Monthly Notices
of the
ROYAL ASTRONOMICAL SOCIETY

MNRAS **00,** 1 (2019) doi:10.1093/mnras/stz1469

GUNDAM: a toolkit for fast spatial correlation functions in galaxy surveys

E. Donoso★

Instituto de Ciencias Astronómicas, de la Tierra, y del Espacio (ICATE), UNSJ, CONICET, San Juan, Argentina

Accepted 2019 May 27. Received 2019 May 27; in original form 2018 February 28

ABSTRACT

We describe the capabilities of a new software package to calculate two-point correlation functions (2PCFs) of large galaxy samples. The code can efficiently estimate 3D/projected/angular 2PCFs with a variety of statistical estimators and bootstrap errors, and is intended to provide a complete framework (including calculation, storage, manipulation, and plotting) to perform this type of spatial analysis with large redshift surveys. GUNDAM implements a very fast skip list/linked list algorithm that efficiently counts galaxy pairs and avoids the computation of unnecessary distances. It is several orders of magnitude faster than a naive pair counter, and matches or even surpass other advanced algorithms. The implementation is also embarrassingly parallel, making full use of multicore processors or large computational clusters when available. The software is designed to be flexible, user friendly and easily extensible, integrating optimized, well-tested packages already available in the astronomy community. Out of the box, it already provides advanced features such as custom weighting schemes, fibre collision corrections and 2D correlations. GUNDAM will ultimately provide an efficient toolkit to analyse the large-scale structure 'buried' in upcoming extremely large data sets generated by future surveys.

Key words: galaxies: general – quasars: general – large-scale structure of Universe.

1 INTRODUCTION

The two-point correlation function is one of the main statistical tools to study the large-scale structure of galaxies in the universe and the first-order measure for characterizing deviations from a uniform distribution. It can quantitatively tell us the degree of clustering of a certain population as a function of scale, constrain the bias, initial conditions, and models of structure formation. Its importance in cosmology is perhaps best exemplified when we consider it as the Fourier transform of the power spectrum of the density field.

In recent years, the advent of wide-area redshift surveys enabled to estimate correlation functions of large galaxy populations with unprecedented accuracy, thanks to high-quality data with high-redshift completeness spanning over large volumes. Surveys such as the Sloan Digital Sky Survey (SDSS; York et al. 2000; Alam et al. 2015) has catalogued 200 million galaxies with photometric redshifts, including 3 million spectra of quasars and galaxies. The Wide-field Infrared Survey Explorer (WISE; Wright et al. 2010) has already produced a database of 0.75 billion sources, most of them galaxies, and upcoming surveys such as the Large Synoptic Survey Telescope (LSST; Ivezić et al. 2008; Abell et al. 2009) will measure over 37 billion galaxies and stars from about 350 billion single epoch detections.

Whether we want to investigate the 3D or 2D distribution of galaxies (or of any other kind of point-like objects) extracted from large surveys or numerical simulations, a fundamental problem is how to count pairs of objects in a reasonable time. Whereas modern computer processors become faster in every generation, the number of floating-point operations required increase substantially for larger samples, up to the point of making some calculations unfeasible without expensive, high-performance computer clusters. While some scientific institutions have access to large computational facilities, many other small institutes and university departments lack such equipment. Recent advances in the application of GPU technology for certain kinds of massive parallel operations are slowly improving on this situation, but at the cost of significant programming efforts to retrofit algorithms, sometimes requiring to recompile complex scientific libraries. One of the design goals of GUNDAM is to make the computation of 2PCFs more *affordable*, even using a single laptop or desktop computer.

Computing n-point correlation functions is a type of generalized n-body problems from a wide family that ranges from kernel density estimation to nearest neighbour searches. All such problems can be decomposed in a series of evaluations of kernel functions on a set of points in a certain metric space. In its most generic form, estimating a 2PCF involves comparing the distances of each point in data set with each other point and counting the number of pairs that are within a certain limit distance. If we have N data points, it can be solved naively by computing the N^2 distances

★ E-mail: edonoso@conicet.gov.ar

between them, but in practice N is often large enough to make such computation unfeasible. This is more than evident for present astronomical surveys that already detect hundreds of millions of sources, and that will soon reach several billion objects in upcoming survey databases. Furthermore, the problem gets severely magnified if the 2PCF computation is embedded in a larger Monte Carlo or Bayesian inference framework where it has to be repeated hundreds or thousands of times.

In view of this situation, it is mandatory to use more clever algorithms and adopt acceleration techniques to speed up these computations. Perhaps the most well-known family of methods relies on binary trees in order to partition the space and efficiently find all neighbours of a given point without scanning the entire data set (e.g. *kd*-trees, Gray & Moore 2001; *ball*-trees, Omohundro 1991). In this work, we present a new counting algorithm based on linked lists and its implementation as part of a feature-rich software toolkit to estimate spatial correlation functions. No less important, this package and its source code is made freely available to the community.[1]

This paper is organized as follows. In Section 2, we explain the methodology adopted to calculate the two-point correlation function and the techniques employed to accelerate pair computation. Section 3 presents single-core and multicore benchmarks along with two examples of application to real astronomical data sets that highlight the features of our code. Finally, in Section 4 we summarize our results and discuss further improvements. Tables in the Appendix section list the most useful routines included in code as well as examples of input and output dictionaries.

Throughout the paper, we assume a flat ΛCDM cosmology, with $\Omega_{\rm m} = 0.25$ and $\Omega_\Lambda = 0.75$. Unless otherwise stated, we adopt $h = H_0/(100\ \mathrm{km\ s^{-1}\ Mpc^{-1}})$ and present the results in units of $h^{-1}\,\mathrm{Mpc}$ with $h = 1$. While GUNDAM is designed primarily for galaxy surveys, we will refer indistinctly to galaxies, particles or objects.

2 IMPLEMENTATION

2.1 The two-point correlation function

For a set of N discrete points in a finite volume V, the probability to find one point in an infinitesimal volume $\mathrm{d}V$ is

$$\langle \mathrm{d}P \rangle = \overline{n}\,\mathrm{d}V, \tag{1}$$

where $\overline{n}$ is the mean number of objects per unit volume, i.e. $N = \overline{n}\,V$ (Peebles 1980). Now, the joint probability of finding two points inside volumes $\mathrm{d}V_1$ and $\mathrm{d}V_2$ at positions r_1 and r_2 separated by distance $r = r_1 - r_2$ can be defined through the relation

$$\langle \mathrm{d}P \rangle = \overline{n}^2\,\mathrm{d}V_1\,\mathrm{d}V_2[1 + \xi(r)], \tag{2}$$

where $\xi(r)$ is the spatial two-point correlation function or 2PCF. For a pure Poisson process the probabilities of both particles are independent and $\xi \equiv 0$. In the more general case when $\xi(r) > 0$, we can clearly see that the 2PCF represents the excess probability over random of finding a pair of objects separated a distance r. While in principle it depends on the positions r_1 and r_2, for homogeneous fields the 2PCF depends only on the separation r. Therefore, for a discrete set of points we can define

$$\xi(r) = \frac{\mathrm{DD}(r)}{\mathrm{RR}(r)} - 1, \tag{3}$$

where $\mathrm{DD}(r)$ and $\mathrm{RR}(r)$ are the pair counts of the data and random samples, respectively. Real galaxy surveys often have complicated boundaries and complex selection functions, i.e. different parts of the sky are mapped at different depths and some objects or regions have higher preference over others. An effective technique to deal with such difficulties is to compare observed pairs counts with artificial catalogues of randomly distributed objects that mimic the angular and radial selection functions. These random samples are usually 10–20 times larger than data sample itself in order to keep the shot noise under control at small separations.

Other than equation (3), there are several statistical estimators for the 2PCF that present different bias, variance properties, computational advantages, and caveats. An elaborate discussion about estimators is out of the scope of this paper (e.g. see Kerscher, Szapudi & Szalay 2000; Vargas-Magaña et al. 2013). Here, we just present the estimators that have been implemented in GUNDAM, namely

$$\xi_N(r) = c_1 \frac{\mathrm{DD}(r)}{\mathrm{RR}(r)} - 1 \tag{4}$$

$$\xi_{\rm H}(r) = c_2 \frac{\mathrm{DD}(r)\mathrm{RR}(r)}{\mathrm{DR}(r)^2} - 1 \tag{5}$$

$$\xi_{\rm L}(r) = \frac{c_2\mathrm{DD}(r) - 2c_3\mathrm{DR}(r) + \mathrm{RR}(r)}{\mathrm{RR}(r)} \tag{6}$$

$$\xi_{\rm D}(r) = c_4 \frac{\mathrm{DD}(r)}{\mathrm{DR}(r)} - 1 \tag{7}$$

that correspond to the natural (Peebles & Hauser 1974), Hamilton (1993), Landy & Szalay (1993), and Davis & Peebles (1983) estimators. For n data points and $n_{\rm r}$ random points, the normalization constants can be written as $c_1 = n_{\rm r}(n_{\rm r} - 1)/n(n - 1)$, $c_2 = 4n(n - 1)/n_{\rm r}(n_{\rm r} - 1)$, $c_3 = (n_{\rm r} - 1)/2n$ and $c_4 = 2n_{\rm r}/(n - 1)$.

For cross-correlations, there are analogue versions of these estimators that can be calculated readily with the individual pair counting routines (see Appendix A1). GUNDAM directly implements a variation of ξ_N for cross-correlations given by $\xi_N^{\rm cc}(r) = c\,\mathrm{QD}(r)/\mathrm{QR}(r) - 1$ (e.g. Shanks et al. 1983), where D and R are the data and random samples, and Q is the sample to get cross-correlated. Estimating a 2PCF usually requires calculating a set of DD, RR, and DR pair counts, and in most cases the RR pair counts will vastly dominate the total computing time.

2.1.1 *Coordinates and distances*

There are several ways to estimate the distance between galaxy pairs which of course depend on the problem and the geometry adopted. GUNDAM calculates the radial (π), projected ($r_{\rm p}$), and redshift-space distance (s) between two galaxies i and j as

$$\pi = |\mathrm{dc}_i - \mathrm{dc}_j| \tag{8}$$

$$r_{\rm p}^2 = 4\,\mathrm{dc}_i\,\mathrm{dc}_j\,[(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2] \tag{9}$$

$$s^2 = \pi^2 + r_{\rm p}^2, \tag{10}$$

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

$$x = 0.5\cos(\mathrm{dec})\sin(\mathrm{ra}) \tag{11}$$

[1] https://github.com/samotracio/gundam

$$y = 0.5\cos(\mathrm{dec})\cos(\mathrm{ra}) \tag{12}$$

$$z = 0.5\sin(\mathrm{dec}). \tag{13}$$

Computationally, this strategy for calculating distances is significantly faster than others, because it avoids a large number of trigonometric operations inside loops. Nevertheless, with slight modifications it is possible to operate directly with spherical coordinates. Note also it is more efficient to operate with r_p^2 instead of r_p, saving a costly square root computation.

2.2 Linked lists and skip lists

In computer science, linked lists are some of the most used data structures in a wide range of applications. In its simplest form, a singly linked list is a sequence of dynamically allocated nodes, each containing data and a reference (pointer) to the next element in the list. A special node defines the start of the list, which is traversed node by node until a special null pointer determines its ending.

Compared to a traditional array, such a structure is highly efficient at element insertion or removal because there is no need to reallocate the entire data set. Arrays are in most cases contiguous blocks of memory that cannot be reshaped or expanded without performing expensive operations. An element at any point of a linked list of any length can be inserted or removed in $O(1)$ time. On the other hand, to access a specific element of a linked list requires to traverse all previous nodes starting from the first one, i.e. a linear search time. Thus, simple linked lists cannot compete against the nearly instantaneous random access time of arrays.

Moreover, as in principle the nodes are allocated in non-contiguous chunks of memory, traversing a linked list is very likely to cause cache misses that can severely impact the execution time in current microprocessor architectures. A cache is a small pool of memory that operates faster than standard RAM and temporally stores instructions and data that otherwise would need to be fetched from RAM. Many modern microprocessors have three cache levels (L1, L2, L3) with typical latencies of 4, 12, and 36 clock cycles,[2] respectively, whereas RAM latencies are 50–100 times higher than the L1 memory. Which information is loaded into the cache depends on sophisticated algorithms and linked lists trace an unpredictable memory access pattern, offsetting the benefits of the locality of reference.

One approach to improve on these performance issues is by using skip lists. A skip list is a data structure that allows to quickly locate a given node, by traversing another sub-sequence that links specific nodes and skipping elements in between. Thus, a skip list is a (linked) list of pointers that signals shortcut nodes, allowing fast searches within an ordered sequence of elements. Furthermore, it is also possible to build a hierarchy of nested skip lists where each successive level skips fewer elements than the previous one.

For the purpose of counting galaxy pairs in surveys or mock catalogues, the data consists of two angular coordinates (RA, DEC), and a radial coordinate such as the redshift Z (in the 3D case), which is transformed into a comoving distance. A regular grid of nodes or cells and a linked list is constructed for all galaxies so that the pointers always lead to the next element in increasing DEC or comoving distance. Then, a second linked list – the skip list – is built by identifying and linking the particle with the lowest DEC value or comoving distance of each cell. Fig. 1 illustrates these concepts, showing how the skip list jumps between cells while the linked list jumps between the particles inside.

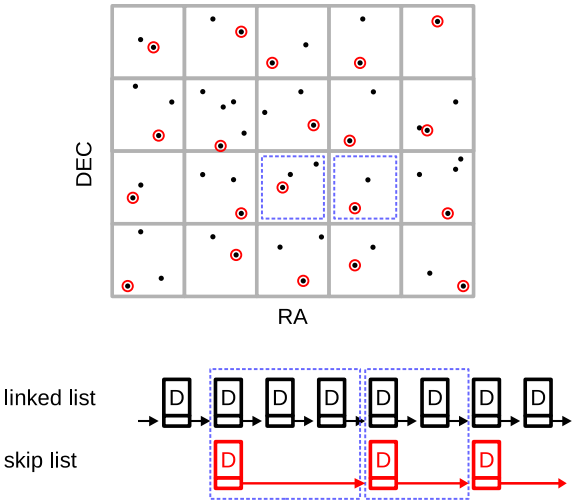

Figure 1. Schematic representation of the linked list–skip list algorithm implemented in GUNDAM. Particles are assigned to a grid of cells and a linked list is built pointing to the next particle in DEC order. The skip list provides a shortcut to jump to the following cell, skipping all particles in between, and therefore unnecessary distance calculations.

Such a counting scheme also allows to quickly identify neighbouring cells around a given particle, discarding objects located in cells that are too far apart to contribute within the range requested for analysis.

2.3 Improving data locality: pixel sorting

One strategy to improve the locality of reference in linked lists is to sort the data divided in cells according to an order given by the cell index before actually constructing the linked list itself. This way, data points in a given cell sit closer in memory, largely increasing the efficiency of the cache. Since these cells do not necessarily have to match the cells employed by the skip list, we refer to these as pixels instead of cells (either 2D or 3D).

After arranging the data in a grid of RA, DEC, and Z pixels (when pertinent), GUNDAM sorts data first according to the pixel index in Z, then according to the index in DEC, and then according the RA pixel index. This effectively (yet partially) translates the spatial proximity of objects into memory proximity of array data points. This scheme of presorting input data is notably faster than simply sorting all input objects by a single coordinate, such as RA or DEC, and has a negligible impact of 1–2 per cent in the total compute time.

Note the main sorting key is the DEC pixel index (or Z in the 3D case) because this is the coordinate associated with the linked list order. The sorting grid is chosen to match the one adopted for the skip list in all dimensions. In Section 3, we will analyse the effect of this methodology.

This method of sorting pixels is similar to Z-curve techniques used in computer science to map multidimensional data into a 1D index. GUNDAM includes options to experiment with alternative ordering methods such as Morton or Hilbert in 2D and 3D (for a review see Samet 2006). Fig. 2 illustrates these techniques with a sample of 700 000 galaxies extracted from SDSS. The coordinate

[2] https://www.7-cpu.com/cpu/Haswell.html

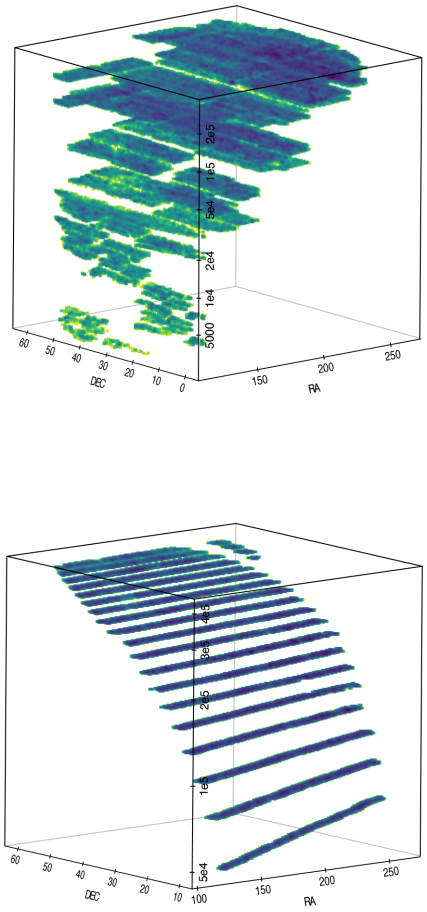

Figure 2. Data sorting techniques implemented in GUNDAM for 700 000 galaxies extracted from SDSS. The vertical coordinate is the position in memory of a given galaxy after reordering its coordinate arrays using the Morton technique (top) and our preferred pixel sorting method (bottom). As shown in Section 3, this sorting in advance impacts strongly in the performance of the pair counting algorithm.

perpendicular to RA–DEC represents the sequential position of a given galaxy in memory right after being sorted with Morton (top) and our pixel sorting method (bottom). We found that while these more complex algorithms can sort the data better, i.e. the subsequent pair counting is slightly faster, at some point the cost of complex sorting counterbalances the gain in speed. Our simple pixel sorting technique is not as good in improving data locality, but performs faster in overall as it requires significant less overhead time.

2.4 Improving grid dimensions

The skip list–linked list algorithm of GUNDAM relies in dividing data into a grid of cells, so that it is reasonable to expect its performance will depend on the number of cells and/or the number of particles per cell. Thus, given an arbitrary input sample it is important to know, at least approximately, what is the best gridding scheme.

However, this is not easy to find out a priori, since it would require knowing the distances between particles. At some point, the GUNDAM implementation consists of 10 nested loops with numerous branching conditions, which makes very difficult to predict the relative average costs of each section of code. In addition, this will also depend on the hardware, as different CPU have different cache size, architecture, and branch prediction algorithms.

In order to find out the 'right' dimensions to reduce execution times, we run a battery of tests for samples of different size N over a contiguous (rectangular) patch of sky, using grids of various sizes and recording the grid parameters that lead to the lowest compute time. For example, for angular pairs we find that a roughly constant density of 22 particles per cell result in the best timings for $N > 10^5$ objects, with a slight linear dependence with N for $10^4 < N < 10^5$. Once a target cell density is set, we find the number of DEC cells from the best-fitting empirical relation between N and DEC cell size as derived from our tests, and adjust the number of RA cells to reach the target cell density. The procedure for 3D space grids is sightly more complicated, but very similar.

We shall note that with this procedure, the optimal grid parameters that are estimated depend on the sample, the number of bins requested and of course on the hardware. We have used uniform random samples (because counting RR pairs will dominate the computing time of a 2PCF) and a reasonable set of bins (e.g. a couple dozen bins up to $\theta = 10$ deg). The fine-tuning should be effective in most desktop processors with 8MB of L3 cache memory (see Section 3 for CPU details).

This procedure to automatically find out the best grid dimensions is encoded in two PYTHON routines, which are easy to modify by the user, and a single flag (`autogrid`) control its application. The user can also modify the target density (e.g. to accommodate larger caches) or directly provide the dimensions of the grid. Section 3 analyses the impact in performance of this feature.

2.5 Other optimizations

2.5.1 *Reverse bin checking*

After the distance d_{ij} between galaxies i and j is calculated, we need to loop though the distance bins, testing if d_{ij} falls inside and increment the corresponding pair count of that bin. This operation, while extremely simple, is very important considering a typical (r_p, π) run can involve billions of pair distances tested typically over 100–1000 bins or even more. As there are many more pairs at large separations than close pairs, they are much more likely to fall into the last separation bins. Therefore, GUNDAM checks bins in reverse order starting from the last one, i.e. at the largest separation, and ending in the first one, i.e. the smallest separation considered. This is significantly more efficient than checking bins the other way around.

2.5.2 *Early loop termination*

Related to the previous optimization, once the bin corresponding to d_{ij} is found, there is no need to keep testing. In such cases GUNDAM immediately branches out, saving the cost of executing unnecessary comparisons.

2.5.3 *Loop unrolling*

After initialization of a loop, every individual iteration involves some overhead work, i.e. incrementing the loop counter, checking exit conditions, etc. For example, a vector of 20 numbers x_i can be squared by looping 20 times and squaring each number in turn, or by looping 10 times and squaring x_i and x_{i+10} at each iteration. This technique is called loop unrolling, by a factor 2 for this particular case.

Under the right conditions, saving loop iterations can lead to performance gains. A high unroll factor can make a loop faster,

but also increases the number of instructions in the compiled code (wasting the instruction cache) and requires to know a priori the number of iterations. GUNDAM uses individual unrolling of specific loop iterations. After the distance d_{ij} between galaxies i and j is calculated, we need to loop though the distance bins, testing if d_{ij} falls inside and increment the corresponding pair count. For example, if GUNDAM has to check θ_1–θ_n angular bins, it individually checks the bins θ_n–θ_{n-4} in order before entering the loop for the remaining bins. As an overwhelming majority of pairs will fall in last few bins, it reduces the overhead of setting up unnecessary loop constructions. In many cases, this simple arrangement can count pairs faster than a regular loop through all bins.

2.5.4 Loop nesting order

The memory storage scheme for multidimensional arrays determines an optimum way of nesting loops in order to minimize cache misses. Fortran is a column major language, where 2D arrays are stored columnwise so that when an element is accessed, a block of references to adjacent elements in that column are cached also. These references will be readily available while processing the next element, saving the excursion to retrieve them again from the always slower RAM memory.

This is particularly important for counting pairs in large galaxy samples, where nested loops can push the array access count by trillions or more. We have made sure that the Fortran routines implemented in GUNDAM loop over data in the proper order, meaning the first index (which varies fastest) is the most deeply nested. In this way, page faults are minimized and cache paging activity is reduced.

A related minor optimization consists in reducing cache competition within nested loops. When a data vector corresponding to a galaxy i is referenced, an entire block around it is promoted higher in the memory hierarchy, reducing the cache space available for data of galaxy j. Therefore, we carry temporary scalar variables from outer i loops to inner j loops, instead of directly accessing the data vectors from within the nested loop.

2.6 Storage

From a strict point of view, a 2PCF is no more than a set of points in the plane or space, and perhaps suitable error bars. A simple tabular representation in an ASCII file could be enough to store and/or present such information. However, in practice we work with dozens, hundreds, and even thousands of 2PCFs, each with its own set of DD/RR/DR/total/bootstrap counts arrays with different dimensionality. We would also like to keep a detailed yet unobtrusive record of runtime information and a flexible representation of both input parameters and output results. Thus, a more rich-full representation of a 2PCF run is highly desirable for an efficient workflow.

GUNDAM uses simple but powerful containers based on PYTHON dictionaries with attribute-style access. A PYTHON dictionary is a set of (key: value) pairs, indexed by the key index, where value can be virtually any other object such an array, a text log file and even complete plots. Such a dictionary can be saved ('pickled', in PYTHON jargon) in a single file and later read back or shared with collaborators. In particular, GUNDAM employs dictionaries that can be accessed as attributes, i.e. using a dot-style notation typical of Java object-oriented language.

By default, GUNDAM output consists of a set of ASCII files for the various count arrays, logs, input parameters, and the 2PCF estimated. All this information is also stored as (key: value) pairs in a single output dictionary. In fact, the set of input parameters that control all aspects of GUNDAM algorithms are also encoded in a single parameter dictionary. See Appendix B1 for a detailed description of input/output keys.

2.7 Parallel approaches

Unlike other calculations such as simulations of particle force interactions, the task of computing a 2PCF is embarrassingly parallel. It is relatively straightforward to split the computation into pieces that can be processed independently and added together at the end. GUNDAM follows this approach, i.e. multiple threads or processes (local or remote) are spawned, each assigned to count pairs using the next available core or thread in a small region of sky determined by the gridding along the DEC dimension.

For computations in a single node, the code uses OpenMP[3] instructions to create and manage multiple threads of execution sharing a common memory space. This allows to significantly reduce the memory footprint when compared to multiprocessing. Since iterations can take different amounts of time to complete (e.g. the survey footprint is highly irregular along the DEC dimension, or the density varies widely between neighbouring cells) the typical workload is strongly imbalanced. To address this, the code loops employ dynamic scheduling, where the amount of work scheduled to each thread is dynamically distributed at runtime. The result is an optimum use of thread pools, when compared with static or guided chunk size scheduling.

For very large data sets it is better, if not mandatory, to distribute the computations among multiple nodes. For this purpose, the code takes advantage of the powerful architecture for parallel and distributed computing given by IPYPARALLEL.[4] This is a well-tested package that enables many kinds of parallelism such as SIMD (single program multiple data), MPMD (multiple program multiple data), and MPI (Message Passing Interface), among others. It also supports most job queue management systems and launchers used in high-performance clusters (e.g. SGE, Torque, LSF, SSH, etc.). Briefly, IPYPARALLEL architecture consist of (i) a central hub that keeps track of engine connections, clients, task requests and results; (ii) task schedulers that route tasks towards engines; (iii) a set of compute engines that listens for requests, run code, and return results, and (iv) a client that provides the interface with the end user. Such a scheme is very flexible, as each of these components are individual processes that run in local or remote hosts. In addition, the schedulers also hide any possible blocking due to the code running in an engine, thus providing a useful framework for asynchronous computation. An in-depth review of IPYPARALLEL is out of the scope of this work and we refer the reader to its documentation. In GUNDAM, this parallelism is controlled by a single boolean flag that pushes data and code to execute in all available compute engines of a running IPYPARALLEL cluster.

There are two working versions of the code, one using highly optimized OpenMP code for single node parallel computations, and another one using the multiprocessing capabilities of IPYPARALLEL, more suitable for multinode computing. In this paper, we report the results for the OpenMP version only. The next version of GUNDAM will combine both approaches in a single package.

[3]OpenMP Standard: http://www.openmp.org
[4]github.com/ipython/ipyparallel

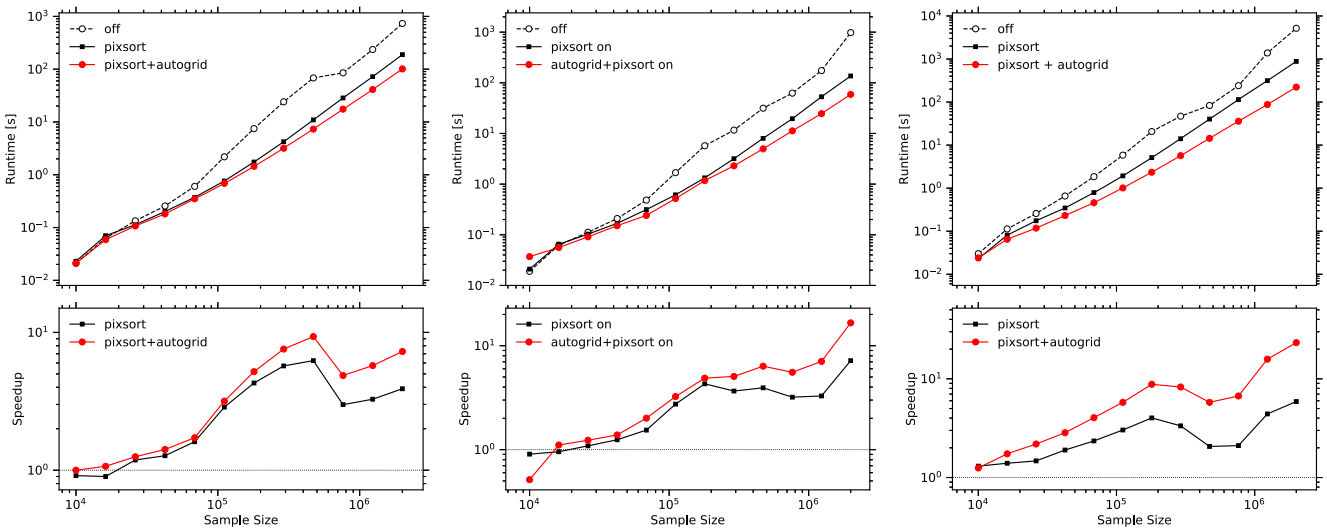

Figure 3. Single-core run time to calculate DD pairs with GUNDAM for samples up to 2×10^6 objects with redshifts between 0.01 and 0.2, extracted from a 60×60 deg^2 light cone constructed from the Millenium simulation and the SAGE semi-analytic model. Performance is evaluated in (r_p, π) space (left), redshift space (middle), and angular space (right). The base skip list algorithm (dashed) counts objects in a fixed $20 \times 100[\times 10]$ grid in DEC–RA–[Z] dimensions, which is compared to improvements by sorting objects within pixels (solid black), and by automatically optimizing the grid size for each sample using the option `autogrid = True` (solid red). Bottom panels show the corresponding speed-up factors over the base skip list algorithm.

2.8 Custom weighting

To account for instrumental and selection biases of astronomical surveys, it is common to weight each galaxy pair according to some prescription. This allows, for example, to correct for different sampling rates in certain regions of the sky, account for undersampled close pairs due to fibre collisions in multi-object spectrographs, correct for differences in the maximum observable volume of magnitude-limited surveys, etc.

Thus, our code allows to specify user-defined weights for each particle considered, so that for a given pair (i, j), the product of their weight $w_i * w_j$ is accumulated. Additive weights can be also considered just by altering a few lines of code in the appropriate routines.

GUNDAM employs this weighting scheme to correct SDSS data for collisions due to the fibre physical diameter, which does not allow to acquire two spectra closer than 55 arcsec within the same SDSS plate ($0.1\,h^{-1}$ Mpc at $z \sim 0.1$). This manifests as a decline in the clustering signal below the collision scale that must be corrected in order to study the clustering at small separations, though it can also affect larger scales (Zehavi et al. 2005). To calculate the weight for a pair of objects separated a distance r_p, the code uses the ratio $(1 + w_{\rm phot}(\theta))/(1 + w_{\rm spec}(\theta))$, where $w_{\rm spec}$ and $w_{\rm phot}$ are the pair counts in the spectroscopic and its photometric parent sample; and θ corresponds to r_p at the median redshift of the survey. This method works relatively well, but can be readily modified by the user to incorporate more sophisticated corrections (e.g. Guo, Zehavi & Zheng 2012).

2.9 Error estimates

Of crucial importance for many applications is to estimate suitable error bars for correlation functions. GUNDAM uses a bootstrap resampling methodology, where a large set of samples (typically 50–200) are randomly drawn with replacement from the data itself, and the statistics of interest, namely the 2PCF, is calculated for each one. Then, a dispersion measure such as the standard deviation of all these samples can be used to infer an error estimate.

The implementation of this technique is key to result in an acceptable performance. Repeatedly running the entire 2PCF procedure 50 or 100 times is clearly out of the question. Instead, the code generates a set of bootstrap resampling weights, i.e. for n data points an array of $(n, n_{\rm boot})$. This weight array is carried deep inside the counting loops where they contribute (or not if weight $= 0$) to another array that accumulates the pair counts at each separation for each bootstrap sample. In this way, bootstrap counts are considered just alongside data counts, saving the need to build multiple linked list and taking advantage of all the optimizations described before.

3 PERFORMANCE

To measure the performance of the code we employ two samples. First, a sample of 2×10^6 galaxies with DEC$\leq$64 deg constructed over the northern galactic cap of SDSS DR7 (Abazajian et al. 2009), covering an almost contiguous area of 7200 deg^2. Galaxies are placed randomly within this footprint and their redshifts assigned to match the redshift distribution of the SDSS main galaxy sample between $z = 0.01$ and $z = 0.3$, and a mean redshift $z \sim 0.1$. This sample covers a sufficiently large area to be representative of typical 2PCFs derived from wide surveys. Secondly, a 60×60 deg^2 light cone composed of 2×10^6 galaxies extracted from the Millenium simulation and the Semi-Analytic Galaxy Evolution Model (SAGE), with redshifts between 0.01 and 0.2. We only report results for the latter, as they are qualitatively the same for both data sets.

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 `march = native -ftree-vectorize`.

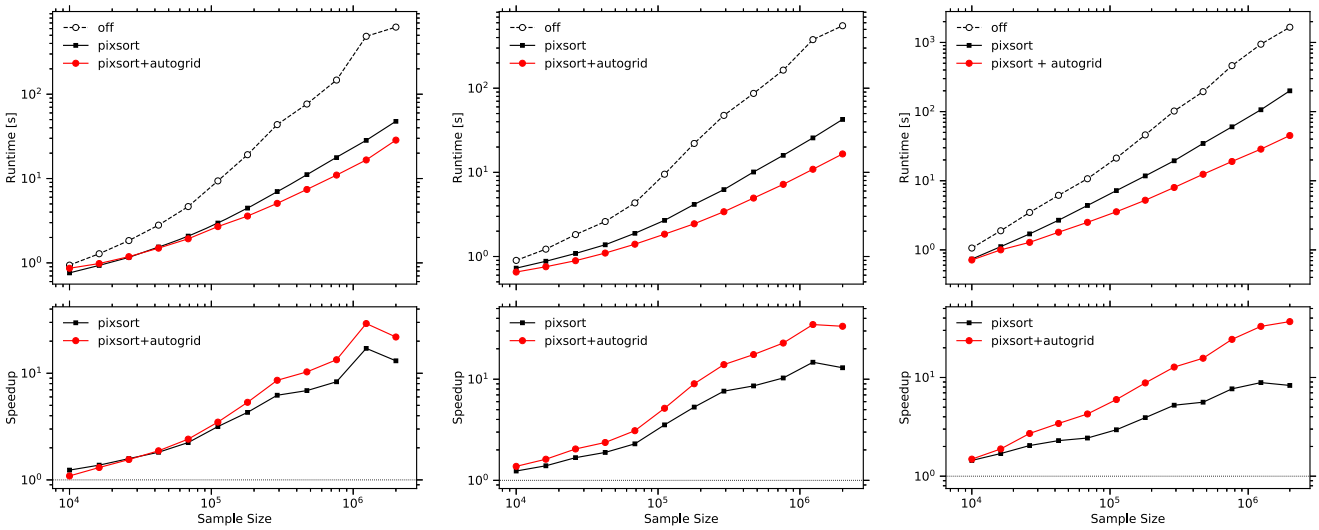

Figure 4. Single-core run time to calculate DR pairs between 10^5 data sources and samples up to 2×10^6 objects extracted from an SDSS DR7 random sample covering 7200 deg^2. Performance is evaluated in (r_p, π) space (left), redshift space (middle), and angular space (right). The base skip list algorithm counts objects in a fixed 16^3 or 16^2 grid (dashed), which is compared against improvements by sorting objects within pixels (solid black), and by automatically optimizing the grid size for each sample using the option `autogrid = True` (solid red). Bottom panels show the corresponding speed-up factors over the base skip list algorithm.

3.1 Single thread

Fig. 3 shows the performance of GUNDAM running under the described set-up in a single thread, to count an increasing number of galaxies in (r_p, π), redshift, and angular space. For each respective case, we requested counts in a matrix of 14×40 (r_p, π) bins with $0.1 < r_p < 25\ h^{-1}$ Mpc and $0 < \pi < 40\ h^{-1}$ Mpc; in 14 redshift-space bins between 0.1 and 25 h^{-1} Mpc; and in 12 angular bins between 0.01 and 5°. We also tested the case of requesting a single 40 h^{-1} Mpc π bin instead of 40 narrow ones, that is, when we are interested in the integrated counts DD(r_p) instead of DD(r_p, π). The impact in execution speed is negligible.

We show the performance of the base skip list–linked list algorithm (dashed line), compared against improvements by sorting pixels (black line) and using an option to fine-tune the grid size automatically (red line). It can be seen that for samples larger than 10^5, the benefit of rearranging data into sorted pixels to make better use of cache memory becomes more and more relevant. The speed-up factors reach 4, 7, and ∼6 for 10^6 particles in the three geometries implemented. If on top of that we let the code choose more wisely the number of cells in each dimension, we find another significant jump in performance. For samples with a large number of objects, there is a combined boost factor of 7 in the (r_p, π) case, 16 in redshift space, and 23 in the angular case. The absolute value of these gain factors depend indeed on the grid size chosen for comparison, in this case a $20 \times 100[\times 10]$ grid in DEC–RA–[radial] dimensions. For reference, when using automatic gridding the code splits our 2×10^6 particle sample into a $74 \times 696 \times 13$ grid, which makes much more sense than using only 20 divisions.

Fig. 4 shows a similar analysis, but while cross-counting pairs between 2×10^5 objects and our 2×10^6 test sample, a typical set-up for calculating the DR term of the Landy–Szalay estimator of the 2PCF. We observe similar, yet slightly higher (combined) boost factors of 23, 33, and 36, most likely due to: (1) a higher stress in cache utilization, i.e. more data points need to be available close to CPU so that intelligent cache usage is more relevant, and (2) minor implementation differences between cross-counting and normal pair counting.

In terms of absolute run time we can see that in about 100 s, GUNDAM can count pairs distant up to 25 and 40 h^{-1} Mpc in (r_p, π) space for about 2×10^6 galaxies. We compare these numbers against the performance of CORRFUNC (Sinha & Garrison, 2017; v2.0.0). This is possibly among the most efficient codes publicly available in the astronomical community. It accelerates 2PCF computation by maximizing the utilization of cache memory, and by exploiting the use of wide vector registers available through intrinsic AVX (Advanced Vector Extensions) instructions of some CPUs. AVX is a recent SIMD technology that allows to execute multiple floating-point operations in a single clock cycle.

Fig. 5 shows the corresponding timings for two cases: the normal case of counting pairs of i and j objects with unitary weights; and the weighted case, accumulating the product of the weights $w_i * w_j$, selected randomly (in this case) from a Gaussian distribution with $(\sigma, \mu) = (1.0, 0.1)$. This is a mock-up of a common scenario for deriving 2PCFs from real surveys, where numerous bias and observational effects are accounted by means of properly chosen weights. From an implementation point of view, this means dealing with extra weight vectors and at least an extra floating-point multiplication deeply nested inside loops. For this reason, GUNDAM uses slightly different versions of the counting algorithm to accommodate these two cases. From the figure it can be seen that our code, which is based in a completely different algorithm, performs as well and even better than CORRFUNC. In (r_p, π) space, it can be up to 1.4 × faster. In angular space GUNDAM can be slightly slower. However, if non-unitary weights are required, our code can run up to 30 per cent faster. The timings for very few particles, e.g. below few times 10^4, should be ignored as the dominant component there is most likely the overhead of grid creation or array manipulation. GUNDAM, as well as CORRFUNC, are intended primarily for processing much larger samples.

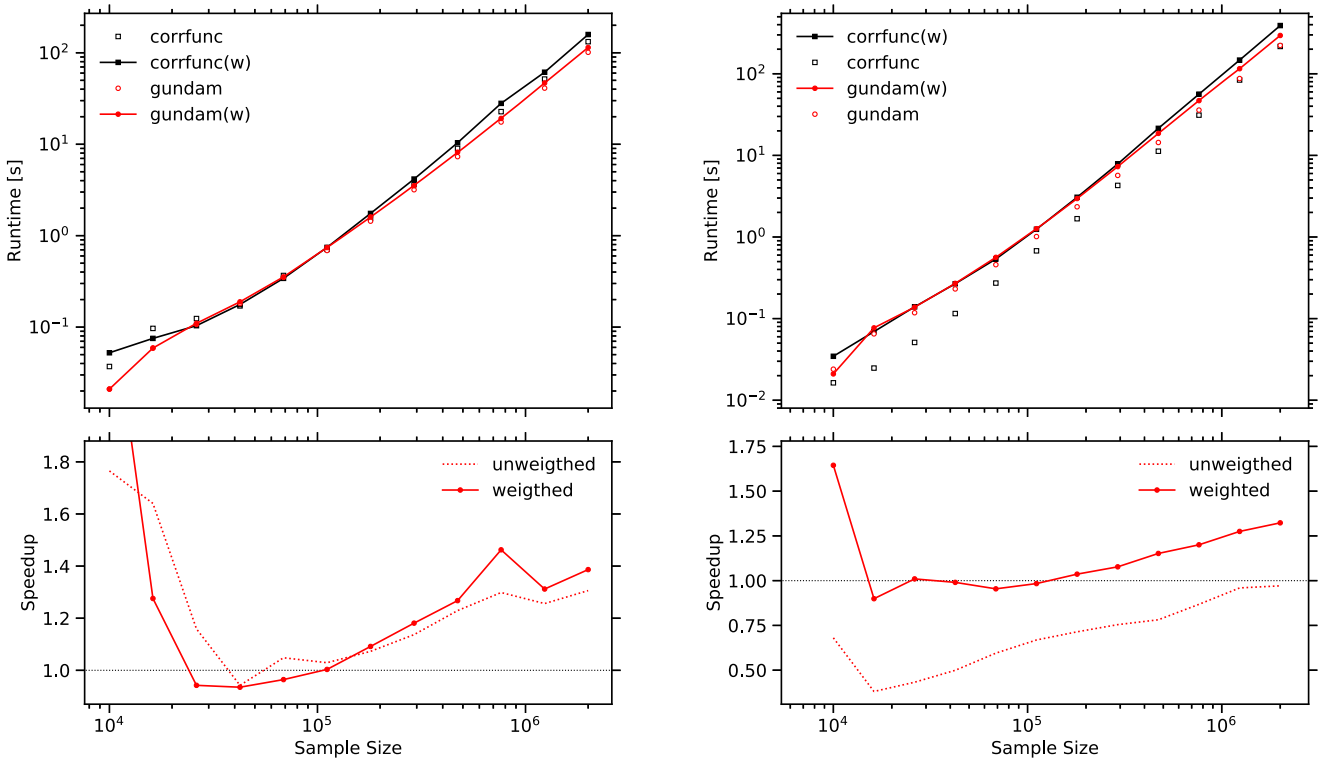

Figure 5. Single-core run time to calculate DD pairs with GUNDAM and CORRFUNC codes for samples up to 2×10^6 objects, using random Gaussian weights or a single constant weight equal to 1 (unweighted pair count case). Performance is evaluated in (r_p, π) space (left) and angular space (right). Bottom panels show the speed-up factor of GUNDAM respect to CORRFUNC for the weighted and unweighted case.

3.2 Multithread

We run a series of tests in the described set-up, running the same pair counts with 2, 3, and 4 threads that execute in each of the physical CPU cores. There is little benefit in using extra virtual threads (i.e. the Hyper-Threading technology of Intel CPUs) because the code is already computationally intensive, making extensive use of available CPU resources and floating-point pipelines. Also, since logical processors share the caches, they might actually compete for cache utilization, which can lead to a degradation of performance. Therefore, we only report up to four threads, noting that eight threads can potentially bump the performance by 25 per cent.

In Fig. 6, we can see that below 3–4 $\times$ 10^4 objects there is a bias towards slightly longer run times independent of the number of threads. As said before, this is due to the overhead of allocating memory and arranging data into convenient structures. In fact, a naive pair counting algorithm might probably run faster for very small samples. For larger samples, the overhead is less relevant and the advantage of using multiple threads in parallel starts to become significant. For around 10^6 particles in (r_p, π) space, we get a close to constant speed-up factor of 1.9, 2.8, and 3.6, when we duplicate, triplicate, or quadruplicate the number of threads. We observe a similar scaling in angular pair counts. For reference, we also show the scaling of CORRFUNC with four threads. These tests suggest that the multithreading efficiency of our implementation is particularly high.

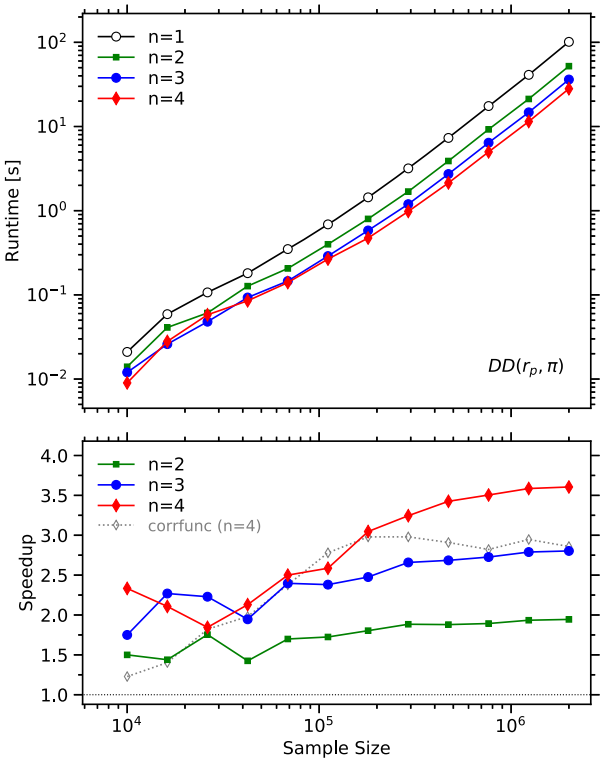

Figure 6. Comparison of GUNDAM parallel performance, by calculating for pair counts in (r_p, π) space with n threads in a single multicore CPU (max. one thread per physical core). The bottom panel show the speed-up factor over the single-thread case and the scaling of CORRFUNC, for reference.

3.3 Bootstrap errors

As described in Section 2.9, GUNDAM can automatically produce bootstrap error estimates. In Fig. 7, we show the cost of cal-

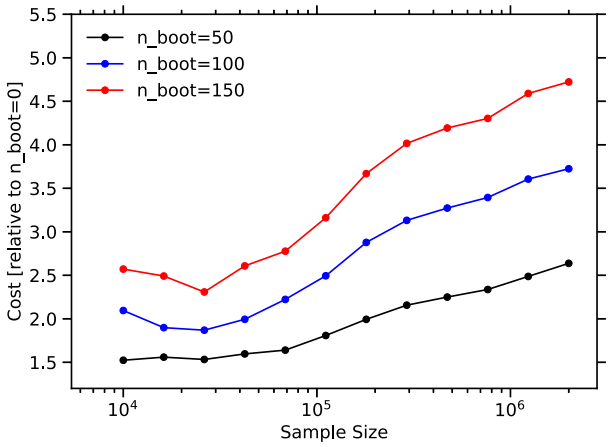

Figure 7. Incidence of estimating bootstrap errors (single core) in (r_p, π) pair counts, for samples of different size and for an increasing number of bootstrap samples drawn.

culating error bars for samples of different size and requesting `nboot` = [50,100,150] bootstrap random samples. The median cost is a factor of 1.9, 2.7, and 3.4 in each respective case, with a mild dependence on the size of the sample. Note, however, that these costs to derive error bars are quite small, considering we are basically performing the pair counts not once, but `nboot` times.

3.4 Use cases

To demonstrate the capabilities of our code, we show two examples of application with real astronomical data sets. These are representative of the typical tasks that can be accomplished with few lines of PYTHON code that make use of our toolkit.

For the first case, we chose a sample of 10^5 luminous red galaxies (LRG) selected from the SDSS DR7 by Kazin et al. (2010), along with 1.6×10^6 random objects. These are galaxies selected on the basis of colour and magnitude to match a population of passive, red, early-type galaxies, characteristic of highly biased environments such as the central regions of clusters and large groups. We use their DR7-Full sample at $z \sim 0.32$, with $-23.2 < M_g < -21.2$. Fig. 8 shows the 2D $\xi(r_p, \pi)$ in 0.5 h^{-1} Mpc bins, smoothed with a Gaussian kernel of width $w = 5$. At small scales, the distortion along the π direction due to random motions within central regions of clusters is clearly visible. At large scales, the Kaiser effect due to the coherent in-fall motions into virialized regions is also evident by the contour squashing. Note that to go from raw catalogue data to this plot the user only needs to call a couple lines of code.

For the second example, we used a combination of the SDSS main galaxy sample with $14.5 < m_r < 17.7$ and $0.02 < z < 0.3$, and the catalogue of physical properties derived by the MPA/JHU[5] team (Kauffmann et al. 2003; Brinchmann et al. 2004). As shown in Fig. 9, we trace the dependence of the clustering on stellar mass by estimating the projected correlation function $w(r_p)$, integrated 40 h^{-1} Mpc along π. We do this in five stellar mass bins from $\log(M/M_\odot) = 9.5$ up to $\log(M/M_\odot) = 12$. The well-known result of more massive galaxies populating denser environments is nicely reproduced here. At scales below 1 h^{-1} Mpc and for increasing stellar mass, we can clearly observe the emergence of the one-halo term due to galaxy pairs residing within the same dark matter halo.

[5] Available at http://www.mpa-garching.mpg.de/SDSS/DR7/

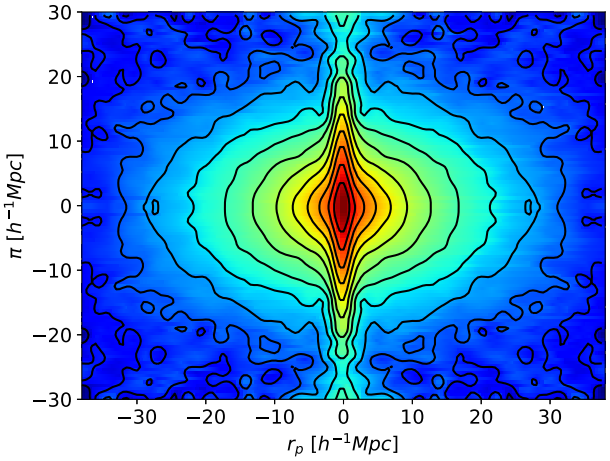

Figure 8. Estimate of the 2PCF in (r_p, π) space calculated by our code, corresponding to 10^5 luminous red galaxies selected from SDSS DR7 by Kazin et al. (2010). Redshift-space distortions, such as the Finger of God effect and the Kaiser effect are clearly visible. The figure was constructed out of the box, with just a couple lines of Python code using the functionality of the GUNDAM toolkit.

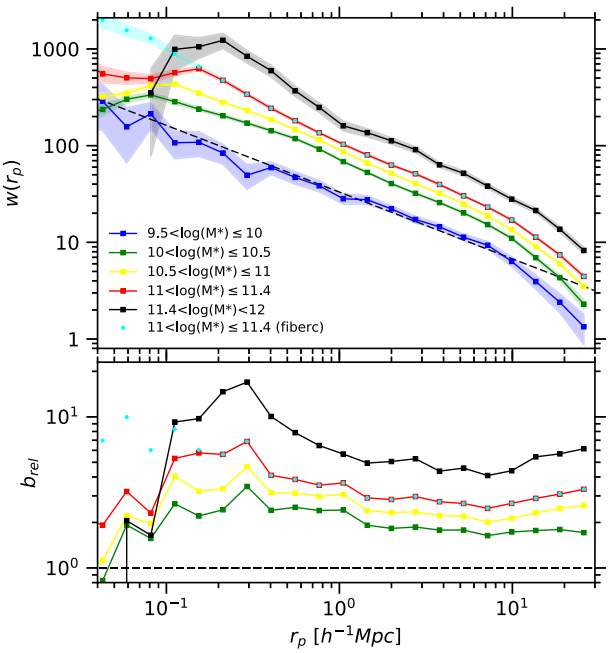

Figure 9. Projected 2PCF of SDSS DR7 main sample galaxies, as function of their stellar mass. As expected, more massive galaxies are found preferentially in denser environments and inhabit massive dark matter halos. The one-halo term due to pairs residing in within the same halo is clearly visible, as well the lack of pairs due to fibre positioning constrains at small separations. This can be corrected readily by setting the option `wbif = 1`. The bottom panel shows the relative bias respect the lowest mass sub-sample. The entire computation of this figure was produced in just about seven lines of code.

The effect of fibre collisions becomes apparent at scales below 0.1–0.2 h^{-1} Mpc, but can be easily corrected by turning on the option `wfib = 1` to up-weight close objects (cyan curve). In the bottom panel, we show the relative bias of each sub-sample respect to the lowest mass bin population, which is fitted here by a power law (dashed curve). While these results are well known, all it takes to construct this figure from the data is about seven lines of code.

4 SUMMARY AND PROSPECTS

In this work, we describe a new package to estimate two-point correlation functions, a fundamental statistical tool used in astronomy. Our code employs an efficient skip list/linked list algorithm to count pairs as fast as possible in projected, redshift, and angular coordinates. A special pixel sorting technique significantly improves the cache friendliness of linked lists that, together with various loop optimization techniques, allow for an increased performance on par and even better than the most advanced array-based algorithms. The code runs in parallel with multithreading or multiprocessing approach, therefore making available the expensive, lengthy computations that large astronomical surveys demand, to small research groups or individuals with limited resources.

The toolkit, implemented in a mixture of PYTHON and FORTRAN languages, is not only fast but easy to use and extend. This is important, as other sophisticated packages that rely on vectorization, such as CORRFUNC, would require to write or modify AVX kernels, a difficult task usually outside the skill set of astronomers or astrophysicists. Any of these counting algorithms could potentially be written using GPU programming, but again that requires significant investment in time and human resources.

A future enhancement is the joint implementation of multiprocessing (e.g. though IPYPARALLEL or MPI) to efficiently distribute tasks among nodes, plus multithreading (OpenMP) for optimal in-node performance. Another possible improvement is vectorization. Traditional array-based processing can benefit almost instantly from special CPU instructions that work in parallel on vectors rather than individual numbers. To vectorize a loop, a compiler first unrolls it by a given vector length, and then packs multiple scalar instructions into a single vector instruction. An efficient vectorization of at least part of GUNDAM algorithms, would be a challenge, but one with large potential gains. We are analysing possible modifications to achieve that goal.

ACKNOWLEDGEMENTS

We would like to thank the anonymous referee for useful suggestions and insight to improve this paper. This work was supported by the Consejo Nacional de Investigaciones Científicas y Técnicas de la República Argentina (CONICET). This research uses the SDSS Archive, funded by the Alfred P. Sloan Foundation, the Participating Institutions, the National Aeronautics and Space Administration, the National Science Foundation, the US Department of Energy, the Japanese Monbukagakusho, and the Max Planck Society.

REFERENCES

Abazajian K. N. et al., 2009, ApJS, 182, 543
Abell P. A. et al., 2009, LSST Science Book, arXiv:0912.0201
Alam S. et al., 2015, ApJS, 219, 12
Brinchmann J., Charlot S., White S. D. M., Tremonti C., Kauffmann G., Heckman T., Brinkmann J., 2004, MNRAS, 351, 1151
Davis M., Peebles P. J. E., 1983, ApJ, 267, 465
Gray A. G., Moore A. W., 2001, in Leen T. K., Dietterich T. G., Tresp V., eds, Advances in Neural Information Processing Systems (NIPS) 13 (Dec 2000). MIT Press, Cambridge, MA, United States
Guo H., Zehavi I., Zheng Z., 2012, ApJ, 756, 127
Hamilton A. J. S., 1993, ApJ, 417, 19
Ivezić Z. et al., 2008, SerAJ, 176, 1
Kauffmann G. et al., 2003, MNRAS, 341, 33
Kazin E. A. et al., 2010, ApJ, 710, 1444
Kerscher M., Szapudi I., Szalay A. S., 2000, ApJ, 535, L13
Landy S. D., Szalay A. S., 1993, ApJ, 412, 64
Omohundro S., 1991, in LippmannR. P., Moody J. E., Touretzky D. S., eds, Advances in Neural Information Processing Systems 3. Morgan Kaufmann, San Francisco, CA, United States
Peebles P. J. E., 1980, The large-scale structure of the universe. Princeton Univ. Press, Princeton, NJ
Peebles P. J. E., Hauser M. G., 1974, ApJS, 28, 19
Samet H., 2006, Foundations of Multidimensional and Metric Data Structures. Morgan Kaufmann, San Francisco, CA, United States
Shanks T., Bean A. J., Ellis R. S., Fong R., Efstathiou G., Peterson B. A., 1983, ApJ, 274, 529
Sinha M., Garrison, L. , 2017, Astrophysics Source Code Library, ascl.net/1703.003
Vargas-Magaña M. et al., 2013, A&A, 554, A131
Wright E. L. et al., 2010, AJ, 140, 1868
York D. G. et al., 2000, AJ, 120, 1579
Zehavi I. et al., 2005, ApJ, 630, 1

APPENDIX A: ROUTINES

Table A1 lists the main routines available in GUNDAM to calculate and visualize 2PCFs. There are additional auxiliary routines that provide useful extra functionality and the full documentation is available online.

APPENDIX B: INPUT/OUTPUT PARAMETERS

Table B1 lists an input parameter dictionary used by GUNDAM to calculate a projected correlation function, and Table B2 lists the keys in the output dictionary.

Table A1. Main routines implemented in GUNDAM.

Name	Description
pcf	Calculate the projected auto-correlation function given two input tables (D, R samples)
pccf	Calculate the projected cross-correlation function given three input tables (D, R, C samples)
rppi_A	Count pairs in projected-radial space (r_p - π) given an input table (D sample)
rppi_C	Count cross-pairs in projected-radial space (r_p - π) given two input tables (D, R samples)
rcf	Calculate the redshift-space autocorrelation function given two input tables (D, R samples)
rccf	Calculates the redshift-space cross-correlation function given three input tables (D, R, C samples)
s_A	Count pairs in redshift-space given an input table (D sample)
s_C	Count cross-pairs in redshift-space given two input tables (D, R samples)
acf	Calculate the angular auto-correlation function given two input tables (D, R samples)
accf	Calculate the angular cross-correlation function given two input tables (D, R, C samples)
th_A	Count pairs in angular-space given an input table (D sample)
th_C	Count cross-pairs in angular-space given two input tables (D, R samples)
plotcf	Plot a 2PCF by providing (x,y) arrays of points
cntplot	Plot a 2PCF direcly from a counts output dictionary (either read from disc or passed directly)
cntplot2D	Idem before but for a 2D $\xi(r_p, \pi)$, with optional Gaussian smoothing and contour levels
comparecf	Plot multiple 2PCFs and (optionally) and ratios of each respect to a control correlation
qprint	Quick nice printout of input/output dictionaries to the PYTHON console

Table B1. Example dictionary of input parameters for a projected correlation function.

Name	Description
`kind`	Kind of correlation function (pcf)
`description`	Short description of the run. Only informative
`file,file1`	File name of data and random sample. Only informative
`estimator`	Statistical estimator of the correlation function
`cra,cdec,cred,cwei,cdcom`	Column name in data sample table for coord., redshift, weights, and comov. distance
`cra1,cdec1,cred1,cwei1,cdcom1`	Column name in random sample table coord., redshift, weights, and comov. distance
`h0,omegam,omegal`	H_0[km s^{-1} Mpc^{-1}], Ω_λ and $\Omega_{\rm matter}$ cosmology parameters
`calcdist`	If `calcdist = False`, adopt comov. distances from input tables. Otherwise calculate them
`outfn`	Base name for all ouput files (e.g. /home/myuser/redagn)
`nsepp`	No. of projected separation bins
`seppmin`	Minimum projected distance to consider [Mpc h^{-1}]
`dsepp`	Size of projected bins (in dex if `logsepp = 1`)
`logsepp`	If `logsepp = 1` use log-spaced bins. Otherwise use linear-spaced bins
`nsepv`	No. of radial separation bins
`sepvmin`	Minimum radial distance to consider [Mpc h^{-1}]
`dsepv`	Size of radial bins [Mpc h^{-1}].
`autogrid`	If `True` guess the optimum nr. of cells `(mxh1,mxh2,mxh3)` of the skip table (SK)
`dens`	Custom nr. of particles per SK cell used when autogrid = True
`mxh1,mxh2,mxh3`	No. of DEC, RA, and comov. dist. cells of the SK table. Only relevant if `autogrid = False`
`doboot`	If `doboot = 1`, calculate bootstrap counts and error bars
`nbts`	No. of bootstrap samples. Only relevant if `doboot = 1`
`bseed`	Seed for boostrap random number generator
`wfib`	Apply SDSS fiber correction for pairs closer than 55 arcsec. See `wfiber` function

Table B2. Example dictionary of output keys for a projected correlation function.

Name	Description
`npt,npt1`	No. of points in data (D) and random sample (R), respectively
`rpl,rpm,rpr`	Left-, middle-, and right side of projected bins
`wrp,wrperr`	Projected correlation function and its error (integrated along all radial bins)
`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 projected and radial bins
`log`	Log record of PYTHON routines
`logfortran`	Log record of FORTRAN routines
`par`	Input parameter dictionary

This paper has been typeset from a TEX/LATEX file prepared by the author.

List of astronomical key words (Updated on 2017 March)

This list is common to *Monthly Notices of the Royal Astronomical Society, Astronomy and Astrophysics*, and *The Astrophysical Journal*. In order to ease the search, the key words are subdivided into broad categories. No more than *six* subcategories altogether should be listed for a paper.

The subcategories in boldface containing the word 'individual' are intended for use with specific astronomical objects; these should never be used alone, but always in combination with the most common names for the astronomical objects in question. Note that each object counts as one subcategory within the allowed limit of six.

The parts of the key words in italics are for reference only and should be omitted when the keywords are entered on the manuscript.

General
editorials, notices
errata, addenda
extraterrestrial intelligence
history and philosophy of astronomy
miscellaneous
obituaries, biographies
publications, bibliography
sociology of astronomy
standards

Physical data and processes
acceleration of particles
accretion, accretion discs
asteroseismology
astrobiology
astrochemistry
astroparticle physics
atomic data
atomic processes
black hole physics
chaos
conduction
convection
dense matter
diffusion
dynamo
elementary particles
equation of state
gravitation
gravitational lensing: micro
gravitational lensing: strong
gravitational lensing: weak
gravitational waves
hydrodynamics
instabilities
line: formation
line: identification
line: profiles
magnetic fields
magnetic reconnection
(magnetohydrodynamics) MHD
masers
molecular data
molecular processes
neutrinos
nuclear reactions, nucleosynthesis, abundances
opacity
plasmas
polarization
radiation: dynamics
radiation mechanisms:general
radiation mechanisms: non-thermal
radiation mechanisms: thermal
radiative transfer
relativistic processes
scattering
shock waves
solid state: refractory
solid state: volatile
turbulence
waves

Astronomical instrumentation, methods and techniques
atmospheric effects
balloons
instrumentation: adaptive optics
instrumentation: detectors
instrumentation: high angular resolution
instrumentation: interferometers
instrumentation: miscellaneous
instrumentation: photometers
instrumentation: polarimeters
instrumentation: spectrographs
light pollution
methods: analytical
methods: data analysis
methods: laboratory: atomic
methods: laboratory: molecular
methods: laboratory: solid state
methods: miscellaneous
methods: numerical
methods: observational
methods: statistical
site testing
space vehicles
space vehicles: instruments
techniques: high angular resolution
techniques: image processing
techniques: imaging spectroscopy
techniques: interferometric
techniques: miscellaneous
techniques: photometric
techniques: polarimetric
techniques: radar astronomy
techniques: radial velocities
techniques: spectroscopic
telescopes

Astronomical data bases
astronomical data bases: miscellaneous
atlases
catalogues
surveys
virtual observatory tools

Astrometry and celestial mechanics
astrometry
celestial mechanics
eclipses
ephemerides
occultations
parallaxes
proper motions
reference systems
time

The Sun
Sun: abundances
Sun: activity
Sun: atmosphere
Sun: chromosphere
Sun: corona
Sun: coronal mass ejections (CMEs)
Sun: evolution
Sun: faculae, plages
Sun: filaments, prominences
Sun: flares
Sun: fundamental parameters
Sun: general
Sun: granulation
Sun: helioseismology
Sun: heliosphere
Sun: infrared
Sun: interior
Sun: magnetic fields
Sun: oscillations
Sun: particle emission
Sun: photosphere
Sun: radio radiation
Sun: rotation
(Sun:) solar–terrestrial relations
(Sun:) solar wind
(Sun:) sunspots
Sun: transition region
Sun: UV radiation
Sun: X-rays, gamma-rays

Planetary systems
comets: general

comets: individual: . . .
Earth
interplanetary medium
Kuiper belt: general

Kuiper belt objects: individual: . . .
meteorites, meteors, meteoroids
minor planets, asteroids: general

minor planets, asteroids: individual: . . .
Moon
Oort Cloud
planets and satellites: atmospheres
planets and satellites: aurorae
planets and satellites: composition
planets and satellites: detection
planets and satellites: dynamical evolution and stability
planets and satellites: formation
planets and satellites: fundamental parameters
planets and satellites: gaseous planets
planets and satellites: general

planets and satellites: individual: . . .
planets and satellites: interiors
planets and satellites: magnetic fields
planets and satellites: oceans
planets and satellites: physical evolution
planets and satellites: rings
planets and satellites: surfaces
planets and satellites: tectonics
planets and satellites: terrestrial planets
planet–disc interactions
planet–star interactions
protoplanetary discs
zodiacal dust

Stars
stars: abundances
stars: activity
stars: AGB and post-AGB
stars: atmospheres
(stars:) binaries *(including multiple)*: close
(stars:) binaries: eclipsing
(stars:) binaries: general
(stars:) binaries: spectroscopic
(stars:) binaries: symbiotic
(stars:) binaries: visual
stars: black holes
(stars:) blue stragglers
(stars:) brown dwarfs
stars: carbon
stars: chemically peculiar
stars: chromospheres
(stars:) circumstellar matter
stars: coronae
stars: distances
stars: dwarf novae
stars: early-type
stars: emission-line, Be
stars: evolution
stars: flare
stars: formation
stars: fundamental parameters
(stars:) gamma-ray burst: general
(stars:) **gamma-ray burst: individual: . . .**
stars: general
(stars:) Hertzsprung–Russell and colour–magnitude diagrams
stars: horizontal branch
stars: imaging
stars: individual: . . .
stars: interiors

stars: jets
stars: kinematics and dynamics
stars: late-type
stars: low-mass
stars: luminosity function, mass function
stars: magnetars
stars: magnetic field
stars: massive
stars: mass-loss
stars: neutron
(stars:) novae, cataclysmic variables
stars: oscillations *(including pulsations)*
stars: peculiar *(except chemically peculiar)*
(stars:) planetary systems
stars: Population II
stars: Population III
stars: pre-main-sequence
stars: protostars
(stars:) pulsars: general
(stars:) **pulsars: individual: . . .**
stars: rotation
stars: solar-type
(stars:) starspots
stars: statistics
(stars:) subdwarfs
(stars:) supergiants
(stars:) supernovae: general
(stars:) **supernovae: individual: . . .**
stars: variables: Cepheids
stars: variables: Scuti
stars: variables: general
stars: variables: RR Lyrae
stars: variables: S Doradus
stars: variables: T Tauri, Herbig Ae/Be
(stars:) white dwarfs
stars: winds, outflows
stars: Wolf–Rayet

Interstellar medium (ISM), nebulae
ISM: abundances
ISM: atoms
ISM: bubbles
ISM: clouds
(ISM:) cosmic rays
(ISM:) dust, extinction
ISM: evolution
ISM: general
(ISM:) HII regions
(ISM:) Herbig–Haro objects

ISM: individual objects: . . .
(except planetary nebulae)
ISM: jets and outflows
ISM: kinematics and dynamics
ISM: lines and bands
ISM: magnetic fields
ISM: molecules
(ISM:) photodissociation region (PDR)
(ISM:) planetary nebulae: general
(ISM:) **planetary nebulae: individual: . . .**
ISM: structure
ISM: supernova remnants

The Galaxy
Galaxy: abundances
Galaxy: bulge
Galaxy: centre
Galaxy: disc
Galaxy: evolution
Galaxy: formation
Galaxy: fundamental parameters
Galaxy: general
(Galaxy:) globular clusters: general
(Galaxy:) **globular clusters: individual: . . .**
Galaxy: halo
Galaxy: kinematics and dynamics
(Galaxy:) local interstellar matter
Galaxy: nucleus
(Galaxy:) open clusters and associations: general
(Galaxy:) **open clusters and associations: individual: . . .**
(Galaxy:) solar neighbourhood
Galaxy: stellar content
Galaxy: structure

Galaxies
galaxies: abundances
galaxies: active
(galaxies:) BL Lacertae objects: general
(galaxies:) **BL Lacertae objects: individual: . . .**
galaxies: bulges
galaxies: clusters: general

galaxies: clusters: individual: . . .
galaxies: clusters: intracluster medium
galaxies: distances and redshifts
galaxies: dwarf
galaxies: elliptical and lenticular, cD
galaxies: evolution
galaxies: formation
galaxies: fundamental parameters
galaxies: general
galaxies: groups: general

galaxies: groups: individual: . . .
galaxies: haloes
galaxies: high-redshift

galaxies: individual: . . .
galaxies: interactions
(galaxies:) intergalactic medium
galaxies: irregular
galaxies: ISM
galaxies: jets
galaxies: kinematics and dynamics
(galaxies:) Local Group
galaxies: luminosity function, mass function
(galaxies:) Magellanic Clouds
galaxies: magnetic fields
galaxies: nuclei
galaxies: peculiar
galaxies: photometry
(galaxies:) quasars: absorption lines
(galaxies:) quasars: emission lines
(galaxies:) quasars: general

(galaxies:) **quasars: individual: . . .**
(galaxies:) quasars: supermassive black holes
galaxies: Seyfert
galaxies: spiral
galaxies: starburst
galaxies: star clusters: general

galaxies: star clusters: individual: . . .
galaxies: star formation
galaxies: statistics
galaxies: stellar content
galaxies: structure

Cosmology
(cosmology:) cosmic background radiation
(cosmology:) cosmological parameters
(cosmology:) dark ages, reionization, first stars
(cosmology:) dark energy
(cosmology:) dark matter
(cosmology:) diffuse radiation
(cosmology:) distance scale
(cosmology:) early Universe
(cosmology:) inflation
(cosmology:) large-scale structure of Universe
cosmology: miscellaneous
cosmology: observations
(cosmology:) primordial nucleosynthesis
cosmology: theory

Resolved and unresolved sources as a function of wavelength
gamma-rays: diffuse background
gamma-rays: galaxies
gamma-rays: galaxies: clusters
gamma-rays: general
gamma-rays: ISM
gamma-rays: stars
infrared: diffuse background
infrared: galaxies
infrared: general
infrared: ISM
infrared: planetary systems
infrared: stars
radio continuum: galaxies
radio continuum: general
radio continuum: ISM
radio continuum: planetary systems
radio continuum: stars
radio continuum: transients
radio lines: galaxies
radio lines: general
radio lines: ISM
radio lines: planetary systems
radio lines: stars
submillimetre: diffuse background
submillimetre: galaxies
submillimetre: general
submillimetre: ISM
submillimetre: planetary systems
submillimetre: stars
ultraviolet: galaxies
ultraviolet: general
ultraviolet: ISM
ultraviolet: planetary systems
ultraviolet: stars
X-rays: binaries
X-rays: bursts
X-rays: diffuse background
X-rays: galaxies
X-rays: galaxies: clusters
X-rays: general
X-rays: individual: . . .
X-rays: ISM
X-rays: stars