Monthly Notices
of the
ROYAL ASTRONOMICAL SOCIETY

MNRAS **498,** 2095–2113 (2020)
Advance Access publication 2020 July 28

doi:10.1093/mnras/staa2201

SCAMPY – A sub-halo clustering and abundance matching based PYTHON interface for painting galaxies on the dark matter halo/sub-halo hierarchy

Tommaso Ronconi,[1,2,3]★ Andrea Lapi,[1,2,3,4] Matteo Viel[1,2,3,4] and Alberto Sartori[1]

[1]*SISSA, Via Bonomea 265, I-34136 Trieste, Italy*
[2]*IFPU, Via Beirut 2, I-34014 Trieste, Italy*
[3]*INFN-Sezione di Trieste, via Valerio 2, I-34127 Trieste, Italy*
[4]*INAF-OATS, via Tiepolo 11, I-34131 Trieste, Italy*

Accepted 2020 July 21. Received 2020 July 21; in original form 2020 February 17

ABSTRACT

We present a computational framework for 'painting' galaxies on top of the dark matter halo/sub-halo hierarchy obtained from *N*-body simulations. The method we use is based on the sub-halo clustering and abundance matching (SCAM) scheme which requires observations of the 1- and 2-point statistics of the target (observed) population we want to reproduce. This method is particularly tailored for high redshift studies and thereby relies on the observed high-redshift galaxy luminosity functions and correlation properties. The core functionalities are written in C++ and exploit Object Oriented Programming, with a wide use of polymorphism, to achieve flexibility and high computational efficiency. In order to have an easily accessible interface, all the libraries are wrapped in PYTHON and provided with an extensive documentation. We validate our results and provide a simple and quantitative application to reionization, with an investigation of physical quantities related to the galaxy population, ionization fraction, and bubble size distribution. The library is publicly available at https://github.com/TommasoRonconi/scampy with full documentation and examples at https://scampy.readthedocs.io.

Key words: methods: numerical – cosmology: theory – dark ages, reionization, first stars – large-scale structure of Universe.

1 INTRODUCTION

Cosmological *N*-body simulations are a fundamental tool for assessing the non-linear evolution of the large-scale structure (LSS). With the increasing power of computational facilities, cosmological *N*-body simulations have grown in size and resolution, allowing to study extensively the formation and evolution of dark matter (DM) haloes (Springel et al. 2005; Boylan-Kolchin et al. 2009; Klypin, Trujillo-Gomez & Primack 2011; Angulo et al. 2012; Klypin et al. 2016). Our confidence on the reliability of these simulations stands on the argument that the evolution of the non-collisional matter component only depends on the effect of gravity and on the initial conditions. While for the first, we can rely on a solid theoretical background, with analytical solutions for both the classical gravitation theory and for a wide range of its modifications, for the latter, we have measurements at high accuracy (Planck Collaboration VI 2018) of the primordial power spectrum of density fluctuations.

The formation and evolution of the luminous component (i.e. galaxies and intergalactic baryonic matter) are far from being understood at the same level as the DM. Several possible approaches have been attempted so far to asses this modelling issue, which can be divided into two main categories. On one side, ab initio models, such as *N*-body simulations with full hydrodynamical treatment and semi-analytical models, that should incorporate all the relevant astrophysical processes, are capable of tracing back the evolution in time of galaxies within their DM host haloes (see Somerville & Davé 2015; Naab & Ostriker 2017, for reviews).

On the other side, *empirical* (or *phenomenological*) models are designed to reproduce observable properties of a target (observed) population of objects at a given moment of their evolution (see e.g. Wechsler & Tinker 2018, for a review). This latter class of methods is typically cheaper in terms of computational power and time required for running. The development of an approach to model the luminous component without prior assumptions on the baryon physics have emerged from the advent of large galaxy surveys (York et al. 2000; Colless et al. 2001; Lilly et al. 2007; Driver et al. 2011; Grogin et al. 2011; McCracken et al. 2012).

Building an empirical model of galaxy occupation requires to define the hosted-object/hosting-halo connection for associating to the underlying DM distribution its baryonic counterpart. This has been achieved by exploiting several approaches that span from the classical mass-based methods, such as the halo occupation distribution (HOD) scheme (Peacock & Smith 2000; Seljak 2000; White 2001; Berlind & Weinberg 2002; Yang, Mo & van den Bosch 2003; Zehavi et al. 2004; Tinker et al. 2005; Zheng et al. 2005; Brown et al. 2008; Leauthaud et al. 2012) or the sub-halo abundance matching (SHAM) scheme (Mo & White 1996; Wechsler et al. 1998; Vale & Ostriker 2004; Conroy, Wechsler & Kravtsov 2006; Wang et al. 2006, 2007; Behroozi, Conroy & Wechsler 2010; Guo et al. 2010; Moster et al. 2010; Trujillo-Gomez et al. 2011), to more sophisticated parametrizations that follow the halo evolution in time (Conroy & Wechsler 2009; Yang et al. 2012; Behroozi, Wechsler & Conroy 2013b; Moster, Naab & White 2013, 2018; Zhu, Avestruz &

★ E-mail: tronconi@sissa.it

Gnedin 2020) also allowing for adaptive complexity (Behroozi et al. 2019). At the same time the number of observables that can be generated with such methods increased including galaxy luminosity (e.g. Rodríguez-Puebla et al. 2017; Moster et al. 2018; Somerville et al. 2018), gas (e.g. Popping, Behroozi & Peeples 2015), metallicity (e.g. Rodríguez-Puebla et al. 2016), and dust (e.g. Imara et al. 2018).

Given their capability to target galaxy formation without biasing the model with baryon physics uncertainties, empirical models complement and help to constrain ab initio models. The power of empirical approaches comes from the possibility to infer the DM density field from observations of the biased luminous component (Monaco & Efstathiou 1999; Jasche & Lavaux 2019; Kitaura et al. 2019). The mock catalogues obtained can be used to build precise co-variance matrices in preparation for assessing the uncertainties on cosmological parameters estimates, that will be inferred from next generation LSS observational campaigns, such as DESI (Levi et al. 2013) and *Euclid* (Amendola et al. 2018). Via the usage of empirical models, it is possible to considerably speed up the construction of mock catalogues and are the natural framework for forward modelling of the LSS observable properties (see e.g. Nuza et al. 2014; Leclercq, Jasche & Wandelt 2015; Kitaura et al. 2019). Furthermore, where ab initio models have struggled to obtain tight parameter constraints (e.g. on the mechanism for galaxy quenching), empirical models are capable of revealing possibly new unexpected physics (see e.g. Behroozi, Wechsler & Conroy 2012; Behroozi & Silk 2015).

Ab initio approaches are tuned to reproduce the LSS of the Universe at the present time, and therefore their reliability in the high redshift regime has to be proven. On the other hand, empirical models are by design particularly suitable for addressing the modelling of the high redshift Universe, but they rely on the availability of high redshift observations of the population to be modelled.

Our motivation for the original development of the Application Programming Interface (API) we present in this work is to study a particular window in the high redshift Universe. Specifically, our aim is twofold: (i) provide a physically robust and efficient way of modelling galaxy populations in the high redshift Universe from a DM-only *N*-body simulation; (ii) test applications, such as the modelling of the distribution of the first sources that started to shed light on the neutral medium, triggering the process called *reionization*. We expect that this tool could have further applications, especially in the context of cross-correlation of different tracers and/or diffuse backgrounds.

SCAMPY provides a PYTHON interface that uses the sub-halo clustering and abundance matching (SCAM) prescription for 'painting' galaxies on top of DM-only simulations. The SCAM algorithm is an extension of the classical HOD for defining the galaxy–halo connection. This class of methods is widely used in the scientific community but specialized software exists only within larger software packages (e.g. the Halotools package from Hearin et al. 2017, which is part of the Astropy library collection). Our intent is to provide the user with a light and versatile interface able to provide performances and extensibility with as little dependence to external software as possible.

We have carefully designed the software to exploit the best features of PYTHON and C++ language. Our intent was not only to achieve high performances of our code but also to make it more accessible, to ease cross-platform installation, and to generally set-up a flexible tool. Since the API we present has been designed to be easily extensible, in the future we will also be able to evolve our current research towards novel directions. Furthermore, this effort would hopefully also encourage new users to adopt our tool. As much as experiments are accurately designed to have the longest life-span possible, we have taken care of designing our software for a long term use.

The API relies on an optimized C++ core implementation of the most computationally expensive sections of the algorithm. This allows, on the one hand, to exploit the performances of a compiled language. On the other hand, it overcomes the limit on the usage of multithreading for shared memory parallelization, as otherwise imposed by the PYTHON standard library.

SCAMPY embeds two main functionalities: on the one side, it is designed for handling and building mock-galaxy catalogues, based on a user-defined parametrization. On the other side, it provides an extremely efficient implementation of the halo-model, which is used to infer the parameters required by the SCAM algorithm.

We provide a framework for loading a DM halo/sub-halo hierarchy, where the haloes are obtained by means of a friends-of-friends algorithm run on top of cosmological *N*-body simulations, while the sub-structures are identified using the SUBFIND algorithm (Springel et al. 2001). None the less, thanks to its extensible design, adapting the API for working with simulations obtained by means of approximated methods, such as COLA (Tassev, Zaldarriaga & Eisenstein 2013) or PINOCCHIO (Monaco, Theuns & Taffoni 2002), would be straightforward.

Once the SCAMPY parameters, which regulate the occupation of structures, have been set, we can easily produce the output mock-catalogue by calling the dedicated functions from the same framework we used for loading the DM halo/sub-halo hierarchy.

This work is organized as follows. In Section 2, we describe the SCAM technique. We describe the main components and algorithms that implement the aforementioned scheme inside our API in Section 3. In Section 4, we show the results of the several tests we have performed in order to validate the functionalities of our API. We have tested our instrument in a proof-of-concept application of the target problem: in Section 5, we study the effect of individual sources injecting ionizing photons in the neutral intergalactic medium at high redshift. Finally, in Section 6, we provide a summary of this work and anticipate the developments we are planning to pursue.

2 SUB-HALO CLUSTERING AND ABUNDANCE MATCHING

Our approach for the definition of the hosted-object/hosting-halo connection is based on the SCAM technique (Guo et al. 2016). With the standard HOD approach, hosted objects are associated with each halo employing a prescription which is based on the total halo mass, or on some other mass proxy (e.g. halo peak velocity and velocity dispersion). On the other side, the SHAM assumes a monotonic relation between some observed object property (e.g. luminosity or stellar mass of a galaxy) and a given halo property (e.g. halo mass). While the first approach is capable of, and extensively used for, reproducing the spatial distribution properties of some target population, the second is the standard for providing plain DM haloes and sub-haloes with observational properties that would otherwise require a full-hydrodynamical treatment of the simulation from which these are extracted.

The SCAM prescription aims to combine both approaches, providing a parametrized model to fit both some observable abundance and the clustering properties of the target population. The approach is nothing more than applying HOD and SHAM in sequence:

(i) the occupation functions for central, $N_{\rm cen}(M_{\rm h})$, and satellite galaxies, $N_{\rm sat}(M_{\rm h})$, depend on a set of defining parameters which can vary in number depending on the shape used. These functions

depend on a proxy of the total mass of the host halo. We sample the space of the defining parameters using a Markov-chain Monte Carlo (MCMC) to maximize a likelihood built as the sum of the χ^2 of the two measures we want to fit, namely the two-point angular correlation function at a given redshift, $\omega(\theta, z)$, and the average number of sources at a given redshift, $n_g(z)$:

$$\log \mathcal{L} \equiv -\frac{1}{2}\left(\chi^2_{\omega(\theta,z)} + \chi^2_{n_g(z)}\right), \tag{1}$$

The analytic form of both $\omega(\theta, z)$ and $n_g(z)$, depending on the same occupation functions $N_{cen}(M_h)$ and $N_{sat}(M_h)$, can be obtained with the standard halo model (see Cooray & Sheth 2002, for a review), which we describe in detail in Section 2.1. How these occupation functions are used to select which sub-haloes will host our mock objects is reported in Section 3.1.1.

(ii) Once we get the host halo/sub-halo hierarchy with the abundance and clustering properties we want, as guaranteed by equation (1), we can apply our SHAM algorithm to link each mass (or, equivalently, mass-proxy) bin with the corresponding luminosity (or observable property) bin.

When these two steps have been performed, the mock-catalogue is built.

2.1 The halo model

The modern formulation of the halo-model theory (see Cooray & Sheth 2002, for a review) provides a halo-based description of non-linear gravitational clustering that is widely used in literature to infer the underlying DM statistical properties at both low and high redshift. The key assumption of this model is that the number of galaxies, N_g, in a given DM halo only depends on the halo mass, M_h. Specifically, if we assume that $N_g(M_h)$ follows a Poisson distribution with mean proportional to the mass of the halo M_h, we can write

$$\langle N_g \rangle (M_h) \propto M_h \tag{2}$$

$$\langle N_g (N_g - 1) \rangle (M_h) \propto M_h^2 \tag{3}$$

From these assumptions it is possible to derive correlations of any order as a sum of the contributions of each possible combination of objects identified in single or in multiple haloes. To get the models required by equation (1) we only need the 1-point and the 2-point statistics. We derive the first as the mean abundance of objects at a given redshift. The average mass density in haloes at redshift z is given by

$$\overline{\rho}(z) = \int M_h \; n(M_h, z) \; \mathrm{d}M_h \tag{4}$$

where $n(M_h, z)$ is the halo mass function. With equation (2) we can then define the average number of objects at redshift z, hosted in haloes with mass $M_{min} \leq M_h \leq M_{max}$, as

$$n_g(z) \equiv \int_{M_{min}}^{M_{max}} \langle N_g \rangle (M_h) \; n(M_h, z) \; \mathrm{d}M_h \; . \tag{5}$$

Deriving the 2-point correlation function, $\xi(r, z)$, would require to treat with convolutions, we therefore prefer to obtain it by inverse Fourier-transforming the non-linear power spectrum $P(k, z)$, who's derivation can instead be treated with simple multiplications:

$$\xi(r, z) = \frac{1}{2\pi^2} \int_{k_{min}}^{k_{max}} \mathrm{d}k k^2 P(k, z) \frac{\sin(kr)}{kr} \; . \tag{6}$$

$P(k, z)$ can be expressed as the sum of the contribution of two terms:

$$P(k, z) = P_{1h}(k, z) + P_{2h}(k, z) \, , \tag{7}$$

where the first, dubbed *1-halo term*, results from the correlation among objects belonging to the same halo, while the second, dubbed *2-halo term*, gives the correlation between objects belonging to two different haloes.

The 1-halo term in real space is the convolution of two similar profiles of shape

$$\widetilde{u}(k, z|M_h) = \frac{4\pi\rho_s r_s^3}{M_h} \left\{ \sin(kr_s)\left[\mathrm{Si}((1+c)kr_s) - \mathrm{Si}(kr_s)\right] - \frac{\sin(ckr_s)}{(1+c)kr_s} - \cos(kr_s)\left[\mathrm{Ci}((1+c)kr_s) - \mathrm{Ci}(kr_s)\right] \right\} , \tag{8}$$

where c is the halo concentration, ρ_s and r_s are, respectively, the scale density and radius of the NFW profile and the sine and cosine integrals are defined as

$$\mathrm{Ci}(x) = \int_t^\infty \frac{\cos t}{t} \, \mathrm{d}t \text{ and } \mathrm{Si}(x) = \int_0^x \frac{\sin t}{t} \, \mathrm{d}t \; . \tag{9}$$

Equation (8) provides the Fourier transform of the DM distribution within a halo of mass M_h at redshift z. Weighting this profile by the total number density of pairs, $n(M_h)(M_h/\overline{\rho})^2$, contributed by haloes of mass M_h, leads to the expression for the 1-halo term:

$$\begin{aligned} P_{1h}(k, z) &\equiv \int n(M_h, z) \left(\frac{M_h}{\overline{\rho}}\right)^2 \left|\widetilde{u}(k, z|M_h)\right|^2 \mathrm{d}M_h \\ &= \frac{1}{n_g^2(z)} \int_{M_{min}}^{M_{max}} \langle N_g (N_g - 1) \rangle (M_h) \; n(M_h, z) \; \left|\widetilde{u}(k, z|M_h)\right|^2 \mathrm{d}M_h, \end{aligned} \tag{10}$$

with $n(M_h, z)$ the halo mass function for host haloes of mass M_h at redshift z and where, in the second equivalence, we used equations (3) and (5) to substitute the ratio $\left(M_h/\overline{\rho}\right)^2$.

The derivation of the 2-halo term is more complex and for a complete discussion the reader should refer to Cooray & Sheth (2002). Let us just say that, for most of the applications, it is enough to express the power spectrum in its linear form. Corrections to this approximation are mostly affecting the small-scales which are almost entirely dominated by the 1-halo component. This is mostly because the 2-halo term depends on the biasing factor which on large scales is deterministic. We therefore have that, in real-space, the power coming from correlations between objects belonging to two separate haloes is expressed as the product between the convolution of two terms and the biased linear correlation function (i.e. $b'_h(M'_h) b_h(M''_h) \xi_{lin}(r, z)$). The two terms in the convolution provide the product between the Fourier-space density profile $\widetilde{u}(k, z|M_h)$, weighted by the total number density of objects within that particular halo (i.e. $n(M_h)(M_h/\overline{\rho})$). In Fourier space, we therefore have

$$\begin{aligned} P_{2h}(k, z) &\equiv \int n(M'_h) \, \frac{M'_h}{\overline{\rho}} \, \widetilde{u}(k, z|M'_h) \, b(M'_h) \; \mathrm{d}M'_h \\ &\int n(M''_h) \, \frac{M''_h}{\overline{\rho}} \, \widetilde{u}(k, z|M''_h) \, b(M''_h) \, P_{lin}(k, z) \; \mathrm{d}M''_h = \\ &= \frac{P_{lin}(k, z)}{n_g^2(z)} \left[\int_{M_{min}}^{M_{max}} \langle N_g \rangle (M_h) \; n(M_h) \; b(M_h, z) \; \widetilde{u}_h(k, z|M_h) \; \mathrm{d}M_h\right]^2 \end{aligned} \tag{11}$$

with $b(M_h)$ the halo bias and $P_{lin}(k, z)$ the linear matter power spectrum evolved up to redshift z. For going from the first to the second equivalence we have to make two assumptions. First we assume self-similarity between haloes. This means that the two nested integrals in $\mathrm{d}M'_h$ and $\mathrm{d}M''_h$ are equivalent to the square of

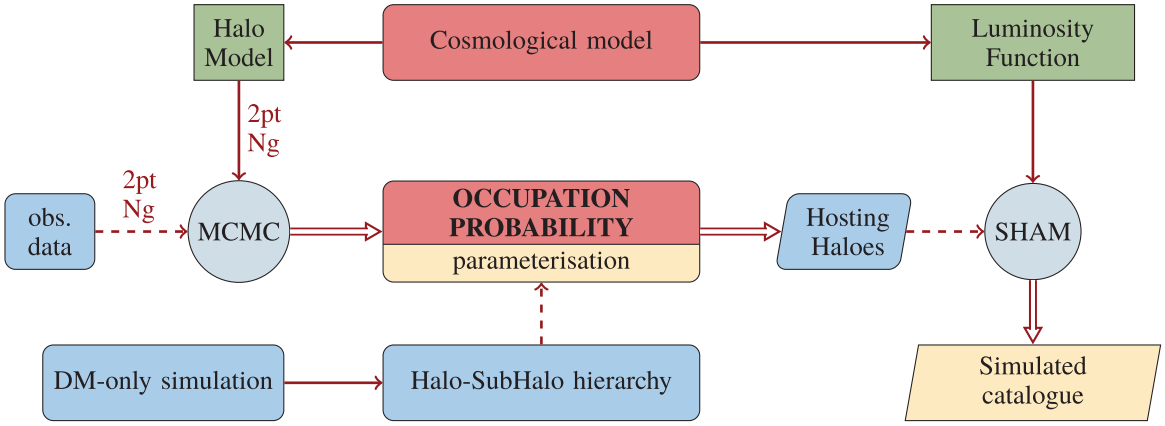

Figure 1. Flowchart describing the main components of the algorithm. In red the two main kernel modules. Green rectangles dub models from which the workflow depends. Round grey circles are for engines that operate on some inputs. Cyan is for inputs, yellow and parallelograms for outputs.

the integral in the rightmost expression. Secondly, we make use of equations (2) and (5) to substitute the ratio $M_h/\overline{\rho}$.

The average number of galaxies within a single halo can be decomposed into the sum

$$\langle N_g \rangle(M_h) \equiv N_{cen}(M_h) + N_{sat}(M_h) \tag{12}$$

where $N_{cen}(M_h)$ is the probability to have a central galaxy in a halo of mass M_h, while $N_{sat}(M_h)$ is the average number of satellite galaxies per halo of mass M_h. These two quantities are precisely the occupation functions we already mentioned in Section 2. Given that no physics motivated functional form exists for $N_{cen}(M_h)$ and $N_{sat}(M_h)$, usually, they are parametrized. By tuning this parametrization we obtain the prescription for defining the hosted-object/hosting-halo connection.

With the decomposition of equation (12), we can approximate equation (3) to

$$\begin{aligned}\langle N_g(N_g-1)\rangle(M_h) &\approx \langle N_{cen}N_{sat}\rangle(M_h) + 2\,\langle N_{sat}(N_{sat}-1)\rangle(M_h) \\ &\approx N_{cen}(M_h)\,N_{sat}(M_h) + N^2_{sat}(M_h)\end{aligned} \tag{13}$$

Thus we can further decompose the 1-halo term of the power spectrum as the combination of power given by *central-satellite* couples (cs) and *satellite-satellite* couples (ss):

$$P_{1h}(k,z) \approx P_{cs}(k,z) + P_{ss}(k,z)\,, \tag{14}$$

When dealing with observations, it is often more useful to derive an expression for the projected correlation function, $\omega(r_p, z)$, where r_p is the projected distance between two objects, assuming flat sky. From the Limber approximation (Limber 1953), we have

$$\begin{aligned}\omega(r_p,z) &= \mathcal{A}\big[\xi(r,z)\big] = \mathcal{A}\big\{\mathcal{F}\big[P(k,z)\big]\big\} = \mathcal{H}_0\big[P(k,z)\big] \\ &= \frac{1}{2\pi}\int k\;P(k,z)\;J_0(r_p k)\;\mathrm{d}k,\end{aligned} \tag{15}$$

where $J_0(x)$ is the 0th-order Bessel function of the first kind. Reading the expression above from left to right, we can get the projected correlation function by Abel-projecting the 3D correlation function $\xi(r, z)$. From the definition in equation (6), $\omega(r_p, z)$ is therefore obtained by Abel-transforming the Fourier transform of the power spectrum. This is equivalent to perform a zeroth-order Hankel transform of the power spectrum, which leads to the last equivalance in equation (15).

Equation (15) though, is valid as long as we are able to measure distances directly in an infinitesimal redshift bin, which is not realistic. Our projected distance depends on the angular separation, θ, and the cosmological distance, $d_C(z)$, of the observed object

$$r_p(\theta, z) = \theta \cdot d_C(z)\,. \tag{16}$$

By projecting the objects in our light cone on a flat surface at the target redshift, we are summing up the contribution of all the objects along the line of sight. Therefore, the two-point angular correlation function can be expressed as

$$\omega(\theta, z) = \int \frac{\mathrm{d}V(z)}{\mathrm{d}z}\mathcal{N}^2(z)\;\omega[r_p(\theta,z),z]\,\mathrm{d}z \tag{17}$$

where $\frac{\mathrm{d}V(z)}{\mathrm{d}z}$ is the comoving volume unit and $\mathcal{N}(z)$ is the normalized redshift distribution of the target population. If we assume that $\omega(\theta, z)$ is approximately constant in the redshift interval $[z_1, z_2]$, we can then write

$$\omega(\theta, z) \approx \left[\int_{z_1}^{z_2} \mathrm{d}z\,\frac{\mathrm{d}V(z)}{\mathrm{d}z}\mathcal{N}^2(z)\right]\cdot\omega[r_p(\theta),\overline{z})] \tag{18}$$

where $\overline{z}$ is the mean redshift of the objects in the interval.

3 THE SCAMPY LIBRARY

In this section, we introduce SCAMPY, our highly optimized and flexible API for 'painting' an observed population on top of the DM-halo/sub-halo hierarchy obtained from DM-only N-body simulations. We will give here a general overview of the algorithm on which our API is based. We refer the reader to Appendix A for a description of the key aspects of our hybrid C++/PYTHON implementation, where we point out how the package is intended for future expansion and further optimization. We also provide, at the end of this section, a brief discussion of the API performances. For a detailed analysis, the reader should refer to Appendix B. The source code and a guide for installing the library can be obtained by cloning the GitHub repository of the project (github.com/TommasoRonconi/scampy). The full documentation of SCAMPY, with a set of examples and tutorials, is available on the website (scampy.readthedocs.io) of the package.

3.1 Algorithm overview

In Fig. 1, we give a schematic view of the main components of the SCAMPY package. All the framework is centred around the occupation probabilities, namely $N_{cen}(M_h)$ and $N_{sat}(M_h)$, which define the average numbers of, respectively, central and satellite galaxies hosted

within each halo. Several parametrizations of these two functional forms exist. One of the most widely used is the standard 5-parameters HOD model (Zheng, Coil & Zehavi 2007; Zheng et al. 2009), with the probability of having a central galaxy given by an activation function and the number distribution of satellite galaxies given by a power law

$$N_{\rm cen}(M_{\rm h}) = \frac{1}{2}\left[1 + {\rm erf}\left(\frac{\log M - \log M_{\rm min}}{\sigma_{\log M_{\rm h}}}\right)\right] \tag{19}$$

$$N_{\rm sat}(M_{\rm h}) = \left(\frac{M_{\rm h} - M_{\rm cut}}{M_1}\right)^{\alpha_{\rm sat}}, \tag{20}$$

where $M_{\rm min}$ is the characteristic minimum mass of haloes that host central galaxies, $\sigma_{\log M_{\rm h}}$ is the width of this transition, $M_{\rm cut}$ is the characteristic cut-off scale for hosting satellites, M_1 is a normalization factor, and $\alpha_{\rm sat}$ is the power-law slope. Our API provides users with both an implementation of the equations (19) and (20), and the possibility to use their own parametrization by inheriting from a base `occupation_p` class.[1] Given that both the modelling of the observable statistics (Section 2.1) and the HOD method used for populating DM haloes depend on these functions, we implemented an object that can be shared by both these sections of the API. As outlined in Fig. 1, the parameters of the occupation probabilities can be tuned by running an MCMC sampling. By using a likelihood as the one exposed in Section 2, the halo-model parametrization that best fits the observed 1- and 2-point statistics of a target population can be inferred.[2]

The chosen cosmological model acts on top of our working pipeline. Besides providing the user with a set of cosmographic functions for modifying and analysing results on the fly, it plays two significant roles in the API. On the one hand, it defines the cosmological functions that are used by the halo model, such as the halo-mass function or the DM density profile in Fourier space. On the other hand, it provides a set of luminosity functions that the user can associate to the populated catalogue through the SHAM procedure. This approach is not the only one possible, as users are free to define their own observable property distribution and provide it to the function that is responsible for applying the abundance matching algorithm.

Algorithm 1 Schematic outline of the steps required to obtain a mock galaxy catalogue with SCAMPY.

```
// Load Halo/Subhalo hierarchy
// (e.g. from SUBFIND algorithm)
halo_cat = catalogue( chosen from file )

// Choose occupation probability function
OPF = OPF(HOD parameters)

// Populate haloes
gxy_array = halo_cat.populate(model = OPF)

// Associate luminosities
gxy_array = SHAM(gxy_array, SHAM parameters)
```

Once the HOD parametrization and the observable-property distribution have been set, it is possible to populate the halo/sub-halo hierarchy of a DM-only catalogue. In algorithm 1, we outline the steps required to populate a halo catalogue with mock observables. We start from a halo/sub-halo hierarchy obtained by means of some algorithm (e.g. SUBFIND) that have been run on top of a DM-only simulation. This is loaded into a `catalogue` structure that manages the hierarchy dividing the haloes in *central* and *satellite* sub-haloes.[3]

Our `catalogue` class comes with a `populate()` member function that takes an object of type occupation probability as argument and returns a trimmed version of the original catalogue in which only the central and satellite haloes hosting an object of the target population are left. We give a detailed description of this algorithm in Section 3.1.1. When this catalogue is ready, the SHAM algorithm can be run on top of it to associate at each mass a mock-observable property. Cumulative distributions are monotonic by construction. Therefore it is quite easy to define a bijective relation between the cumulative mass distribution of haloes and the cumulative observable property distribution of the target population. This algorithm is described in Section 3.1.2.

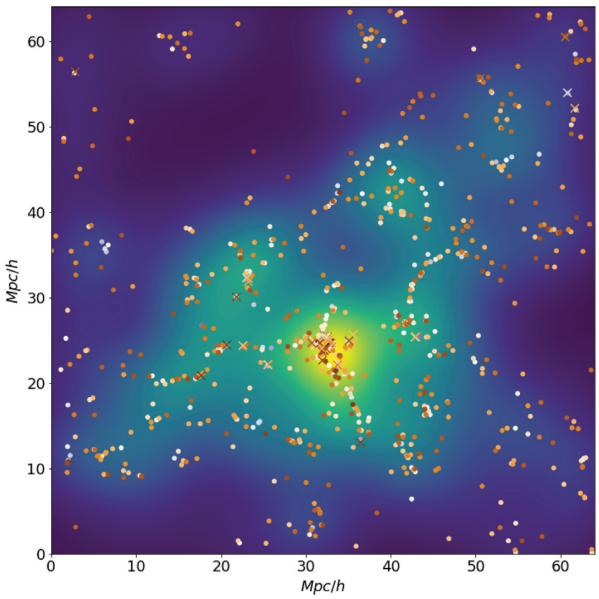

Figure 2. A 4 Mpc h^{-1} thick slice of a populated catalogue obtained from a DM-only simulation with 64 Mpc h^{-1} box side length. The colour code on the background shows the smoothed DM density field (with density increasing going from darker to brighter regions) while the markers show our mock galaxies (with colour representing lower to higher luminosity going from brighter to darker). Circles are for centrals and crosses for satellites.

3.1.1 *Populating algorithm*

Input sub-halo catalogues are trimmed into hosting sub-halo catalogues by passing to the `populate()` member function of the class `catalogue` an object of type `occupation_p`.

In algorithm 2, we describe this halo occupation routine. For each halo i in the catalogue, we compute the values of $\langle N_{\rm cen}\rangle(M_i)$ and $\langle N_{\rm sat}\rangle(M_i)$. To account for the assumptions made in our derivation of

[1]The documentation of the library comprehends a tutorial on how to achieve this.

[2]In the documentation website we will provide a step-by-step tutorial using `emcee` (Foreman-Mackey et al. 2013).

[3]For the case of SUBFIND run on top of a GADGET snapshot this can be done automatically using the `catalogue.read_from_gadget()` function. We plan to add similar functions for different halo-finders [e.g. ROCKSTAR, Behroozi, Wechsler & Wu (2013a), and SPARTA, Diemer (2017)] in future extensions of the library.

the halo model, we select the number of objects each halo will host by extracting a random number from a Poisson distribution. For the occupation of the central halo this reduces to extracting a random variable from a binomial distribution: $N_{\rm cen} = \mathscr{B}(1, \langle N_{\rm cen}\rangle_i)$. While, in the case of satellite sub-haloes, we extract a random Poisson variable $N_{\rm sat} = \mathscr{P}(\langle N_{\rm sat}\rangle_i)$, then we randomly select $N_{\rm sat}$ satellite sub-haloes from those residing in the $i^{\rm th}$ halo.

Algorithm 2 Description of the `populate` (model = OPF) function. This is an implementation of the HOD prescription, where the assumptions made to define the halo model (i.e. the average number of objects within a halo follows a Poisson distribution with mean $\langle N_g\rangle(M_h)$) are accounted for.

```
// Iterate over all the haloes in catalogue
for halo in catalogue do

    // Compute probability of central
    p_cen ← model.N_cen( halo.mass )

    // Define a binomial random variable
    select ← random.Binomial(1, p_cen)
    if select then
        halo ← central

    // Compute average number of satellites
    N̄_sat ← model.N_sat( halo.mass )

    // Define a Poisson random variable
    N_sat = random.Poisson(N̄_sat)
    halo ← select randomly N_sat objects among satellites
```

In Fig. 2, we show a 4 Mpc h^{-1} thick slice of a simulation with box side lenght of 64 Mpc h^{-1}, the background colour code represents the density field traced by all the sub-haloes found by the SUBFIND algorithm, smoothed with a Gaussian filter, while the markers show the positions of the sub-haloes selected by the populating algorithm. We will show in Section 4.1 that this distribution of objects reproduces the observed statistics. It is possible to notice how the markers trace the spatial distribution of the underlying DM density field.

3.1.2 Abundance matching algorithm

When the host sub-haloes have been selected we can run the last step of our algorithm. The `abundance_matching()` function implements the SHAM prescription to associate with each sub-halo an observable property (e.g. a luminosity or the star formation rate of a galaxy). This is achieved by defining a bijective relation between the cumulative distribution of sub-haloes as a function of their mass and the cumulative distribution of the property we want to associate them.

An example of this procedure is shown in Fig. 3. We want to set, for each sub-halo, the UV luminosity of the galaxy it hosts. In the left-hand panel, we show the cumulative mass distribution of sub-haloes, $\mathrm{d}N(M_{\rm subhalo})$, with the dashed green region being the mass resolution limit of the DM sub-haloes in our simulation after the populating algorithm has been applied. On the right-hand panel we show the cumulative UV luminosity function, which is given by the integral

$$\Phi(M^{\rm UV} < M^{\rm UV}_{\rm lim}) = \int_{-\infty}^{M^{\rm UV}_{\rm lim}} \frac{\mathrm{d}\Phi}{\mathrm{d}M^{\rm UV}}\,\mathrm{d}M^{\rm UV} \tag{21}$$

where $M^{\rm UV}_{\rm lim}$ is the limiting magnitude of the survey data we want to reproduce (marked by a dashed red region in Fig 3). We find the abundance corresponding to each mass bin (grey step line in the left-hand panel) and we compute the corresponding luminosity by inverting the cumulative luminosity function obtained with equation (21):

$$M^{\rm UV}(M_{\rm subhalo}) = \Phi^{-1}[\,\mathrm{d}N(M_{\rm subhalo})]\,. \tag{22}$$

The result of this matching is shown by the orange crosses in the right-hand panel of Fig. 3. At the time we are writing, the scatter around the distribution of luminosities can be controlled by tuning the bin-width used to measure $\mathrm{d}N(M_{\rm subhalo})$. We plan to extend this functionality of the API by adding a parameter for tuning this scatter to the value chosen by the user.

In Fig. 2, the colour gradient of markers highlights the increase in their associated luminosity (from lighter to darker colour, going from fainter to brighter object).

3.2 Performance discussion

In this section, we comment some design choices made for optimizing the performances of the library. A detailed discussion on the implementation is provided in Appendix A. For some benchmarking measures of the API performances we refer the reader to Appendix B.

With our hybrid implementation we have deployed a library that exploits both the performance efficiency of a compiled language (C++) as well as the flexibility of an interpreted language (PYTHON). The C++ core of the library has multithreaded sections to run in parallel the most computationally demanding calculations of the workflow. This choice has been made in order to circumvent the Global Interpreter Lock (GIL) which would otherwise force the PYTHON application to run on a single thread. Spawning threads directly from the C++ core is more efficient than using most of the PYTHON packages for multithreading. None the less this behaviour can be suppressed by setting the corresponding environment variable accordingly.

Functions that do not spawn threads by default are those meant for modelling cosmological statistics (e.g. clustering and abundances) and functions that have a pure PYTHON implementation.

Since the former functions would ideally be used in an MCMC framework, they run on a single thread. In this way all the cores of the CPU are available for parallelizing the parameter space sampling. Nevertheless, we have carefully optimized and benchmarked such functions (more details in Appendices A and B, respectively): on a single thread, one single computation of the likelihood in equation (1) for a typical problem size (i.e. a data set with approximately 10 degrees of freedom) takes approximately 1 ms, running on a common laptop.

Concerning the pure PYTHON implementation, the `catalogue` class and the `populate` function are the only sections of the library that would gain from a multithreaded core, given that they operate on lists of independent objects. Such functionality has not been implemented yet but would be a natural evolution of our library. At the current state of the implementation, both these operations take times in the order of 1/10 s for a sub-halo table with $10^{4/5}$ objects. Loading and populating sub-haloes have $\mathcal{O}(N_{\rm sub})$ scaling, where $N_{\rm sub}$ is the number of sub-haloes loaded into the catalogue. The dependence of these time measurements to the machine architecture is negligible.

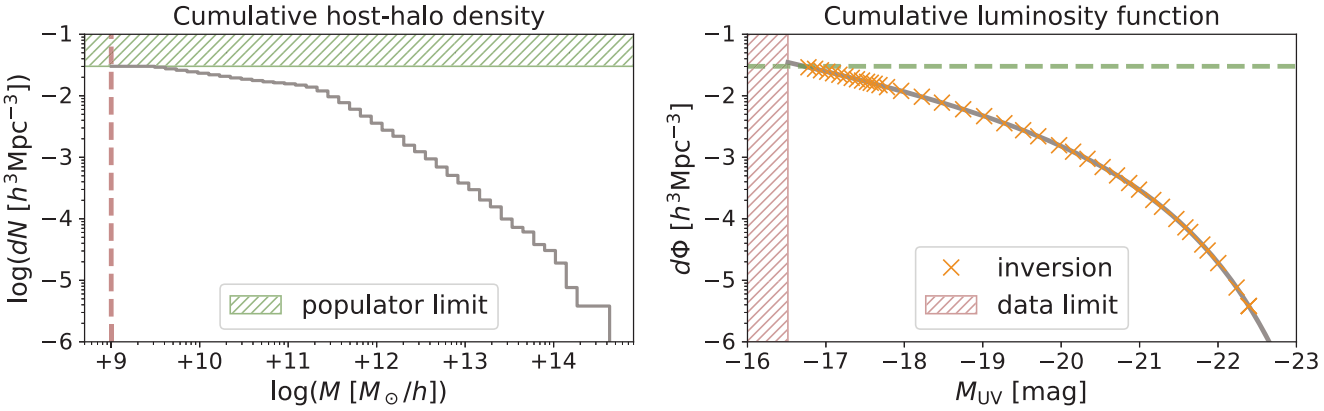

Figure 3. Abundance matching scheme. *Left-hand panel:* cumulative number density of host sub-haloes divided into regular logarithmic bins (solid grey step-line). The green dashed band shows the limit imposed by the resolution in mass of the simulation, which is inherited by the host catalogue obtained with the populator algorithm. *Right-hand panel:* cumulative luminosity function (solid grey line). The dashed red band shows the limit imposed by the magnitude limit of the observed target population. Orange crosses mark the positions, bin-per-bin, of the abundances measured in each bin of the left-hand panel. In both the left-hand and the right-hand panel we reported with a dashed line of corresponding colour the limit imposed by the resolution of the catalogue (dashed green line in right panel) and by the magnitude limit of the survey (dashed red line in the left-hand panel).

Table 1. Fiducial values of the HOD parameters at the different redshifts inspected. The HOD model is defined by equations (19) and (20).

z	$M_{\rm min}$	$\sigma_{\log M}$	$M_{\rm cut}$	M_1	$\alpha_{\rm sat}$
	$[10^{10}\,{\rm M}_\odot\,h^{-1}]$		$[{\rm M}_\odot\,h^{-1}]$	$[10^{12}\,{\rm M}_\odot\,h^{-1}]$	
0	5.0	0.3	0.0	1.0	1.0
2	2.0	0.3	0.0	1.0	1.0
4	2.0	0.3	0.0	1.0	1.0
6	1.0	0.3	0.0	0.5	1.2
8	1.5	0.2	0.0	0.5	1.2

4 VERIFICATION AND VALIDATION

We have extensively tested all the functions building up our API in all their unitary components. We developed a testing machinery, included in the official repository of the project, to run these tests in a continuous integration environment. This will both guarantee consistency during future expansions of the library, as long as providing users with a quick check that the build have been completed successfully.

In this section, we show that our machinery is producing the expected results. Specifically, in Section 4.1 we show that the mock-catalogues obtained with SCAMPY reproduce the observables we want. We have also tested our API for the accuracy in reproducing cross-correlations in Section 4.2. Even though there is no instruction in the algorithm that guarantees this behaviour, using the halo model it is trivial to obtain predictions for the cross-correlation of two different populations of objects.

All the validation tests have been obtained by assuming a set of reasonable values for the HOD parameters of equations (19) and (20). All the parameters used are listed in Table 1. We will refer to these sets of parameters as *fiducial model* in the rest of this manuscript.

The resulting occupation probabilities have been then used to populate a set of halo/sub-halo catalogues. These catalogues have been obtained by running on the fly the friends of friends (FoF) and SUBFIND (Springel et al. 2001) algorithms on top of a set of cosmological N-body simulations. The DM snapshots have been obtained by running the (non-public) P-GADGET-3 N-body code (which is derived from the GADGET-2 code, Springel 2005). In

Table 2. Our set of cosmological simulations with the corresponding relevant physical quantities: $N_{\rm part}$ is the total number of DM particles; $M_{\rm part}$ is the mass of each particle; $L_{\rm box\text{-}side}$ is the side length of the simulation box; $z_{\rm min}$ is the minimum redshift up to which the simulation has been evolved.

Name	$N_{\rm part}$	$M_{\rm part}$	$L_{\rm box\text{-}side}$	$z_{\rm min}$
`lowres`	512^3	$8.13\times10^7\,{\rm M}_\odot\,h^{-1}$	$64\,{\rm Mpc}\,h^{-1}$	0
`midres`	1024^3	$1.02\times10^7\,{\rm M}_\odot\,h^{-1}$	$64\,{\rm Mpc}\,h^{-1}$	2
`highres`	1024^3	$1.27\times10^6\,{\rm M}_\odot\,h^{-1}$	$32\,{\rm Mpc}\,h^{-1}$	2

Table 3. Fiducial cosmological parameters of the N-body simulations used in this work.

h	$\Omega_{\rm CDM}$	$\Omega_{\rm b}$	Ω_Λ	σ_8	n_s
0.7	0.3	0.045	0.7	0.8	0.96

Table 2, we list the different simulation boxes we used for testing the library. Given the large computational cost of running high resolution N-body codes, only the `lowres` simulation box has been evolved up to redshift $z = 0$, while we stopped the others at redshift $z = 2$. The cosmological parameters used for all these simulations are summarized in Table 3.

4.1 Observables

Here, we present measurements obtained after both the populating algorithm of Section 3.1.1 and the abundance matching algorithm of Section 3.1.2 have been applied to the DM-only input catalogue. Applying the abundance matching algorithm does not modify the content of the populated catalogue, besides associating to each mass an additional observable property.

In the two panels of Fig. 4, we show the abundances of central and satellite sub-haloes, as a function of the halo mass. The dashed yellow step-lines show the distribution in the DM-only input catalogue, while the grey solid line marks the distribution defined by the fiducial occupation probability functions. By applying the populating algorithm (Section 3.1.1) to the input catalogue we obtain the distributions marked by the solid red step-lines, which are in perfect agreement with the expected distribution.

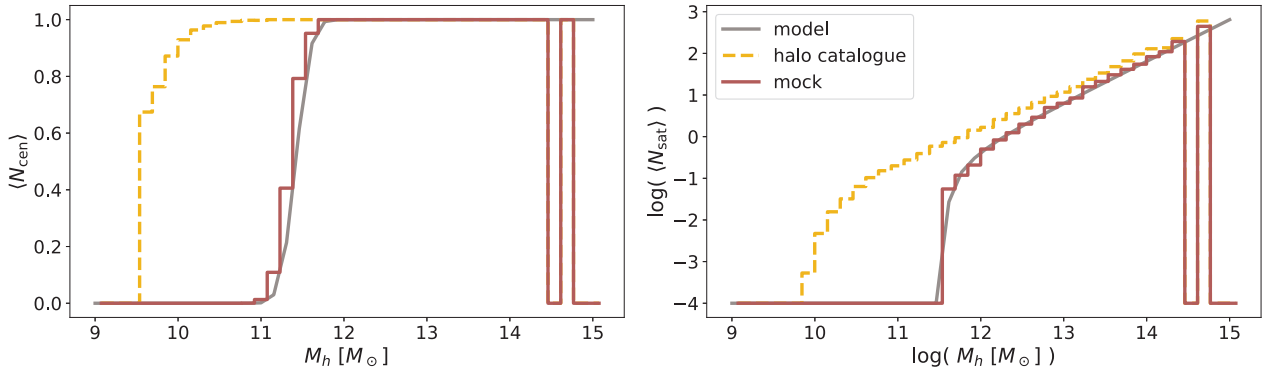

Figure 4. Occupation probability functions for central sub-haloes (*left-hand panel*) and satellite sub-haloes (*right-hand panel*). The grey solid line marks the $z = 0$ fiducial model of Table 1. The yellow step-wise dashed line and the red step-wise solid line mark the distributions measured on the sub-halo catalogue before and after having applied our populating algorithm.

We then draw a random Gaussian sample around the halo model estimate for the objects abundance and their clustering using the above selection of occupation probabilities (equations 5 and 6, respectively). These random samples build up our *mock data set*. We then run an MCMC sampling of the parameter space, with the likelihood of equation (1), to infer the set of parameters that best fit the mock data set. For sampling the parameter space we use the EMCEE (Foreman-Mackey et al. 2013) Affine Invariant MCMC Ensemble sampler, along with the SCAMPY PYTHON interface to the halo model estimates of $n_g(z)$ and $\xi(r, z)$.

After having obtained the best-fitting parameters, we produce 10 runs of the full pipeline described in algorithm 1. In doing this, we are producing 10 different realizations of the resulting mock catalogue. Since the selection of the host sub-haloes is not deterministic, this procedure allows to obtain an estimate of the errors resulting from the assumptions of the halo model. Finally, we use the LandySzalay (Landy & Szalay 1993) estimator in each of the populated catalogues to measure the 2-point correlation function:

$$\xi(r) = \frac{DD(r) - 2DR(r) + RR(r)}{RR(r)} \tag{23}$$

where $DD(r)$ is the normalized number of unique pairs of sub-haloes with separation r, $DR(r)$ is the normalized number of unique pairs between the populated catalogue and a mock sample of objects with random positions, and $RR(r)$ is the normalized number of unique pairs in the random objects catalogue. We then measure, with equation (23), the clustering in each of the 10 realizations and we compute the mean and standard deviation of these measurements in each radii bin.

The results are shown in Fig. 5, for redshift $z = 0$, and in Fig. 6, for redshifts $z = 2$, 4, 6, and 8. In the upper panel of Fig. 5 we show the mock data set with triangle markers and errors, the lines show the halo model best-fitting estimate of the 2-point correlation function (with the different contributes of the 1- and 2-halo terms). The circle markers show the average measure obtained from our set of mock catalogues.

In the lower panel of Fig. 5 and in the four panels of Fig. 6 we show the relative distance between the measure performed on the catalogue populated with the best-fitting parametrization of the occupation probabilities ($\xi_{\rm b.fit}$) and on a catalogue populated with the fiducial value of the parameters ($\xi_{\rm fid}$). Comparing the measurements in the two different populated catalogues, instead of comparing with the model itself, guarantees that discrepancies due to box-size and resolution of the simulation used are mitigated in the distance ratio plot.

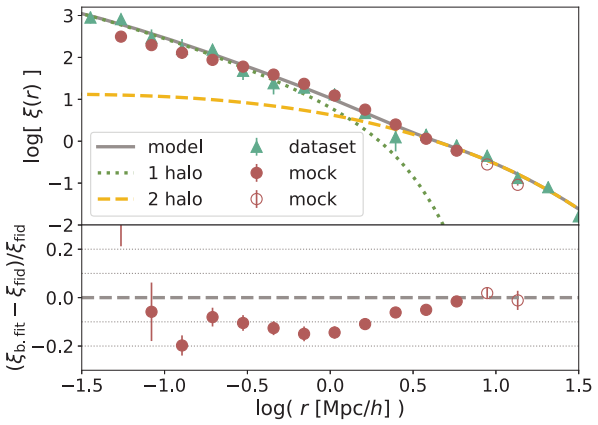

Figure 5. Validation of the two-point correlation function at redshift $z = 0$. *Upper panel:* comparison between the mock clustering data set (triangles), the best-fitting halo-model prediction (solid line) and the mean and standard deviation of the clustering measured with the Landy–Szalay estimator on the 10 realizations (circles and errors), we also show the modelled 1-halo (dotted line) and 2-halo (dashed line) terms for reference. *Lower panel:* red markers show the distance ratio between the measurement obtained on the mock catalogue with the best-fitting parameters and the averaged measurement obtained on the mock catalogue with fiducial parameters. The dashed line shows 0 per cent distance between the two. In both panels, empty circles mark measurements at $r > 6.4$ Mpc h^{-1}, where the limited box size affects the precision of the result.

As it is shown in the lower panel of Fig. 5, at redshift $z = 0$, the catalogue populated with the best-fitting parameters reproduces the clustering properties of the fiducial catalogue with a distance lower than 15 per cent on most of the scales inspected.

The discrepancies at small scales could depend on several effects. In particular:

(i) Implementation of the populating algorithm: galaxies can be assigned to satellite haloes randomly or by rank ordering them based on some property of the sub-halo. The two methods might produce different levels of clustering within the halo.

(ii) The sub-halo finder used: SUBFIND is known for not resolving completely the sub-halo hierarchy closer to the halo centre.

(iii) Simulation resolution: discretization of the DM distribution limits the smallest sub-halo that can be modelled. This also results in a larger scatter when going at higher redshift (as shown in Fig. 6) where the average size of a halo gets smaller.

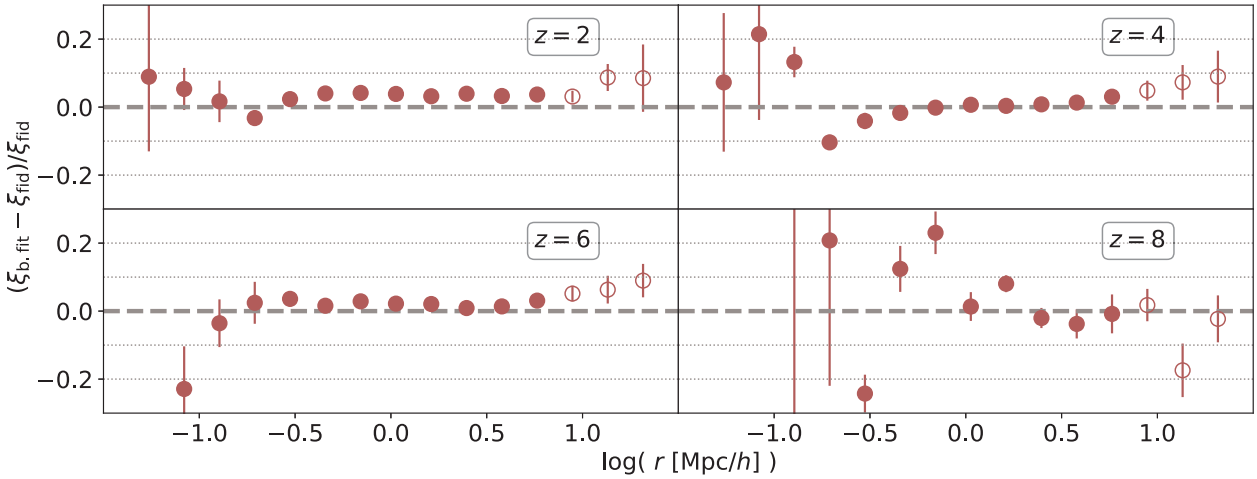

Figure 6. Validation of the two-point correlation function at redshift $z = 2$ (upper left panel), $z = 4$ (upper right panel), $z = 6$ (lower left panel), and $z = 8$ (lower right panel). Red markers show the distance ratio between the averaged-measurement obtained on the mock catalogue with the best-fitting parameters and the measurement obtained on the mock catalogue with fiducial parameters. The dashed line shows 0 per cent distance between the two. As in Fig. 5, empty circles mark measurements at $r > 6.4$ Mpc h^{-1}.

Concerning the latter point, it is worth to mention that, even thought the measurement is less precise, the average distance from the expected result is still lower than 10\15 per cent. Finally, in both Figs 5 and 6, we mark with empty circles measurements taken at radii $r > 6.4$ Mpc h^{-1}, where the limited size of the simulation box affects the statistics.

All the discrepancies we find are a known weakness of the HOD method. In literature there have been a lot of effort in quantifying and correcting this effect (see e.g. Beltz-Mohrmann, Berlind & Szewciw 2020; Hadzhiyska et al. 2020, for two recent works), which, as already mentioned, is thought to result from a concurrence of box-size effects, cosmic variance, and assembly bias. Hadzhiyska et al. (2020), in particular, find an average distance of 15 per cent between the HOD prediction and the clustering measured in hydrodynamical N-body simulations.

The requirement of reproducing the 1-point statistics of the original catalogue is necessary to have the expected observational property distribution in the output mock-catalogue. This requirement guarantees that the abundance matching scheme will start associating the observational property from the right position in the cumulative distribution, i.e. from the abundance corresponding to the limiting value that said property has in the survey.

In Fig. 7, we show the example case of the UV luminosity function. We mark with orange circles the cumulative luminosity function measured on the mock-catalogue after the application of our API. For comparison we also show the luminosity function model we are matching (grey solid line) and the observation limit of the target population (red dashed region).

As it is shown in the lower panel of Fig. 7, the distance ratio between the expected distribution and the mock distribution is lower than ≈ 10 per cent over all the range of magnitudes.

The halo-model prediction for the total abundance of sources is $n_{\rm g}^{\rm hm}(z) = 3.49 \times 10^{-2}$ $[h^3{\rm Mpc}^{-3}]$, while we measure $n_{\rm g}^{\rm pop}(z) = (3.06 \pm 0.04) \times 10^{-2}$ $[h^3{\rm Mpc}^{-3}]$ in the populated catalogue. Such a miss-match is somehow expected. In fact, it has been shown (e.g. Sinha et al. 2018) that the HOD model presents difficulties when fitting multiple statistics. For the distribution of sources shown in Fig. 7, we correct this error by extrapolating the limiting magnitude (marked by the hatched region) to the value matching the abundance of the populated galaxies.

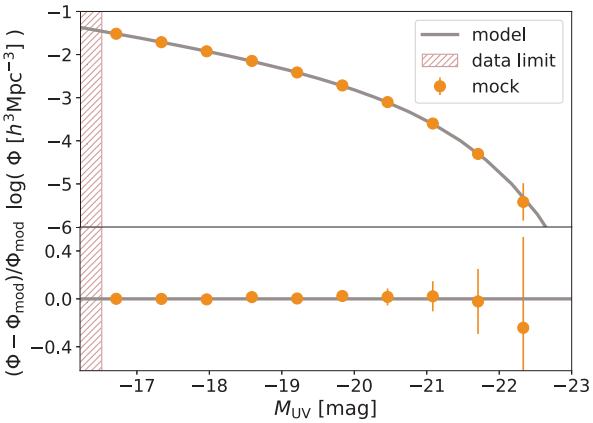

Figure 7. Cumulative luminosity function at redshift $z = 0$. *Upper panel:* the grey solid line marks the model prediction while the orange circles with errors mark the distribution measured on the populated catalogue. The hatched red region marks the limiting magnitude $M_{\rm lim}^{\rm UV}$. *Lower panel:* distance ratio between the luminosity function measured on the populated catalogue and the model.

Sinha et al. (2018) also show that the clustering statistics with the higher constraining power depends on the galaxy population that has to be modelled. Since we are running our analysis on a simulated data set, with the aim of validating the computational framework, we are not pushing this analysis further. Nevertheless, in a real-life application, the likelihood of equation (1) should be adjusted considering these observations, depending on the scientific goal of the application. The `halo_model` class of our library provides a wide range of cosmological statistics that can be modelled, leaving to the users the responsibility of choosing the one that better suits their needs.

4.2 Multiple populations cross-correlation

Even though in our API there is no prescription for this purpose, it is interesting to test how the framework performs in predicting the cross-correlation between two different populations. This quantity

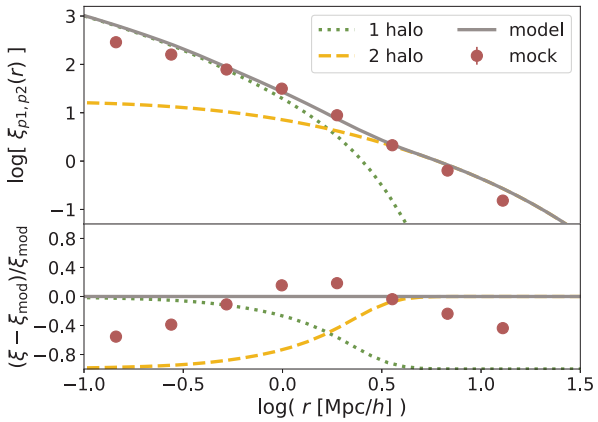

Figure 8. Comparison between the cross-correlation function, measured with the modified Landy–Szalay estimator of equation (26), between two dummy mock-populations at redshift $z = 0$. The lower panel shows the distance ratio between the measurement and the model prediction.

measures the fractional excess probability, relative to a random distribution, of finding a mock-source of population 1 and a mock-source of population 2, respectively, within infinitesimal volumes separated by a given distance.

It is simple to modify equations (10) and (11) to get the expected power spectrum of the cross-correlation (Cooray & Sheth 2002). For the 1-halo term this is achieved by splitting the $(M_h/\overline{\rho})^2$ of equation (10) in the contribution of the two different populations, which leads to the following equation:

$$P_{1h}^{(1,2)}(k,z) = \frac{1}{n_g^{(1)}(z) n_g^{(2)}(z)} \cdot \int_{M_{min}}^{M_{max}} N_g^{(1)}(M_h) N_g^{(2)}(M_h) n_h(M_h) |\widetilde{u}_h(k, M_h, z)|^2 dM_h \quad (24)$$

where quantities referring to the two different populations are marked with the superscripts (1) and (2).

For the case of the 2-halo term, obtaining an expression for the cross-correlation requires to divide the two integrals of equation (11) in the contributions of the two different populations, leading to

$$P_{2h}^{(1,2)}(k,z) = \frac{P_m(k,z)}{n_g^{(1)}(z) n_g^{(2)}(z)} \cdot \left[\int_{M_{min}}^{M_{max}} N_g^{(1)}(M_h) n_h(M_h) b_h(M_h, z) \widetilde{u}_h(k, M_h, z) dM_h \right] \cdot \left[\int_{M_{min}}^{M_{max}} N_g^{(2)}(M_h) n_h(M_h) b_h(M_h, z) \widetilde{u}_h(k, M_h, z) dM_h \right] \quad (25)$$

We get the cross-correlation of the two mock-populations using a modification (González-Nuevo et al. 2017) of the Landy–Szalay estimator

$$\xi^{(1,2)}(r) = \frac{D_1D_2(r) - D_1R_2(r) - D_2R_1(r) + R_1R_2(r)}{R_1R_2(r)}, \quad (26)$$

where $D_1D_2(r)$, $D_1R_2(r)$, $D_2R_1(r)$, and $R_1R_2(r)$ are the normalized data1-data2, data1-random2, data2-random1, and random1-random2, respectively, pair counts for a given distance r.

In Fig. 8, the red circles show the cross-correlation measured with equation (26) for two different mock-populations with a dummy choice of the occupation probabilities parameters. Errors are measured using a bootstrap scheme with 10 sub-samples. For comparison, we also show the halo-model prediction of the two-point correlation function, obtained with equations (24) and (5), separated in 1-halo and 2-halo term contribution. The lower panel of Fig. 8 shows the distance ratio between the measure and the model prediction, which is lower than 40 per cent over almost all the scales inspected.

5 PROOF-OF-CONCEPT APPLICATION: IONIZING PHOTONS PRODUCTION FROM LYMAN-BREAK GALAXIES AT HIGH REDSHIFT

In the context of high redshift cosmology, one of the most compelling open problems is the process of reionization, that brought the Universe from the optically thick state of the Dark Ages to the transparent state we observe today (see e.g. Choudhury & Ferrara 2006; Wise 2019, for reviews). Modelling this phase of the Universe evolution is a tricky task, especially using methods tuned to reproduce the observations at low redshift, such as hydrodynamical simulations and semi-analytical models. Instead, if we trust the capability of N-body simulations to capture the evolution of DM haloes up to the highest redshifts, an empirical method such as SCAMPY is more likely to correctly predict the distribution of sources.

We expect reionization to occur as a non-homogeneous process in which patches of the Universe ionize and then merge, prompted by the formation of the first luminous sources (Barkana & Loeb 2001). In order to map the spatial distribution of these ionized bubbles to the underlying DM distribution we apply our method to reproduce the observations of eligible candidates for the production of the required ionizing photon budget. It is commonly accepted that the primary role in the production of ionizing photons at high redshift has been played by primordial, star-forming galaxies (Shapiro & Giroux 1987; Miralda-Escude & Ostriker 1990; Barkana & Loeb 2001; Steidel, Pettini & Adelberger 2001). The best candidates for these objects are Lyman-break galaxies (LBGs) which are selected in surveys using their differing appearance in several imaging filters, due to the position of the Lyman limit (Steidel et al. 2001, 2018; Lapi et al. 2017; Matthee et al. 2018).

The first galaxies that started to inject ionizing radiation in the intergalactic medium were hosted in small DM haloes with masses up to a minimum of 10^8 $M_\odot$ h^{-1}. In order to paint a population of LBGs on top of a DM simulation, we need, first of all, a high resolution N-body simulation to provide the halo/sub-halo hierarchy required by SCAMPY. We therefore run the FoF and SUBFIND algorithms on top of 25 snapshots in the redshift range $4 \leq z \leq 10$ with thinness $\Delta z = 0.25$. The DM snapshots have been obtained by running the (non-public) P-GADGET-3 N-body code (which is derived from the GADGET-2 code, Springel 2005) on the two simulation dubbed `highres` and `midres` in Table 2.

Simulating the reionization process in more detail would required larger simulations at the same level of mass resolution as obtained for the `highres` and `midres` catalogues of our sample, the computational cost becoming out of reach for the test we want to perform here. Overcoming the computational cost of N-body simulations could be obtained by applying up-sampling techniques of sub-grid modelling. Such methods use a low resolution density field and build mock halo catalogues either by matching the theoretical predictions of the halo mass function (de la Torre & Peacock 2013; Angulo et al. 2014) or the bias evolution in time (Nasirudin, Iliev & Ahn 2020). This goes beyond the aim of the test-bench application we want to present here and we reserve to exploit it in future extensions of this work.

To set the occupation probabilities parameters with the likelihood in equation (1), we need the 1- and 2-point statistics of LBGs at high redshift. The observational constrains of these high redshift statistics,

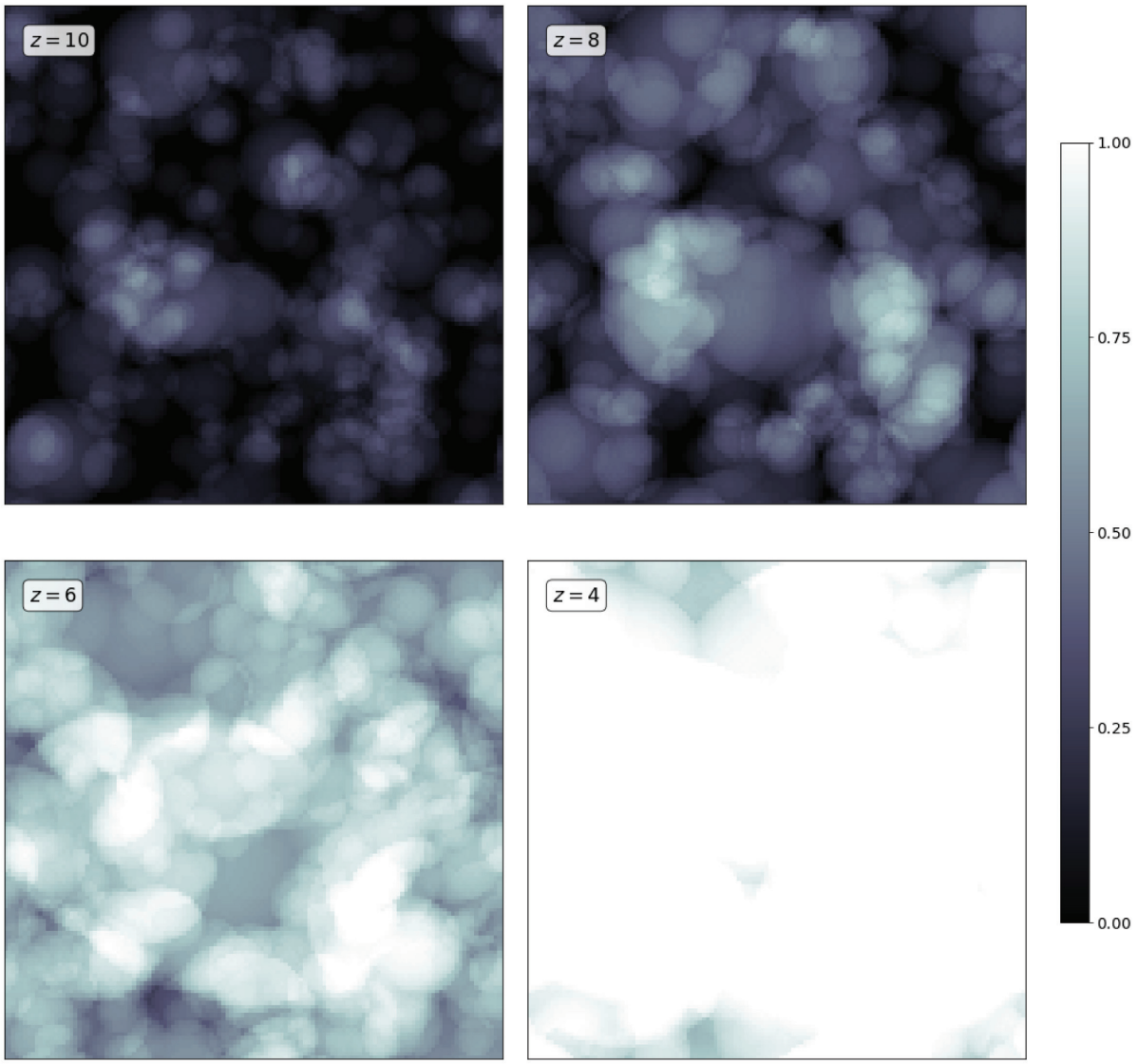

Figure 9. Four snapshots at different redshift (shown in the top left angle of each tile) of the ionization fraction obtained by projecting the values in each voxel along one dimension. The value of $X_{\rm re}$ increases from darker to lighter shades of grey.

due to the high distances involved, are not sufficiently tight. We have therefore extrapolated the available measures as follows:

(i) in Bouwens et al. (2019) the luminosity function of LBGs is fitted up to redshift $z = 10$ and $M_{\rm UV} \approx -16$. We extrapolate this fit up to $M_{\rm UV} = -13$ and integrate to obtain an estimate of the number density of LBGs at high redshift.

(ii) Harikane et al. (2016) provide measurements of the angular 2-point correlation function in the redshift range $4 \leq z \leq 7$. We assume the clustering at redshift $z \geq 7$ to be constant and equal to the measurement obtained for redshift $z = 7$.

Both the observables assumed above are simplistic approximations which are though sufficient for test-benching our model. We will investigate further the limits imposed by the lack of statistics at high redshift in future extensions of this work.

Once the parameters of the model have been set for each redshift, we run the algorithm described in Section 3.1 and obtain a set of LBG mock catalogues. As a first approximation, let us define a neutral hydrogen distribution on top of each of our snapshots and consider the ionized region that should form around each source of our mock catalogues. Each mock-LBG in our simulation is producing an amount of ionizing photons which is proportional to its UV luminosity, $M_{\rm UV}$. Namely, the rate of ionizing photons that escape from each UV source is

$$\dot{N}_{\rm ion}(M_{\rm UV}) \approx f_{\rm esc}\, k_{\rm ion}\, {\rm SFR}(M_{\rm UV}) \tag{27}$$

where $k_{\rm ion} \approx 4 \times 10^{53}$ is the number of ionizing photons $s^{-1}\,(M_\odot/yr)^{-1}$, with the quoted value appropriate for a Chabrier initial mass function (IMF), $f_{\rm esc}$ is the average escape fraction for ionizing photons from the interstellar medium of high-redshift galaxies (see e.g. Mao et al. 2007; Dunlop et al. 2013; Robertson et al. 2015; Lapi et al. 2017; Chisholm et al. 2018; Matthee et al. 2018; Steidel et al. 2018), and $\log({\rm SFR}(M_{\rm UV})) \approx -7.4 - 0.4\,M_{\rm UV}$ is the star formation rate of each source. The volume of the Strömgren sphere, that forms around each mock-LBG, is then given by

$$V_{\rm S} \equiv \frac{\dot{N}_{\rm ion}(M_{\rm UV})}{\overline{n}_{\rm H}(z)}\, t_{\rm rec}\, (1 - {\rm e}^{-t/t_{\rm rec}}) \tag{28}$$

where $\overline{n}_{\rm H}(z) \approx 2 \times 10^{-7}\ (\Omega_{\rm b}(z)h^2/0.022)\ {\rm cm}^{-3}$ is the mean comoving hydrogen number density at given redshift while t is the cosmic time at given redshift and $t_{\rm rec}$ is the cosmic time at the epoch the source started producing a steady flux of ionizing photons.

We make the following simplistic assumptions:

(i) at each snapshot we do not provide any information about the previous reionization history: at each redshift sources have to completely ionize the medium and the value of $t_{\rm rec}$ is fixed at the cosmic time corresponding to $z = 20$.

(ii) the escape fraction is set to $f_{\rm esc} = 0.1$, which is a conservative value with respect to what recent observations suggest.

With the aforementioned simplifications, we can build spheres around each source at each redshift, therefore producing an approximated map of the ionization state of our snapshots, without having to rely on radiative transfer. In Fig. 9 we show the projection along one dimension of 4 snapshots at redshift $z = 10$, 8, 6 and 4 for the `highres` simulation. To get the point-by-point ionization fraction we divide our snapshots into voxels of fixed size. If a voxel is embedded within the Strömgren sphere belonging to some source, it is set as ionized, otherwise it is considered neutral. Voxels that lie in the overlapping of two or more Strömgren spheres are counted only once. This further approximation implies *losing* an amount of ionizing photons which is proportional to the overlapping volume of the whole simulation box. This approximation mainly affects the lower redshifts, as shown in the next Section. In Fig. 9 we set the voxel-side to 0.25 Mpc h^{-1}, resulting in 128^3 voxels in total.

In the remaining part of this Section we will show the results of some measurements that can be obtained from these mock ionization snapshots.

5.1 Ionized fraction measurement

At each redshift, we measure the ionization fraction resulting from our pipeline by counting the number of voxels marked as ionized over the total number of voxels in which the snapshot is divided. The results for both the `midres` and the `highres` simulations are marked with empty squares and red circles, respectively, in Fig. 10. We compare our measurement with models of reionization history from recent literature.

The grey shaded region shows the tanh-model used in Planck Collaboration VI (2018) while the orange one delimits the prediction of the same model with a larger value of the parameter that regulates the steepness of the ionization fraction evolution ($\Delta_z = 1.5$ instead of $\Delta_z = 0.5$, as from Lewis 2008). The solid green line shows the model from Kulkarni et al. (2019) which is obtained by computing with the ATON code (Aubert & Teyssier 2008, 2010), the radiative transfer a posteriori on top of a gas density distribution obtained using the P-GADGET-3 code with the QUICK_LYALPHA approximation from Viel, Haehnelt & Springel (2004).

Our mock ionization boxes predict reionization to reach half-completion ($X_{\rm re} = 0.5$) at redshift $z = 6.88^{+0.12}_{-0.13}$, which is a lower value with respect to the Planck Collaboration VI (2018) prediction of $z = 7.68 \pm 0.79$, but still within the error bars. Comparing to the extremely steep model used in Planck Collaboration VI (2018), the evolution in our simulations is way shallower, closer to the lower limit of the modified tanh-model. None the less, our measurements seem to agree fairly well with the measurements of Kulkarni et al. (2019) up to redshift $z \approx 6$. With respect to the other authors, our simulation reaches completeness (i.e. $X_{\rm re} = 1$) at redshift $z \approx 4$. As anticipated, this issue at the lowest redshifts is by some extent expected. The overall ionizing photon budget is indeed underestimated in our approximation as the result of how we treat the overlapping region between different Strömgren spheres. We plan to address this point, by implementing a physically motivated treatment of these overlapping regions (e.g. in Zahn et al. 2007) in future extensions of our analysis.

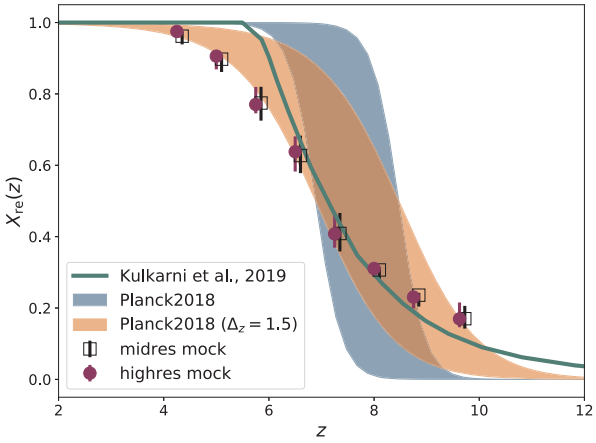

Figure 10. Evolution of the hydrogen ionization fraction $X_{\rm re}$ with redshift. Red circles and empty squares mark the measurements obtained with our method on the `midres` and `highres` simulation, respectively. The `midres` distribution has been shifted with an offset of $\delta z = 0.1$ along the x-axis direction to better distinguish it from the `highres` one. The shaded regions delimit the model used in Planck Collaboration VI (2018) and a modification for widening the reionization window. The solid line shows the prediction from Kulkarni et al. (2019).

Taking into account the strong approximations made in this proof-of-concept application, the measurement we obtain for the evolution of $X_{\rm re}$ is surprisingly consistent with equivalent measures in literature.

5.2 Ionized bubble size distribution

It is accepted that reionization results from the percolation of ionized H II bubbles as well as from their growth in radius (Miralda-Escude & Ostriker 1990; Furlanetto, Zaldarriaga & Hernquist 2004; Wang & Hu 2006) in the neutral intergalactic medium. A relevant statistics for cosmological studies is the size distribution of the individual bubbles forming around ionizing radiation sources. Obtaining precise measurements of this statistics could help constraining future experiments, such as CMB-S4 (Roy et al. 2018) and 21 cm intensity mapping (e.g. Mesinger, Furlanetto & Cen 2011).

In our framework, getting estimates of the bubble size distribution is straightforward. In Fig. 11, we present measurements of two different definitions for the bubble size probability.

On the left-hand panel we plot the fraction of bubbles of given size over the total number of bubbles in the simulation box. The measurement has been obtained at redshift $4 \leq z \leq 10$, with bin size $\delta z = 1$, we plot results only for $z = 4$, 6, 8, and 10 for clarity. The distribution shown presents a lognormal shape that we fit with the model from Roy et al. (2018)

$$P(R) = \frac{1}{R}\,\frac{1}{\sqrt{2\pi\sigma^2_{\ln r}}}\,\exp\left\{-\frac{\left[\ln(R/\overline{R})\right]^2}{2\sigma^2_{\ln r}}\right\}; \tag{29}$$

the model is regulated by two free parameters: the characteristic bubble size $\overline{R}$ (in Mpc h^{-1}) and the standard deviation $\sigma_{\ln r}$. We list the best-fitting values of these parameters in Table 4, for the different redshifts considered. While the value of the characteristic radius is almost constant in time with a value of $\overline{R} \approx 0.2$ Mpc h^{-1},

Table 4. Best-fitting parameters of the lognormal model defined in equation (29) obtained from our measures on the `highres` mock ionized bubble catalogue.

z	$\overline{R}$ [Mpc h^{-1}]	$\sigma^2_{\ln r}$
10	0.229 ± 0.006	0.614 ± 0.025
9	0.181 ± 0.006	0.786 ± 0.033
8	0.222 ± 0.004	0.810 ± 0.024
7	0.213 ± 0.003	0.974 ± 0.022
6	0.165 ± 0.003	1.340 ± 0.033
5	0.178 ± 0.003	1.616 ± 0.052
4	0.208 ± 0.003	1.983 ± 0.052

the standard deviation increases significantly from higher to lower redshift.

On the right-hand panel of Fig. 11, we show the fraction of ionized voxels as a function of the bubble radius over the total number of ionized voxels in the simulation box (normalized to 1). The solid lines show the measurements obtained from the `highres` box, while the dashed ones mark the distribution obtained from the `midres` box. The results on the two boxes are consistent between the two simulations, especially at lower redshifts. Compared to the left-hand panel, the measurements obtained for the bubble size probability definition of the right-hand panel are more consistent with what can be found in literature (e.g. Zahn et al. 2007). In particular, the characteristic radius seems to grow from higher to lower redshift, reaching values in the order of $1\backslash 10$ Mpc h^{-1}. We could not fit the distributions on the right-hand panel of Fig. 11 with the same lognormal model of equation (29). This is probably due to not having considered bubble overlapping in our measurements. We will investigate further on this topic in future work.

6 SUMMARY AND CONCLUSIONS

We have here presented SCAMPY, our application for painting observed populations of objects on top of DM-only N-body cosmological simulations. With the provided PYTHON interface, users can load and populate DM haloes and sub-haloes obtained by means of the FoF and SUBFIND algorithms applied to DM snapshots at any redshift. We foresee to extend this framework to the usage with DM halo and sub-halo catalogues obtained with alternative algorithms.

The main requirements that guided the design of SCAMPY were to provide a flexible and optimized framework for approaching a wide variety of problems, while keeping the computation fast and efficient. To this end, we stick to the simple, yet physically robust, SCAM prescription for providing the recipe to populate DM haloes and sub-haloes. In Section 2, we presented an overview of the theoretical background of this methodology, while, in Section 3 we provided a detailed description of the components and main algorithms implemented in SCAMPY.

In Section 4, we have shown a set of measurements obtained from simulations populated with galaxies using SCAMPY. We have demonstrated that the output mock-galaxies have the expected abundance and clustering properties. Furthermore, we have also proven that the same API could be used to 'paint' multiple populations on top of the same DM simulation and that the cross-correlation between these populations is also well mimicked.

In the context of reionization, this could help in studying the spatial distribution properties and evolution in time of the ionized bubbles that might have developed around sources of ionizing radiation. In Section 5, we have performed a preliminary study, under simplistic assumptions, on the ionization properties of the high redshift Universe which result from the injection in the medium of ionizing photons from Lyman Break Galaxies (LBG). We are now able to measure locally on simulations the ionized hydrogen filling factor at different redshifts. This also allows to perform a tomographic measure of the ionization state of the medium at varying cosmic time. Furthermore, we can also directly measure the ionized bubble size distribution, which is a quantity that, up to now, has been either modelled indirectly (Roy et al. 2018) or measured assuming radiative transfer (Zahn et al. 2007).

While a specific problem prompted the development of the API, extensibility has been a crucial design choice. SCAMPY features a modular structure exploiting Object-Oriented programming, both in C++ and in PYTHON. With a wise usage of polymorphism, we obtained a flexible application that can be both used by itself as well as along with other libraries.

We are working on adding to the API further miscellaneous functionalities, such as the possibility to download and install it with

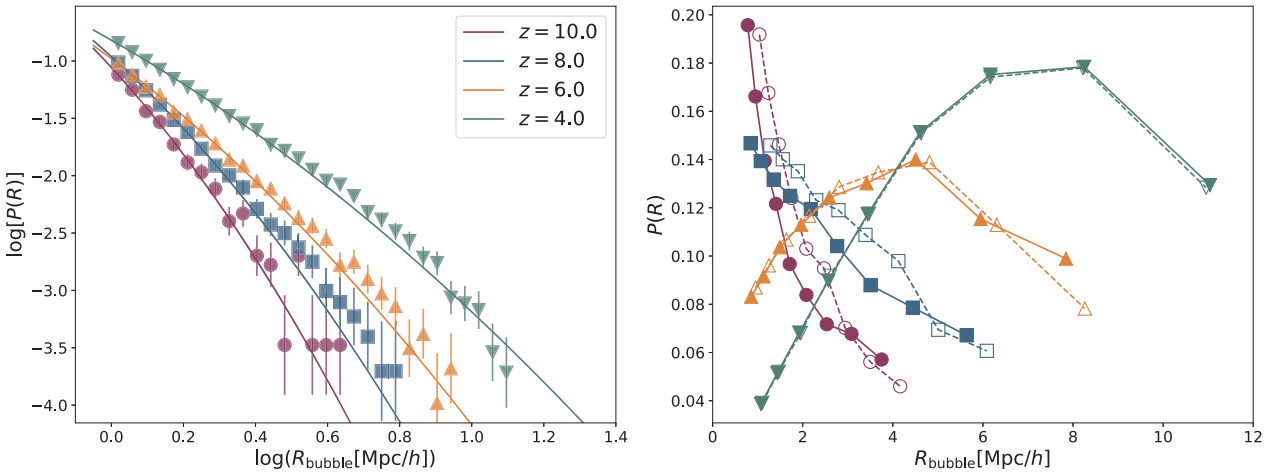

Figure 11. Bubble size probability distributions. *Left-hand panel:* fraction of the bubbles with given size over the total number of bubbles, markers are measured from the `highres` simulation while the solid lines show the model of equation (29) fitted on these data (the best-fitting parameters are listed in Table 4). *Right-hand panel:* fraction of ionized voxels embedded in bubbles with given size over the total number of ionized voxels. Dashed lines mark the measurements obtained from the `midres` simulation, while solid lines have been obtained from the `highres` simulation.

both `pip` and `conda`, in the next months. The online documentation is continuously updated, and more examples and tutorials are ready to be uploaded in the next months.

To conclude, we are planning to extend our work both on the scientific side, by exploring new directions, and on the computational side, by implementing an efficient k^d-tree algorithm for the optimization of the neighbour search in both DM-halo catalogues and mock-galaxy catalogues. A possible list of the directions we are planning to pursue follows.

(i) *Application of the algorithm to multiple populations/different tracers of the LSS* – An application of the same pipeline to the other major players that are known to be involved on the reionization of hydrogen, e.g. AGNs, would be trivial, provided that suitable data sets are available. Another possibility would be the cross-correlation with intensity mapping like e.g. Spinelli et al. (2020).

(ii) *Application to the reionization process* – Up to now we have mainly applied SCAMPY functionalities to a proof-of-concept framework. None the less, with the data set at our disposal, an extensive study of the role played by LBG in reionization is possible, provided that larger *N*-body simulations with high resolution are available. This would require DM halo catalogues on large simulation boxes, with a complete sampling of the lower masses (down to $10^8\,M_\odot$). Such catalogues could be obtained by running high resolution *N*-body simulations or, more realistically, by applying a subresolution scheme, such as the halo-bias models proposed by Nasirudin et al. (2020).

(iii) *Extension to different cosmological models* – By now all our investigations have been performed assuming a lambda cold dark matter cosmology. To extend these results to other cosmological models would only require modest modifications of the source code and to obtain the corresponding DM-only *N*-body simulations, similar to high-redshift studies performed e.g. in warm Dm scenarios or massive neutrino cosmologies (Maio & Viel 2015; Fontanot et al. 2015).

(iv) *Machine learning extension of the halo occupation model* – Using the HOD is straightforward and a lot of literature is available on the topic. This approach, though, comes with the limit that all the properties of the observed population have to be inferred from the mass of the host halo. This is known to be a rough approximation. To overcome this limit, we plan to use, instead, a neural network (NN) model of the host halo occupation properties where the inputs of the NN are a set of known features of the halo/sub-halo hierarchy, such as the local environment around the halo and the dispersion velocity within the halo. Matter density based approaches for extending DM only *N*-body simulations have already been attempted. Recent efforts in this sense include work from He et al. (2019) which use NNs to predict the non-linear evolution of matter perturbations beyond the Zel'dovic approximation. In another work, Yip et al. (2019), paint baryons on top of simulations only run with DM. We plan instead to investigate the possibility to post process the baryonic effects on top of DM-only simulations in a halo-based framework.

ACKNOWLEDGEMENTS

The authors warmly thank the referee, Boryana Hadzhiyska, for the constructive report. TR is thankful to Federico Marulli, Alfonso Veropalumbo, and Luigi Danese for useful discussion. This work has been partially supported by PRIN MIUR 2017 prot. 20173ML3WW 002, 'Opening the ALMA window on the cosmic evolution of gas, stars and supermassive black holes'. AL acknowledges the MIUR grant 'Finanziamento annuale individuale attivitá base di ricerca' and the EU H2020-MSCA-ITN-2019 Project 860744 'BiD4BEST: Big Data applications for Black hole Evolution STudies'. MV and TR are supported by the INFN-PD51 INDARK grant. MV is also supported by financial contribution from the agreement ASI-INAF no. 2017-14-H.0.

DATA AVAILABILITY

No new relevant data were generated or analysed in support of this research. All the information for reproducing our results are available in the article and on the website of the presented package (scampy.readthedocs.io).

REFERENCES

Amendola L. et al., 2018, Living Rev. Relativ., 21, 2
Angulo R. E., Springel V., White S. D. M., Jenkins A., Baugh C. M., Frenk C. S., 2012, MNRAS, 426, 2046
Angulo R. E., Baugh C. M., Frenk C. S., Lacey C. G., 2014, MNRAS, 442, 3256
Astropy Collaboration, 2013, A&A, 558, A33
Astropy Collaboration, 2018, AJ, 156, 123
Aubert D., Teyssier R., 2008, MNRAS, 387, 295
Aubert D., Teyssier R., 2010, ApJ, 724, 244
Barkana R., Loeb A., 2001, Phys. Rep., 349, 125
Behroozi P. S., Silk J., 2015, ApJ, 799, 32
Behroozi P. S., Conroy C., Wechsler R. H., 2010, ApJ, 717, 379
Behroozi P. S., Wechsler R. H., Conroy C., 2012, ApJ, 762, L31
Behroozi P. S., Wechsler R. H., Wu H.-Y., 2013a, ApJ, 762, 109
Behroozi P. S., Wechsler R. H., Conroy C., 2013b, ApJ, 770, 57
Behroozi P., Wechsler R. H., Hearin A. P., Conroy C., 2019, MNRAS, 488, 3143
Beltz-Mohrmann G. D., Berlind A. A., Szewciw A. O., 2019, MNRAS, 491, 5771
Berlind A. A., Weinberg D. H., 2002, ApJ, 575, 587
Bouwens R. J., Stefanon M., Oesch P. A., Illingworth G. D., Nanayakkara T., Roberts-Borsani G., Labbé I., Smit R., 2019, ApJ, 880, 25
Boylan-Kolchin M., Springel V., White S. D. M., Jenkins A., Lemson G., 2009, MNRAS, 398, 1150
Brown M. J. I. et al., 2008, ApJ, 682, 937
Chisholm J. et al., 2018, A&A, 616, A30
Choudhury T. R., Ferrara A., 2006, preprint (arXiV:astro-ph/0603149)
Colless M. et al., 2001, MNRAS, 328, 1039
Conroy C., Wechsler R. H., 2009, ApJ, 696, 620
Conroy C., Wechsler R. H., Kravtsov A. V., 2006, ApJ, 647, 201
Cooray A., Sheth R., 2002, Phys. Rep., 372, 1
de la Torre S., Peacock J. A., 2013, MNRAS, 435, 743
Diemer B., 2017, ApJS, 231, 5
Driver S. P. et al., 2011, MNRAS, 413, 971
Dunlop J. S. et al., 2013, MNRAS, 432, 3520
Fontanot F., Villaescusa-Navarro F., Bianchi D., Viel M., 2015, MNRAS, 447, 3361
Foreman-Mackey D., et al., 2013, PASP, 125, 306
Furlanetto S. R., Zaldarriaga M., Hernquist L., 2004, ApJ, 613, 1
Galassi M., Davies J., Theiller J., Gough B., Jungman G., 2009, GNU Scientific Library Reference Manual. 3rd edn., Network Theory Ltd
González-Nuevo J. et al., 2017, J. Cosmol. Astropart. Phys., 2017, 024
Grogin N. A. et al., 2011, ApJS, 197, 35
Guo Q., White S., Li C., Boylan-Kolchin M., 2010, MNRAS, 404, 1111
Guo H. et al., 2016, MNRAS, 459, 3040
Hadzhiyska B., Bose S., Eisenstein D., Hernquist L., Spergel D. N., 2020, MNRAS, 493, 5506
Hamilton A. J. S., 2000, MNRAS, 312, 257
Harikane Y. et al., 2016, ApJ, 821, 123

He S., Li Y., Feng Y., Ho S., Ravanbakhsh S., Chen W., Póczos B., 2019, Proc. Natl. Acad. Sci., 116, 13825
Hearin A. P. et al., 2017, AJ, 154, 190
Imara N., Loeb A., Johnson B. D., Conroy C., Behroozi P., 2018, ApJ, 854, 36
Jasche J., Lavaux G., 2019, A&A, 625, A64
Kitaura F.-S., Ata M., Rodriguez-Torres S. A., Hernandez-Sanchez M., Balaguera-Antolinez A., Yepes G., 2019, preprint (arXiv:1911.00284)
Klypin A. A., Trujillo-Gomez S., Primack J., 2011, ApJ, 740, 102
Klypin A., Yepes G., Gottl'ober S., Prada F., Heß S., 2016, MNRAS, 457, 4340
Kulkarni G., Keating L. C., Haehnelt M. G., Bosman S. E. I., Puchwein E., Chardin J., Aubert D., 2019, MNRAS, 485, L24
Landy S. D., Szalay A. S., 1993, ApJ, 412, 64
Lapi A., Mancuso C., Celotti A., Danese L., 2017, ApJ, 835, 37
Leauthaud A. et al., 2012, ApJ, 746, 95
Leclercq F., Jasche J., Wandelt B., 2015, J. Cosmol. Astropart. Phys., 2015, 015
Levi M. et al., 2013, preprint (arXiv:1308.0847)
Lewis A., 2008, Phys. Rev. D, 78, 023002
Lilly S. J. et al., 2007, ApJS, 172, 70
Limber D. N., 1953, ApJ, 117, 134
Maio U., Viel M., 2015, MNRAS, 446, 2760
Mao J., Lapi A., Granato G. L., de Zotti G., Danese L., 2007, ApJ, 667, 655
Marulli F., Veropalumbo A., Moresco M., 2016, Astron. Comput., 14, 35
Matthee J., Sobral D., Gronke M., Paulino-Afonso A., Stefanon M., Röttgering H., 2018, A&A, 619, A136
McCracken H. J. et al., 2012, A&A, 544, A156
Mesinger A., Furlanetto S., Cen R., 2011, MNRAS, 411, 955
Miralda-Escude J., Ostriker J. P., 1990, ApJ, 350, 1
Mo H. J., White S. D. M., 1996, MNRAS, 282, 347
Monaco P., Efstathiou G., 1999, MNRAS, 308, 763
Monaco P., Theuns T., Taffoni G., 2002, MNRAS, 331, 587
Moster B. P., Somerville R. S., Maulbetsch C., van den Bosch F. C., Macciò A. V., Naab T., Oser L., 2010, ApJ, 710, 903
Moster B. P., Naab T., White S. D. M., 2013, MNRAS, 428, 3121
Moster B. P., Naab T., White S. D. M., 2018, MNRAS, 477, 1822
Naab T., Ostriker J. P., 2017, ARA&A, 55, 59
Nasirudin A., Iliev I. T., Ahn K., 2020, MNRAS, 494, 3294
Nuza S. E., Kitaura F.-S., Heß S., Libeskind N. I., Müller V., 2014, MNRAS, 445, 988
Peacock J. A., Smith R. E., 2000, MNRAS, 318, 1144
Planck Collaboration VI, 2018, preprint (arXiv:1807.06209)
Popping G., Behroozi P. S., Peeples M. S., 2015, MNRAS, 449, 477
Robertson B. E., Ellis R. S., Furlanetto S. R., Dunlop J. S., 2015, ApJ, 802, L19
Rodríguez-Puebla A., Behroozi P., Primack J., Klypin A., Lee C., Hellinger D., 2016, MNRAS, 462, 893
Rodríguez-Puebla A., Primack J. R., Avila-Reese V., Faber S. M., 2017, MNRAS, 470, 651
Roy A., Lapi A., Spergel D., Baccigalupi C., 2018, J. Cosmol. Astropart. Phys., 2018, 014
Seljak U., 2000, MNRAS, 318, 203
Shapiro P. R., Giroux M. L., 1987, ApJ, 321, L107
Sinha M., Berlind A. A., McBride C. K., Scoccimarro R., Piscionere J. A., Wibking B. D., 2018, MNRAS, 478, 1042
Somerville R. S., Davé R., 2015, ARA&A, 53, 51
Somerville R. S. et al., 2018, MNRAS, 473, 2714
Spinelli M., Zoldan A., De Lucia G., Xie L., Viel M., 2019, MNRAS, 493, 5434
Springel V., 2005, MNRAS, 364, 1105
Springel V., White S. D. M., Tormen G., Kauffmann G., 2001, MNRAS, 328, 726
Springel V. et al., 2005, Nature, 435, 629
Steidel C. C., Pettini M., Adelberger K. L., 2001, ApJ, 546, 665
Steidel C. C., Bogosavljević M., Shapley A. E., Reddy N. A., Rudie G. C., Pettini M., Trainor R. F., Strom A. L., 2018, ApJ, 869, 123
Tassev S., Zaldarriaga M., Eisenstein D. J., 2013, J. Cosmol. Astropart. Phys., 2013, 036
Tinker J. L., Weinberg D. H., Zheng Z., Zehavi I., 2005, ApJ, 631, 41
Trujillo-Gomez S., Klypin A., Primack J., Romanowsky A. J., 2011, ApJ, 742, 16
Vale A., Ostriker J. P., 2004, MNRAS, 353, 189
Viel M., Haehnelt M. G., Springel V., 2004, MNRAS, 354, 684
Wang X., Hu W., 2006, ApJ, 643, 585
Wang L., Li C., Kauffmann G., De Lucia G., 2006, MNRAS, 371, 537
Wang L., Li C., Kauffmann G., De Lucia G., 2007, MNRAS, 377, 1419
Wechsler R. H., Tinker J. L., 2018, ARA&A, 56, 435
Wechsler R. H., Gross M. A. K., Primack J. R., Blumenthal G. R., Dekel A., 1998, ApJ, 506, 19
White M., 2001, MNRAS, 321, 1
Wise J. H., 2019, Contemporary Physics, 60, 145
Yang X., Mo H. J., van den Bosch F. C., 2003, MNRAS, 339, 1057
Yang X., Mo H. J., van den Bosch F. C., Zhang Y., Han J., 2012, ApJ, 752, 41
Yip J. H. T. et al., 2019, preprint (arXiv:1910.07813)
York D. G. et al., 2000, AJ, 120, 1579
Zahn O., Lidz A., McQuinn M., Dutta S., Hernquist L., Zaldarriaga M., Furlanetto S. R., 2007, ApJ, 654, 12
Zehavi I. et al., 2004, ApJ, 608, 16
Zheng Z. et al., 2005, ApJ, 633, 791
Zheng Z., Coil A. L., Zehavi I., 2007, ApJ, 667, 760
Zheng Z., Zehavi I., Eisenstein D. J., Weinberg D. H., Jing Y. P., 2009, ApJ, 707, 554
Zhu H., Avestruz C., Gnedin N. Y., 2020, ApJ, 899, 137

APPENDIX A: API STRUCTURE AND BUSHIDO

In the context of software development for scientific usage and, in general, whenever the development is intended for the use in Academia, the crucial aspects that would make the usage flexible are often overlooked.

In the development of SCAMPY, we have considered the good practices in software development, such as cross-platform testing and the production of reasonable documentation for the components of the API. We have outlined a strategy for keeping the software ordered and easy to read while maintaining efficient the computation. The usage of advanced programming techniques, along with the design of a handy class dedicated to interpolation, also allowed to boost the performances of our code.

In this Appendix, we describe the framework we have developed, highlighting the best programming practices used, and commenting on the design choices.

The overall structure can be divided broadly into four main components:

(i) **C++ core** – it mainly deals with the most computationally expensive sections of the algorithm.
(ii) **PYTHON interface** – it provides the user interface and implements sections of the algorithm that do not need to be severely optimized.
(iii) **Tests**, divided into **unit tests** and **integration tests**, are used for validation and consistency during code development.
(iv) **Documentation** provides the user with accessible information on the library's functionalities.

The organization of the source code is modular. Test and documentation sections are treated internally as modules of the library, and their development is, to some extent, independent to the rest of the API. Furthermore, not being essential for the API operation, their build is optional.

The Meson Build System deals with compilation and installation of the library. Much like the well-known CMake (reference website), it allows to ease the compilation and favours portability while automatizing the research and eventual download of external dependencies.

A1 Modularization

The C++ and PYTHON implementations are treated separately and have different modularization strategies. As we already anticipated, the C++ language is adopted to exploit the performances of a compiled language. None the less, it also allows for multithreading parallelization on shared memory architectures. This would not be normally possible in standard PYTHON because of the Global Interpreter Lock, which limits the processor to execute exactly only one thread at a time.

Each logical piece of the algorithm (see Section 3.1 and Fig. 1) has been implemented in a different module. This division has been maintained both in the core C++ implementation and in the PYTHON interface. Bridging over the two languages has been obtained through the implementation of source C++ code with a C-style interface enclosed in an `extern 'C'` scope to produce shared libraries with C-style mangling. To wrap the compiled C++ libraries in PYTHON we use the CTypes module. This choice was made because CTypes is part of the PYTHON standard. Therefore no external libraries or packages are needed. This choice favours portability and eases compilation.

All of the C++ modules are organized in different sub-directories with similar structure:

(i) `src` sub-directory, containing all the source files (`.cpp` extension);
(ii) `include` sub-directory, containing all the header files (`.h` extension);
(iii) a `meson.build` script for building.

All the PYTHON implementation is hosted in a dedicated sub-directory of the repository. Each module of the PYTHON interface to the API is coded in a separate file. The PYTHON dependencies to the C++ implementation are included in the source files at compile time by the build system.

In Table A1, we list all the Python-modules provided to the user. They are divided between the C++ wrapped and the PYTHON only ones. All of them are part of the `scampy` package that users can import by adding a

```
/path/to/install_directory/python
```

to their PYTHONPATH.

A2 External dependencies

Scientific codes often severely depend on external libraries. Even though a golden rule when programming, especially with a HPC intent, is to *not reinvent the wheel*, external dependencies have to be treated carefully. If the purpose of the programmer is to provide their software with a wide range of functionalities, while adopting external software where possible, the implementation can quickly become a *dependency hell*.

For this reason, we decided to keep the dependence on external libraries to a reasonable minimum. The leitmotiv being, trying not to be stuck on bottlenecks requiring us to import external libraries while maintaining the implementation open to the usage along with the most common scientific software used in our field.

Table A1. PYTHON modules of the API. The first column lists the module names and the second provides a short description of the module purpose. We divided the table in two blocks, separating the modules of the package that depend on the C++ implementation from those that have a pure PYTHON implementation.

Module	Purpose
	Wrapped from C-interface
`interpolator`	Templated classes and functions for cubic-spline interpolation in linear and logarithmic space.
`cosmology`	Provides the interface and an implementation for cosmological computations that span from cosmographic to Power-Spectrum dependent functions, computations are boosted with interpolation.
`halo_model`	Provides classes for computing the halo-model derivation of non-linear cosmological statistics.
`occupation_p`	Provides the occupation probability functions implementation.
	Python-only
`objects`	Defines the objects that can be stored in the class `catalogue` of the `scampy` package, namely `host_halo`, `halo`, and `galaxy`.
`gadget_file`	Contains a class for reading the halo/sub-halo hierarchy from the outputs of the SUBFIND algorithm of GADGET.
`catalogue`	It provides a class for organizing a collection of host-haloes into an hierarchy of central and satellite haloes. It also provides functionalities for automatic reading of input files and to populate the DM haloes with objects of type `galaxy`.
`abundance_matching`	Contains routines used for running the SHAM algorithm.

The C++ section of the API depends on the following external libraries:

(i) **GNU scientific library** (version 2 or greater Galassi et al. 2009): this library is widely used in the community and compiled binary packages are almost always available in HPC platforms.
(ii) **FFTLog** (Hamilton 2000): also this library is a must in the cosmology community. In our API, we provide a C++ wrap of the functions written in Fortran90. We have developed a patch for the original implementation that allows to compile the project with Meson (see fftlog_patch on GitHub for details).
(iii) **OpenMP**: one of the most common APIs for multithreading in shared memory architectures. It is already implemented in all the most common compilers, thus it does not burden on the user to include this dependency.

We are aware that a vast collection of libraries for cosmological calculations is already available to the community (Marulli, Veropalumbo & Moresco 2016; Astropy Collaboration 2013, 2018). The intent of our `cosmology` module is not to substitute any of these but to provide an optimized set of functions integrated in the API without adding a further dependence on external libraries. By using polymorphism (both static and dynamic) we tried to keep our API as much flexible as possible. We explicitly decided to not force the dependence to any specific Boltzmann-solver to obtain the linear power spectrum of matter perturbations (see Section 2.1), the choice is left to the user.

Furthermore, the choice of PYTHON to build the user interface, allowed to easily implement functions that do not require any other specific library to work. An example is the `abundance_matching` module, which is almost completely independent to the rest of the API: the only other internal module needed is the `scampy.object` but all its functionalities can be obtained by using PYTHON lambdas and NUMPY arrays.

The only other PYTHON libraries used in ScamPy are:

(i) **CTypes** which is part of the PYTHON standard and is used for connecting the C-style binaries to the PYTHON interface.

(ii) **Numpy** that, despite not being part of the standard, is possibly the most common PYTHON library on Earth and provides a large number of highly optimized functions and classes for array manipulation and numerical calculations.

A3 Extensibility

Simplifying the addition of new features has been one of our objectives from the first phases of development. We wanted to be able to expand the functionalities of the API, both on the C++ side, in order to boost the performances, and on the PYTHON side, in order to use the API for a wide range of cosmological applications.

This is easily achieved with the modular structure we have built up. Adding a new C++ module reduces to including a new set of headers and source files in a dedicated sub-directory. Further details on the structure said sub-directory should have and on the way its build is integrated in the API will be provided in the library website.

Adding new modules to the PYTHON interface is even simpler, as it only requires to add a new dedicated file in the `python/scampy` sub-directory. Eventually, it can be also appended to the `__all__` list in the `python/scampy/__init__.py` file of the package. In this case, it is not necessary to operate on the build system as it will automatically install the new module along with the already existing ones.

APPENDIX B: PERFORMANCES AND BENCHMARKING

We have measured the performances of our API's main components and benchmarked the scaling and efficiencies of the computation at varying precision and work-load. We will show here a set of time measurements performed on the two main components of the library: the `cosmology` class and the `halo_model` class. These are the two classes that would most affect the performances in real-life applications of our API.

B1 Wrapping benchmark

First of all, in Table B1, we show the execution time of the same function called from different languages. Since our hybrid implementation requires to bridge through C to wrap in PYTHON the optimizations obtained in C++, it is interesting to compare their respective execution time. Along with the PYTHON execution time, we also provide the ratio with respect to the reference C++ time, t_{C++}, for the same function. All the times are expressed in nanoseconds.

We are showing three typical member calls that are representative of the functionalities provided by the two classes. For both of them, we measured the constructor time (c.tor), the time for executing a function that returns a scalar ($d_C(z)$ and $n_g(z)$) and the execution time for a function returning an array ($n(M, z)$ and $\xi(r, z)$). It can be noticed that, especially for the `cosmology` class, by calling the same function in PYTHON, the execution time increases. The worst case is the `cosmology` class constructor time that looses a factor

Table B1. Execution time in nanoseconds of the same function in different languages. For the PYTHON case, we also show the ratio with respect to the C++ execution time. The timings reported are the average of 10 runs on the 4 physical cores with hyper-threading disabled of a laptop with Intel® Core™i7-7700HQ 2.80GHz CPU.

Function	C++	PYTHON	t_{py}/t_{C++}
	`cosmology` class		
c.tor	2.520e + 05	8.892e + 05	3.528
$d_C(z)$	2.083e + 03	5.984e + 03	2.873
$n(M, z)$	1.186e + 08	1.197e + 08	1.009
	`halo_model` class		
c.tor	3.011e + 09	2.961e + 09	0.983
$n_g(z)$	1.917e + 04	2.851e + 04	1.488
$\xi(r, z)$	3.396e + 06	1.784e + 06	0.525

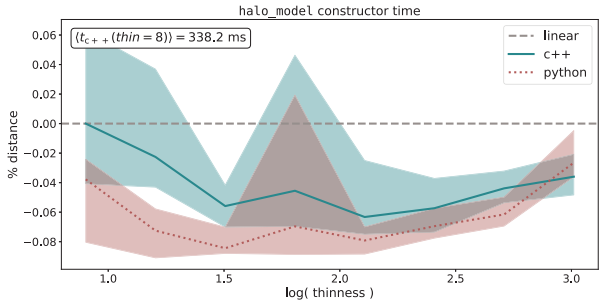

Figure B1. Percent distance between the constructor time scaling at varying thinness and the linear scaling case. For the PYTHON case, the percentage is computed with respect to the C++ time to ease the comparison. For reference, we also show in the white text-box the measured constructor time with thinness = 8.

∼3.5 in PYTHON. It has to be noticed though, that the execution time is lower than a millisecond and, since the constructor is the member function that is called the less, this is not severely affecting the overall performance of the PYTHON interface.

None the less, because of the larger number of function calls required by moving from one language to another, losing some performance is expected. What we did not expect is the gain in performance we are getting when moving to PYTHON, as it is shown in the last column of the `halo_model` class box of Table B1. This behaviour might be due to the different way memory is allocated, accessed and copied in PYTHON with respect to C++/C. Moreover, the timers used for measuring the execution in the different languages are different. Even by comparing measures taken with the same precision, it is not guaranteed to have the same accuracy.

B2 Halo-model performances

We have then tested the execution time of the `halo_model` constructor and member functions at varying work-load. In our implementation, the `halo_model` class requires to define a set of interpolating functions at construction time, these functions can be defined using our `interpolator` class (see Table A1). The interpolation accuracy depends on the resolution of the interpolation grid. In the `halo_model` class, at fixed limits of the interpolation interval, this is controlled by the *thinness* input parameter, which takes typical values 50/200 in real-life applications.

In Fig. B1, we show how the constructor-time varies with varying thinness in the range $10 < \text{thin} < 10^3$. The plot is obtained by calling 10 times the constructor per each thinness value and then averaging (solid and dotted lines). The shaded region marks the best and worst

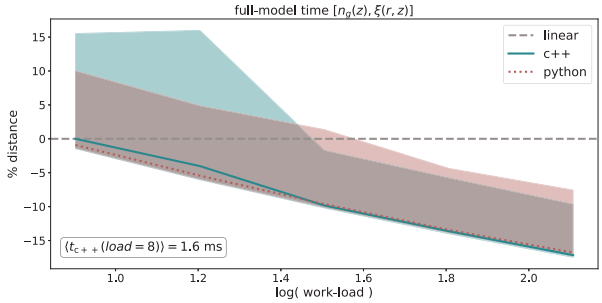

Figure B2. Percent distance between the full-model time scaling at varying work-load and the linear scaling case. For the PYTHON case, the percentage is computed with respect to the C++ time to ease the comparison. For reference, we also show in the white text-box the measured constructor time with work-load = 8.

execution time among the 10 runs. Instead of the actual execution time we show the per cent distance with respect to perfect linear scaling (dashed line) for both the C++ case (blue) and the PYTHON case (red). We define the per cent distance at given thinness as

$$\text{per cent distance (thin)} \equiv 100 \cdot \frac{t(\text{thin}) - t_{\text{lin}}(\text{thin})}{t_{\text{lin}}(\text{thin})} \tag{B1}$$

where t_{lin}(thin) is the execution time for given thinness in the linear scaling case, computed with respect to the C++ case. In the white text box of Fig. B1, we also show the C++ constructor time for thin = 8, as a reference. As the picture shows, the scaling is almost perfectly linear, with a maximum distance of the 0.06 per cent in the cC++ case.

Possibly the most crucial computational bottleneck of the whole API is the time taken by the computation of a full model. With the term 'full model' we mean the execution of the two functions for computing the halo-model estimate of the 1- and 2-point statistics, namely $n_{\text{g}}(z)$ and $\xi(r, z)$. In an MCMC framework, while the constructor is called only once, these two functions are called tens of thousands of times. This is a necessary step to set the parametrization of the SCAM algorithm.

As also shown in Table B1, the execution of the two single functions takes an amount of time which is in the order of the millisecond in the C++ case. We can also notice that the execution time of a full model is dominated by the computation of the two point correlation function, $\xi(r, z)$. Since this function is operating on a vector and returning a vector, it is reasonable to expect that its execution time varies with the work-load, i.e. with the vector size.

In Fig. B2, we show the per cent distance, defined as in equation (B1), of the average full-model execution time at varying work-load (solid lines) with respect to the perfect linear scaling case (dashed line), in the range $2^3 \leq$ load $\leq 2^{14}$. The measurements are obtained by averaging the results of 10 runs in both C++ (blue) and PYTHON (red). The shaded regions mark the best and the worst performance among all the runs at varying work-load. It can be noticed that, by increasing the work-load, the average execution time gets up to 15 per cent worse than perfect linear scaling. This is due to some latency introduced by the necessity of Fourier transforming the power spectrum to model clustering. We have to point out though, that the typical work-load is in the range 5/15 for real-life applications and that the execution time in this case is of the order of the millisecond.

Finally, we have measured how the constructor time-scales with increasing number of multithreading processors. We did not perform this measure for the full-model computation because, in the perspective of using it in an MCMC framework with parallel walkers, the full-model will be computed always serially.

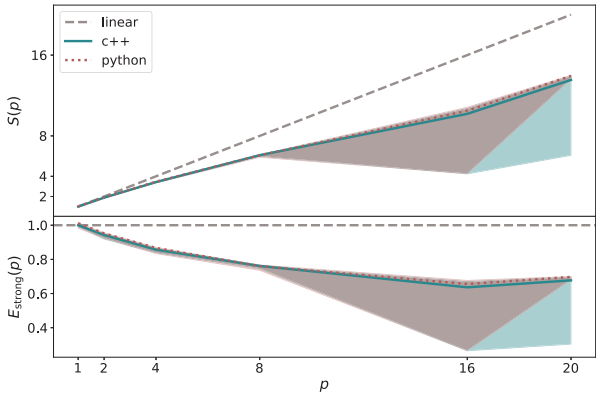

Figure B3. Strong-scaling speed-up (*upper panel*) and efficiency (*lower panel*) of the constructor time at fixed thinness and varying number of multithreading processors. The solid line marks the average of 100 runs while the shaded region marks the best and worst result area. We run the tests on a full computing-node of the SISSA Ulysses cluster.

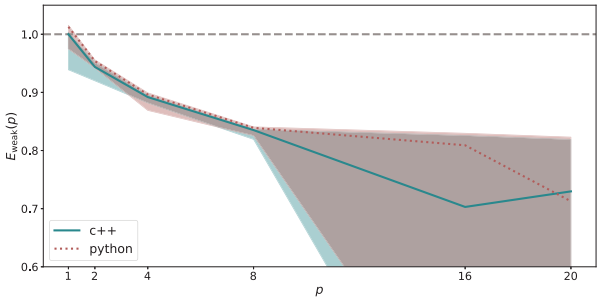

Figure B4. Weak-scaling efficiency of the constructor time at thinness growing proportionally to the number of multithreading processors. The solid line marks the average of 100 runs while the shaded region marks the best and worst result area. We run the tests on a full computing-node of the SISSA Ulysses cluster.

We present measurements of both the constructor time *strong scaling* and *weak scaling*. While the first measures the scaling with processor number at fixed thinness, the latter measures the scaling at thinness increasing proportionally with the processor number.

First of all, let us define the *speed-up*

$$S(p) = \frac{t(1)}{t(p)}, \tag{B2}$$

where p is the number of processors and $t(p)$ is the time elapsed running the code on p processors. This quantity measures the gain in performances one should expect when having access to larger parallel systems.

We also define the efficiency for the strong and the weak scaling case:

$$\begin{aligned} E_{\text{strong}}(p) &= \frac{S(p)}{p} \\ E_{\text{weak}}(p) &= S(p). \end{aligned} \tag{B3}$$

This quantity roughly measures the percentage of exploitation of the parallel system used. Thus, providing a hint of how much the serial part of the code is affecting the gain we can expect from spawning multiple threads.

We run these measures on a node from the `regular` partition of the SISSA Ulysses cluster.[4] Each of these nodes provide two shared memory sockets with 10 processors each. We measured the constructor time by averaging the results of 100 runs where the threads number has been controlled by setting

```
export OMP_NUM_THREADS = $ii
export OMP_PLACES = cores
export OMP_PROC_BIND = close
```

where `ii` varies in the set $\{1, 2, 4, 8, 16, 20\}$ and where the last two commands control the affinity of the processes spawned.

In Fig. B3, we show the speed-up (upper panel) and efficiency (lower panel) of the strong scaling. The dashed line marks perfect linear speed-up in the upper panel, and 100 per cent efficiency in the lower panel. Even though it is far from being perfect, the speed-up shows a constantly increasing trend. The efficiency seems to get constant around the 60 per cent for $p \geq 16$, but a larger parallel system would be necessary for getting a more precise measurement.

To conclude, in Fig. B4, we show the weak scaling efficiency case. The thinness, at given processors number p, is set to thin $= 50 \cdot p$. As the picture shows, the efficiency seems to become almost constant at $p \gtrsim 8$ for both the C++ and PYTHON case, with a value between 70 per cent and 80 per cent.

[4]Please refer to the website for detailed informations.

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