILE_22} {WEIGHTS_DIR} {COVARIANCE_DIR} {N_MU_BINS} {N_
↪SUBSAMPLES} {OUTPUT_DIR}
```

Additional Jackknife Input Parameters

- {XI_JACKKNIFE_FILE}, {XI_JACKKNIFE_FILE_11}, {XI_JACKKNIFE_FILE_12}, {XI_JACKKNIFE_FILE_22}: Input ASCII file containing the correlation function estimates $\xi_A^J(r, \mu)$ for each jackknife region, as created by the *Jackknife Matrix Correlation Functions (for the JACKKNIFE mode only)* script. This has the format specified in *File Inputs*.
- {WEIGHTS_DIR}: Directory containing the jackknife weights and pair counts, as created by the *Computing Jackknife Weights* script. This must contain `jackknife_weights_n{N}_m{M}_j{J}_{INDEX}.dat` and `binned_pair_counts_n{N}_m{M}_j{J}_{INDEX}.dat` using the relevant covariance matrix binning scheme.

Output

The output `.npz` file contains the following additional columns;

- `jackknife_theory_covariance` (np.ndarray): Theoretical jackknife covariance matrix estimate $\hat{C}_{ab}^J(\alpha^*)$.
- `jackknife_data_covariance` (np.ndarray): Data-derived jackknife covariance matrix $\hat{C}_{ab}^{J,\text{data}}$, computed from the individual unrestricted jackknife correlation function estimates.
- `jackknife_theory_precision` (np.ndarray): Associated precision matrix to the theoretical jackknife covariance matrix estimate, $\Psi_{ab}^J(\alpha^*)$.

For any queries regarding the code please contact [Oliver Philcox](#).