

ScamPy – A sub-halo clustering & 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 Sartori¹

¹SISSA, Via Bonomea 265, 34136 Trieste, Italy

²IFPU, Via Beirut 2, 34014 Trieste, Italy

³INFN-Sezione di Trieste, via Valerio 2, 34127 Trieste, Italy

⁴INAF-OATS, via Tiepolo 11, 34131 Trieste, Italy

Accepted XXX. Received YYY; in original form ZZZ

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.

Key words: methods: numerical – cosmology: theory, large scale structure of the universe, dark ages, reionization, first stars

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. 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 dark matter. Several possible approaches have been attempted so far to assess this modeling issue, which can be divided into two main categories. On one side, *ab initio* models, such as N-body simulations with a 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. 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 (Moster et al. 2018; Behroozi et al. 2019; Zhu et al. 2020). This latter class of methods is typically cheaper in terms of computational power and time required for running.

The former class of approaches is tuned to reproduce the LSS of the Universe at the present time, and therefore its 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.

★ E-mail: tronconi@sissa.it (SISSA)

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 with varying and adaptive parameterisation and complexity (Moster et al. 2018; Behroozi et al. 2019, and references therein). At the same time the number of observables that can be generated with such methods increased including galaxy luminosity, gas, metallicity and dust (e.g. Popping et al. 2015; Rodríguez-Puebla et al. 2016; Rodríguez-Puebla et al. 2017; Moster et al. 2018; Somerville et al. 2018; Imara et al. 2018). Empirical modelling offers several advantages with respect to *ab initio* modelling. It allows 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 modeling of the LSS observable properties (see, e.g. Nuza et al. 2014; Leclercq et al. 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 un-expected physics (see, e.g. Behroozi et al. 2012; Behroozi & Silk 2015).

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 Halo Occupation Distribution (HOD) for defining the galaxy-halo connection. This class of methods is widely used in the scientific community. However, according to the authors’ knowledge, the literature lacks a library developed explicitly for this task.

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 multi-threading for shared memory parallelisation, 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 an user-defined parameterisation. 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 substructures are identified using the SUBFIND algorithm (Springel et al. 2001). Nonetheless, thanks to its extensible design, adapting the API for working with simulations obtained by means of approximated methods, such as COLA (Tassev et al. 2013) or PINOCCHIO (Monaco et al. 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 Sub-halo Clustering and Abundance Matching 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 inter-galactic 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 & ABUNDANCE MATCHING

Our approach for the definition of the hosted-object/hosting-halo connection is based on the Sub-halo Clustering and Abundance Matching (SCAM) technique (Guo et al. 2016). With the standard HOD approach, hosted objects are associated to each halo employing a prescription which is based on the total halo mass, or on some other mass proxy (e.g. halo peak velocity, velocity dispersion). On the other side, Sub-halo abundance matching (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 prop-

erties 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 parameterised 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, $N_{\text{cen}}(M_h)$, and satellite galaxies, $N_{\text{sat}}(M_h)$, 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, $n_g(z)$:

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

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

(ii) Once we get the host halo/subhalo hierarchy with the abundance and clustering properties we want, as guaranteed by Eq. (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

It provides a halo-based description of nonlinear gravitational clustering and is used at both low and high redshift (Zehavi et al. 2005; Zheng et al. 2007; van den Bosch et al. 2013; More et al. 2015; Bullock et al. 2002; Hamana et al. 2004; Ouchi et al. 2005; Lee et al. 2006, 2009; Hildebrandt, H. et al. 2007; Bian et al. 2013; Harikane et al. 2016). The key assumption of this model is that the number of galaxies, N_g , in a given dark matter 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

$$\langle N_g \rangle (M_h) \propto M_h \quad (2)$$

$$\langle N_g(N_g - 1) \rangle (M_h) \propto M_h^2 \quad (3)$$

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 Eq. (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

$$\bar{\rho}(z) = \int M_h n(M_h, z) dM_h \quad (4)$$

where $n(M_h, z)$ is the halo mass function. With Eq. (2) we

can then define the average number of objects at redshift z , hosted in haloes with mass $M_{\text{min}} \leq M_h \leq M_{\text{max}}$, as

$$n_g(z) \equiv \int_{M_{\text{min}}}^{M_{\text{max}}} \langle N_g \rangle (M_h) n(M_h, z) dM_h. \quad (5)$$

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 $P(k, z)$, whose derivation can instead be treated with simple multiplications:

$$\xi(r, z) = \frac{1}{2\pi^2} \int_{k_{\text{min}}}^{k_{\text{max}}} dk k^2 P(k, z) \frac{\sin(kr)}{kr}. \quad (6)$$

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

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

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

$$\begin{aligned} \bar{u}(k, z|M_h) = & \frac{4\pi\rho_s r_s^3}{M_h} \left\{ \sin(kr_s) [\text{Si}((1+c)kr_s) - \text{Si}(kr_s)] \right. \\ & \left. - \frac{\sin(ckr_s)}{(1+c)kr_s} - \cos(kr_s) [\text{Ci}((1+c)kr_s) - \text{Ci}(kr_s)] \right\}, \end{aligned} \quad (8)$$

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

$$\text{Ci}(x) = \int_t^\infty \frac{\cos t}{t} dt \quad \text{and} \quad \text{Si}(x) = \int_0^x \frac{\sin t}{t} dt. \quad (9)$$

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

$$\begin{aligned} P_{1\text{h}}(k, z) & \equiv \int n(M_h, z) \left(\frac{M_h}{\bar{\rho}} \right)^2 |\bar{u}(k, z|M_h)|^2 dM_h = \\ & = \frac{1}{n_g^2(z)} \int_{M_{\text{min}}}^{M_{\text{max}}} \langle N_g(N_g - 1) \rangle (M_h) n(M_h, z) |\bar{u}(k, z|M_h)|^2 dM_h, \end{aligned} \quad (10)$$

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 Eqs. (3) and (5) to substitute the ratio $(M_h/\bar{\rho})^2$.

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. $b'_h(M'_h)b_h(M''_h)\xi_{\text{lin}}(r, z)$). The two terms in the convolution provide the product between the Fourier-space density profile $\tilde{u}(k, z|M_h)$, weighted by the total number density of objects within that particular halo (i.e. $n(M_h)(M_h/\bar{\rho})$). In Fourier space, we therefore have

$$\begin{aligned} P_{2h}(k, z) &\equiv \int n(M'_h) \frac{M'_h}{\bar{\rho}} \tilde{u}(k, z|M'_h) b(M'_h) dM'_h \\ &\quad \int n(M''_h) \frac{M''_h}{\bar{\rho}} \tilde{u}(k, z|M''_h) b(M''_h) P_{\text{lin}}(k, z) dM''_h = \\ &= \frac{P_{\text{lin}}(k, z)}{n_g^2(z)} \left[\int_{M_{\text{min}}}^{M_{\text{max}}} \langle N_g \rangle(M_h) n(M_h) b(M_h, z) \tilde{u}_h(k, z|M_h) dM_h \right]^2 \end{aligned} \quad (11)$$

with $b(M_h)$ the halo bias and $P_{\text{lin}}(k, z)$ 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 dM'_h and dM''_h are equivalent to the square of the integral in the rightmost expression. Secondly, we make use of Eqs. (2) and (5) to substitute the ratio $M_h/\bar{\rho}$.

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

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

where $N_{\text{cen}}(M_h)$ is the probability to have a central galaxy in a halo of mass M_h , while $N_{\text{sat}}(M_h)$ 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 $N_{\text{cen}}(M_h)$ and $N_{\text{sat}}(M_h)$, usually, they are parameterised. By tuning this parameterisation we obtain the prescription for defining the hosted-object/hosting-halo connection.

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

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

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):

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

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

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

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 Eq. (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 equivalence in Eq. (15).

Eq. (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

$$r_p(\theta, z) = \theta \cdot d_C(z). \quad (16)$$

By projecting the objects in our lightcone 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

$$\omega(\theta, z) = \int \frac{dV(z)}{dz} \mathcal{N}^2(z) \omega[r_p(\theta, z), z] dz \quad (17)$$

where $\frac{dV(z)}{dz}$ 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

$$\omega(\theta, z) \approx \left[\int_{z_1}^{z_2} dz \frac{dV(z)}{dz} \mathcal{N}^2(z) \right] \cdot \omega[r_p(\theta, \bar{z})] \quad (18)$$

where \bar{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/subhalo 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. 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 Figure 1, we give a schematic view of the main components of the ScamPy package. All the framework is centred around the occupation probabilities, namely $N_{\text{cen}}(M_h)$ and $N_{\text{sat}}(M_h)$, which define the average numbers of, respectively, central and satellite galaxies hosted within each halo. Several parameterisations of these two functional forms exist. One of the most widely used is the standard 5-parameters HOD model, 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:

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

$$N_{\text{sat}}(M_h) = \left(\frac{M_h - M_{\text{cut}}}{M_1} \right)^{\alpha_{\text{sat}}} \quad (20)$$

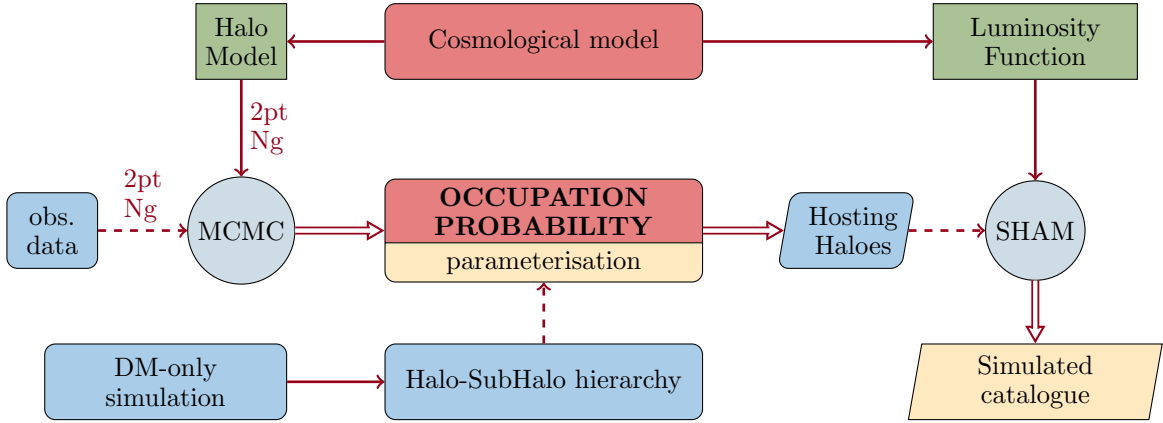


Figure 1. Flowchart describing the main components of the algorithm. In red the two main kernel modules. Green rectangles dub models from which the workflow depends. Round gray circles are for engines that operate on some inputs. Cyan is for inputs, yellow and parallelograms for outputs.

where M_{\min} is the characteristic minimum mass of halos that host central galaxies, $\sigma_{\log M_n}$ is the width of this transition, M_{cut} is the characteristic cut-off scale for hosting satellites, M_1 is a normalization factor, and α_{sat} is the power-law slope. Our API provides users with both an implementation of the Eqs. (19) and (20), and the possibility to use their own parameterisation by inheriting from a base `occupation_p` class.¹ Given that both the modeling 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 parameterisation 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.

Once the HOD parameterisation and the observable-property distribution have been set, it is possible to populate the halo/sub-halo hierarchy of a DM-only catalogue. In Alg. 1, we outline the steps required to populate a halo catalogue with mock observables. We start from a halo/subhalo hierarchy obtained by means of some algorithm (e.g. SUBFIND) that have been run on top of a DM-only simulation.

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 )
```

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.

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 et al. (2013), and SPARTA, Diemer (2017)) in future extensions of the library.

3.1.1 Populating algorithm

Input subhalo catalogues are trimmed into hosting subhalo 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

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
    pcen ← model.Ncen( halo.mass )

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

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

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

of $\langle N_{\text{cen}} \rangle(M_i)$ and $\langle N_{\text{sat}} \rangle(M_i)$. To account for the assumptions made in our derivation of 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: $N_{\text{cen}} = \mathcal{B}(1, \langle N_{\text{cen}} \rangle_i)$. While, in the case of satellite subhaloes, we extract a random Poisson variable $N_{\text{sat}} = \mathcal{P}(\langle N_{\text{sat}} \rangle_i)$, then we randomly select N_{sat} satellite subhaloes from those residing in the i^{th} halo.

In Fig. 2, we show a 4 Mpc/h thick slice of a simulation with box side length of 64 Mpc/h, the background colour code represents the density field traced by all the subhaloes found by the SUBFIND algorithm, smoothed with a Gaussian filter, while the markers show the positions of the subhaloes 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 subhaloes have been selected we can run the last step of our algorithm. The `abundance_matching()` function implements the SHAM prescription to associate to each subhalo 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

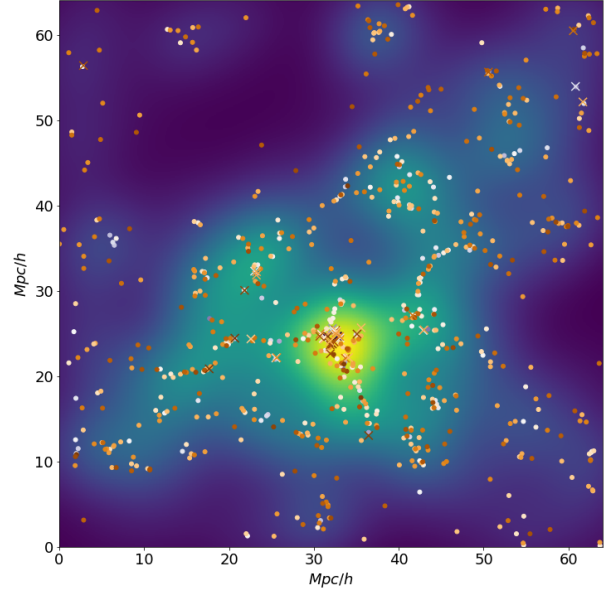


Figure 2. A 4 Mpc/h thick slice of a populated catalogue obtained from a DM-only simulation with 64 Mpc/h box side length. The colour code on the background shows the smoothed DM density field (with density increasing going from darker to brighter regions) while the markers show our mock galaxies (with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

subhaloes 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 subhalo, the UV luminosity of the galaxy it hosts. In the left panel, we show the cumulative mass-distribution of subhaloes, $dN(M_{\text{subhalo}})$, with the dashed green region being the mass resolution limit of the DM subhaloes in our simulation after the populating algorithm has been applied. On the right panel we show the cumulative UV luminosity function, which is given by the integral

$$\Phi(M^{\text{UV}} < M_{\text{lim}}^{\text{UV}}) = \int_{-\infty}^{M_{\text{lim}}^{\text{UV}}} \frac{d\Phi}{dM^{\text{UV}}} dM^{\text{UV}} \quad (21)$$

where $M_{\text{lim}}^{\text{UV}}$ 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 (gray step line in the left panel) and we compute the corresponding luminosity by inverting the cumulative luminosity function obtained with Eq. (21):

$$M^{\text{UV}}(M_{\text{subhalo}}) = \Phi^{-1}[dN(M_{\text{subhalo}})] \quad (22)$$

The result of this matching is shown by the orange crosses in the right 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 $dN(M_{\text{subhalo}})$. 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 Figure 2, the colour gradient of markers highlights the increase in their associated luminosity (from lighter to darker colour, going from fainter to brighter object).

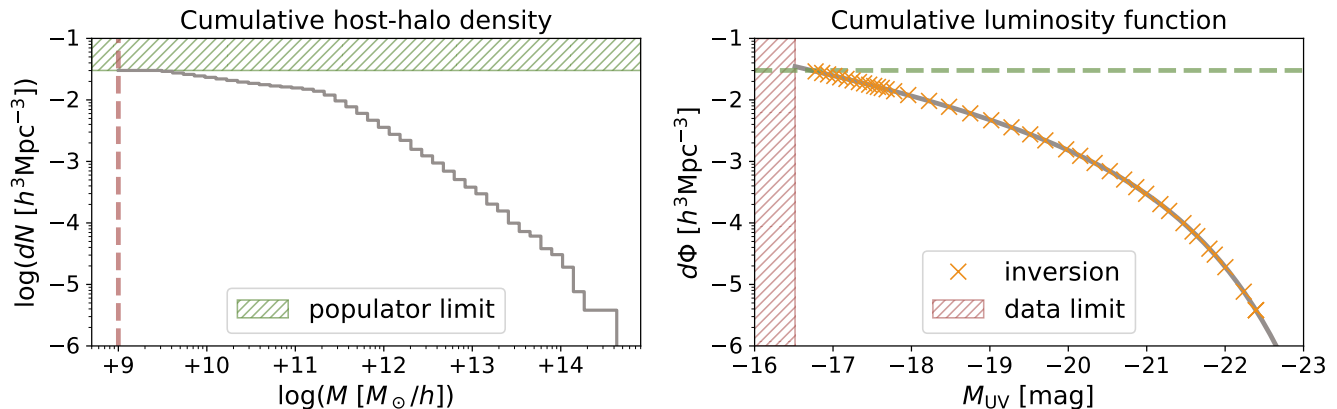


Figure 3. Abundance matching scheme. *Left panel:* cumulative number density of host subhaloes divided into regular logarithmic bins (solid gray step-line). The green dashed band shows the limit imposed by the resolution in mass of the simulation, which is inherited by the host catalogue obtained with the populator algorithm. *Right panel:* cumulative luminosity function (solid gray 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 panel. In both the left and the right 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 panel).

4 VERIFICATION & 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. For some benchmarking measures of the API performances we refer the reader to Appendix B.

All the validation tests have been obtained by assuming a set of reasonable values for the HOD parameters. The resulting occupation probabilities have been then used to populate a set of halo/subhalo catalogues. These catalogues have been obtained by running on the fly the 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 1 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 summarised in Table 2.

Table 1. Our set of cosmological simulations with the corresponding relevant physical quantities: N_{part} is the total number of DM particles; M_{part} is the mass of each particle; $L_{\text{box-side}}$ is the side length of the simulation box; z_{min} is the minimum redshift up to which the simulation has been evolved.

name	N_{part}	M_{part}	$L_{\text{box-side}}$	z_{min}
lowres	512^3	$8.13 \times 10^7 M_\odot/h$	64 Mpc/h	0
midres	1024^3	$1.02 \times 10^7 M_\odot/h$	64 Mpc/h	2
highres	1024^3	$1.27 \times 10^6 M_\odot/h$	32 Mpc/h	2

Table 2. Fiducial cosmological parameters of the N-body simulations used in this work.

h	Ω_{CDM}	Ω_b	Ω_Λ	σ_8	n_s
0.7	0.3	0.045	0.7	0.8	0.96

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 subhaloes, as a function of the halo mass. The dashed yellow step-lines show the distribution in the DM-only input catalogue, while the gray solid line marks the distribution defined by the 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.

We then draw a random Gaussian sample around the halo model estimate for the objects abundance and their

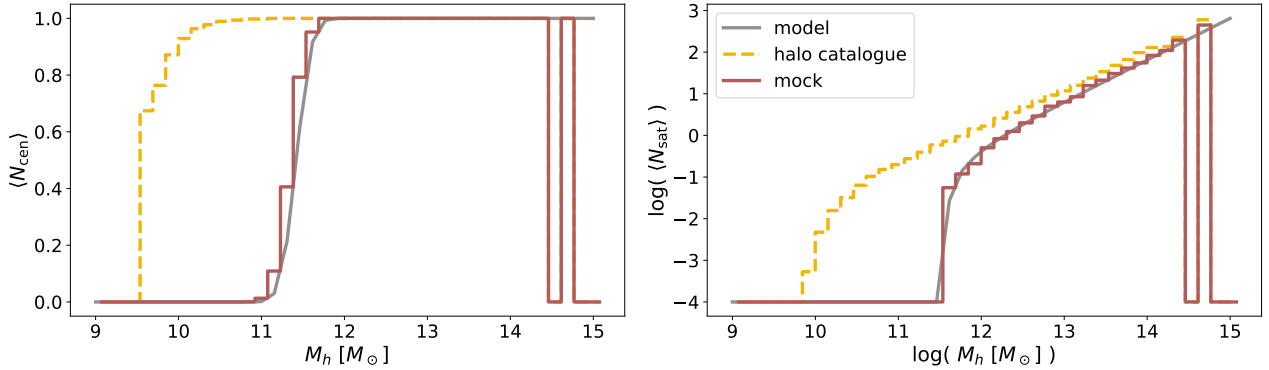


Figure 4. Occupation probability functions for central subhaloes (*left panel*) and satellite subhaloes (*right panel*). The gray solid line marks the model we want to reproduce. The yellow step-wise dashed line and the red step-wise solid line mark the distributions measured on the subhalo catalogue before and after having applied our populating algorithm.

clustering using the above selection of occupation probabilities (Eqs. (5) and (6) respectively). These random samples build up our *mock dataset*. We then run an MCMC sampling of the parameter space, with the likelihood of Eq. (1), to infer the set of parameters that best fit the mock dataset. 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-fit parameters, we produce 10 runs of the full pipeline described in Alg. 1. In doing this, we are producing 10 different realisations of the resulting mock catalogue. Since the selection of the host-subhaloes 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 Landy-Szalay (Landy & Szalay 1993) estimator in each of the populated catalogues to measure the 2-point correlation function:

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

where $DD(r)$ is the normalised number of unique pairs of subhaloes with separation r , $DR(r)$ is the normalised number of unique pairs between the populated catalogue and a mock sample of objects with random positions, and $RR(r)$ is the normalised number of unique pairs in the random objects catalogue. We then measure, with Eq. (23), the clustering in each of the 10 realisations 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 $z = 2, 4, 6, 8$. In the upper panel of Fig. 5 we show the mock dataset with triangle markers and errors, the lines show the halo model 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 distance of both the mock-dataset and of the average measurement, with respect to the inferred model. First of all, let us notice that the models inferred are in good agreement with the mock dataset. The accuracy of the measure with respect to the model, instead, decreases as redshift increases. At the largest scales we have a decrease in power of the measured 2-point correlation function. This

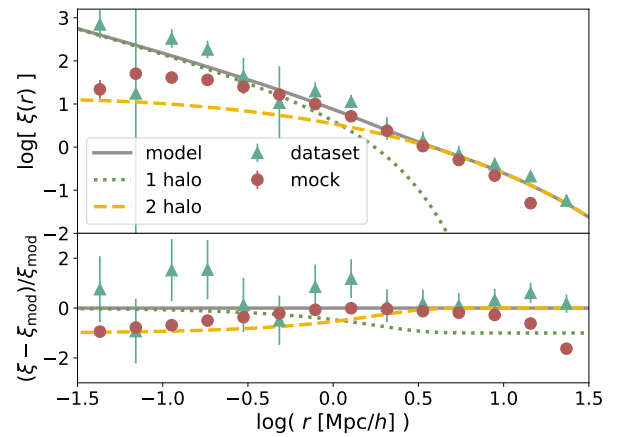


Figure 5. Validation of the two point correlation function at redshift $z = 0$. *Upper panel:* Comparison between the mock clustering dataset (triangles), the best-fit halo-model prediction (solid line) and the mean and standard deviation of the clustering measured with the Landy-Szalay estimator on the 10 realisations (circles and errors). *Lower panel:* distance ratio between the best-fit halo-model prediction and the mock-dataset/averaged-measurement. In both panels, we also show the modeled 1-halo (dotted line) and 2-halo (dashed line) terms for reference.

is 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 et al. 2019; Hadzhiyska et al. 2019, for two recent works), which is thought to result from a concurrence of box-size effects, cosmic-variance and assembly bias.

On the other hand, the limited spatial resolution of our simulation box is responsible for the discrepancies at the smaller scales. Since at redshift greater than $2 \div 3$ the number of sub-structures within haloes found by the SUBFIND algorithm becomes substantially smaller, the discrepancy becomes larger.

The discrepancies at small scales might be partially due to the transition between 1- and 2-halo term, as the bump in the measures seems to suggest. These discrepancies could be corrected by applying the same pipeline to an N-body simulation with higher resolution.

The requirement of reproducing the 1-point statistics

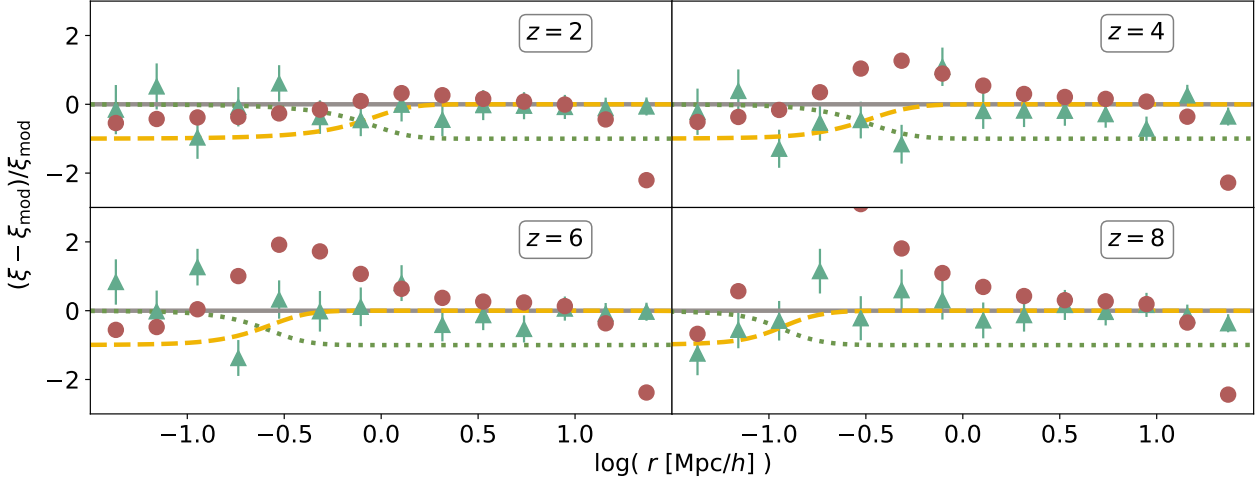


Figure 6. Same as lower panel of Fig. 6 for redshift $z = 2, 4, 6, 8$.

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. The halo-model prediction for the total abundance of sources is $n_g^{\text{hm}}(z) = 3.49 \cdot 10^{-2} [h^3 \text{Mpc}^{-3}]$, while we measure $n_g^{\text{pop}}(z) = (3.06 \pm 0.04) \cdot 10^{-2} [h^3 \text{Mpc}^{-3}]$ in the populated catalogue.

In Fig. 7, 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 (gray 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 $\approx 10\%$ over all the range of magnitudes.

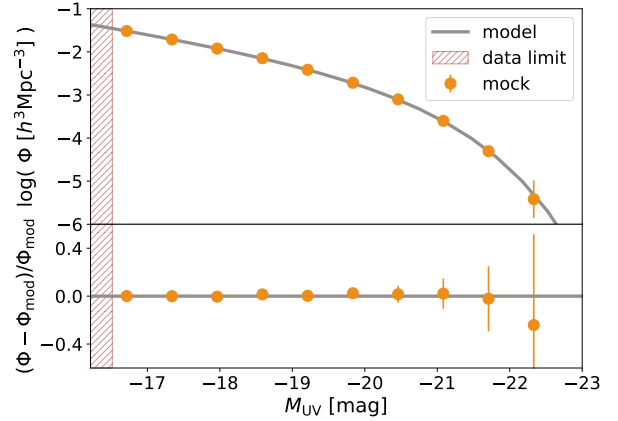


Figure 7. Cumulative luminosity function at redshift $z = 0$. *Upper panel:* the gray solid line marks the model prediction while the orange circles with errors mark the distribution measured on the populated catalogue. The hatched red region marks the limiting magnitude $M_{\text{lim}}^{\text{UV}}$.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

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 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 Eqs. (10) and (11) to get the expected power spectrum of the cross-correlation. For the 1-halo term this is achieved by splitting the $(M_h/\bar{\rho})^2$ of Eq. (10) in the contribution of the two different populations, which

leads to the following equation:

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

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 Eq. (11) in the contributions of the two different pop-

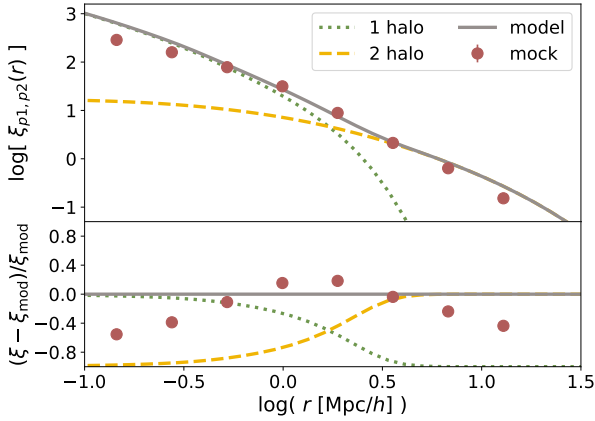


Figure 8. Same comparison as in Fig. 5 for the cross-correlation function, measured with the modified Landy-Szalay estimator of Eq. (26), between two dummy mock-populations at redshift $z = 0$.

ulations, leading to

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

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

$$\xi^{(1,2)}(r) = \frac{D_1 D_2(r) - D_1 R_2(r) - D_2 R_1(r) + R_1 R_2(r)}{R_1 R_2(r)} \quad (26)$$

where $D_1 D_2(r)$, $D_1 R_2(r)$, $D_2 R_1(r)$ and $R_1 R_2(r)$ are the normalized data1-data2, data1-random2, data2-random1 and random1-random2 pair counts for a given distance r .

In Fig. 8 the red circles show the cross-correlation measured with Eq. (26) for two different mock-populations with a dummy choice of the occupation probabilities' parameters. Errors are measured using a bootstrap scheme with 10 subsamples. For comparison, we also show the halo-model prediction of the two-point correlation function, obtained with Eqs. 24 and 25, separated in 1- 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% 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. 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. In order to map the spatial distribution of these ionized bubbles to the underlying dark matter 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. 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.

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 $10^8 M_\odot/h$. 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 1.

To set the occupation probabilities parameters with the likelihood in Eq. (1), we need the 1- and 2-point statistics of LBGs at high redshift. We can infer the abundance measure by integrating the luminosity function of this population of objects up to the luminosity of -13 mag. We use the most recent estimates for this statistics as presented in Bouwens et al. (2019). For the 2-point correlation function we use instead the measurements at redshift $4 \leq z \leq 7$ provided by Harikane et al. (2016).

Once the parameters of the model have been set for each redshift, we run the algorithm described in Sec. 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_{UV} . Namely, the rate of ionizing photons that escape from each UV source is

$$\dot{N}_{\text{ion}}(M_{UV}) \approx f_{\text{esc}} k_{\text{ion}} \text{SFR}(M_{UV}) \quad (27)$$

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

of each source. The volume of the Strömgren sphere, that forms around each mock-LBG, is then given by

$$V_S \equiv \frac{\dot{N}_{\text{ion}}(M_{\text{UV}})}{\bar{n}_H(z)} t_{\text{rec}} (1 - e^{-t/t_{\text{rec}}}) \quad (28)$$

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

We make the following simplistic assumptions:

- 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_{rec} is fixed at the cosmic time corresponding to $z = 20$.
- the escape fraction is set to $f_{\text{esc}} = 0.1$, 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 Figure 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. In Figure 9 we set the voxel-side to 0.25 Mpc/h, 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 Figure 10. We compare our measurement with models of reionization history from recent literature.

The gray 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 et al. (2004).

Our mock ionization boxes predict reionization to reach half-completion ($X_{\text{re}} = 0.5$) 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

Table 3. Best-fit parameters of the log-normal model defined in Eq. (29) obtained from our measures on the **highres** mock ionized bubble catalogue.

z	\bar{R} [Mpc/h]	$\sigma_{\ln r}^2$
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

Planck Collaboration VI (2018), the evolution in our simulations is way shallower, closer to the lower limit of the modified **tanh**-model. Nonetheless, 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. $X_{\text{re}} = 1$) at redshift $z \approx 4$. This issue at the lowest redshifts is by some extent expected, considering the first of the assumptions we listed above.

Taking into account the strong approximations made in this proof-of-concept application, the measurement we obtain for the evolution of X_{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 HII bubbles as well as from their growth in radius (Furlanetto et al. 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 21cm intensity mapping (e.g. Mesinger et al. 2011).

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

On the left 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 log-normal shape that we fit with the model from Roy et al. (2018)

$$P(R) = \frac{1}{R} \frac{1}{\sqrt{2\pi\sigma_{\ln r}^2}} \exp\left\{-\frac{[\ln(R/\bar{R})]^2}{2\sigma_{\ln r}^2}\right\}; \quad (29)$$

the model is regulated by two free parameters: the characteristic bubble size \bar{R} (in Mpc/h) and the standard deviation $\sigma_{\ln r}$. We list the best-fitting values of these parameters in Table 3, for the different redshifts considered. While the value of the characteristic radius is almost constant in time with a value of $\bar{R} \approx 0.2$ Mpc/h, the standard deviation increases significantly from higher to lower redshift.

On the right panel of Figure 11, we show the fraction of ionized voxels as a function of the bubble radius over the

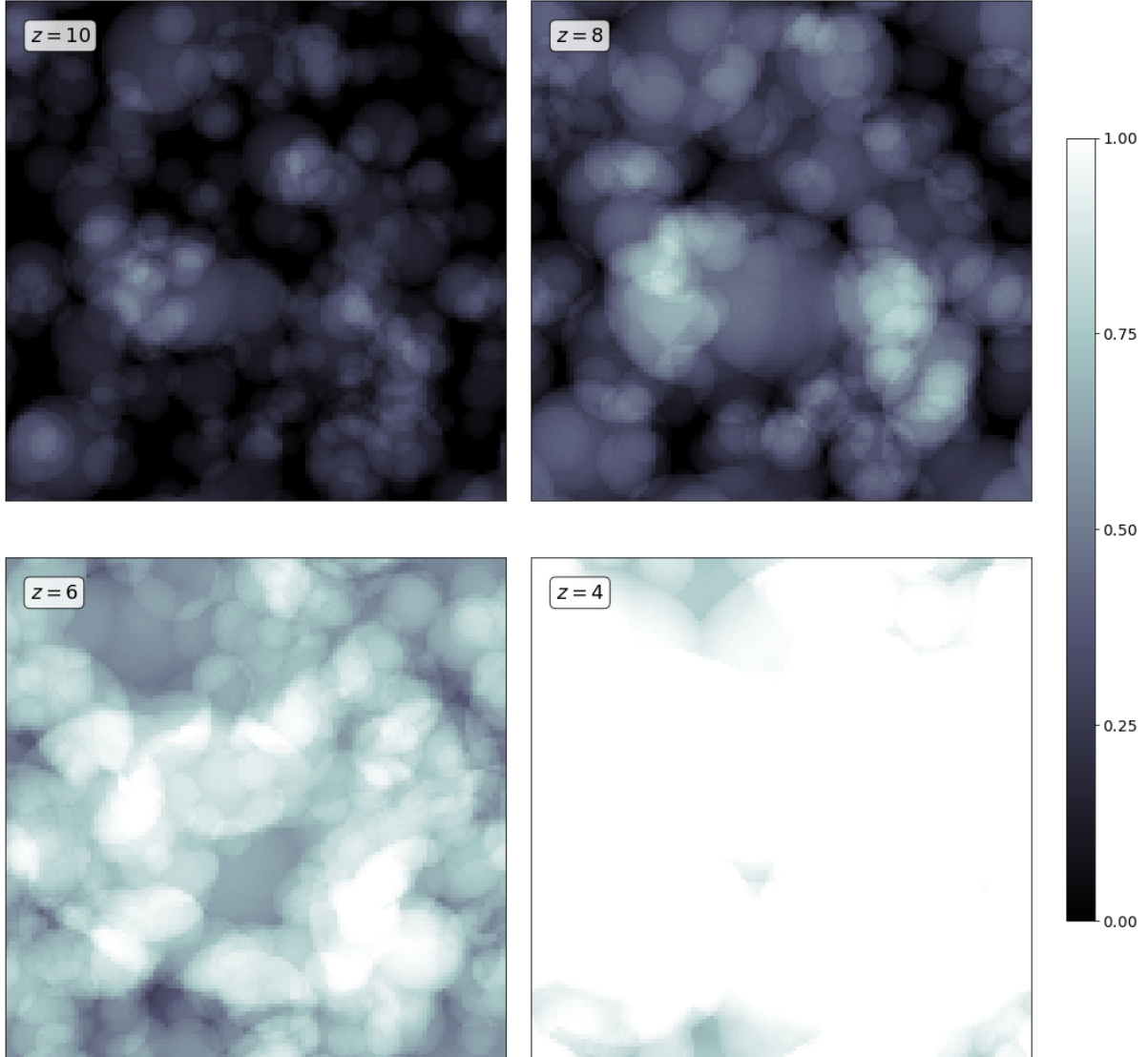


Figure 9. Four snapshots at different redshift (shown in the top left angle of each tile) of the ionization fraction obtained by projecting the values in each voxel along one dimension. The value of X_{re} increases from darker to lighter shades of gray.

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 panel, the measurements obtained for the bubble size probability definition of the right 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 $1 \div 10 \text{ Mpc}/h$. We could not fit the distributions on the right panel of Figure 11 with the same log-normal model of Eq. (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 & 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 pro-

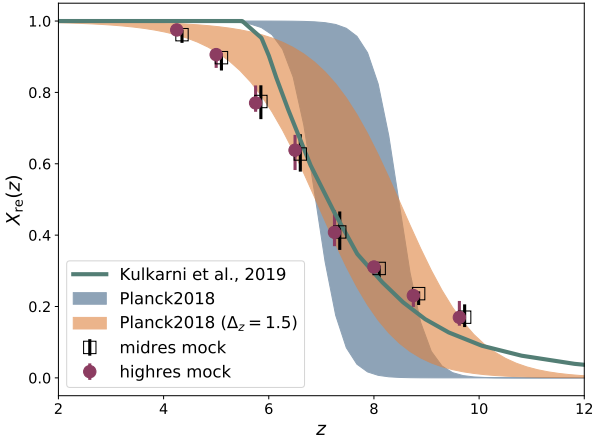


Figure 10. Evolution of the hydrogen ionization fraction X_{re} with redshift. Red circles and empty squares mark the measurements obtained with our method on the `midres` and `highres` simulation, respectively. The `midres` distribution has been shifted with an offset of $\delta z = 0.1$ along the x-axis 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. \(2019\)](#).

vided 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 modeled 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 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 optimisation of the neighbor search in both DM-halo catalogues and mock-galaxy catalogues. A possible list of the directions we are planning to pursue follows.

- *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 datasets are available. Another possibility would be the cross-correlation with intensity mapping like e.g. [Spinelli et al. \(2019\)](#).

- *Application to the reionization process* - Up to now we have mainly applied ScamPy functionalities to a proof-of-concept framework. Nonetheless, with the dataset 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.

- *Extension to different cosmological models* - By now all our investigations have been performed assuming a Λ CDM 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 dark matter scenarios or massive neutrino cosmologies [Maio & Viel \(2015\)](#); [Fontanot et al. \(2015\)](#).

- *Machine learning extension of the halo occupation model* - Using the halo occupation distribution (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.

ACKNOWLEDGEMENTS

T.R. 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’. A.L. acknowledges the MIUR grant ‘Finanziamento annuale individuale attività 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 n.2017-14-H.0.

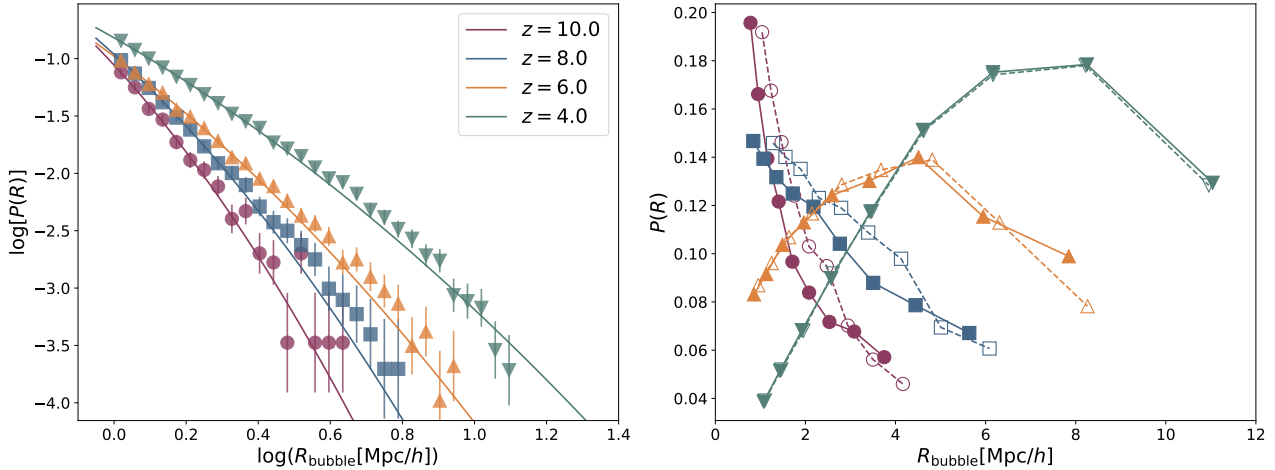


Figure 11. Bubble size probability distributions. *Left panel:* fraction of the bubbles with given size over the total number of bubbles, markers are measured from the *highres* simulation while the solid lines show the model of Eq. (29) fitted on these data (the best fitting parameters are listed in Table 3). *Right 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.

REFERENCES

- Amendola L., et al., 2018, *Living Reviews in Relativity*, 21, 2
- Astropy Collaboration et al., 2013, *A&A*, 558, A33
- Astropy Collaboration et al., 2018, *AJ*, 156, 123
- Aubert D., Teyssier R., 2008, *MNRAS*, 387, 295
- Aubert D., Teyssier R., 2010, *ApJ*, 724, 244
- Behroozi P. S., Silk J., 2015, *The Astrophysical Journal*, 799, 32
- Behroozi P. S., Wechsler R. H., Conroy C., 2012, *The Astrophysical Journal*, 762, L31
- Behroozi P. S., Wechsler R. H., Wu H.-Y., 2013, *ApJ*, 762, 109
- 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, arXiv e-prints, p. arXiv:1908.11448
- Bian F., et al., 2013, *The Astrophysical Journal*, 774, 28
- Bouwens R. J., Stefanon M., Oesch P. A., Illingworth G. D., Nanayakkara T., Roberts-Borsani G., Labbé I., Smit R., 2019, *ApJ*, 880, 25
- Bullock J. S., Wechsler R. H., Somerville R. S., 2002, *Monthly Notices of the Royal Astronomical Society*, 329, 246
- Chisholm J., et al., 2018, *A&A*, 616, A30
- Cooray A., Sheth R., 2002, *Phys. Rep.*, 372, 1
- Diemer B., 2017, *The Astrophysical Journal Supplement Series*, 231, 5
- 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., et al., 2009, GNU Scientific Library Reference Manual - Third Edition
- González-Nuevo J., et al., 2017, *J. Cosmology Astropart. Phys.*, 2017, 024
- Guo H., et al., 2016, *MNRAS*, 459, 3040
- Hadzhiyska B., Bose S., Eisenstein D., Hernquist L., Spergel D. N., 2019, arXiv e-prints, p. arXiv:1911.02610
- Hamana T., Ouchi M., Shimasaku K., Kayo I., Suto Y., 2004, *Monthly Notices of the Royal Astronomical Society*, 347, 813
- Hamilton A. J. S., 2000, *MNRAS*, 312, 257
- Harikane Y., et al., 2016, *ApJ*, 821, 123
- Hildebrandt, H. Pielorz, J. Erben, T. Schneider, P. Eifler, T. Simon, P. Dietrich, J. P. 2007, *A&A*, 462, 865
- Imara N., Loeb A., Johnson B. D., Conroy C., Behroozi P., 2018, *The Astrophysical Journal*, 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, arXiv e-prints, p. arXiv:1911.00284
- 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
- Leclercq F., Jasche J., Wandelt B., 2015,] 10.1088/1475-7516/2015/06/015, 2015, 015
- Lee K.-S., Giavalisco M., Gnedin O. Y., Somerville R. S., Ferguson H. C., Dickinson M., Ouchi M., 2006, *The Astrophysical Journal*, 642, 63
- Lee K.-S., Giavalisco M., Conroy C., Wechsler R. H., Ferguson H. C., Somerville R. S., Dickinson M. E., Urry C. M., 2009, *The Astrophysical Journal*, 695, 368
- Levi M., et al., 2013, arXiv e-prints, p. arXiv:1308.0847
- Lewis A., 2008, *Phys. Rev. D*, 78, 023002
- Limber D. N., 1953, *ApJ*, 117, 134
- Maior U., Viel M., 2015, *MNRAS*, 446, 2760
- Marulli F., Veropalumbo A., Moresco M., 2016, *Astronomy and Computing*, 14, 35
- Matthee J., Sobral D., Gronke M., Paulino-Afonso A., Stefanon M., Röttgering H., 2018, *A&A*, 619, A136
- Mesinger A., Furlanetto S., Cen R., 2011, *MNRAS*, 411, 955
- Monaco P., Efstathiou G., 1999, *Monthly Notices of the Royal Astronomical Society*, 308, 763
- Monaco P., Theuns T., Taffoni G., 2002, *MNRAS*, 331, 587
- More S., Miyatake H., Mandelbaum R., Takada M., Spergel D. N., Brownstein J. R., Schneider D. P., 2015, *The Astrophysical Journal*, 806, 2
- Moster B. P., Naab T., White S. D. M., 2018, *MNRAS*, 477, 1822
- Nuza S. E., Kitaura F.-S., Heß S., Libeskind N. I., Mäijller V., 2014, *Monthly Notices of the Royal Astronomical Society*, 445, 988
- Ouchi M., et al., 2005, *The Astrophysical Journal*, 635, L117
- Planck Collaboration VI 2018, arXiv e-prints, p. arXiv:1807.06209
- Popping G., Behroozi P. S., Peebles M. S., 2015, *Monthly Notices of the Royal Astronomical Society*, 449, 477

- Robertson B. E., Ellis R. S., Furlanetto S. R., Dunlop J. S., 2015, *ApJ*, **802**, L19
- Rodríguez-Puebla A., Primack J. R., Avila-Reese V., Faber S. M., 2017, *MNRAS*, **470**, 651
- Rodríguez-Puebla A., Behroozi P., Primack J., Klypin A., Lee C., Hellinger D., 2016, *Monthly Notices of the Royal Astronomical Society*, 462, 893
- Roy A., Lapi A., Spergel D., Baccigalupi C., 2018, *J. Cosmology Astropart. Phys.*, **2018**, 014
- Somerville R. S., et al., 2018, *MNRAS*, **473**, 2714
- Spinelli M., Zoldan A., De Lucia G., Xie L., Viel M., 2019, arXiv e-prints, p. [arXiv:1909.02242](https://arxiv.org/abs/1909.02242)
- Springel V., 2005, *MNRAS*, **364**, 1105
- Springel V., White S. D. M., Tormen G., Kauffmann G., 2001, *Monthly Notices of the Royal Astronomical Society*, 328, 726
- 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. Cosmology Astropart. Phys.*, **2013**, 036
- Viel M., Haehnelt M. G., Springel V., 2004, *Monthly Notices of the Royal Astronomical Society*, 354, 684
- Wang X., Hu W., 2006, *The Astrophysical Journal*, 643, 585
- Zahn O., Lidz A., McQuinn M., Dutta S., Hernquist L., Zaldarriaga M., Furlanetto S. R., 2007, *ApJ*, **654**, 12
- Zehavi I., et al., 2005, *The Astrophysical Journal*, 630, 1
- Zheng Z., Coil A. L., Zehavi I., 2007, *The Astrophysical Journal*, 667, 760
- Zhu H., Avestruz C., Gnedin N. Y., 2020, arXiv e-prints, p. [arXiv:2001.02233](https://arxiv.org/abs/2001.02233)
- van den Bosch F. C., More S., Cacciato M., Mo H., Yang X., 2013, *Monthly Notices of the Royal Astronomical Society*, 430, 725

APPENDIX A: API STRUCTURE & 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 4 main components:

- **c++ core** - it mainly deals with the most computationally expensive sections of the algorithm.
- **Python interface** - it provides the user interface and implements sections of the algorithm that do not need to be severely optimised.
- **Tests**, divided into **unit tests** and **integration tests**, are used for validation and consistency during code development.
- **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. Nonetheless, it also allows for multi-threading parallelisation 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 Figure 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:

- **src** sub-directory, containing all the source files (`.cpp` extension);
- **include** sub-directory, containing all the header files (`.h` extension);
- 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

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	
<code>interpolator</code>	Templated classes and functions for cubic-spline interpolation in linear and logarithmic space
<code>cosmology</code>	Provides the interface and an implementation for cosmological computations that span from cosmographic to Power-Spectrum dependent functions, computations are boosted with interpolation
<code>halo_model</code>	Provides classes for computing the halo-model derivation of non-linear cosmological statistics.
<code>occupation_p</code>	Provides the occupation probability functions implementation.
Python-only	
<code>objects</code>	Defines the objects that can be stored in the class <code>catalogue</code> of the <code>scampy</code> package, namely <code>host_halo</code> , <code>halo</code> and <code>galaxy</code> .
<code>gadget_file</code>	Contains a class for reading the halo/sub-halo hierarchy from the outputs of the SUBFIND algorithm of GaDGET.
<code>catalogue</code>	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 Dark Matter haloes with objects of type <code>galaxy</code> .
<code>abundance_matching</code>	Contains routines used for running the SHAM algorithm.

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.

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

- **GNU Scientific Library** (Galassi et al. 2009, version 2 or greater): this library is widely used in the community and compiled binary packages are almost always available in HPC platforms.
- **FFTL** (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).

- **OpenMP**: one of the most common APIs for multi-threading 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 et al. 2016; Astropy Collaboration et al. 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 in 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:

- **CTypes** which is part of the Python standard and is used for connecting the C-style binaries to the Python interface.
- **Numpy** which, 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.

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

APPENDIX B: PERFORMANCES & 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 Tab. 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 optimisations 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 3 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 loses a factor ~ 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.

Nonetheless, 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 Tab. 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

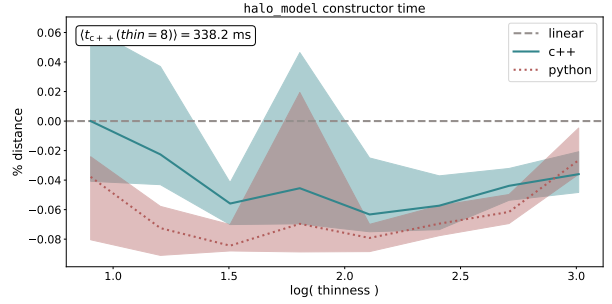


Figure B1. Percent distance between the constructor time scaling at varying thinness and the linear scaling case. For the python case, the percentage is computed with respect to the C++ time to ease the comparison. For reference, we also show in the white text-box the measured constructor time with thinness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

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 $50 \div 200$ in real-life applications.

In Fig. B1 we show how the constructor-time varies with varying thinness in the range $10 < thin < 10^3$. 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 execution time among the 10 runs. Instead of the actual execution time we show the percent distance with respect to perfect linear scaling (dashed line) for both the C++ case (blue) and the python case (red). We define the percent distance at given thinness as

$$\% \text{ distance}(thin) \equiv 100 \cdot \frac{t(thin) - t_{lin}(thin)}{t_{lin}(thin)} \quad (\text{B1})$$

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% in the C++ 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 $n_g(z)$ 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 parameterisation of the SCAM algorithm.

As also shown in Tab. B1, the execution of the two

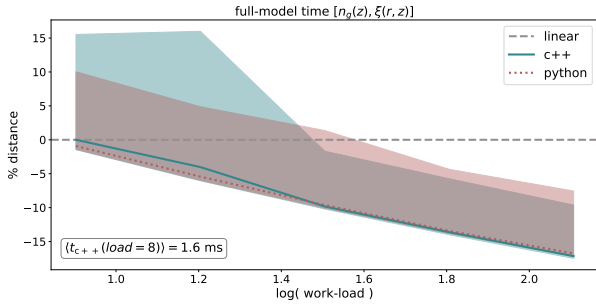


Figure B2. Percent distance between the full-model time scaling at varying work-load and the linear scaling case. For the python case, the percentage is computed with respect to the C++ time to ease the comparison. For reference, we also show in the white text-box the measured constructor time with work-load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

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 percent distance, defined as in Eq. (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 $2^3 \leq \text{load} \leq 2^{14}$. 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 worst performance among all the runs at varying workload. It can be noticed that, by increasing the work-load, the average execution time gets up to 15% 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 $5 \div 15$ for real-life applications and that the execution time in this cases is of the order of the millisecond.

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

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*

$$S(p) = \frac{t(1)}{t(p)} \quad (\text{B2})$$

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

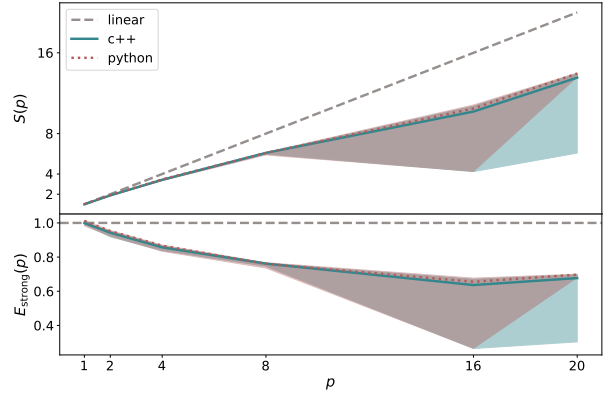


Figure B3. Strong-scaling speed-up (*upper panel*) and efficiency (*lower-panel*) of the constructor time at fixed thinness and varying number of multi-threading processors. The solid line marks the average of 100 runs while the shaded region marks the best and worst result area. We run the tests on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

scaling case:

$$E_{\text{strong}}(p) = \frac{S(p)}{p} \quad (\text{B3})$$

$$E_{\text{weak}}(p) = S(p)$$

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% 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% 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 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% and 80%.

This paper has been typeset from a $\text{\TeX}/\text{\LaTeX}$ file prepared by the author.

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

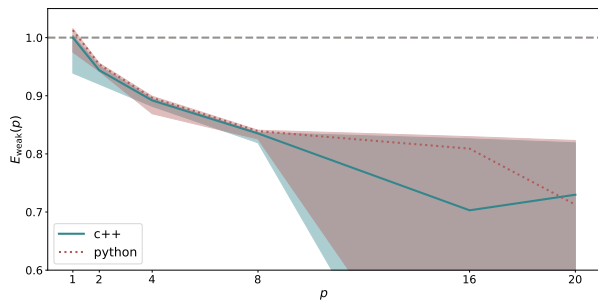


Figure B4. Weak-scaling efficiency of the constructor time at thinness growing proportionally to the number of multi-threading processors. The solid line marks the average of 100 runs while the shaded region marks the best and worst result area. We run the tests on a full computing-node of the SISSA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