Gundam Documentation

Release 0.8

Emilio Donoso

Feb 28, 2018

Contents

Contents:

CHAPTER 1

GUNDAM : A Toolkit For Fast Two-Point Correlation Functions

Gundam is a package to **count pairs** and **calculate spatial two-point correlation functions** of large galaxy samples. Its main features are :

Speed By calling Fortran routines that implement efficient skip-list/linked-list algorithms, Gundam can be extremely fast

Parallel Can automatically run in parallel to use all cores available. It employs the OpenMP framework to make use of multi-core CPUs

User-friendly By carefully wrapping Fortran code in a suitable Python framework, Gundam is very easy to use. A typical run consists of just 3 lines of code : (1) read data, (2) define parameters, (3) get counts

Error estimates, user-defined weights, fiber corrections Gundam can estimate bootstrap errors, weight pair counts, and even correct counts for fiber collisions

Plotting functions Gundam can produce nice, paper ready plots for 1D and 2D correlations, complete with ratios, labels and even power-law fits

Extensible Desgined in 3 layers of main, auxiliary and wrapper routines, it is quite easy to extend functionality by novice as well as seasoned users

Pair counts and correlation functions can be saved in ASCII files, as well as in a dictionary-like object that holds all calculations, input parameters and log messages. Share this object with your collaborators instead of just the final plot.

Though intended primarily for redshift surveys, it can also be adapted for simulation data and ultimately for any set of points in space.

@author: Emilio Donoso <edonoso@conicet.gov.ar>

CHAPTER 2

Installation

To install Gundam, you have two choices: (1) build from scratch, or (2) use pip. I recommend method (1), since it will allow easy access to modify or extend the Fortran counting routines. In any case, make sure to fulfill the required *Dependencies* before installation.

2.1 Option 1: Building from source

Just install the required dependencies, clone the Gundam repository and type *make*

```
git clone https://github.com/el_samotracio/gundam.git
cd gundam
make
```

By default this will compile and build the library in-place. Feel free to modify the *Makefile* to suit your needs. After compilation, you can optionally install the library in your default global-site packages directory

```
python setup.py install
```

or in your default user packages directory

```
python setup.py install --user
```

2.2 Option 2: Using pip (not yet functional!)

```
pip install gundam
```

2.3 Dependencies

0. Python: 3.5 or later
1. GCC compiler (Fortran & C)
2. Numpy, matplotlib, astropy, scipy
3. munch
4. pymorton (only needed when playing with Morton ordering, instead of the default "pixel" sorting)

Any of the above can be installed with pip or conda. A few other codes for Morton and Hilbert ordering are also included if you wish to experiment with alternative sorting methods, but are not really needed if you stick with the default sorting scheme).

Gundam has been tested under Anaconda 4.4.0, Python 3.6.1, Numpy 1.12.1 and gcc 7.1.1, running OpenSuse in an Intel platform.

CHAPTER 3

Introduction

3.1 1. Input Data

Input data for the main routines are astropy tables (see Data Tables), which provide all the functionality to manipulate tabular data, and to read/write in a variety of formats (ASCII, VOTable, FITS tables, etc.)

These input tables should have columns for **at least** angular coordinates and weights, but note the default names of columns can be overridden so there is no need to rename your original table (e.g. that infamous "RAJ_2000" instead of "ra"). Extra columns are welcome though as the name implies, are indeed extra and consume extra memory

Default name	Description
ra,dec	Right ascension and declination [deg]
z	Redshift (not needed in angular correlations)
wei	Weight of source
dcom	Comoving distance. Only used if `calcdist=False`

For example, data in FITS format can be read simply by

```
from astropy.table import Table
gals = Table.read('redgals.fits')  # read data
# If gals does not have a column for weights, just create one filled with 1's
gals['wei'] = 1.0
```

3.2 2. Set Up Input Parameters

Since there are quite a few parameters to deal, Gundam employs a special dictionary (see Munch) to pack and pass all of them at once. This dictionary also has attribute-like access with dot notation, meaning to access parameters you just type `par.omegam` (print matter density), `par.h0=100.` (set Hubble constant), etc. If you are used to ipython+tab completion you will certainly love this.

While you can create an input parameter dictionary from scratch, it is far easier to use `gundam.packpars()` to create a skeleton with default values, and then customize it to your needs. For example

```
import gundam as gun
par = gun.packpars(kind='pcf')     # Get defaults for a proj. corr. function
par.h0     = 69.5       # Change H0 [km/s/Mpc]
par.nsepp = 24          # Set 24 bins in projected separation [Mpc]
par.dsepp = 0.1         # Each of size 0.1 dex
par.estimator = 'LS'   # Pick Landy-Szalay estimator

# Can also specify values while creating the par object
par2 = gun.packpars(kind='acf', outfn='/test/angularcf')

# Quickly check all parameters by printing a nicely formated list
par2.qprint()
```

3.3 3. Get Counts & Correlations

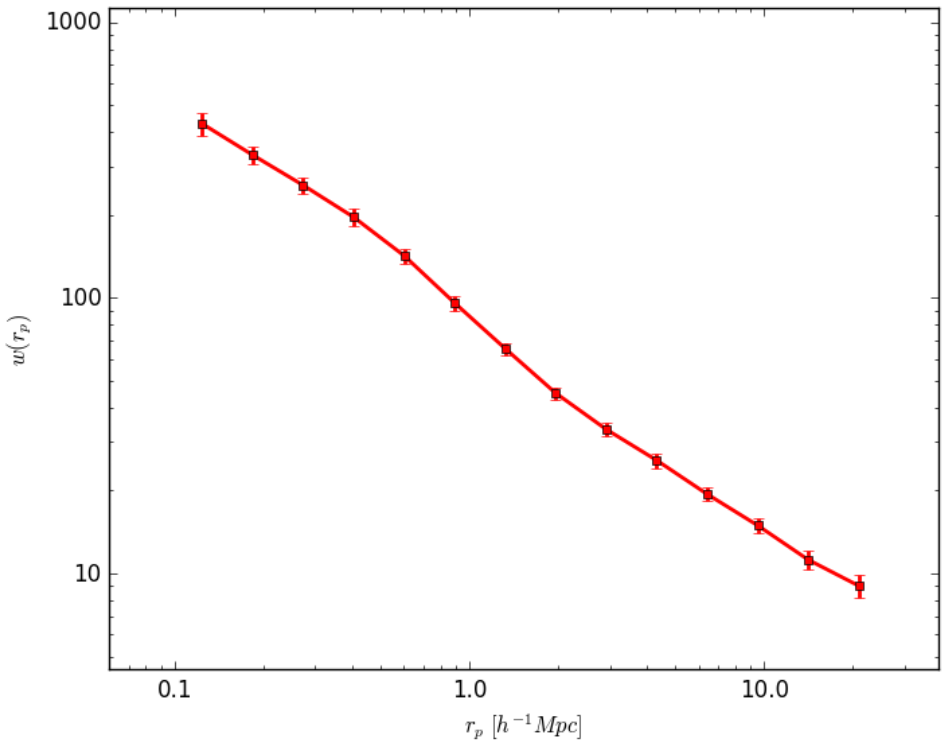
Having read your data (*gals* and *rans*) and set up input parameters *par*, getting a projected correlation function is as easy as

```
cnt = gun.pcf(gals, rans, par, nthreads=1)
```

The output object `cnt` is, again, a Munch dictionary that holds the correlation function `cnt.wrp`, the projected bins mid-point `cnt.rpm`, and many others. Here we show that the data and random samples had ~84k and 400k galaxies, respectively.

```
cnt.qprint()

# =================  Projected Correlation  =================
# Description :: Full projected autocorrelation
# npt         :: 84383
# npt1        :: 400000
# rpl         :: [ 0.1 ... 1.58113882522 ... 16.8522765348 ]
# rpm         :: [ 0.124173954464 ... 1.96336260485 ... 20.9261381905 ]
# rpr         :: [ 0.148347908929 ... 2.34558638447 ... 24.9999998462 ]
# wrp         :: [ 214.431631437 ... 22.5684208374 ... 4.51112595359 ]
# wrperr      :: [ 20.4245915831 ... 1.15641601016 ... 0.435505255 ]
# dd          :: [ 2082.0 ... 84.0 ... 26.0 ]
#                [ ...  ... ]
#                [ 1428490.29 ... 1136520.2 ... 823343.33 ]
# rr          :: [ 1084.0 ... 918.0 ... 749.0 ]
#                [ ...  ... ]
#                [ 26894309.95 ... 22249661.39 ... 17933389.42 ]
# ......
```

A plot is worth a thousand words, so let's do a good graphic of $w(r_p)$ by typing `gun.cntplot(cnt, factor=2.)` (the 2x factor is due to xxxx)



3.4 4. Going Parallel

To speed things up, Gundam can count pairs in parallel using multiple cores. Just set `nthreads` as in

```
cnt = gun.pcf(gals, rans, par, nthreads=8)
```

That's all. Under the hood, the software divides the counting process in several declinations stripes, computes the pairs in each, and adds everything up at the end. OpenMP threads are created and scheduled by the underlying Fortran code.

3.5 5. Typical Use Cases

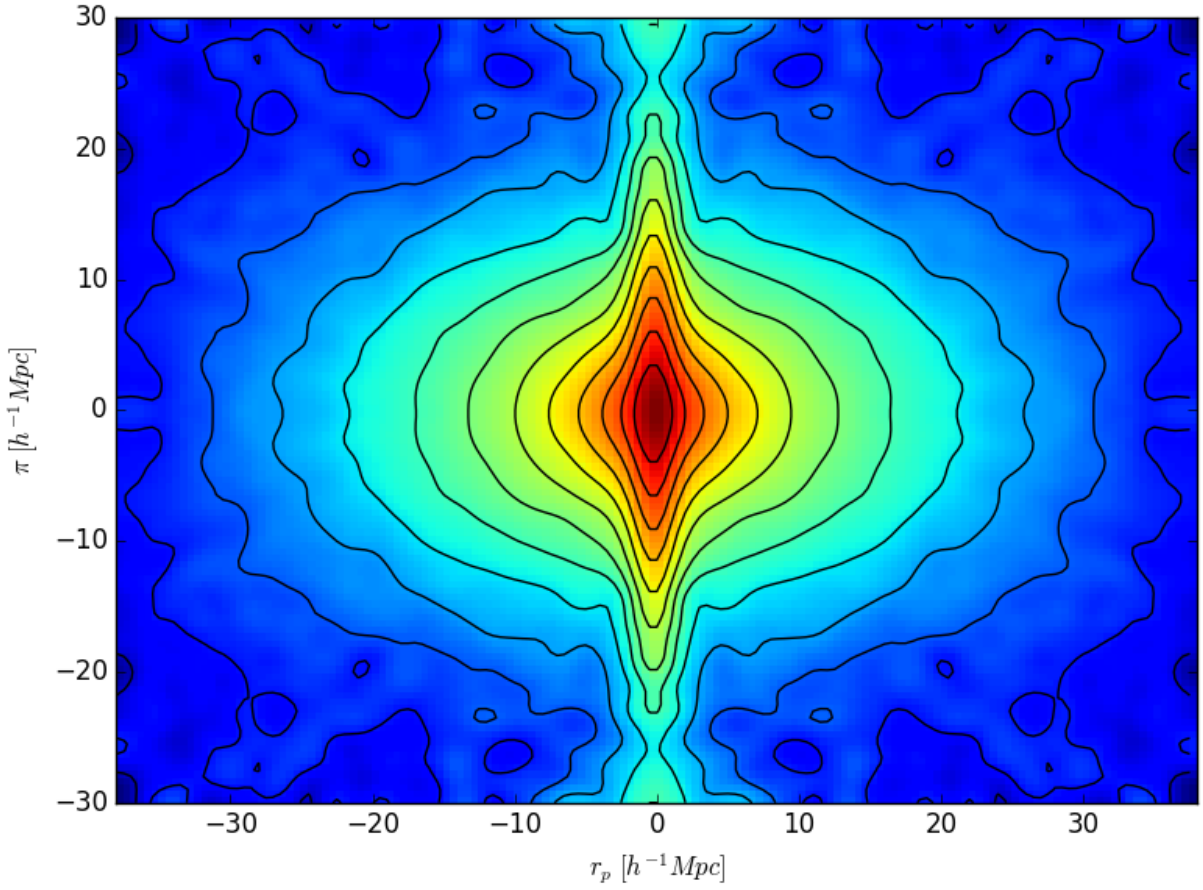
3.5.1 Check God's Fingers

Gundam can calculate and plot 2D correlation functions in a few lines. Let's see a self-explanatory example for 100k luminous red galaxies from SDSS DR7 (included in /examples directory)

```
from astropy.table import Table
import gundam as gun

# READ DATA
gals = Table.read('./examples/DR7-lrg.fits')
rans = Table.read('./examples/DR7-lrg-rand.fits')
gals['wei'] = 1.0
rans['wei'] = 1.0
```

```
# DEFINE INPUT PARAMETERS
par = gun.packpars(kind='pcf')
par.outfn     = './examples/LRGs'  # Base name of output files
par.estimator = 'LS'     # Choose Landy-Szalay estimator
par.nsepp     = 76       # Number of bins in projected separation rp
par.seppmin   = 0.01     # Minimum rp [Mpc/h]
par.dsepp     = 0.5      # Bin size in rp [Mpc/h]
par.logsepp   = False    # Use linear spaced bins
par.nsepv     = 60       # Number of bins in radial separation pi
par.dsepv     = 0.5      # Bin size in pi [Mpc/h]

# GET PCF
cnt = gun.pcf(gals, rans, par)

# PLOT A SMOOTHED 2D PCF
gun.cntplot2D(cnt, slevel=8)
```

which produces this cool figure. Anything familiar? Perhaps the Fingers of God? Kaiser squashing?



3.5.2 Lessons on Integration

So far so good, but how do you set the radial integration limit of w(rp)? There are two ways:

- **The long way** : you set radial bins (`nsepv`, `dsepv`) accordingly. For example, to integrate up to 40 Mpc make

40 bins of 1 Mpc with `nsepv=40`, `dsepv=1.0`

- **The short way** : you set radial bins (`nsepv`, `dsepv`) accordingly. For example, to integrate up to 40 Mpc make 1 bin of 40 Mpc with `nsepv=1`, `dsepv=40.`

No need to point out that the short way is faster. Hence, if you don't mind about intermediate bins just go straight with a single "fat" bin.

Note, however, that if you request a set of radial bins, i.e. `nsepv>1`, the code will: (1) calculate projected correlation function at each radial bin, and (2) sum each contribution. This can be different from adding the counts from all radial bins and then applying the estimator because empty bins are not necessarily the same in the DD, RR and DR terms. A single fat bin will have higher signal and less noise, especially at small separations.

3.5.3 Printing Nicely

While the dictionaries that store counts and/or parameters are useful objects, they do not print nicely due to amount and dimensions of the various arrays inside. Again, there are two ways to go around:

- Use `gun.qprint()` method

```
In [22]: cnt.qprint()
================  Projected Cross-correlation  ================
npt         :: 250000
npt1        :: 500000
npt2        :: 150000
rpl         :: [ 0.01 ... 0.56234132519 ... 23.7137370566 ]
rpm         :: [ 0.0116676071608 ... 0.656117767261 ... 27.6682568292 ]
rpr         :: [ 0.0133352143216 ... 0.749894209332 ... 31.6227766017 ]
wrp         :: [ 34.3333333333 ... 84.183601623 ... -3.04940643393 ]
wrperr      :: [ 0.0 ... 0.0 ... 0.0 ]
cd          :: [ 33.0 ... 0.0 ... 0.0 ]
               [ ...  ... ]
               [ 1107801.0 ... 1045875.0 ... 980030.0 ]
cr          :: [ 2.0 ... 0.0 ... 0.0 ]
               [ ...  ... ]
               [ 2291303.0 ... 2269425.0 ... 2228177.0 ]
par         :: {
               "autogrid": true,
               "bseed": 12345,
               "calcdist": true,
               "cdcom": "dcom",
               ...
               }
```

- Use `gun.cnttable()` routine to pop up a table of counts in your browser

rpl	rpm	rpr	wrp	wrperr	cd_000	cd_001	cd_002	cd_003	cd_004	cd_005	cd_
0.0100	0.0117	0.0133	34.3333	0.0000	33.00	47.00	19.00	23.00	6.00	9.00	2.00
0.0133	0.0156	0.0178	187.5000	0.0000	59.00	44.00	31.00	21.00	12.00	5.00	3.00
0.0178	0.0207	0.0237	286.4000	0.0000	75.00	72.00	51.00	36.00	23.00	10.00	15.00
0.0237	0.0277	0.0316	307.5333	0.0000	116.00	93.00	74.00	61.00	44.00	36.00	22.00
0.0316	0.0369	0.0422	345.6577	0.0000	172.00	155.00	98.00	97.00	44.00	39.00	36.00
0.0422	0.0492	0.0562	1043.8752	0.0000	347.00	292.00	191.00	150.00	96.00	108.00	64.00
0.0562	0.0656	0.0750	375.8299	0.0000	653.00	421.00	308.00	222.00	209.00	170.00	138.0
0.0750	0.0875	0.1000	382.5289	0.0000	1083.00	821.00	641.00	471.00	391.00	305.00	185.0
0.1000	0.1167	0.1334	378.2244	0.0000	1682.00	1353.00	1037.00	772.00	596.00	473.00	394.0
0.1334	0.1556	0.1778	250.1667	0.0000	2372.00	1915.00	1477.00	1153.00	965.00	721.00	609.0
0.1778	0.2075	0.2371	222.8686	0.0000	3037.00	2611.00	2166.00	1649.00	1371.00	1122.00	900.0
0.2371	0.2767	0.3162	170.7156	0.0000	4419.00	3440.00	3075.00	2495.00	1819.00	1832.00	1395.
0.3162	0.3690	0.4217	142.4552	0.0000	5840.00	5011.00	4151.00	3555.00	2916.00	2487.00	1963.
0.4217	0.4920	0.5623	109.2964	0.0000	7825.00	7038.00	5635.00	4886.00	4041.00	3343.00	2907.
0.5623	0.6561	0.7499	84.1836	0.0000	10941.00	9328.00	7481.00	6451.00	5800.00	4981.00	4162.
0.7499	0.8749	1.0000	60.6537	0.0000	14666.00	12882.00	10602.00	9064.00	7821.00	6913.00	5858.
1.0000	1.1668	1.3335	44.2873	0.0000	20033.00	17067.00	14396.00	12579.00	10482.00	9413.00	8070.
1.3335	1.5559	1.7783	30.7463	0.0000	26554.00	23666.00	20398.00	17559.00	15201.00	12891.00	11293
1.7783	2.0748	2.3714	22.8146	0.0000	36633.00	33350.00	28978.00	25275.00	21639.00	19505.00	17112
2.3714	2.7668	3.1623	16.7931	0.0000	51996.00	48530.00	43207.00	38782.00	34267.00	29905.00	26782
3.1623	3.6896	4.2170	11.8360	0.0000	71817.00	68452.00	63739.00	56775.00	51649.00	46677.00	42022
4.2170	4.9202	5.6234	8.4795	0.0000	102299.00	99221.00	92945.00	86050.00	79749.00	73615.00	67105
5.6234	6.5612	7.4989	5.4641	0.0000	144152.00	141415.00	135461.00	129158.00	121452.00	115060.00	10672
7.4989	8.7495	10.0000	3.3000	0.0000	209071.00	207274.00	201830.00	196371.00	189485.00	182383.00	17316

Or you can always try your luck using (i)python regular print

```
In [21]: cnt
Out[21]:
Munch({'cd': array([[  3.30000000e+01,   4.70000000e+01,   1.90000000e+01, ...,   0.00000000e+00,   0.00000000e+00,
          0.00000000e+00],
       [  5.90000000e+01,   4.40000000e+01,   3.10000000e+01, ...,   0.00000000e+00,   0.00000000e+00,
          0.00000000e+00],
       [  7.50000000e+01,   7.20000000e+01,   5.10000000e+01, ...,   0.00000000e+00,   0.00000000e+00,
          0.00000000e+00],
       ...,
       [  4.49333000e+05,   4.49393000e+05,   4.47641000e+05, ...,   3.21611000e+05,   3.22087000e+05,
          3.22534000e+05],
       [  6.95893000e+05,   6.93519000e+05,   6.94848000e+05, ...,   5.68332000e+05,   5.65434000e+05,
          5.65063000e+05],
       [  1.10780100e+06,   1.10799200e+06,   1.10531600e+06, ...,   9.89658000e+05,   9.83078000e+05,
          9.80030000e+05]]), 'wrp': array([  3.43333333e+01,   1.87500000e+02,   2.86400000e+02,   3.07533333e+02,   3.45657
        1.04387523e+03,   3.75829939e+02,   3.82528871e+02,   3.78224407e+02,   2.50166671e+02,
        2.22868635e+02,   1.70715576e+02,   1.42455216e+02,   1.09296430e+02,   8.41836016e+01,
        6.06536886e+01,   4.42873136e+01,   3.07463370e+01,   2.28146011e+01,   1.67931369e+01,
        1.18360480e+01,   8.47951005e+00,   5.46414581e+00,   3.30001289e+00,   9.92689371e-01,
       -7.42013873e-01,  -2.10577365e+00,  -3.04940643e+00]), 'rpr': array([  1.33352143e-02,   1.77827941e-02,   2.3713737
4.21696503e-02,
        5.62341325e-02,   7.49894209e-02,   1.00000000e-01,   1.33352143e-01,   1.77827941e-01,
        2.37137371e-01,   3.16227766e-01,   4.21696503e-01,   5.62341325e-01,   7.49894209e-01,
        1.00000000e+00,   1.33352143e+00,   1.77827941e+00,   2.37137371e+00,   3.16227766e+00,
        4.21696503e+00,   5.62341325e+00,   7.49894209e+00,   1.00000000e+01,   1.33352143e+01,
        1.77827941e+01,   2.37137371e+01,   3.16227766e+01]), 'cr': array([[  2.00000000e+00,   0.00000000e+00,   0.0000000
2.00000000e+00,
        0.00000000e+00],
       [  1.00000000e+00,   1.00000000e+00,   0.00000000e+00, ...,   0.00000000e+00,   0.00000000e+00,
          1.00000000e+00],
       [  3.00000000e+00,   1.00000000e+00,   0.00000000e+00, ...,   0.00000000e+00,   0.00000000e+00,
          1.00000000e+00],
       ...,
       [  7.61524000e+05,   7.61925000e+05,   7.60559000e+05, ...,   7.40701000e+05,   7.39385000e+05,
          7.39888000e+05]
```

3.5.4 Further Examples

Data and code for 3 examples of using Gundam are provide in the repo (example_lrg.py, example_pcf.py and example_redblue.py).

3.6 6. Coordinates & Distances

The radial, projected and redshift-space distance between two galaxies *i* and *j* are calculated as

$$\pi = |dc_i - dc_j|$$

$$r_p^2 = 4dc_i dc_j[(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2]$$

$$s^2 = \pi^2 + r_p^2$$

where *dc* is the comoving distance in the chosen cosmology and *(x, y, z)* are the rectangular coordinates given by

$$x = 0.5\cos(dec)\sin(ra)$$

$$y = 0.5\cos(dec)\cos(ra)$$

$$z = 0.5\sin(dec)$$

By default the comoving distances are calculated with astropy's cosmology module using a FLRW cosmology with a cosmological constant and curvature (LambdaCDM). If you prefer another, just modify the corresponding code in the main Gundam routines, or even better, append your own distances to the input tables and set `calcdist=False`

3.7 7. Routines, Cells & Counts

All Fortran routines are stored in the *cflibfor* library, under the module called *mod*. Feel free to directly use these, for example

```
import cfibfor as cff
cff.mod.bootstrap(10,4,124567)
```

or through Gundam

```
import gundam as gun
gun.cff.mod.bootstrap(10,4,124567)
```

Of course the number of cells to use (i.e. `mxh1`, `mxh2`, `mxh3`) has some impact in the performance and the optimum values depends on the sample characteristics, the binning adopted and even the hardware employed. Gundam will try to guess values for these parameters based on simple fittings to galaxy data extracted from the Millennium Simulation. They should work well as starting values for many use cases but depending your needs, you might want to fine tune these. Just remember to keep it reasonable. For example, if you have half million objects, setting `mxh1=4` or `mxh1=400` is not wise in most cases. Expect typical variations of 3-30% in performance for a range of *reasonable* values.

Note the counting routines actually return **half** of real pairs, so depending the case you might want to multiply by 2. The estimators for all implemented correlation functions **already do this** for you.

CHAPTER 4

Input Parameters

4.1 Projected Space

4.1.1 Input Parameters Dictionary (pcf)

Dictionary with attribute-style access, that stores all input parameters for the code, plus some useful run time information during output (see outaddrt)

Parameter	Description
kind	Kind of correlation function ('pcf')
h0	Hubble constant [km/s/Mpc]. Default=100.
omegam	Omega matter. Default=0.3
omegal	Omega lambda. Default=0.7
autogrid	If `True` choose the optimum nr. of cells `(mxh1,mxh2,mxh3)` of the skip table (SK). Default=True
dens	Custom nr. of particles per SK cell used when `autogrid=True`. No need to specify unless desired. Default=None
mxh1	Nr. of DEC cells of the SK table. Only relevant if `autogrid=False`. Default=30
mxh2	Nr. of RA cells of the SK table. Only relevant if `autogrid=False`. Default=180
mxh3	Nr. of DCOM cells of the SK table. Only relevant if `autogrid=False`. See notemxh3 below. Default=40
pxorder	Data ordering method. See *Pixel Ordering Methods* for details and options. Default='natural'
doboot	If `True`, calculate bootstrap counts and error bars. Default=False
nbts	Nr. of bootstrap samples. Only relevant if `doboot=True`. Default=50
bseed	Fixed seed for boostrap RNG. Always set `bseed>0` if running in paralell. Default=12345
wfib	If `True` apply SDSS fiber correction for pairs closer than 55". See *wfiber()* Fortran function. Default=False
nsepp	Nr. of projected separation bins. Default=22
seppmin	Minimum projected distance to consider [Mpc/h]. Default=0.01
dsepp	Size of projected bins (in dex if log bins). Default=0.15
logsepp	If `True` use log-spaced bins. Otherwise use linear-spaced bins. Default=True
nsepv	Nr. of radial separation bins. Default=1
dsepv	Size of radial bins (linear) [Mpc/h]. Default=40.
calcdist	If `False`, take comov. distances from input tables instead of calculating them. Default=True
file	File name of **data** sample. Only informative. Default=''
file1	File name of **random** sample. Only informative. Default=''
description	Short description of the run. Only informative. Default=''
estimator	Statistical estimator of the correlation function. Default='NAT' * 'NAT' : Natural estimator -> $DD/RR-1$ * 'HAM' : Hamilton estimator -> $DD*RR/DR^2-1$ * 'LS' : Landy-Szalay estimator -> $(DD-2DR+RR)/RR$ * 'DP' : Davis-Peebles estimator -> $DD/DR-1$
cra,cdec, cred,cwei, cdcom	Column names in **data** sample table (tab). Default=('ra' , 'dec', 'z', 'wei', 'dcom')
cra1,cdec1, cred1,cwei1 cdcom1	Column names in **random** sample (tab1). Default=('ra' , 'dec', 'z', 'wei', 'dcom')
custRA-bound	Specify custom RA boundaries for samples that cross the RA=0 limit. See *Custom RA Boundaries*. Default=None
outfn	Base name for all output files (e.g. /home/myuser/redagn)

4.1.2 Input Parameters Dictionary (pccf)

Dictionary with attribute-style access, that stores all input parameters for the code, plus some useful run time information during output (see outaddrt)

Parameter	Description
kind	Kind of correlation function ('pccf')
h0	Hubble constant [km/s/Mpc]. Default=100.

Continu

Table 4.1 – continued from previous page

Parameter	Description
omegam	Omega matter. Default=0.3
omegal	Omega lambda. Default=0.7
autogrid	If `autogrid=True` choose the optimum nr. of cells `(mxh1,mxh2,mxh3)` of the skip table (S
dens	Custom nr. of particles per SK cell used when `autogrid=True`. No need to specify unless desi
mxh1	Nr. of DEC cells of the SK table. Only relevant if `autogrid=False`. Default=30
mxh2	Nr. of RA cells of the SK table. Only relevant if `autogrid=False`. Default=180
mxh3	Nr. of DCOM cells of the SK table. Only relevant if `autogrid=False`. See notemxh3 below. I
pxorder	Data ordering method. See *Pixel Ordering Methods* for details and options. Default='natural'
doboot	If `True`, calculate bootstrap counts and error bars. Default=False
nbts	Nr. of bootstrap samples. Only relevant if `doboot=True`. Default=50
bseed	Fixed seed for boostrap RNG. Always set `bseed>0` if running in paralell. Default=1245
wfib	If `True` apply SDSS fiber correction for pairs closer than 55". See *wfiber()* Fortran function. Defa
nsepp	Nr. of projected separation bins. Default=22
seppmin	Minimum projected distance to consider [Mpc/h]. Default=0.01
dsepp	Size of projected bins (in dex if log bins). Default=0.15
logsepp	If `True` use log-spaced bins. Otherwise use linear-spaced bins. Default=True
nsepv	Nr. of radial separation bins. Default=1
dsepv	Size of radial bins (linear) [Mpc/h]. Default=40.
calcdist	If `False`, take comov. distances from input tables instead of calculating them. Default=True
file	File name of **data** sample. Only informative. Default=''
file1	File name of **random** sample. Only informative. Default=''
file2	File name of **cross** sample. Only informative. Default=''
description	Short description of the run. Only informative. Default=''
estimator	Statistical estimator of the correlation function. Default='DP' * 'DP' : Davis-Peebles estimator ->
cra,cdec, cred,cwei, cdcom	Column names in **data** sample table (tab). Default=('ra' , 'dec', 'z', 'wei', 'dcom')
cra1,cdec1, cred1,cwei1 cdcom1	Column names in **random** sample (tab1). Default=('ra' , 'dec', 'z', 'wei', 'dcom')
cra2,cdec2, cred2,cwei2 cdcom2	Column names in **cross** sample table (tab2). Default=('ra' , 'dec', 'z', 'wei', 'dcom')
custRAbound	Specify custom RA boundaries for samples that cross the RA=0 limit. See *Custom RA Boundaries*
outfn	Base name for all output files (e.g. /home/myuser/redagn)

4.1.3 Input Parameters Dictionary (rppiA)

Dictionary with attribute-style access, that stores all input parameters for the code, plus some useful run time information during output (see outaddrt)

Parameter	Description
kind	Kind of correlation function ('rppiA')
h0	Hubble constant [km/s/Mpc]. Default=100.
omegam	Omega matter. Default=0.3
omegal	Omega lambda. Default=0.7
autogrid	If `autogrid=True` choose the optimum nr. of cells `(mxh1,mxh2,mxh3)` of the skip table (SK). Default=True
dens	Custom nr. of particles per SK cell used when `autogrid=True`. No need to specify unless desired. Default=None
mxh1	Nr. of DEC cells of the SK table. Only relevant if `autogrid=False`. Default=30
mxh2	Nr. of RA cells of the SK table. Only relevant if `autogrid=False`. Default=180
mxh3	Nr. of DCOM cells of the SK table. Only relevant if `autogrid=False`. See notemxh3 below. Default=40
pxorder	Data ordering method. See *Pixel Ordering Methods* for details and options. Default='natural'
doboot	If `True`, calculate bootstrap counts and error bars. Default=False
nbts	Nr. of bootstrap samples. Only relevant if `doboot=True`. Default=50
bseed	Fixed seed for boostrap RNG. Always set `bseed>0` if running in paralell. Default=1245
wfib	If `True` apply SDSS fiber correction for pairs closer than 55". See *wfiber()* Fortran function. Default=False
nsepp	Nr. of projected separation bins. Default=22
seppmin	Minimum projected separation to consider [Mpc/h]. Default=0.01
dsepp	Size of projected bins (in dex if log bins). Default=0.15
logsepp	If `True` use log-spaced bins. Otherwise use linear-spaced bins. Default=True
nsepv	Nr. of radial separation bins. Default=1
dsepv	Size of radial bins (linear) [Mpc/h]. Default=40.
calcdist	If `False`, take comov. distances from input tables instead of calculating them. Default=True
file	File name of data sample. Only informative. Default=''
description	Short description of the run. Only informative. Default=''
cra,cdec, cred,cwei, cdcom	Column names in th sample table (tab). Default=('ra' , 'dec', 'z', 'wei', 'dcom')
custRAbound	Specify custom RA boundaries for samples that cross the RA=0 limit. See *Custom RA Boundaries*. Default=None
outfn	Base name for all output files (e.g. /home/myuser/redagn)

4.1.4 Input Parameters Dictionary (rppiC)

Dictionary with attribute-style access, that stores all input parameters for the code, plus some useful run time information during output (see outaddrt)

Parameter	Description
kind	Kind of correlation function ('rppiC')
h0	Hubble constant [km/s/Mpc]. Default=100.
omegam	Omega matter. Default=0.3
omegal	Omega lambda. Default=0.7
autogrid	If `autogrid=True` choose the optimum nr. of cells `(mxh1,mxh2,mxh3)` of the skip table (SK). Default=True
dens	Custom nr. of particles per SK cell used when `autogrid=True`. No need to specify unless desired. Default=None
mxh1	Nr. of DEC cells of the SK table. Only relevant if `autogrid=False`. Default=30
mxh2	Nr. of RA cells of the SK table. Only relevant if `autogrid=False`. Default=180
mxh3	Nr. of DCOM cells of the SK table. Only relevant if `autogrid=False`. See notemxh3 below. Default=40
pxorder	Data ordering method. See *Pixel Ordering Methods* for details and options. Default='natural'
doboot	If `True`, calculate bootstrap counts and error bars. Default=False
nbts	Nr. of bootstrap samples. Only relevant if `doboot=True`. Default=50
bseed	Fixed seed for boostrap RNG. Always set `bseed>0` if running in paralell. Default=1245
wfib	If `True` apply SDSS fiber correction for pairs closer than 55". See *wfiber()* Fortran function. Default=False
nsepp	Nr. of projected separation bins. Default=22
seppmin	Minimum projected distance to consider [Mpc/h]. Default=0.01
dsepp	Size of projected bins (in dex if log bins). Default=0.15
logsepp	If `True` use log-spaced bins. Otherwise use linear-spaced bins. Default=True
nsepv	Nr. of radial separation bins. Default=1
dsepv	Size of radial bins (linear) [Mpc/h]. Default=40.
calcdist	If `False`, take comov. distances from input tables instead of calculating them. Default=True
file	File name of **data** sample. Only informative. Default=''
file1	File name of **random** sample. Only informative. Default=''
description	Short description of the run. Only informative. Default=''
cra,cdec, cred,cwei, cdcom	Column names in **data** sample table (tab). Default=('ra' , 'dec', 'z', 'wei', 'dcom')
cra1,cdec1, cred1,cwei1 cdcom1	Column names in **random** sample (tab1). Default=('ra' , 'dec', 'z', 'wei', 'dcom')
custRAbound	Specify custom RA boundaries for samples that cross the RA=0 limit. See *Custom RA Boundaries*. Default=None
outfn	Base name for all output files (e.g. /home/myuser/redagn)

4.1.5 Note on mxh3

Due to performance reasons, the number of cells in the radial (comoving) distance actually used to build the skip table is always set as `mxh3=int((dcmax-dcmin)/rvmax)`. Hence, the parameter `mxh3` supplied at input will be ignored unless it is smaller than this optimum value.

Note however that `mxh3` is only relevant for the performance of the algorithms. It is **not** related with the number of radial bins `nsepv` where we want to get output counts.

4.2 Redshift Space

4.2.1 Input Parameters Dictionary (rcf)

Dictionary with attribute-style access, that stores all input parameters for the code, plus some useful run time information during output (see outaddrt)

Parameter	Description
kind	Kind of correlation function ('rcf')
h0	Hubble constant [km/s/Mpc]. Default=100.
omegam	Omega matter. Default=0.3
omegal	Omega lambda. Default=0.7
autogrid	If `autogrid=True` choose the optimum nr. of cells `(mxh1,mxh2,mxh3)` of the skip table (SK). Default=True
dens	Custom nr. of particles per SK cell used when `autogrid=True`. No need to specify unless desired. Default=None
mxh1	Nr. of DEC cells of the SK table. Only relevant if `autogrid=False`. Default=30
mxh2	Nr. of RA cells of the SK table. Only relevant if `autogrid=False`. Default=180
mxh3	Nr. of DCOM cells of the SK table. Only relevant if `autogrid=False`. See notemxh3 below. Default=40
pxorder	Data ordering method. See *Pixel Ordering Methods* for details and options. Default='natural'
doboot	If `True`, calculate bootstrap counts and error bars. Default=False
nbts	Nr. of bootstrap samples. Only relevant if `doboot=True`. Default=50
bseed	Fixed seed for boostrap RNG. Always set `bseed>0` if running in paralell. Default=12345
wfib	If `True` apply SDSS fiber correction for pairs closer than 55". See *wfiber()* Fortran function. Default=False
nseps	Nr. of redshift space bins. Default=22
sepsmin	Minimum separation to consider [Mpc/h]. Default=0.01
dseps	Size of redshift space bins (in dex if log bins). Default=0.15
logseps	If `True` use log-spaced bins. Otherwise use linear-spaced bins. Default=True
calcdist	If `False`, take comov. distances from input tables instead of calculating them. Default=True
file	File name of **data** sample. Only informative. Default=''
file1	File name of **random** sample. Only informative. Default=''
description	Short description of the run. Only informative. Default=''
estimator	Statistical estimator of the correlation function. Default='NAT' * 'NAT' : Natural estimator -> $DD/RR-1$ * 'HAM' : Hamilton estimator -> $DD*RR/DR^2-1$ * 'LS' : Landy-Szalay estimator -> $(DD-2DR+RR)/RR$ * 'DP' : Davis-Peebles estimator -> $DD/DR-1$
cra,cdec, cred,cwei, cdcom	Column names in **data** sample table (tab). Default=('ra' , 'dec', 'z', 'wei', 'dcom')
cra1,cdec1, cred1,cwei1 cdcom1	Column names in **random** sample (tab1). Default=('ra' , 'dec', 'z', 'wei', 'dcom')
custRA-bound	Specify custom RA boundaries for samples that cross the RA=0 limit. See *Custom RA Boundaries*. Default=None
outfn	Base name for all output files (e.g. /home/myuser/redagn)

4.2.2 Input Parameters Dictionary (rccf)

Dictionary with attribute-style access, that stores all input parameters for the code, plus some useful run time information during output (see outaddrt)

Parameter	Description
kind	Kind of correlation function ('rccf')
h0	Hubble constant [km/s/Mpc]. Default=100.
omegam	Omega matter. Default=0.3
omegal	Omega lambda. Default=0.7
autogrid	If `autogrid=True` choose the optimum nr. of cells `(mxh1,mxh2,mxh3)` of the skip table (SK). Default=True
dens	Custom nr. of particles per SK cell used when `autogrid=True`. No need to specify unless desired. Default=None
mxh1	Nr. of DEC cells of the SK table. Only relevant if `autogrid=False`. Default=30
mxh2	Nr. of RA cells of the SK table. Only relevant if `autogrid=False`. Default=180
mxh3	Nr. of DCOM cells of the SK table. Only relevant if `autogrid=False`. See notemxh3 below. Default=40
pxorder	Data ordering method. See *Pixel Ordering Methods* for details and options. Default='natural'
doboot	If `True`, calculate bootstrap counts and error bars. Default=False
nbts	Nr. of bootstrap samples. Only relevant if `doboot=True`. Default=50
bseed	Fixed seed for boostrap RNG. Always set `bseed>0` if running in paralell. Default=1245
wfib	If `True` apply SDSS fiber correction for pairs closer than 55". See *wfiber()* Fortran function. Default=False
nseps	Nr. of redshift space bins. Default=22
sepsmin	Minimum separation to consider [Mpc/h]. Default=0.01
dseps	Size of redshift space bins (in dex if log bins). Default=0.15
logseps	If `True` use log-spaced bins. Otherwise use linear-spaced bins. Default=True
calcdist	If `False`, take comov. distances from input tables instead of calculating them. Default=True
file	File name of data sample. Only informative. Default=''
file1	File name of random sample. Only informative. Default=''
file2	File name of cross sample. Only informative. Default=''
description	Short description of the run. Only informative. Default=''
estimator	Statistical estimator of the correlation function. Default='DP' * 'DP' : Davis-Peebles estimator -> $CD/CR - 1$
cra,cdec, cred,cwei, cdcom	Column names in **data** sample table (tab). Default=('ra' , 'dec', 'z', 'wei', 'dcom')
cra1,cdec1, cred1,cwei1 cdcom1	Column names in **random** sample (tab1). Default=('ra' , 'dec', 'z', 'wei', 'dcom')
cra2,cdec2, cred2,cwei2 cdcom2	Column names in **cross** sample table (tab2). Default=('ra' , 'dec', 'z', 'wei', 'dcom')
custRAbound	Specify custom RA boundaries for samples that cross the RA=0 limit. See *Custom RA Boundaries*. Default=None
outfn	Base name for all output files (e.g. /home/myuser/redagn)

4.2.3 Input Parameters Dictionary (sA)

Dictionary with attribute-style access, that stores all input parameters for the code, plus some useful run time information during output (see outaddrt)

Parameter	Description
kind	Kind of correlation function ('sA')
h0	Hubble constant [km/s/Mpc]. Default=100.
omegam	Omega matter. Default=0.3
omegal	Omega lambda. Default=0.7
autogrid	If `autogrid=True` choose the optimum nr. of cells `(mxh1,mxh2,mxh3)` of the skip table (SK). Default=True
dens	Custom nr. of particles per SK cell used when `autogrid=True`. No need to specify unless desired. Default=None
mxh1	Nr. of DEC cells of the SK table. Only relevant if `autogrid=False`. Default=30
mxh2	Nr. of RA cells of the SK table. Only relevant if `autogrid=False`. Default=180
mxh3	Nr. of DCOM cells of the SK table. Only relevant if `autogrid=False`. See notemxh3 below. Default=40
pxorder	Data ordering method. See *Pixel Ordering Methods* for details and options. Default='natural'
doboot	If `True`, calculate bootstrap counts and error bars. Default=False
nbts	Nr. of bootstrap samples. Only relevant if `doboot=True`. Default=50
bseed	Fixed seed for boostrap RNG. Always set `bseed>0` if running in paralell. Default=1245
wfib	If `True` apply SDSS fiber correction for pairs closer than 55". See *wfiber()* Fortran function. Default=False
nseps	Nr. of redshift space bins. Default=22
sepsmin	Minimum separation to consider [Mpc/h]. Default=0.01
dseps	Size of redshift space bins (in dex if log bins). Default=0.15
logseps	If `True` use log-spaced bins. Otherwise use linear-spaced bins. Default=True
calcdist	If `False`, take comov. distances from input tables instead of calculating them. Default=True
file	File name of **data** sample. Only informative. Default=''
description	Short description of the run. Only informative. Default=''
cra,cdec, cred,cwei, cdcom	Column names in th sample table (tab). Default=('ra' , 'dec', 'z', 'wei', 'dcom')
custRAbound	Specify custom RA boundaries for samples that cross the RA=0 limit. See *Custom RA Boundaries*. Default=None
outfn	Base name for all output files (e.g. /home/myuser/redagn)

4.2.4 Input Parameters Dictionary (sC)

Dictionary with attribute-style access, that stores all input parameters for the code, plus some useful run time information during output (see outaddrt)

Parameter	Description
kind	Kind of correlation function ('sC')
h0	Hubble constant [km/s/Mpc]. Default=100.
omegam	Omega matter. Default=0.3
omegal	Omega lambda. Default=0.7
autogrid	If `autogrid=True` choose the optimum nr. of cells `(mxh1,mxh2,mxh3)` of the skip table (SK). Default=True
dens	Custom nr. of particles per SK cell used when `autogrid=True`. No need to specify unless desired. Default=None
mxh1	Nr. of DEC cells of the SK table. Only relevant if `autogrid=False`. Default=30
mxh2	Nr. of RA cells of the SK table. Only relevant if `autogrid=False`. Default=180
mxh3	Nr. of DCOM cells of the SK table. Only relevant if `autogrid=False`. See notemxh3 below. Default=40
pxorder	Data ordering method. See *Pixel Ordering Methods* for details and options. Default='natural'
doboot	If `True`, calculate bootstrap counts and error bars. Default=False
nbts	Nr. of bootstrap samples. Only relevant if `doboot=True`. Default=50
bseed	Fixed seed for boostrap RNG. Always set `bseed>0` if running in paralell. Default=1245
wfib	If `True` apply SDSS fiber correction for pairs closer than 55". See *wfiber()* Fortran function. Default=False
nseps	Nr. of redshift space bins. Default=22
sepsmin	Minimum separation to consider [Mpc/h]. Default=0.01
dseps	Size of redshift space bins (in dex if log bins). Default=0.15
logseps	If `True` use log-spaced bins. Otherwise use linear-spaced bins. Default=True
calcdist	If `False`, take comov. distances from input tables instead of calculating them. Default=True
file	File name of data sample. Only informative. Default=''
file1	File name of random sample. Only informative. Default=''
description	Short description of the run. Only informative. Default=''
cra,cdec, cred,cwei, cdcom	Column names in **data** sample table (tab). Default=('ra' , 'dec', 'z', 'wei', 'dcom')
cra1,cdec1, cred1,cwei1 cdcom1	Column names in **random** sample (tab1). Default=('ra' , 'dec', 'z', 'wei', 'dcom')
custRAbound	Specify custom RA boundaries for samples that cross the RA=0 limit. See *Custom RA Boundaries*. Default=None
outfn	Base name for all output files (e.g. /home/myuser/redagn)

4.2.5 Note on mxh3

Due to performance reasons, the number of cells in the radial (comoving) distance actually used to build the skip table is always set as `mxh3=int((dcmax-dcmin)/rvmax)`. Hence, the parameter `mxh3` supplied at input will be ignored unless it is smaller than this optimum value.

Note however that `mxh3` is only relevant for the performance of the algorithms. It is **not** related with the number of radial bins `nsepv` where we want to get output counts.

4.3 Angular Space

4.3.1 Input Parameters Dictionary (acf)

Dictionary with attribute-style access, that stores all input parameters for the code, plus some useful run time information during output (see outaddrt)

Parameter	Description
kind	Kind of correlation function ('acf')
autogrid	If `autogrid=True` choose the optimum nr. of cells `(mxh1,mxh2,mxh3)` of the skip table (SK). Default=True
dens	Custom nr. of particles per SK cell used when `autogrid=True`. No need to specify unless desired. Default=None
mxh1	Nr. of DEC cells of the SK table. Only relevant if `autogrid=False`. Default=30
mxh2	Nr. of RA cells of the SK table. Only relevant if `autogrid=False`. Default=180
pxorder	Data ordering method. See *Pixel Ordering Methods* for details and options. Default='natural'
doboot	If `True`, calculate bootstrap counts and error bars. Default=False
nbts	Nr. of bootstrap samples. Only relevant if `doboot=True`. Default=50
bseed	Fixed seed for boostrap RNG. Always set `bseed>0` if running in paralell. Default=1245
wfib	If `True` apply SDSS fiber correction for pairs closer than 55". See *wfiber()* Fortran function. Default=False
nsept	Nr. of angular bins. Default=22
septmin	Minimum separation to consider [deg]. Default=0.01
dsept	Size of angular bins (in dex if log bins). Default=0.15
logsept	If `True` use log-spaced bins. Otherwise use linear-spaced bins. Default=True
file	File name of **data** sample. Only informative. Default=''
file1	File name of **random** sample. Only informative. Default=''
description	Short description of the run. Only informative. Default=''
estimator	Statistical estimator of the correlation function. Default='NAT' * 'NAT' : Natural estimator -> $DD/RR - 1$ * 'HAM' : Hamilton estimator -> $DD * RR/DR^2 - 1$ * 'LS' : Landy-Szalay estimator -> $(DD - 2DR + RR)/RR$ * 'DP' : Davis-Peebles estimator -> $DD/DR - 1$
cra,cdec,cwei	Column names in **data** sample table (tab). Default=('ra', 'dec', 'wei')
cra1,cdec1,cwei1	Column names in **random** sample table (tab1). Default=('ra', 'dec', 'wei')
custRAbound	Specify custom RA boundaries for samples that cross the RA=0 limit. See *Custom RA Boundaries*. Default=None
outfn	Base name for all output files (e.g. /home/myuser/redagn)

4.3.2 Input Parameters Dictionary (accf)

Dictionary with attribute-style access, that stores all input parameters for the code, plus some useful run time information during output (see outaddrt)

Parameter	Description
kind	Kind of correlation function ('accf')
autogrid	If `autogrid=True` choose the optimum nr. of cells `(mxh1,mxh2,mxh3)` of the skip table (SK). Default=True
dens	Custom nr. of particles per SK cell used when `autogrid=True`. No need to specify unless desired. Default=None
mxh1	Nr. of DEC cells of the SK table. Only relevant if `autogrid=False`. Default=30
mxh2	Nr. of RA cells of the SK table. Only relevant if `autogrid=False`. Default=180
pxorder	Data ordering method. See *Pixel Ordering Methods* for details and options. Default='natural'
doboot	If `True`, calculate bootstrap counts and error bars. Default=False
nbts	Nr. of bootstrap samples. Only relevant if `doboot=True`. Default=50
bseed	Fixed seed for boostrap RNG. Always set `bseed>0` if running in paralell. Default=1245
wfib	If `True` apply SDSS fiber correction for pairs closer than 55". See *wfiber()* Fortran function. Default=False
nsept	Nr. of angular bins. Default=22
septmin	Minimum separation to consider [deg]. Default=0.01
dsept	Size of angular bins (in dex if log bins). Default=0.15
logsept	If `True` use log-spaced bins. Otherwise use linear-spaced bins. Default=True
file	File name of **data** sample. Only informative. Default=''
file1	File name of **random** sample. Only informative. Default=''
file2	File name of **cross** sample. Only informative. Default=''
description	Short description of the run. Only informative. Default=''
estimator	Statistical estimator of the correlation function. Default='DP' * 'DP' : Davis-Peebles estimator -> $CD/CR - 1$
cra,cdec, cwei	Column names in **data** sample table (tab). Default=('ra', 'dec', 'wei')
cra1,cdec1, cwei1	Column names in **random** sample table (tab1). Default=('ra', 'dec', 'wei')
cra2,cdec2, cwei2	Column names in **cross** sample table (tab2). Default=('ra', 'dec', 'wei')
custRAbound	Specify custom RA boundaries for samples that cross the RA=0 limit. See *Custom RA Boundaries*. Default=None
outfn	Base name for all output files (e.g. /home/myuser/redagn)

4.3.3 Input Parameters Dictionary (thA)

Dictionary with attribute-style access, that stores all input parameters for the code, plus some useful run time information during output (see outaddrt)

Parameter	Description
kind	Kind of correlation function ('thA')
autogrid	If `autogrid=True` choose the optimum nr. of cells `(mxh1,mxh2,mxh3)` of the skip table (SK). Default=True
dens	Custom nr. of particles per SK cell used when `autogrid=True`. No need to specify unless desired. Default=None
mxh1	Nr. of DEC cells of the SK table. Only relevant if `autogrid=False`. Default=30
mxh2	Nr. of RA cells of the SK table. Only relevant if `autogrid=False`. Default=180
pxorder	Data ordering method. See *Pixel Ordering Methods* for details and options. Default='natural'
doboot	If `True`, calculate bootstrap counts and error bars. Default=False
nbts	Nr. of bootstrap samples. Only relevant if `doboot=True`. Default=50
bseed	Fixed seed for boostrap RNG. Always set `bseed>0` if running in paralell. Default=1245
wfib	If `True` apply SDSS fiber correction for pairs closer than 55". See *wfiber()* Fortran function. Default=False
nsept	Nr. of angular bins. Default=22
septmin	Minimum separation to consider [deg]. Default=0.01
dsept	Size of angular bins (in dex if log bins). Default=0.15
logsept	If `True` use log-spaced bins. Otherwise use linear-spaced bins. Default=True
file	File name of **data** sample. Only informative. Default=''
description	Short description of the run. Only informative. Default=''
cra,cdec, cred,cwei	Column names in **data** sample table (tab). Default=('ra', 'dec', 'wei')
custRAbound	Specify custom RA boundaries for samples that cross the RA=0 limit. See *Custom RA Boundaries*. Default=None
outfn	Base name for all output files (e.g. /home/myuser/redagn)

4.3.4 Input Parameters Dictionary (thC)

Dictionary with attribute-style access, that stores all input parameters for the code, plus some useful run time information during output (see outaddrt)

Parameter	Description
kind	Kind of correlation function ('thC')
autogrid	If `autogrid=True` choose the optimum nr. of cells `(mxh1,mxh2,mxh3)` of the skip table (SK). Default=True
dens	Custom nr. of particles per SK cell used when `autogrid=True`. No need to specify unless desired. Default=None
mxh1	Nr. of DEC cells of the SK table. Only relevant if `autogrid=False`. Default=30
mxh2	Nr. of RA cells of the SK table. Only relevant if `autogrid=False`. Default=180
pxorder	Pixel ordering method. See *Pixel Ordering Methods* for details and options. Default='natural'
doboot	If `True`, calculate bootstrap counts and error bars. Default=False
nbts	Nr. of bootstrap samples. Only relevant if `doboot=True`. Default=50
bseed	Fixed seed for boostrap RNG. Always set `bseed>0` if running in paralell. Default=12345
wfib	If `True` apply SDSS fiber correction for pairs closer than 55". See *wfiber()* Fortran function. Default=False
nsept	Nr. of angular bins. Default=22
septmin	Minimum separation to consider [deg]. Default=0.01
dsept	Size of angular bins (in dex if log bins). Default=0.15
logsept	If `True` use log-spaced bins. Otherwise use linear-spaced bins. Default=True
file	File name of data sample. Only informative. Default=''
file1	File name of random sample. Only informative. Default=''
description	Short description of the run. Only informative. Default=''
pxorder	Sorting method for data. See *Pixel Ordering Methods* for options. Default='natural'
cra,cdec, cwei	Column names in **data** sample table (tab). Default=('ra', 'dec', 'wei')
cra1,cdec1, cwei1	Column names in **random** sample table (tab1). Default=('ra', 'dec', 'wei')
custRAbound	Specify custom RA boundaries for samples that cross the RA=0 limit. See *Custom RA Boundaries*. Default=None
outfn	Base name for all output files (e.g. /home/myuser/redagn)

CHAPTER 5

Output

5.1 Projected Space

5.1.1 Output Dictionary (pcf)

Dictionary with attribute-style access, containing all counts and correlations accesible by field keys, e.g. counts.dd, counts.rr, counts.wrp, etc. It also stores the log and all input parameters.

Key(s)	Description
rpl,rpm,rpr	Left, mid, and right-points of projected bins (in Mpc)
wrp,wrperr	Projected correlation function and error
dd	DD pair count array in projected and radial bins
rr	RR pair count array in projected and radial bins
dr	DR pair count array in projected and radial bins
bdd	Boostrap DD pair count array in proj. and radial bins
npt,npt1	Nr. of points in data (D) and random sample (R), resp.
log	Log record of Python routines
logfortran	Log record of Fortran routines
par	Input parameters + runtime parameters (see xxxx)

5.1.2 Output Dictionary (pccf)

Dictionary with attribute-style access, containing all counts and correlations accesible by field keys, e.g. counts.cd, counts.cr, counts.wrp, etc. It also stores the log and all input parameters.

Key(s)	Description
rpl,rpm,rpr	Left, mid, and right-points of projected bins (in Mpc)
wrp,wrperr	Projected cross-correlation function and error
cd	CD pair count array in projected and radial bins
cr	CR pair count array in projected and radial bins
bcd	Boostrap CD pair count array in proj. and radial bins
npt,npt1,npt2	Nr. of points in samples D, R, and C, respectively
log	Log record of Python routines
logfortran	Log record of Fortran routines
par	Input parameters + runtime parameters (see xxxx)

5.1.3 Output Dictionary (rppiA)

Dictionary with attribute-style access, containing all counts and correlations accesible by field keys, e.g. counts.dr, counts.bdr, counts.intpi, etc. It also stores the log and all input parameters.

Key(s)	Description
rpl,rpm,rpr	Left, mid, and right-points of projected bins (in Mpc)
dd	DD pair count array in projected and radial bins
bdd	Boostrap DD pair count array in proj. and radial bins
intpi	DD counts integrated along all radial bins
intpib	Boostrap DD counts integrated along all radial bins
npt	Nr. of points in the sample (D)
log	Log record of Python routines
logfortran	Log record of Fortran routines
par	Input parameters + runtime parameters (see xxxx)

5.1.4 Output Dictionary (rppiC)

Dictionary with attribute-style access, containing all counts and correlations accesible by field keys, e.g. counts.dr, counts.bdr, counts.intpi, etc. It also stores the log and all input parameters.

Key(s)	Description
rpl,rpm,rpr	Left, mid, and right-points of projected bins (in Mpc)
dr	DR pair count array in projected and radial bins
bdr	Boostrap DR pair count array in proj. and radial bins
intpi	DR counts integrated along all radial bins
intpib	Boostrap DR counts integrated along all radial bins
npt,npt1	Nr. of points in the samples D and R, respectively
log	Log record of Python routines
logfortran	Log record of Fortran routines
par	Input parameters + runtime parameters (see xxxx)

5.2 Redshift Space

5.2.1 Output Dictionary (rcf)

Dictionary with attribute-style access, containing all counts and correlations accesible by field keys, e.g. counts.dd, counts.rr, counts.xis, etc. It also stores the log and all input parameters.

Key(s)	Description
sl,sm,sr	Left, mid, and right-points of redshift space bins (in Mpc)
xis,xiserr	Redshift space correlation function and error
dd	DD pair count array in projected and radial bins
rr	RR pair count array in projected and radial bins
dr	DR pair count array in projected and radial bins
bdd	Boostrap DD pair count array in proj. and radial bins
npt,npt1	Nr. of points in data (D) and random sample (R), resp.
log	Log record of Python routines
logfortran	Log record of Fortran routines
par	Input parameters + runtime parameters (see xxxx)

5.2.2 Output Dictionary (rccf)

Dictionary with attribute-style access, containing all counts and correlations accesible by field keys, e.g. counts.cd, counts.cr, counts.xis, etc. It also stores the log and all input parameters.

Key(s)	Description
sl,sm,sr	Left, mid, and right-points of redshift space bins (in Mpc)
xis,xiserr	Redshift space cross-correlation function and error
cd	CD pair count array in redshift space bins
cr	CR pair count array in redshift space bins
bcd	Boostrap CD pair count array in redshift space bins
npt,npt1,npt2	Nr. of points in samples D, R, and C, respectively
log	Log record of Python routines
logfortran	Log record of Fortran routines
par	Input parameters + runtime parameters (see xxxx)

5.2.3 Output Dictionary (sA)

Dictionary with attribute-style access, containing all counts and correlations accesible by field keys, e.g. counts.dd, counts.bd, etc. It also stores the log and all input parameters.

Key(s)	Description
sl,sm,sr	Left, mid, and right-points of redshift space bins (in Mpc)
dd	DD pair count array in redshift space bins
bdd	Boostrap DD pair count array in redshift space bins
npt	Nr. of points in the sample (D)
log	Log record of Python routines
logfortran	Log record of Fortran routines
par	Input parameters + runtime parameters (see xxxx)

5.2.4 Output Dictionary (sC)

Dictionary with attribute-style access, containing all counts and correlations accesible by field keys, e.g. counts.dr, counts.bdr, etc. It also stores the log and all input parameters.

Key(s)	Description
sl,sm,sr	Left, mid, and right-points of redshift space bins (in Mpc)
dr	DR pair count array in redshift space bins
bdr	Boostrap DR pair count array in redshift space bins
npt,npt1	Nr. of points in the samples D and R, respectively
log	Log record of Python routines
logfortran	Log record of Fortran routines
par	Input parameters + runtime parameters (see xxxx)

5.3 Angular Space

5.3.1 Output Dictionary (acf)

Dictionary with attribute-style access, containing all counts and correlations accesible by field keys, e.g. counts.dd, counts.rr, counts.wth, etc. It also stores the log and all input parameters.

Key(s)	Description
thl,thm,thr	Left, mid, and right-points of anglar bins (in deg)
wth,wtherr	Angular correlation function and error
dd	DD pair count array in angular bins
rr	RR pair count array in angular bins
dr	DR pair count array in angular bins
bdd	Boostrap DD pair count array in angular bins
npt,npt1	Nr. of points in data (D) and random sample (R), resp.
log	Log record of Python routines
logfortran	Log record of Fortran routines
par	Input parameters + runtime parameters (see xxxx)

5.3.2 Output Dictionary (accf)

Dictionary with attribute-style access, containing all counts and correlations accesible by field keys, e.g. counts.cd, counts.cr, counts.wth, etc. It also stores the log and all input parameters.

Key(s)	Description
thl,thm,thr	Left, mid, and right-points of angular bins (in Mpc)
wht,wtherr	Angular cross-correlation function and error
cd	CD pair count array in angular bins
cr	CR pair count array in angular bins
bcd	Boostrap CD pair count array in angular bins
npt,npt1,npt2	Nr. of points in samples D, R, and C, respectively
log	Log record of Python routines
logfortran	Log record of Fortran routines
par	Input parameters + runtime parameters (see xxxx)

5.3.3 Output Dictionary (thA)

Dictionary with attribute-style access, containing all counts and correlations accesible by field keys, e.g. counts.dd, counts.bdd, etc. It also stores the log and all input parameters.

Key(s)	Description
thl,thm,thr	Left, mid, and right-points of angular bins (deg)
dd	DD pair count array in angular bins
bdd	Boostrap DD pair count array in angular bins
npt	Nr. of points in the sample (D)
log	Log record of Python routines
logfortran	Log record of Fortran routines
par	Input parameters + runtime parameters (see xxxx)

5.3.4 Output Dictionary (thC)

Dictionary with attribute-style access, containing all counts and correlations accesible by field keys, e.g. counts.dr, counts.bdr, etc. It also stores the log and all input parameters.

Key(s)	Description
thl,thm,thr	Left, mid, and right-points of angular bins (deg)
dr	DR pair count array in angular bins
bdr	Boostrap DR pair count array in angular bins
npt,npt1	Nr. of points in the samples D and R, respectively
log	Log record of Python routines
logfortran	Log record of Fortran routines
par	Input parameters + runtime parameters (see xxxx)

5.4 Output Files

If `write=True` the code saves several files named as `outfn` plus an extension. According to the kind of correlation, these files are named as

File extension	Description
.wrp,.xis,.wth	Projected, redshift-space and angular correlation, respectively (ASCII)
.dd,.rr,.dr	DD, RR, DR pair count arrays, in the corresponding space bins (ASCII)
.cd,.cr	CD and CR pair count arrays, in the corresponding space bins (ASCII)
.cnt	Pickled file with the **counts** dictionary object (binary)
.par	Input parameters in JSON style dictionary (ASCII)

In order to properly capture log information from Python and Fortran code in different environments, Gundam **always** generates two log files named as `outfn` plus an extension as

File extension	Description
.log	Log record produced by Python routines (ASCII)
.fortran.log	Log record produced by Fortran routines (ASCII)

If `plot=True`, plots of correlation/counts are shown on screen and also saved to disk in pdf format with extensions **.wrp.pdf.**, **.xis.pdf**, etc.

5.5 Additional Runtime Information in Ouput

Some useful information is added to the **par** dictionary returned within the **counts** dictionary. These are mostly to keep a record of runtime parameters, such as if the run was parallel, if it wrote ouput files, check the boundaries of the survey, etc.

Key(s)	Description
para	Wheter to run in parallel or serial, i.e single core
write	Write files was requested or not
plot	Plot was requested or not
sbound	Survey boundaries as a list (ramin,ramax,decmin, decmax,dcmin,dcmax), as returned for example by *gundam.bound3d()*

CHAPTER 6

Pixel Ordering Methods

To increase the performance of linked list algorithms, spatial data is divided in "pixels" and these pixels are sorted by different methods. The rows of the input data table can be rearranged according to this pixel index, so that nearby structure in real space are also close in computer memory space. Once input data is sorted, traversing the linked list will generate many more cache hits.

This is quite similar to Morton-curve or Hilbert-curve techniques used in computer science to map multidimensional data into a one dimensional index.

Note these "pixels" have nothing to do with the cells used by the linked list algorithm. The parameter `pxorder` controls the method to sort the pixels. Available options are:

- **'natural'** Pixels are sorted in lexical order by three keys, Z-DEC-RA, in that order. This is usually the fastest option as it more closely matches the mechanics of the counting algorithm. The number of pixels are set to `(mxh3,mxh1,mxh2)`
- **'morton_dir'** 2D Morton-curve ordering using the RA-DEC values directly, i.e. no division in pixels
- **'morton2'** Pixels are sorted by 2D Morton-curve in RA-DEC
- **'morton3'** Pixels are sorted by 3D Morton-curve in RA-DEC-Z
- **'hilbert2'** Pixels are sorted by 2D Hilbert-curve in RA-DEC
- **'hilbert3'** Pixels are sorted by 3D Hilbert-curve in RA-DEC-Z

Note these are experimental features mainly for testing purposes. You do not need to fuss with these options unless you are looking for fun. Plus, in most cases the default natural order is the fastest option.

The following images illustrate how pixel sorting rearranges a list of ra-dec coordinates into nearby patches. The vertical axis is the index of the ra-dec point in the list (in log scale for clarity).

6.1 Custom RA Boundaries

The area in the sky occupied by a sample or survey can be composed of multiple scattered "patches" at both sides of the RA=0 limit. In order to efficiently grid and sort such a sample, the area enclosing all of its objects should be minimal, which can be difficult to find out automatically.

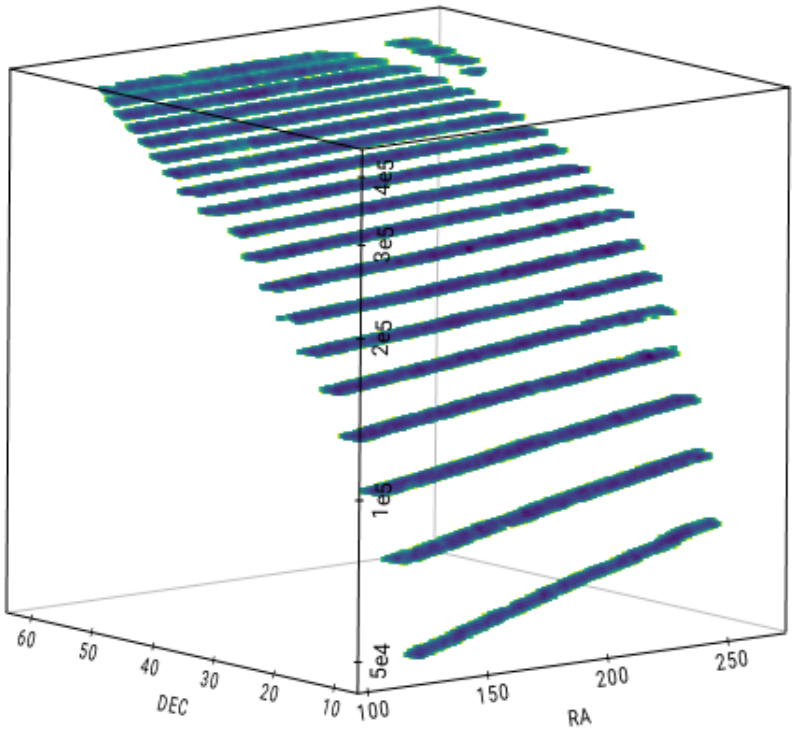


Fig. 6.1: Natural pixel ordering

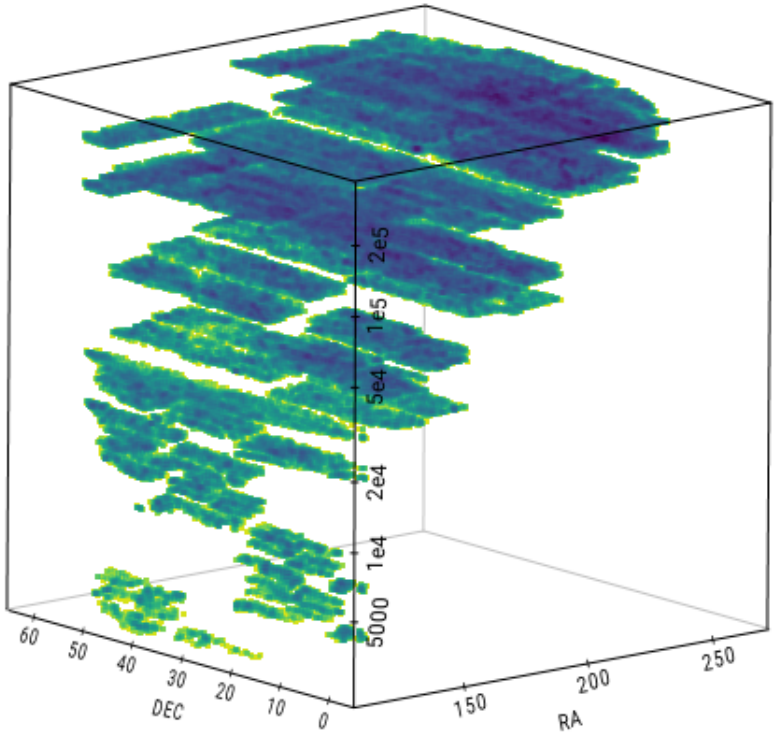


Fig. 6.2: Morton pixel ordering

For example, if a survey is composed of two equatorial 1 deg^2 patches, one beginning at RA=355 deg and the other beginning at RA=5 deg, the grid can extend from 5 to 356 deg or, more conveniently, from 355 to 6 deg. The input parameter `custRAbound` can be set to indicate such limits. In this case, setting `custRAbound = [355., 6.]` will create a much smaller grid and routines will perform faster.

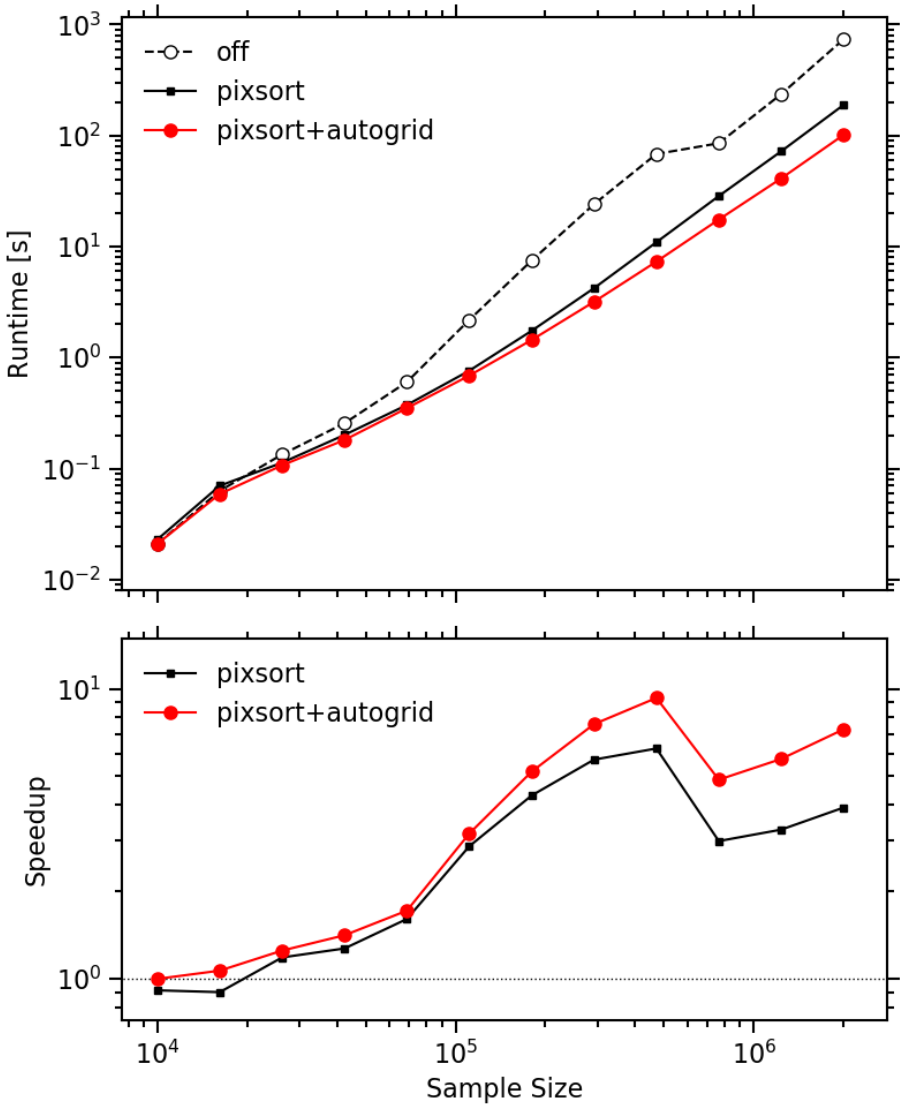
Gundam will try to guess if a sample crosses the RA=0 limit (see *gundam.cross0guess()*).

CHAPTER 7

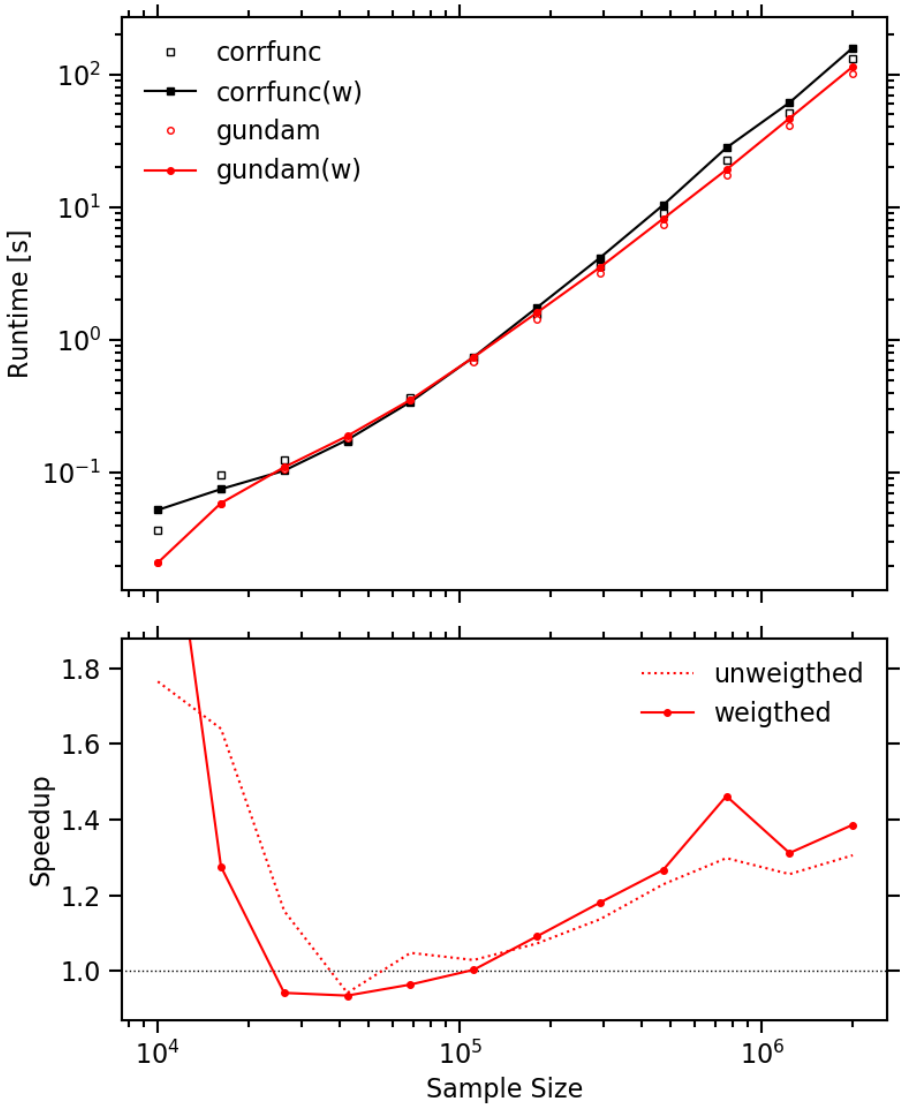
Performance

The Gundam package is optimized both for *speed* and *usability*. Nevertheless, it is among the fastest codes available. Below you will find some benchmarks (single-thread), comparing run time vs size for various samples of 10^4 to 8x10^6 galaxies extracted from a 60x60 deg^2 light cone build from the Millennium Simulation and the Semi-Analytic Galaxy Evolution Model (SAGE).

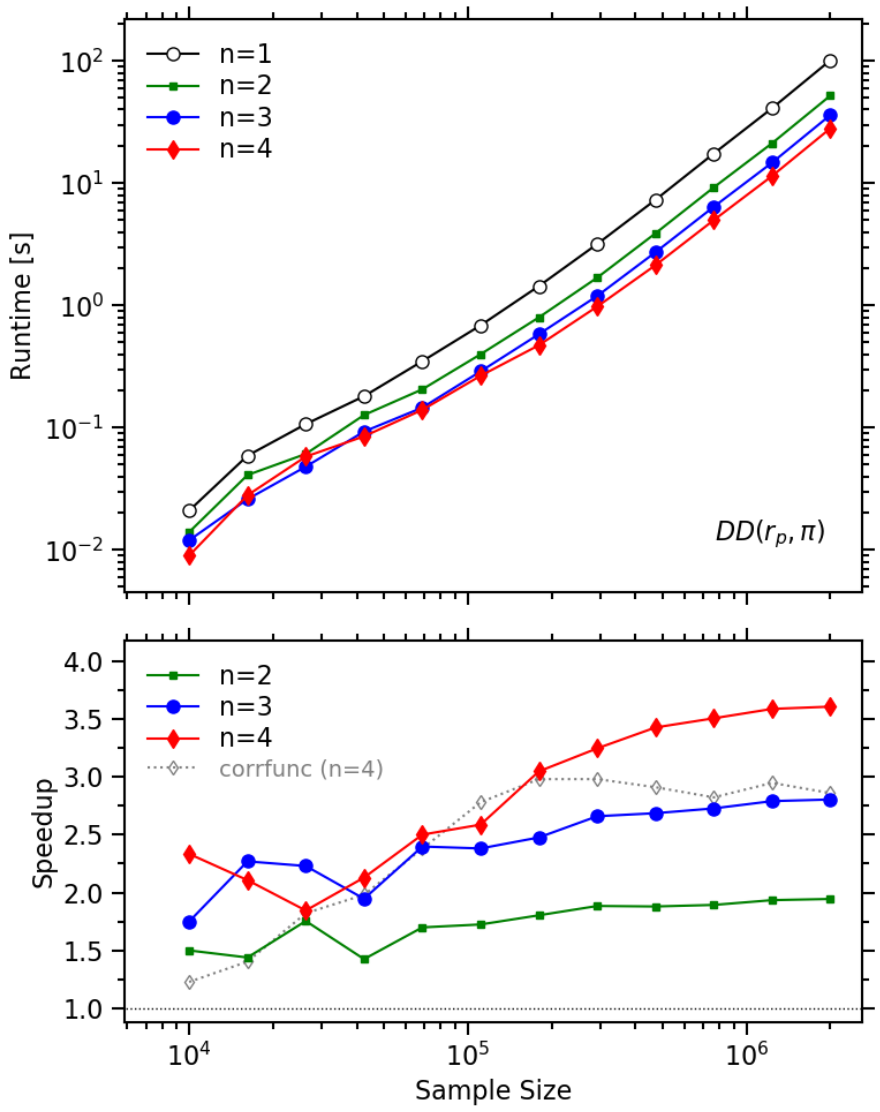
We show here the effect of sorting the data into convenient cells or *pixels* before counting pairs, and the effect of choosing a good grid size for samples of increasing density.



Now we compare with one of the fastest publicity available codes (corrfunc)



And now we compare the performance with *n* threads running in parallel



The equipment employed for testing is based in a four core i7-3770K 3.5GHz CPU (L1 cache: 32KB data + 32KB instruction, L2 cache: 256KB, L3 cache: 8MB shared) with 16GB RAM, running with OpenSuse Linux, GNU Fortran 6.1.1 compiler and Python 3.5.2. Compilation was performed with f2py (numpy 1.11.1) with flags {tt march=native -ftree-vectorize}. We required 14 rp bins from 0.01 to 25 Mpc and 40 radial bins from 0 to 40 Mpc.

CHAPTER 8

API Reference

8.1 Main Routines (projected space)

These are the main routines for counting pairs and estimating correlation functions. They encapsulate all required steps to provide a true end-user experience : you only need to provide data+parameters to receive final results, and perhaps that amazing plot of your next paper

`gundam.`**`pcf`**(*tab*, *tab1*, *par*, *nthreads=-1*, *write=True*, *plot=False*, ***kwargs*)

Given two astropy tables corresponding to **data** and **random** samples, calculate the **two-point projected auto-correlation function (pcf)**

All input parameters that control binning, cosmology, estimator, etc. are passed in a single dictionary `par`, which can be easily generated with default values using `gundam.packpars()` and customized later

Parameters

tab [astropy table] Table with data particles (D)

tab1 [astropy table] Table with random particles (R)

par [Munch dictionary] Input parameters. See indic-pcf for a detailed description

nthreads [integer. Default=-1] Number of threads to use. Default to -1, which means all available cpu cores

write [bool. Default=True]

- True : write several output files for DD/RR/DR counts, correlations, etc.
- False : do not write any output files (except for logs, which are always saved)

plot [bool. Default=False]

- True : generate the CF plot on screen (and saved to disk when `write=True`)
- False : do not generate any plots

kwargs [dict] Extra keywords passed to *`gundam.plotcf()`*

Returns

counts [Munch dictionary] Munch dictionary, containing all counts and correlations accesible by field keys, e.g. counts.dd, counts.rr, counts.wrp, etc. It also stores the complete log and all input parameters. See outdicpcf for a detailed description.

Examples

```
import gundam as gun ; from astropy.table import Table
gals  = Table.read('redgal.fits')                          # read data
rans  = Table.read('redgal_rand.fits')                     # read randoms
par = gun.packpars(kind='pcf',outfn='redgal')              # generate default␣
↪parameters
cnt = gun.pcf(gals, rans, par, write=True, plot=True)    # get pcf and plot
```

gundam.**pccf** (*tab*, *tab1*, *tab2*, *par*, *nthreads=-1*, *write=True*, *plot=False*, ***kwargs*)

Given three astropy tables corresponding to **data**, **random**, and **cross** samples, this routine calculates the **two-point projected cross-correlation function (pccf)**

All input parameters that control binning, cosmology, estimator, etc. are passed in a single dictionary `par`, which can be easily generated with default values using `gundam.packpars()` and customized later

Parameters

tab [astropy table] Table with data particles (D)

tab1 [astropy table] Table with random particles (R)

tab2 [astropy table] Table with cross sample particles (C)

par [Munch dictionary] Input parameters. See indic-pccf for a detailed description

nthreads [integer. Default=-1] Number of threads to use. Default to -1, which means all available cpu cores

write [bool. Default=True]

- True : write several output files for CD/CR counts, correlations, etc.
- False : do not write any output files (except for logs, which are always saved)

plot [bool. Default=False]

- True : generate the CF plot on screen (and saved to disk when `write=True`)
- False : do not generate any plots

kwargs [dict] Extra keywords passed to *gundam.plotcf()*

Returns

counts [Munch dictionary] Munch dictionary, containing all counts and correlations accesible by field keys, e.g. counts.cd, counts.cr, counts.wrp, etc. It also stores the complete log and all input parameters. See outdicpccf for a detailed description

Examples

```
import gundam as gun ; from astropy.table import Table
gals  = Table.read('redgal.fits')                        # read data files
rans  = Table.read('redgal_rand.fits')
qsos  = Table.read('qso.fits')
par = gun.packpars(kind='pccf',outfn='redgal_qso')       # generate default␣
↪parameters
cnt = gun.pccf(gals, rans, qsos, par, write=True )       # get pcf
```

gundam.**rppi_A**(*tab*, *par*, *nthreads=-1*, *write=True*, *plot=False*, ***kwargs*)

Given an astropy data table, count pairs in the projected-radial space (r_p vs π space).

All input parameters that control binning, cosmology, column names, etc. are passed in a single dictionary `par`, which can be easily generated with default values using `gundam.packpars()` and customized later

Parameters

tab [astropy table] Table with data particles

par [Munch dictionary] Input parameters. See indic-rppiA for a detailed description

nthreads [integer. Default=-1] Number of threads to use. Default to -1, which means all available cpu cores

write [bool. Default=True]

- True : write several output files for counts, correlations, etc.
- False : do not write any output files (except for logs, which are always saved)

plot [bool. Default=True]

- True : generate a plot of counts (integrated radially) vs proj. radius
- False : do not generate any plots

kwargs [dict] Extra keywords passed to *gundam.plotcf()*

Returns

counts [Munch dictionary] Munch dictionary, containing all counts and correlations accesible by field keys, e.g. counts.dd, counts.bdd, etc. It also stores the complete log and all input parameters. See outdicrppiA for a detailed description

Examples

```
import gundam as gun ; from astropy.table import Table
gals  = Table.read('redgal.fits')                       # read data
par = gun.packpars(kind='rppiA',outfn='redgalpairs')    # generate default␣
↪parameters
cnt = gun.rppiA(gals, par, write=True, plot=True)       # get pair counts and plot
```

gundam.**rppi_C**(*tab*, *tab1*, *par*, *nthreads=-1*, *write=True*, *plot=False*, ***kwargs*)

Given two astropy data tables, cross-count pairs in the projected-radial space (r_p vs π space).

All input parameters that control binning, cosmology, column names, etc. are passed in a single dictionary `par`, which can be easily generated with default values using `gundam.packpars()` and customized later

Parameters

tab [astropy table] Table with particles (sample D)

tab1 [astropy table] Table with particles (sample R)

par [Munch dictionary] Input parameters. See indic-rppiC for a detailed description

nthreads [integer. Default=-1] Number of threads to use. Default to -1, which means all available cpu cores

write [bool. Default=True]

- True : write several output files for counts, correlations, etc.
- False : do not write any output files (except for logs, which are always saved)

plot [bool. Default=True]

- True : generate a plot of counts (integrated radially) vs proj. radius
- False : do not generate any plots

kwargs [dict] Extra keywords passed to *gundam.plotcf()*

Returns

counts [Munch dictionary] Munch dictionary, containing all counts and correlations accesible by field keys, e.g. counts.dr, counts.bdr, etc. It also stores the complete log and all input parameters. See outdicrppiC for a detailed description

Examples

```
import gundam as gun ; from astropy.table import Table
qsos = Table.read('qso.fits')                          # read data
gals = Table.read('redgal.fits')                       # read data
par = gun.packpars(kind='rppiC',outfn='qso_rg_pairs')  # generate default␣
↪parameters
cnt = gun.rppiC(qsos, gals, par, write=True)           # get pair counts
```

8.2 Main Routines (redshift space)

These are the main routines for counting pairs and estimating correlation functions. They encapsulate all required steps to provide a true end-user experience : you only need to provide data+parameters to receive final results, and even that amazing plot of your next paper

gundam.**rcf**(*tab*, *tab1*, *par*, *nthreads=-1*, *write=True*, *para=False*, *plot=False*, ***kwargs*)

Given two astropy tables corresponding to **data** and **random** samples, this routine calculates the **two-point redshift space auto-correlation function (rcf)**

All input parameters that control binning, cosmology, estimator, etc. are passed in a single dictionary `par`, which can be easily generated with default values using `gundam.packpars()` and customized later

Parameters

tab [astropy table] Table with data particles (D)

tab1 [astropy table] Table with random particles (R)

par [Munch dictionary] Input parameters. See indic-rcf for a detailed description

write [bool. Default=True]

- True : write several output files for DD/RR/DR counts, correlations, etc.
- False : do not write any output files (except for logs, which are always saved)

para [bool. Default=False]

- True : run in parallel over all available ipyparallel engines
- False : run serially. No need for any ipyparallel cluster running

plot [bool. Default=False]

- True : generate the CF plot on screen (and saved to disk when `write=True`)
- False : do not generate any plots

Returns

counts [Munch dictionary] Munch dictionary, containing all counts and correlations accesible by field keys, e.g. counts.dd, counts.rr, counts.xis, etc. It also stores the complete log and all input parameters. See outdicrcf for a detailed description.

Examples

```
import gundam as gun ; from astropy.table import Table
gals  = Table.read('redgal.fits')                       # read data
rans  = Table.read('redgal_rand.fits')                  # read randoms
par = gun.packpars(kind='rcf',outfn='redgal')           # generate default␣
↪parameters
cnt = gun.rcf(gals, rans, par, write=True, plot=True)   # get rcf and plot
```

gundam.**rccf**(*tab*, *tab1*, *tab2*, *par*, *nthreads=-1*, *write=True*, *plot=False*, ***kwargs*)

Given three astropy tables corresponding to **data**, **random**, and **cross** samples, this routine calculates the **two-point redshift space cross-correlation function (rccf)**

All input parameters that control binning, cosmology, estimator, etc. are passed in a single dictionary `par`, which can be easily generated with default values using `gundam.packpars()` and customized later

Parameters

tab [astropy table] Table with data particles (D)

tab1 [astropy table] Table with random particles (R)

tab2 [astropy table] Table with cross sample particles (C)

par [Munch dictionary] Input parameters. See indic-rccf for a detailed description

nthreads [integer. Default=-1] Number of threads to use. Default to -1, which means all available cpu cores

write [bool. Default=True]

- True : write several output files for CD/CR counts, correlations, etc.
- False : do not write any output files (except for logs, which are always saved)

plot [bool. Default=False]

- True : generate the CF plot on screen (and saved to disk when `write=True`)
- False : do not generate any plots

kwargs [dict] Extra keywords passed to *gundam.plotcf()*

Returns

counts [Munch dictionary] Munch dictionary, containing all counts and correlations accesible by field keys, e.g. counts.cd, counts.cr, counts.xis, etc. It also stores the complete log and all input parameters. See outdicrccf for a detailed description

Examples

```
import gundam as gun ; from astropy.table import Table
gals  = Table.read('redgal.fits')                       # read data files
rans  = Table.read('redgal_rand.fits')
qsos  = Table.read('qso.fits')
par = gun.packpars(kind='pccf',outfn='redgal_qso')      # generate default␣
↪parameters
cnt = gun.rccf(gals, rans, qsos, par, write=True )      # get rcf
```

gundam.**s_A**(*tab*, *par*, *nthreads=-1*, *write=True*, *para=False*, *plot=False*, ***kwargs*)

Given an astropy table, count pairs in redshift space

All input parameters that control binning, cosmology, column names, etc. are passed in a single dictionary `par`, which can be easily generated with default values using `gundam.packpars()` and customized later

Parameters

tab [astropy table] Table with data particles

par [Munch dictionary] Input parameters. See indic-sA for a detailed description

write [bool. Default=True]

- True : write several output files for counts, correlations, etc.
- False : do not write any output files (except for logs, which are always saved)

para [bool. Default=False]

- True : run in parallel over all available ipyparallel engines
- False : run serially. No need for any ipyparallel cluster running

plot [bool. Default=False]

- True : generate a plot of counts vs redshift separation
- False : do not generate any plots

Returns

counts [Munch dictionary] Munch dictionary, containing all counts and correlations accesible by field keys, e.g. counts.dd, counts.bdd, etc. It also stores the complete log and all input parameters. See outdicsA for a detailed description

Examples

```
import gundam as gun ; from astropy.table import Table
gals  = Table.read('redgal.fits')                    # read data
par = gun.packpars(kind='sA',outfn='redgalpairs')    # generate default parameters
cnt = gun.s_A(gals, par, write=True, plot=True  )    # get counts and plot
```

gundam.**s_C**(*tab*, *tab1*, *par*, *nthreads=-1*, *write=True*, *para=False*, *plot=False*, ***kwargs*)

Given two astropy tables, cross-count pairs in redshift space

All input parameters that control binning, cosmology, column names, etc. are passed in a single dictionary `par`, which can be easily generated with default values using `gundam.packpars()` and customized later

Parameters

tab [astropy table] Table with particles (sample D)

tab1 [astropy table] Table with particles (sample R)

par [Munch dictionary] Input parameters. See indic-sC for a detailed description

write [bool. Default=True]

- True : write several output files for counts, correlations, etc.
- False : do not write any output files (except for logs, which are always saved)

para [bool. Default=False]

- True : run in parallel over all available ipyparallel engines
- False : run serially. No need for any ipyparallel cluster running

plot [bool. Default=False]

- True : generate a plot of counts vs redshift separation
- False : do not generate any plots

Returns

counts [Munch dictionary] Munch dictionary, containing all counts and correlations accesible by field keys, e.g. counts.dr, counts.bdr, etc. It also stores the complete log and all input parameters. See outdicsC for a detailed description

Examples

```
import gundam as gun ; from astropy.table import Table
qsos = Table.read('qso.fits')                          # read data
gals = Table.read('redgal.fits')                       # read data
par = gun.packpars(kind='sC',outfn='qso_rg_pairs')     # generate default parameters
cnt = gun.rppiC(qsos, gals, par, write=True)           # get pair counts
```

8.3 Main Routines (angular space)

These are the main routines for counting pairs and estimating correlation functions. They encapsulate all required steps to provide a true end-user experience : you only need to provide data+parameters to receive final results, and even that amazing plot of your next paper

`gundam.acf`(*tab*, *tab1*, *par*, *nthreads=-1*, *write=True*, *plot=False*, ***kwargs*)

Given two astropy tables corresponding to **data** and **random** samples, this routine calculates the **two-point angular space auto-correlation function (acf)**

All input parameters that control binning, cosmology, estimator, etc. are passed in a single dictionary `par`, which can be easily generated with default values using `gundam.packpars()` and customized later

Parameters

tab [astropy table] Table with data particles (D)

tab1 [astropy table] Table with random particles (R)

par [Munch dictionary] Input parameters. See indic-acf for a detailed description

nthreads [integer. Default=-1] Number of threads to use. Default to -1, which means all available cpu cores

write [bool. Default=True]

- True : write several output files for DD/RR/DR counts, correlations, etc.
- False : do not write any output files (except for logs, which are always saved)

plot [bool. Default=False]

- True : generate the CF plot on screen (and saved to disk when `write=True`)
- False : do not generate any plots

kwargs [dict] Extra keywords passed to *gundam.plotcf()*

Returns

counts [Munch dictionary] Munch dictionary, containing all counts and correlations accesible by field keys, e.g. counts.dd, counts.rr, counts.xis, etc. It also stores the complete log and all input parameters. See outdicacf for a detailed description.

Examples

```
import gundam as gun ; from astropy.table import Table
gals  = Table.read('redgal.fits')                      # read data
rans  = Table.read('redgal_rand.fits')                 # read randoms
```

```
par = gun.packpars(kind='rcf',outfn='redgal')           # generate default␣
↪parameters
cnt = gun.rcf(gals, rans, par, write=True, plot=True)   # get rcf and plot
```

gundam.**accf**(*tab*, *tab1*, *tab2*, *par*, *nthreads=-1*, *write=True*, *plot=False*, ***kwargs*)

Given three astropy tables corresponding to **data**, **random**, and **cross** samples, this routine calculates the **two-point angular space cross-correlation function (accf)**

All input parameters that control binning, estimator, column names, etc. are passed in a single dictionary `par`, which can be easily generated with default values using `gundam.packpars()` and customized later

Parameters

tab [astropy table] Table with data particles (D)

tab1 [astropy table] Table with random particles (R)

tab2 [astropy table] Table with cross sample particles (C)

par [Munch dictionary] Input parameters. See indic-accf for a detailed description

nthreads [integer. Default=-1] Number of threads to use. Default to -1, which means all available cpu cores

write [bool. Default=True]

- True : write several output files for CD/CR counts, correlations, etc.
- False : do not write any output files (except for logs, which are always saved)

plot [bool. Default=False]

- True : generate the CF plot on screen (and saved to disk when `write=True`)
- False : do not generate any plots

kwargs [dict] Extra keywords passed to *gundam.plotcf()*

Returns

counts [Munch dictionary] Munch dictionary, containing all counts and correlations accesible by field keys, e.g. counts.cd, counts.cr, counts.wth, etc. It also stores the complete log and all input parameters. See outdicaccf for a detailed description

Examples

```
import gundam as gun ; from astropy.table import Table
gals  = Table.read('redgal.fits')                       # read data files
rans  = Table.read('redgal_rand.fits')
qsos  = Table.read('qso.fits')
par = gun.packpars(kind='accf',outfn='redgal_qso')      # generate default␣
↪parameters
cnt = gun.accf(gals, rans, qsos, par, write=True )      # get accf
```

gundam.**th_A**(*tab*, *par*, *nthreads=-1*, *write=True*, *plot=False*, ***kwargs*)

Given an astropy data table, count pairs in angular space

All input parameters that control binning, column names, etc. are passed in a single dictionary `par`, which can be easily generated with default values using `gundam.packpars()` and customized later

Parameters

tab [astropy table] Table with data particles

par [Munch dictionary] Input parameters. See indic-thA for a detailed description

nthreads [integer. Default=-1] Number of threads to use. Default to -1, which means all available cpu cores

write [bool. Default=True]

- True : write several output files for counts, correlations, etc.
- False : do not write any output files (except for logs, which are always saved)

plot [bool. Default=False]

- True : generate a plot of counts vs angular separation
- False : do not generate any plots

kwargs [dict] Extra keywords passed to *gundam.plotcf()*

Returns

counts [Munch dictionary] Munch dictionary, containing all counts and correlations accesible by field keys, e.g. counts.dd, counts.bdd, etc. It also stores the complete log and all input parameters. See outdicthA for a detailed description

Examples

```
import gundam as gun ; from astropy.table import Table
gals  = Table.read('redgal.fits')                    # read data
par = gun.packpars(kind='thA',outfn='redgalpairs')   # generate default parameters
cnt = gun.th_A(gals, par, write=True, plot=True  )   # get counts and plot
```

gundam.**th_C**(*tab*, *tab1*, *par*, *nthreads=-1*, *write=True*, *plot=False*, ***kwargs*)

Given two astropy data tables, cross-count pairs in angular space

All input parameters that control binning, column names, etc. are passed in a single dictionary `par`, which can be easily generated with default values using `gundam.packpars()` and customized later

Parameters

tab [astropy table] Table with particles (sample D)

tab1 [astropy table] Table with particles (sample R)

par [Munch dictionary] Input parameters. See indic-thC for a detailed description

nthreads [integer. Default=-1] Number of threads to use. Default to -1, which means all available cpu cores

write [bool. Default=True]

- True : write several output files for counts, correlations, etc.
- False : do not write any output files (except for logs, which are always saved)

plot [bool. Default=False]

- True : generate a plot of counts vs angular separation

- False : do not generate any plots

kwargs [dict] Extra keywords passed to *gundam.plotcf()*

Returns

counts [Munch dictionary] Munch dictionary, containing all counts and correlations accesible by field keys, e.g. counts.dr, counts.bdr, etc. It also stores the complete log and all input parameters. See outdicthC for a detailed description

Examples

```
import gundam as gun ; from astropy.table import Table
qsos = Table.read('qso.fits')                          # read data
gals = Table.read('redgal.fits')                       # read data
par = gun.packpars(kind='thC',outfn='qso_rg_pairs')    # generate default parameters
cnt = gun.th_C(qsos, gals, par, write=True)            # get pair counts
```

8.4 Auxiliary Routines

These are helper routines that make it easy to work with correlations, visualize count data, optimize grid size, sort data, etc. For examples of how to use them check their individual help, or even better, the source code of *gundam.pcf()*

gundam.**bestSKgrid2d**(*par*, *npts*, *ras*, *dens=None*)

Try to find the optimum size (mxh1,mxh2) of a 2D skip grid, for an arbitrary sample of (ra,dec) points.

This is far from trivial, so for the moment, this routine works as follows:

1. Choose an "optimum" target cell density (around 22 ppcell in many tests)
2. Find the best `mxh1` from an empirical best fit relation of npts vs mxh1
3. Adjust `mxh2` to reach the target cell density

Parameters

par [Munch dictionary] Input parameters dictionary

npts [integer or list of integers] Nr. of objects in the sample(s). For cross-correlations it should be a list of the form [sample_D_size, sample_R,size] and the effective `npts` adopted is their sum

ras [float array or list of arrays] Right ascention of objects in the samples(s). For cross-correlations, the two samples are concatenated

dens [float. Default=None] Target cell density. Note that *dens=None* implicates a default value of 22 if npts>50000 and 16 otherwise.

Returns

mxh1, mxh2 [integer] Nr. of DEC and RA cells of the SK table, respectively

dens [float] The effective cell density adopted

gundam.**bestSKgrid3d**(*par*, *npts*, *ras*, *dens=None*)

Try to find the optimum size (mxh1,mxh2,mxh3) of a 3D skip grid, for an arbitrary sample of (ra,dec,dcom) points.

This is far from trivial, so for the moment, this routine works as follows:

1. Choose an "optimum" target cell density according to the type of correlation
2. Choose the best `mxh3` as `mxh3=int((dcmax-dcmin)/rvmax)`
3. Find the best `mxh1` from an empirical best fit relation of npts vs mxh1
4. Adjust `mxh2` to reach the target cell density

Parameters

par [Munch dictionary] Input parameters dictionary

npts [integer or list of integers] Nr. of objects in the sample(s). For cross-correlations it should be a list of the form [sample_D_size, sample_R,size] and the effective `npts` adopted is their sum

ras [float array or list of arrays] Right ascention of objects in the samples(s). For cross-correlations, the two samples are joined together

dens [float. Default=None] Target cell density. Note that *dens=None* implicates different default values. See function code.

Returns

mxh1, mxh2, mxh3 [integer] Nr. of DEC, RA, DCOM cells of the SK table, respectively

dens [float] The effective cell density adopted

gundam.**cfindex**(*path='./'*)

List file_name + description of all **count** (.cnt) files in a given path. Useful to quickly check a dir with dozens of correlation function runs and therefore hundreds of files.

Parameters

path [string] Path to list descriptions of .cnt files inside

gundam.**cnttable**(*cfi*, *fmt1=None*, *fmt2=None*, *write=False*, *browser=True*)

Shows a nicely formatted tabular view of the count data stored in a **counts** output dictionary. The table is printed in *stdout* and optionally displayed in the default web browser.

Parameters

cfi [string or Munch dictionary] Filepath for the counts (.cnt) file, or the **counts** dictionary itself

fmt1 [string] Ouput format of numeric fields (bins, correlations and errors). Default='.4f'

fmt2 [string] Ouput format of numeric fields (counts). Default='.2f'

write [bool] If `write=True`, save table to disk. Filename will be asked for. Default=False

browser [bool] If `browser=True`, display HTML table in browser. Default=True

Returns

tab [astropy table] Single table with all relevant counts as columns. Use `print(tab)` or `tab.show_in_browser()`

Examples

```
# show info for a projected correlation from file on disk
cnttable('/proj/galred.cnt')
# show info a from variable in the session's memory
cnttable(galred)
```

gundam.**makebins**(*nsep*, *sepmin*, *dsep*, *logsep*)

Create arrays of bins in which Gundam will count pairs and estimate correlation functions, given the number of bins desired, the minimum bin value and the chosen bin width.

Note units are not needed, but should be interpreted according to the input parameters

Parameters

nsep [integer] Number of bins

sepmin [float] Minimum bin location

dsep [float] Bin width (dex if `logsep=1`)

logsep [bool] If `True`, do log-space binning. If `False`, do linear-space binning

Returns

sep [float array] Bin locations used by Gundam (left-side + extra bin at right limit)

sepout [float array] Left, middle and right-side points of each bin. Useful to plot more easily

Examples

```
# Create 18 log bins of size=0.2 dex in redshift and projected space
seps,sepsout = makebins(18,0.01,0.2,1)
sepp,seppout = makebins(18,0.01,0.2,1)
# Create 25 bins of size 2 Mpc in radial space, from 0 to 50 Mpc
sepv = makebins(25,0.,2.,0)[0]
# Create instead 1 bin of size 50 Mpc, e.g. to work out the pcf integrated from 0␣
↪to 50 Mpc
sepv = makebins(1,0.,50.,0)[0]
```

gundam.**pixsort**(*tab*, *colnms*, *par*)

Arrange an astropy table with (ra,dec) or (ra,dec,z) data into a grid of SK pixels and sort the rows of the table according to the pixel index, ordered by a given methodology. This way data in a given pixel sits closer in memory, largely increasing the efficiency of the cache.

Several (experimental) orders such as Morton and Hilbert are available

Parameters

tab: astropy table Data table with ra/dec or ra/dec/z points

colnms: list of strings Colum names for coordinates and redshift, e.g. ['ra','dec','z'], ['RAJ2000','DEJ2000','redshift']

par: Munch dictionary Input parameters dictionary. Must contain `mxh1`, `mxh2`, `mxh3` (when appropiate), the survey boundaries `sbound` and the desired method for ordering `pxorder`. For samples that cross the RA=0 limit, it can is useful to specify a custom RA boundary. See *Custom RA boundaries*

Returns

sidx [array] Sort index. Use `tab=tab[sidx]` to actually sort the data

`gundam.`**`qprint`**(*self*)

Prints a quick, nicely formatted version of Munch dictionaries, such as the **counts** ouput dictionary or the **par** input dictionary used by Gundam routines. Very useful to quickly explore what is stored inside.

This routine is added dynamically to the Munch class, so it can be accessed as `any_munch_obj.qprint()`

Note **counts** output dictionaries can also be explored using *`gundam.cnttable()`* to display a more elaborated tabular view of (exclusively) the counts calculated.

Examples

```
import gundam as gun

# Explore the many arrays and counts of a typical run
cred = gun.readcounts('redgalaxies.cnt')
cred.qprint()

# Check the parameters used to create the run
cred.par.qprint()
```

`gundam.`**`radec2xyz`**(*ra*, *dec*, *r=0.5*)

Converts a (ra,dec) coordinates to rectangular (x,y,z) coordinates in a sphere of radius r.

For `r=0.5`, this allows to speed up subsecuent haversine distance calculations between two points, by simply computing $dhav^2 = (x1 - x2)^2 + (y1 - y2)^2 + (z1 - z2)^2$

Parameters

ra,dec [float arrays] Right ascention an declination coordinates

r [float. Default=0.5] Sphere radius

Returns

(x,y,z) [tuple] Rectangular coordinates

gundam.**tpcf**(*npt*, *nrpt*, *dd*, *bdd*, *rr*, *dr*, *estimator*)

Return the (auto)correlation function for a given estimator and arrays of data and random counts

If DR counts are not needed (e.g. the 'NAT' estimator), just set `dr=0`

If boostrap errors are not needed or available, just set `bdd` to a zero-valued array with null 2nd dimension, e.g. `bdd=np.zeros([len(dd),0])`

Parameters

npt,nrpt [integer] Number of data and random particles

dd [float array] DD counts

bdd [float array] Bootstrap DD counts

rr [float array] RR counts in projected and radial separations

dr [float array] DR counts in projected and radial separations

estimator [string] Statistical estimator of the correlation function. Default='NAT'

- 'NAT' : Natural -> $DD/RR - 1$
- 'HAM' : Hamilton -> $DD * RR/DR^2 - 1$
- 'LS' : Landy-Szalay -> $(DD - 2DR + RR)/RR$
- 'DP' : Davis-Peebles -> $DD/DR - 1$

Returns

xi [float array] Correlation function

xierr [float array] Boostrap error estimate. Set to zero if `bdd` is nulled as explained above

Notes

See this paper for a nice review on estimators and their normalization factors. Here, the normalization factors are derived to : (1) keep estimator formulae clean, (2) avoid having operations such as (npt*(npt-1)) * dd, where counts are multiplied/divided by very big numbers when npt is large.

Examples

```
# Calculate the angular CF using the Landy-Szalay estimator
acf, acferr = gun.tpcf(npt,nrpt,dd,bdd,rr,dr,estimator='LS')
```

gundam.**tpccf**(*npt*, *nrpt*, *cd*, *bcd*, *cr*, *estimator*)

Return the (cross)correlation function for a given estimator and count arrays for data (D), random (R) and cross (C) samples.

For the moment the only estimator implemented is the Davis-Peebles : $\xi = CD/CR - 1$

If bootstrap errors are not needed or available, just set `bdd` to a zero-valued array, e.g. `bdd=np.zeros([len(dd),0])`

Parameters

npt,nrpt [integer] Number of particles in data (D) and random (R) samples

cd [float array] CD counts

bcd [float array] Bootstrap CD counts

cr [float array] CR counts

estimator [string]

- 'DP' : Davis-Peebles -> $CD/CR - 1$

Notes

C and D are data samples while R is random sample corresponding to D

Returns

fxi [float array] Cross-correlation function

fxierr [float array] Boostrap error estimates

Examples

```
import gundam as gun
c = gun.readcounts('qso_gal.cnt')

(ccf,ccferr) = tpccf(c.npt, c.nrpt, c.cd, c.bcd, c.cr, estimator='DP')
```

gundam.**tpcf_wrp**(*npt*, *nrpt*, *dd*, *bdd*, *rr*, *dr*, *dsepv*, *estimator*)

Return the projected (auto)correlation function for a given estimator and arrays of data and random counts

If DR counts are not needed (e.g. the 'NAT' estimator), just set `dr=0`

If boostrap errors are not needed or available, just set `bdd` to a zero-valued array with null 2nd dimension, e.g. `bdd=np.zeros([len(dd),0])`

Parameters

npt,nrpt [integer] Number of data and random particles

dd [float array] DD counts in projected and radial separations

bdd [float array] Bootstrap DD counts in projected and radial separations

rr [float array] RR counts in projected and radial separations

dr [float array] DR counts in projected and radial separations

dsepv [float] Bin size in radial direction

estimator [string] Statistical estimator of the correlation function

- 'NAT' : Natural -> $DD/RR - 1$
- 'HAM' : Hamilton -> $DD * RR/DR^2 - 1$

- 'LS' : Landy-Szalay -> $(DD - 2DR + RR)/RR$
- 'DP' : Davis-Peebles -> $DD/DR - 1$

Returns

wrp [float array] Correlation function

wrperr [float array] Boostrap error estimate. Set to zero if `bdd` is nulled as explained above

Notes

See this paper for a nice review on estimators and their normalization factors. Here, the normalization factors are derived to : (1) keep estimator formulae clean, (2) avoid having operations such as (npt*(npt-1)) * dd, where counts are multiplied/divided by very big numbers when npt is large.

Examples (xxxx TODO)

```
(wrp,wrperr) = tpcf_wrp(npt,nrpt,ddpv,bddpv,rrpv,drpv,dsepv,estimator='HAM')
```

gundam.**tpccf_wrp**(*npt*, *nrpt*, *cd*, *bcd*, *cr*, *dsepv*, *estimator*)

Return the projected (cross)correlation function for a given estimator and count arrays

If boostrap errors are not needed or available, just set `bcd` to a zero-valued array with null 2nd dimension, e.g. `bcd=np.zeros([len(ddXXX),0])`

Parameters

npt [integer] Number of data particles

nrpt [integer] Number of random particles

cd [float array] CD counts in projected and radial separations

bcd [float array] Bootstrap CD counts in projected and radial separations

cr [float array] CR counts in projected and radial separations

dsepv [float] Radial bin size

estimator [string] Statistical estimator of the correlation function

- 'DP' : Davis-Peebles -> $CD/CR - 1$ (C,D data samples, R is random of D)

Returns

wrp [float array] Projected cross-correlation function

wrperr [float array] Boostrap error estimate

Examples

```
# remains to do XXXXX
(wrp,wrperr) = tpccf_wrp(npt,nrpt,cd,bcd,cr,dsepv,estimator='DP')
```

8.5 Plotting Routines

gundam.**cntplot**(*cnt*, ***kwargs*)

Plot a correlation function from a **counts** output dictionary (either read from disk or passed directly). Both axes are set to log-space and axes labels are selected automatically according to the type of correlation (i.e. given by `par.kind`)

This is a wrapper for *gundam.plotcf()*, so all of its parameters can be specified too.

Parameters

cnt [string or Munch dictionary] Filepath for the counts (.cnt) file, or the **counts** dictionary itself

kwargs [keyword list] Any extra [key]=value pairs are passed to the underlying *gundam.plotcf()* routine

Examples

```
import gundam as gun

# Read a pcf run and plot the correlation function
cnt1 = gun.readcounts('/p01/redgalsP.cnt')
cntplot(cnt1)
# Plot the correlation function from a .cnt file
cntplot('/p01/redgalsA.cnt', label='angcf of redgals', fill=True)
```

gundam.**cntplot2D**(*cnt*, *estimator=None*, *slevel=5*, *write=False*, *figfile=None*, *xlabel=None*, *ylabel=None*, *cmap='jet'*, ***kwargs*)

Plot the 2D correlation function in the projected-radial space (r_p vs π space) with optional gaussian smoothing and contour levels

Parameters

cnt [string or Munch dictionary] Filepath for the counts (.cnt) file or the **counts** output dictionary

estimator [string. Default=None] Estimator for the correlation function. Any of ('NAT','LS','HAM','DP'). If `estimator=None`, then it is taken from `cnt.par.estimator`

slevel [float. Default=5] Smoothing level (namely the size of the Gaussian smothing kernel)

write [bool. Default=False] Save the figure to disk (default format is pdf). See *Notes* to save in other graphic formats

figfile [string. Default=None] Specify an alternative file name for the figure. If `None`, then choose `cnt.par.outfn` as default. Do not add extension.

xlabel, ylabel [string. Default=None] X-axis and Y-axis labels. If supplied, they override the default labels (r_p $[h^{-1}Mpc]$ and π $[h^{-1}Mpc]$)

cmap [string. Default='jet'] Colormap for the plot

kwargs [keyword list] Any extra [key]=value pairs are passed to `matplolib.pyplot.pcolor()` Use this to customize shading, edges, alpha, etc.

Notes

- The graphic format can be changed by passing the `figformat` key in `kwargs`, e.g. `figformat='pdf'`. Any format supported by matplotlib is valid.

Examples

```
# Check some nice Fingers of God and the Kaiser squashing
cntplot2D('lum_red_gals.cnt', cmap='viridis')
```

gundam.**comparecf**(*clist1*, *clist2=None*, *shift=0.0*, *fac=1.0*, *ploterrbar=True*, *fill=False*, *filtneg=False*, *label=None*, *plotratio=False*, *ratioxrange=None*, *color1=None*, *marker1=None*, *markers1=None*, *linestyle1=None*, *linewidth1=None*, *color2=None*, *marker2=None*, *markers2=None*, *linestyle2=None*, *linewidth2=None*, *f=None*, *ax1=None*, *ax2=None*)

Plot multiple correlation functions in a single figure for easy comparison. Optionally show an additional lower panel displaying the ratio of each function respect to a single "control" correlation (many-to-one) or to multiple correlations (one-to-one).

Parameters

clist1 [list of Munch dictionaries / list of strings] Count dictionaries or filepaths of .cnt files of correlations

clist2 [list of Munch dictionaries / list of strings. Default=None] List of control correlations. When `plotratio=True`, the y-values of each correlation curve in `clist1` are divided by those in `clist2` (one-to-one) and plotted in a lower panel. If `clist2` has a single element, the ratios are from all `clist1` divided by the single `clist1`. See *Notes* for more details

shift [float. Default=0.0] Fraction of bin size by which `x` values are shifted. Useful to slightly separate overlapping curves

fac [float. Default=1.0] Multiplication factor for `y` and `yerr`

ploterrbar [bool. Default=True] If `ploterrbar=True`, plot error bars according to `yerr`

fill [bool. Default=False] If `fill=True`, plot a filled semi-transparent error region instead of the usual error bars

filtneg [bool. Default=False] If `filtneg=True`, filter out points where (y-yerr)<0, i.e. those with large errors in a log plot

label: list Optional list of strings to label each correlation function. If ommited, the values are taken from the `outfn` key stored in the count objects

plotratio [bool. Default=False] If `plotratio=True`, plot also a lower panel with ratios of `clist1` respect to `clist2`

ratioxrange [list of form [xmin,xmax]] Only plot the ratio between xmin and xmax

color1,marker1,markers1,linestyle1,linewidth1 [lists] List of colors, marker symbols, marker sizes, line styles and line widths for curves in `clist1`

color2,marker2,markers2,linestyle2,linewidth2 [lists] List of colors, marker symbols, marker sizes, line styles and line widths for control curves in `clist2`

f [figure instance] Handle of existing Figure instance

ax1,ax2 [axes instances] Handles of correlation plot and ratio plot axes

Notes

The correlation curves in `clist2` are **not** plotted in the correlation function panel while curves present in **both clists** are **not shown** in the ratio panel (i.e. to avoid ratios of curves respect to themselves).

Returns

(f,ax1,ax2) or (f,ax1) [tuple of handles] Handles of figure, correlation axis (ax1) and ratio axis (ax2), if present

Examples

```
# Compare two w(rp) correlations
comparecf(['galred.cnt', 'galblue.cnt'])
# Compare one acf on disk with another passed as a counts ouput dictionary
comparecf(['/proj/galred.cnt', qso])
# Compare multiple samples and plot sample/control ratios
f,ax1,ax2 = comparecf(['galred.cnt', 'galblue.cnt', 'galgreen.cnt'], clist2=[
↪'allgals.cnt'], fill=True, plotratio=True)
# Add another curve to previous plot
comparecf(['qso.cnt'], clist2=['control_qso.cnt'], color2=['k'], f=f, ax1=ax1,␣
↪ax2=ax2, plotratio=True)
```

gundam.**fitpowerlaw**(*x, y, yerr, iguess=[1.0, -1.0], fitrange=None, plot=False, markfitrange=False, **kwargs*)

Fit a power-law of the form ax^{γ} to a correlation function over a given x-coordinate range. Optionally plot the fitted curve

Parameters

x [float array] x-coordinates such as cnt.rpm, cnt.thm, etc.

y [float array] x-coordinates such as cnt.wrp, cnt.wth, etc.

yerr [float array] Errors in y-coordinates such as cnt.wrperr, cnt.wtherr, etc.

iguess [list of floats. Default=[1., -1.]] Initial guesses for a and γ

fitrange [float array of form [xmin,xmax]. Default=None] Restrict fit to points inside the given interval

plot [bool. Default=False] Plot the fitted power-law curve

markfitrange [bool. Default=False] Overlay marks for points actually used in the fitting

kwargs [keyword list] Any extra [key]=value pairs are passed to `matplolib.pyplot.plot()` Use this to customize colors, linestyles, markers, etc.

Examples

```
import gundam as gun

c1 = gun.readcounts('galaxies.cnt')
cntplot(c1)
gun.fitpowerlaw(c1.rpm, c1.wrp, c1.wrperr, plot=True)
```

gundam.**plotcf**(*x*, *y*, *yerr*, *fac=1.0*, *write=False*, *figfile=None*, *par=None*, *angunit='deg'*, *xlabel=None*, *ylabel=None*, *label=None*, *shift=0.0*, *ploterrbar=True*, *fill=False*, *filtneg=False*, ***kwargs*)

Plot a correlation function from arrays of *x*, *y* and *yerr* values. Both axes are set to log-space and axes labels are selected automatically according to the type of correlation (i.e. given by `par.kind`)

Parameters

x,y,yerr [float arrays] x, y coordinates and corresponding errors of the correlation function. If `yerr=0` or all elements of yerr are <=0 or `ploterrbar=False`, no errorbar is plotted

fac [float. Default=1.0] Multiplication factor for `y` and `yerr`

write [bool. Default=False] Save the figure to disk (default format is pdf). See *Notes* to save in other graphic formats

figfile [string. Default=None] Specify an alternative file name for the figure. If specified, overrides the default which is to take it from `par.outfn`. Do not add extension.

par [dictionary of type Munch. Default=None] Used to pass `outfn` name to name saved figures when `write=True`

angunit [string. Default='deg']

- 'arcsec' : set ouput axis in arcsec (`x` values are unscaled)
- 'arcmin' : set ouput axis in arcmin (`x` values are scaled as `x`/60)
- 'deg' : set ouput axis in degrees (`x` values are scaled as `x`/3600)

xlabel, ylabel [string. Default=None] X-axis and Y-axis labels. If supplied, they override the default labels deduced from `par.kind`

label [string. Default=None] Label for the legend of the curve. If supplied, it will override the default label which is the basename of `par.outfn`. Note you have to issue at least a *plt.legend()* to actually display the legend box

shift [float. Default=0.0] Fraction of bin size by which `x` values are shifted. Useful to slightly separate overlapping curves

ploterrbar [bool. Default=True] If `ploterrbar=True`, plot error bars according to `yerr`

fill [bool. Default=False] If `fill=True`, plot a filled semi-transparent error region instead of the usual error bars

filtneg [bool. Default=False] If `filtneg=True`, filter out points where (y-yerr)<0, i.e. those with large errors in a log plot

kwargs [keyword list] Any extra [key]=value pairs are passed to the underlying `matplotlib.pyplot.plot()` routine, except for `alpha` which is passed to `pyplot.fill_between()`, `capsize` which is passed to `pyplot.errorbar()`, and `figformat` which is passed to `pyplot.savefig()`. Use this to customize colors, linestyles, markers, etc.

Notes

- Sucessive calls cycle between 4 predefined styles (for color, marker, linewidth, etc.) that can be overrriden by passing the corresponding [key]=value pairs in `kwargs`
- The output graphic format can be changed by passing the `figformat` key in `kwargs`, e.g. `figformat='pdf'`. Any format supported by matplotlib is valid.

Examples

```
import gundam as gun
c1 = gun.readcounts('redgalPCF.cnt')
c2 = gun.readcounts('redgalRCF.cnt')

plt.figure(1)    # Plot w(rp)
gun.plotcf(c1.rpm,c1.wrp,c1.wrperr,par=c1.par)
plt.figure(2)    # Plot xi(s)
gun.plotcf(c2.sm,c2.xis,c2.xiserr,par=c2.par,color='yellow')
```

8.6 Comprehensive Auxiliary Routines

These are helper routines that perform many common tasks (in the Python side) during a correlation run, such as collecting counts, i/o functions, initialization, logging, etc. For a nice example of how to use them, see source code of *gundam.pcf()*

gundam.**addlog**(*file*, *key*, *m*)

Read a text file and dump it into a key of a given dictionary. Useful to add entire logs from disk into Munch objects

Parameters

file [string] Complete path of file, usually any log file

key [string] Name of the key to be created

m [Munch dictionary] The dictionary where the key `m.key` will be created

gundam.**allequal**(*v*)

Fast way to check if all elements of a 1D-array have the same value. Useful to detect when all weights are set to 1.0, and hence to call faster versions of the counting routines

Parameters

v [array_like] Array to be checked

Returns

res [bool] True if all elements have the same value

gundam.**addpvcols**(*x*, *cfo*, *basecolname*, ***kwargs*)

Auxiliary function used by *gundam.cnttable()* to append columns to a table populated with the fields of **counts** ouput dictionary that store pair counts, all with nice column names. Works with 1d counts or 2d counts arrays (e.g. those from a pcf run when `nsepv>1`).

Parameters

x [astropy table] Table to add data

cfo [Much dictionary] Counts dictionary with the count arrays, e.g. cfo.dd, cfo.rr, etc.

basecolname [string] The name of the field to add, e.g. *dd*, and also the prefix for the column name, which if needed will be appended with *_001*, *_002*, etc. for each radial bin

kwargs : Any [key]=value pair to pass to the astropy Column constructor. Intended to pass a format specification for the column, such as `format='.4f'`

Returns

None, it modifies the input table `x`

gundam.**bound2d**(*decs*)

Return the (maximal) survey boundaries in (RA,DEC) for multiple samples. Note in the RA direction, the limits are **always** set as ramin=0. and ramax=360. to make the pair counting valid for surveys that cross the origin of coordinates

Parameters

decs [float array or list of arrays] DEC coordinates of one or more samples [deg]

Returns

bound [tuple] The limits (ramin,ramax,decmin,decmax) that enclose all input samples

gundam.**bound3d**(*decs*, *dcs*)

Return the (maximal) survey boundaries in (RA,DEC,DCOM) for multiple samples. Note in the RA direction, the limits are **always** set as ramin=0. and ramax=360. to make the pair counting valid for surveys that cross the origin of coordinates

Parameters

decs [float array or list of arrays] DEC coordinates of one or more samples [deg]

dcs [float array or list of arrays] Comoving distances of one or more samples [Mpc/h]

Returns

bound [tuple] The limits (ramin,ramax,decmin,decmax,dcmin,dcmax) that enclose all input samples

gundam.**cross0guess**(*ra*)

Guess if a set of RA coordinates cross the RA=0 division (by finding one source 1deg to the left and another 1deg to the right of RA=0, at least)

Parameters

ra [array] Right ascention coordiantes

Returns

res [bool] True if the sample seems to cross the RA=0 boundary

gundam.**buildoutput**(*par*, *npts=[]*, *binslmr=[]*, *dd=None*, *rr=None*, *dr=None*, *cd=None*, *cr=None*, *bootc=None*, *cf=None*, *cferr=None*)

Given a set of pair count arrays, assemble the final **counts** output dictionary, adding also bins, input parameters and relevant sample sizes.

This is for the main Gundam functions that calculate a multiple pair counts, i.e. *gundam.pcf()*, *gundam.pccf()*, *gundam.rcf()*, *gundam.rccf()*, *gundam.acf()*, *gundam.accf()*

Parameters

par [Munch dictionary] Input parameters

npts [list of int. Default=[]] A list with the nr. of points in each sample, i.e. [npt, npt1]. For example npts=[400, 4000] in cross-count cases; npts=[400] for auto-count cases

binslmr [list. Default=[]] The left, mid and right-side locations of each bin, returned of example by *gundam.makebins()*

dd,rr,dr [array. Default=None] The "dd", "rr" and "dr" count arrays, if available

cd,cr [array. Default=None] The "cd" and "cr" count arrays, if available

bootc [array. Default=None] The boostrap count array, if available

cf,cferr [array. Default=None] The correlation function and its error

Returns

counts [Munch dictionary] The ouput dictionary

gundam.**buildoutputC**(*par*, *npts=[]*, *binslmr=[]*, *dd=None*, *dr=None*, *bootc=None*, *intpi=None*, *intpib=None*)

Given a set of pair count arrays, assemble the final **counts** output dictionary, adding also bins, input parameters and relevant sample sizes.

This is for the main Gundam functions that calculate a single pair count, i.e. *gundam.rppi_A()*, *gundam.rppi_C()*, *gundam.s_A()*, *gundam.s_C()*, *gundam.th_A()*, *gundam.th_C()*

Parameters

par [Munch dictionary] Input parameters

npts [list of int. Default=[]] A list with the nr. of points in each sample, i.e. [npt, npt1]. For example npts=[400, 4000] in cross-count cases; npts=[400] for auto-count cases

binslmr [list. Default=[]] The left, mid and right-side locations of each bin, returned of example by *gundam.makebins()*

dd,dr [array. Default=None] The "dd" and "dr" count arrays, if available

bootc [array. Default=None] The boostrap count array, if available

intpi, intpib [array. Default=None] "dd" counts and boostrap "dd" counts integrated along all radial bins, if available

Returns

counts [Munch dictionary] The ouput dictionary

gundam.**check_kind**(*par*, *kind*)

Check that `par.kind = kind`. Useful to test, for example, if you are passing the right **par** object before really counting pairs

Parameters

par [Munch dictionary] Input parameters dictionary for Gundam routines

kind [string] The *kind* to check against, e.g. 'pcf', 'accf', 'rppiA', etc.

gundam.**closelog**(*log*, *runspyder=True*)

Close the log machinery used by the main counting routines by removing the handlers

Parameters

log [logging object] The log object

runspyder [bool. Default=True] If `runspyder=True`, remove the extra handler for stdout added when running under a Spyder console

gundam.**finalize**(*log*, *logf*, *logff*, *Ltime*, *t0*, *counts*)

Perform some finalization tasks common to all correlation runs, by logging loop/total times and adding the contents of log files to the **counts** output dictionary

Parameters

log [logging object] The log object

logfile [string] The complete path of the *.log* file

Ltime, Ttime [float] Loop time and total compute time of a correlation run

counts [Munch dictionary] Output dictionary containing all counts and correlations

gundam.**initialize**(*kind*, *par*, *nthreads=None*, *write=None*, *plot=None*)

Perform some initialization tasks common to all correlation function runs, such as reading **par** from disk file if needed, check the parameter *kind*, find out if running under Spyder, initialize the logs, etc.

Parameters

kind [string] The *kind* to check against, e.g. 'pcf', 'accf', 'rppiA', etc.

par [Munch dictionary] Input parameters dictionary for Gundam routines

nthreads [integer.] Number of threads to use. Passed just to store it under **par**

write [bool. Default=True] Flag to generate output count files. Passed just to store it under **par**

plot [bool. Default=True] Flag to generate plot. Passed just to store it under **par**

Returns

par [Munch dictionary] Input parameters dictionary for Gundam routines

log [logging object] The log object

logfile, logfilefort [string] The complete path of the *.log* file and the *.fotran.log* file

runspyder [bool. Default=True] Detect if running under a Spyder console

gundam.**logcallinfo**(*log*, *par*, *npts=[]*)

Write to log some useful runtime parameters of the main counting routines

Parameters

log [logging object] The log object

par [Munch dictionary] Input parameters dictionary for Gundam routines

npts [list of 1, 2 or 3 integers] The nr of objects in the input data table, random table, and cross sample table

gundam.**logtimming**(*log*, *cntid*, *t*)

Write a message to a log instance, showing the compute time for the counts identified with a certain ID string

Parameters

log [logging object] Log object

cntid [string] String to ID a certain type of counts, e.g. 'DD', 'RR', DR', etc.

t [float] The time elapsed [s]

gundam.**readcounts**(*cfile*, *silent=False*)

Read from disk the **counts** dictionary generated by the main counting routines.

Parameters

cfile [string] Filepath for the counts (.cnt) file

silent [bool] If False, print a status message indicating the file was read. Default=False

Returns

counts [Munch dictionary] The counts object

gundam.**readpars**(*filename*)
Load from a JSON (.par) file the input parameters dictionary used by many Gundam routines.

Parameters

filename [string] Filepath of .par file

gundam.**savecounts**(*cnt*, *altname=None*)
Save to disk the **counts** output dictionary returned by the main counting routines.

The default file name is `cnt.par.outfn` + *.cnt*, which can be overriden as `altname` + *.cnt*, if supplied

Parameters

cnt [Munch dictionary] The **counts** object

altname [string. Default=None] If supplied, use an alternative file name instead of `cnt.par.outfn`

gundam.**savepars**(*par*, *altname=None*)
Save the parameters dictionary **par**, such as the one generated by `gundam.packpars()`, in a JSON file. By default it is named as `par.outfn` + *.par*

Parameters

par [Munch dictionary] Input parameters dictionary for Gundam routines

altname [string. Default=None] If supplied, use an alternative file name instead of `par.outfn` + *.par*

Examples

```
import gundam as gun

# Get default values for an angular CF run and save to disk
par = gun.packpars(kind='acf', outfn='/proj/acfrun01')
gun.savepars(par)
```

gundam.**setlogs**(*par*, *runspyder=True*)
Set up the log machinery used by the main counting routines by creating the logger object, adding the required handlers and cleaning previous logs if present

Parameters

par [Munch dictionary] Input parameters dictionary for Gundam routines

runspyder [bool. Default=True] If `runspyder=True`, add an extra handler for stdout when running under a Spyder console

Returns

log [logging object] The log object

logfile [string] The complete path of the .log file

`gundam.`**`writeasc_cf`**(*lb*, *mb*, *rb*, *f*, *ferr*, *par*, *fmt='%17.5f'*, *altname=None*)

Write an ASCII file for w(rp) / xi(s) / w(th) ouput counts produced by the code

Parameters

lb,mb,rb [float arrays] Left, mid and rigth-side of bins

f, ferr [float arrays] Correlation function and its error

par [Munch dictionary] Used to pass `par.outfn` to name the output file

fmt [string. Default='%17.5f'] Numeric formatting string

altname [string. Default=None] If supplied, use an alternative file name instead of `par.outfn`

Examples

```
import gundam as gun
c1 = gun.readcounts('redgals.cnt')

writeasc_cf(c1.rpl, c1.rpm, c1.rpr, c1.wrp, c1.wrperr, c1.par)
```

`gundam.`**`writeasc_rppicounts`**(*lb*, *mb*, *rb*, *rppi*, *par*, *fmt='%17.5f'*, *cntid=None*, *altname=None*)

Write an ASCII file for a rp-pi count array produced by the code. This is a 2D array of counts in the projected (rp) and radial (pi) directions.

The columns in the output will be [*lb mb rb tot_counts rppi*] where the first 3 are the left, mid and right-side of bins, *tot_counts* are the counts integrated for all radial bins, and *rppi* has one column for each radial bin

Parameters

lb,mb,rb [float] Left, mid and rigth-side of bins

rppi [float array] 2-dimensional count array. Usually this is one of the fields cnt.dd, cnt.rr, etc. of a projected correlation run

par [Munch dictionary] Used to pass various data, including `par.outfn` to name the output file

fmt [string. Default='%17.5f'] Numeric formatting string

cntid [string. Default=None] ID string for column headers. Usually can be 'dd', 'rr', 'dr', etc. Also appended as the extension of the ouput file (when `altname=None`)

altname [string. Default=None] If supplied, use an alternative file name instead of `par.outfn` + *.cntid*

Examples

```
import gundam as gun
c1 = gun.readcounts('redgals.cnt')

c1.par.outfn
'/home/myuser/sdss/redgals'

# Write the DD counts in rp-pi dimensions
gun.writeasc_rppicounts(c1.rpl, c1.rpm, c1.rpr, c1.dd, c1.par, cntid='dd')

# Inspect the output file
with open('redgals.dd', 'r') as f:
    print(f.read(), end="")

#  lb        mb        rb        dd         dd_001    dd_002    dd_003    ...
#  0.10000   0.12417   0.14835   11509.00   2082.00   1500.00   1168.00   ...
#  0.14835   0.18421   0.22007   20273.00   3122.00   2378.00   1899.00   ...
#  0.22007   0.27327   0.32647   36169.00   4940.00   3845.00   3283.00   ...
#  0.32647   0.40539   0.48431   64866.00   8453.00   6302.00   5236.00   ...
#  ...
```

gundam.**writeasc_counts**(*lb*, *mb*, *rb*, *c*, *par*, *fmt='%17.5f'*, *cntid=None*, *altname=None*)

Write an ASCII file for a (1-dimensional) counts array produced by the code.

The columns in the output will be [*lb mb rb c*] where the first 3 are the left, mid and right-side of bins and *c* are the counts

Parameters

lb,mb,rb [float] Left, mid and rigth-side of bins

c [float array] Counts array. Usually this is one of the fields cnt.dd, cnt.dr, cnt.rr, etc. of a correlation run

par [Munch dictionary] Used to pass `par.outfn` to name the output file

fmt [string. Default='%17.5f'] Numeric formatting string

cntid [string. Default=None] ID string for column header. Usually can be 'dd', 'rr', 'dr', etc. Also appended as the extension of the ouput file (when `altname=None`)

altname [string. Default=None] If supplied, use an alternative file name instead of `par.outfn` + *.cntid*

Examples

```
import gundam as gun
c1 = gun.readcounts('bluegals.cnt')

# Write the DD counts in angular dimensions. Use an alternative file name
gun.writeasc_counts(c1.thl, c1.thm, c1.thr, c1.dd, c1.par, cntid='dd', altname=
↪'akounts')

# Inspect the output file
```

```
with open('akounts', 'r') as f:
    print(f.read(), end="")

# lb        mb        rb          dd
# 0.01000   0.01206   0.01413     3178.00
# 0.01413   0.01704   0.01995     6198.00
# 0.01995   0.02407   0.02818    12765.00
# 0.02818   0.03400   0.03981    24888.00
# 0.03981   0.04802   0.05623    49863.00
# 0.05623   0.06783   0.07943    98883.00
...
```

8.7 Fortran Wrapper Routines

These are the wrappers that call Fortran pair counting routines. They are intended to: (1) choose the fastest counting routine depending whether weights, bootstrap errors, etc. are requested or not; and (2): concentrate all mayor Fortran calling in one place

Unless you are building your own custom script, heavily modifying, or extending the core algorithms, you normally should not need to use these functions

`gundam.`**`pairs_auto`**(*par*, *wunit*, *logff*, *tab*, *x*, *y*, *z*, *sk*, *ll*, *dc=None*)

Wrapper for calling Fortran counting routines (for auto-pairs)

This function isolates the call of external Fortran routines, choosing the correct one based on geometry, and choosing the fastest depending whether, weights, bootstrap errors, etc. are requested or not.

Parameters

par [Munch dictionary] Input parameters

wunit [bool]

- True : all weights are equal to 1
- False : at least one weight is not 1

logff [string] File for logging the output of Fortran routines

tab [astropy table] Table with data particles

x,y,z [arrays] Rectangular coordinates of data particles

sk,ll [arrays] Skip table (SK) and linked list table (see `gundam.skll2d()` or `gundam.skll3d()`)

dc [array [optional]] Array of comoving distances. Not needed for angular counts

Returns

tt [list of ndarray] Ouput counts as returned by Fortran counting routines

`gundam.`**`pairs_cross`**(*par*, *wunit*, *logff*, *tab*, *x*, *y*, *z*, *tab1*, *x1*, *y1*, *z1*, *sk1*, *ll1*, *dc=None*, *dc1=None*)

Wrapper for calling Fortran counting routines (for cross-pairs among two tables called D(data) and Random(R))

This function isolates the call of external Fortran routines, choosing the correct one based on geometry, and choosing the fastest depending whether, weights, bootstrap errors, etc. are requested or not.

Parameters

par [Munch dictionary] Input parameters

wunit [bool]

- True : all weights are equal to 1
- False : at least one weight is not 1

logff [string] File for logging the output of Fortran routines

tab,tab1 [astropy tables] Table with data particles (D) and random particles (R), respectively

x,y,z,x1,y1,z1 [arrays] Rectangular coordinates of particles in D table and R rable, respectively

tab1 [astropy table] Table with data particles (D)

sk1,ll1 [arrays] Skip table (SK) and linked list table (see `gundam.skll2d()` or `gundam.skll3d()`)

dc,dc1 [arrays [optional]] Array of comoving distances of particles in D and R tables. Not needed for angular counts

Returns

tt [list of ndarray] Ouput counts as returned by Fortran counting routines

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Index