Facultad de Ciencias

The variance of the nucleator: a simulation study
(Estudio de la varianza del nucleador mediante simulaciones Monte Carlo)

Trabajo de Fin de Máster
para acceder al

MÁSTER EN MATEMÁTICAS Y COMPUTACIÓN

Autor: Javier González Villa

Directores: Marcos Cruz Rodríguez
Domingo Gómez Pérez

Julio - 2015

Agradecimientos

Espero no dejarme a nadie en el tintero a la hora de nombrar a toda esa gente que me ha ayudado durante estos años, tras los cuales aquí me encuentro.

Primero he de agradecer a mi familia, pero en especial a mis padres Constantino Gonzalez y María Teresa Villa los cuales siempre han respaldado mis decisiones y me han impulsado a desarollar los conocimientos que más me atraen. También he de agradecer a mi tío y a mi abuela Triunfo González y María Luz González por prestarme su apoyo todos estos años. A mi novia Estela López por confiar en mí y tener la paciencia suficiente para aguantarme en todo momento.

Por otro lado, agradecer a mis amigos, tanto los antiguos como los que he ido haciendo a lo largo del master, los buenos momentos y las charlas distendidas con las que tanto he aprendido.

Por último y no por ello menos importante, he de agradecer a Marcos Cruz y a Domingo Gómez, tutores de mí tesis, y a Luis Manuel Cruz la posibilidad que me han brindado a la hora de realizarlo y por guiarme en todo momento.

A todos vosotros, muchas gracias.

Resumen

La estereología es la rama de la ciencia que, a través de la interpretación tridimensional de secciones, provee técnicas prácticas para extraer información cuantitativa sobre objetos tridimensionales. Por lo tanto, conocer de manera precisa el funcionamiento de esas técnicas, así como su precisión y comportamiento en diferentes situaciones es de gran interés.

En esta tesis analizamos el método nucleador, que proporciona una estimación insesgada del volumen de un objeto. Por otro lado también analizamos dos estimadores teóricos de la varianza de las estimaciones que genera el método nucleador. Realizamos el análisis con diferentes objetos simulados, con el propósito de concluir cuando el método es adecuado y cuando estos estimadores teóricos nos pueden ayudar a la hora de saber cuan fiable es la estimación.

La estimación de la varianza en muestreo sistemático es un problema complicado porque los elementos de la muestra son dependientes y generalmente no existe un estimador insesgado. Por tanto replicar el proceso un gran número de veces en objetos simulados parece ser la única alternativa para comprobar la precisión de los estimadores. Aquí simulamos tres objetos distintos y comprobamos que uno de los estimadores analizados puede resultar de cierta utilidad aunque hay un margen amplio de mejora.

Palabras Clave: Estereología, Monte Carlo, estimación de volumen, varianza en muestreo sistemático, nucleador, geometría computacional.

Abstract

Volume estimation is a classic stereological problem. There are several unbiased estimators, such as Cavalieri test planes and slabs, fakir test lines, or the nucleator, which is the one we analyse in this thesis. It is based on pseudosystematic sampling on the sphere.

Error variance estimation in systematic or pseudosystematic sampling is nontrivial problem since the observations are dependent in general. There are no variance estimators which are always unbiased. We study two analytical variance estimators and check their performance through Monte Carlo replications on simulated particles.

We simulate three different objects and conclude that one of the estimators can be useful in some cases but it can be largely improved.

Keywords: Stereology, Monte Carlo, volume estimation, variance under systematic sampling, nucleator, computational geometry.

Contents

List of Figures

List of Tables

1 Introduction

Design unbiased estimators of geometrical parameters are widely available in stereology for individual particles — for references see for instance Baddeley and Jensen Baddeley and Vedel Jensen (2004). There are several stereological estimators for number, length, area or volume of particles. We will focus on volume estimation.

There are different stereological methods to estimate volume such as for instance Cavalieri test planes and slabs, fakir test lines, or the nucleator, see Howard and Reed (2004). Here we study in detail the nucleator method (Gundersen (1988)).

The nucleator volume estimator is based on measuring distances between an arbitrary fixed origin and the surface of the particle using pseudosystematic sampling on the sphere. The estimator is unbiased and its variance has been analytically estimated (Gual-Arnau and Cruz-Orive (2002)). The variance predictor is a mathematical approximation, but not an unbiased estimator. Its performance depends on particle shape, and therefore no general statements can be made about its statistical properties: Monte Carlo resampling seems to be the only choice to check the statistical quality of the variance predictor.

Here we study the analytical variance estimators on three different simulated particles. In order to check for consistency the first particle is a Gaussian-like particle used in Gual-Arnau and Cruz-Orive (2002). The two other particles are simulated asteroid-like particles.

2 Background

2.1 Spherical coordinate system

The spherical coordinate system (Arfken and George B (2013),Chapter 2) is a coordinate system for three-dimensional space where a point is defined by a tuple (r, ϕ, θ) where r is the radial distance of that point from a fixed origin, θ is the polar angle measured from a fixed zenith (point on the sphere vertically above a given origin) direction and ϕ is the azimuth angle of its orthogonal projection on a reference plane that passes through the origin and is orthogonal to the zenith, measured from a fixed reference direction on that plane.

These angles ϕ and θ are necessary restricted to obtain a unique set of spherical coordinates for each point. These intervals are $0 \leq \phi < 2\pi$ and $0 \leq \theta \leq \pi$. Therefore, we denote a ray in the direction $u = (\phi, \theta)$ as a set of all points for any $0 \leq r < \infty$ in the defined direction. Hence, the antipode of a ray $u = (\phi, \theta)$ is defined as $(\phi + \pi, \pi - \theta)$.

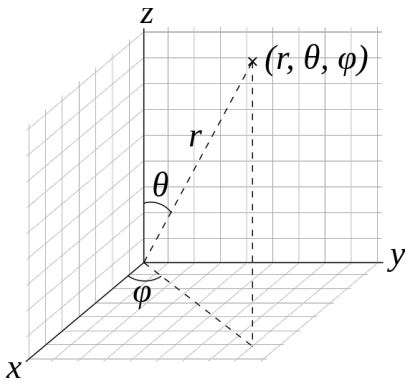

Figure 1: Representation of a point on the spherical coordinate system (Wikipedia (2015)).

As we can see in the Fig. 1, the chosen zenith corresponds with the Z axis in the Cartesian coordinate system and the reference plane that passes through the origin and is orthogonal to the zenith refers to the plane XY. This convention will be followed for the rest of the thesis.

2.2 Monte Carlo methods

Monte Carlo methods are a broad class of computational algorithms that apply techniques in which a large quantity of randomly generated numbers are studied using a probabilistic model to find an approximate solution to a numerical problem that would be difficult to solve by other methods, see Leobacher and Pillichshammer (2014).

These algorithms are complicated because they require a large number of simulations. However these algorithms produce approximations of the integral of functions $f : \mathbb{R} \to \mathbb{R}$ by an equal weight quadrature rule of the form:

$$Q_N(f) = \frac{1}{N} \sum_{n=0}^{N-1} f(x_n), \tag{1}$$

where N is the number of total experiments or realisations and $f(x_n)$ is the produced value by the function f for each quadrature point x_n. These realisations can be modelled as a set of independent and uniform distributed random variables for each x_n or as a set of systematic random samples with particular periods.

This method is unbiased because the expectation value is equal to the computed integral of the function f.

$$\mathbb{E}[Q_N(f)] = \frac{1}{N} \sum_{n=0}^{N-1} \mathbb{E}[f] = \mathbb{E}[f] = \int_{\mathbb{R}} f(x)dx. \tag{2}$$

Therefore we have a unbiased method and if we know the variance of the function f we can obtain the variance of the produced values by the Monte Carlo method, as follows:

$$\mathrm{Var}[Q_N(f)] = \frac{\mathrm{Var}[f]}{N}. \tag{3}$$

In conclusion, Monte Carlo methods are designed to solve problems, when deterministic algorithms are inefficient and solving exact mathematical models is very costly. If the probabilistic model is correctly selected and applied these algorithms provide a good estimation of the real solution.

2.3 Unbiased Stereology concepts

Understanding unbiased stereology as well as its application in our specific problem requires a previous definition of concepts, which refers to probability and geometry.

Focusing on geometric concepts, we define a particle as a 3D set of points, edges and planes, which conform a mesh. These particles can be convex or star convex. A particle is convex if for every pair of points within the particle, every point on the straight line segment that joins the pair of points is also in the particle. A particle is star convex if there exists a point in the particle such that the line segment from this point to any point in the particle is also contained. Therefore a convex particle is always star convex but a star convex particle is not always convex.

The measurement tools are the instruments to obtain quantitative properties of these particles (volumes, perimeters, areas and surfaces) in 3D space. In our case the measurement tools are isotropic lines. A line or ray is isotropic when its properties (ϕ and θ are uniform in any interval of the sphere) are the same in all directions.

Hence, we are going to use a systematic random sampling method, see 3.5, which starts from a random seed to generate a uniform distributed set of samples. This method, like random sampling, has drawbacks for exceptional cases. For example, random sampling methods can generate a set of spatially concentrated samples as well as systematic random sampling can produce samples with similar properties due to the periodicity and the object shape.

Finally, we must consider the bias of an estimator. This property is defined as the difference between the expected value of an estimator and the true value of the parameter being estimated. Hence, an unbiased method is defined as a method, with bias equal to zero. Consequently, the mean square error depends only on the variance value provided by any unbiased method.

3 Material & Methods

3.1 The nucleator

The object of interest is a fixed particle, namely a bounded, nonvoid subset $Y \subset \mathbb{R}^3$ with piecewise smooth boundary ∂Y. Our target parameter is the volume V of Y and the design adopted to estimate V is the nucleator Gundersen (1988). The basic idea underlying the nucleator (Cruz-Orive (1987), Appendix B), was based on a ray emanating from a fixed point $O \in \mathbb{R}^3$ in a direction $u \in \mathbb{S}^2$.

If such ray hits Y, then the corresponding intersection will in general consist of say $m(u) \geq 1$ separate intercept segments. The distances of the end points of these intercepts from the origin O, arranged in increasing order of magnitude, may be denoted as follows,

$$\{l_{i-}(u), l_{i+}(u);\ i = 1, 2, \ldots, m(u)\}. \tag{4}$$

Integration of the conic volume element associated with a ray leads to the nucleator representation of volume, namely,

$$V = \frac{1}{3} \int_{\mathbb{S}^2} \sum_{i=1}^{m(u)} (l_{i+}^3(u) - l_{i-}^3(u)) du. \tag{5}$$

where du is the area element on the unit hemisphere (Fig. 2). If $O \in Y$, then $l_{1-}(u) = 0$ for all u. If the particle Y is moreover star convex with respect to $O \in Y$, then

$$V = \frac{1}{3} \int_{\mathbb{S}^2} l_+^3(u) du, \tag{6}$$

where $l_+(u) = l_{1+}(u)$ is the intercept length determined by a ray u.

More generally, the problem is to estimate the integral Q of a function $f : \mathbb{S}^2 \to [0, +\infty)$ called the measurement function, which is a nonrandom function defined on the unit sphere and square integrable on it:

$$Q = \int_{\mathbb{S}^2} f du = \int_0^{2\pi} \int_0^{\pi} f(\phi, \theta) \sin\theta d\theta d\phi, \tag{7}$$

$$= \int_0^{2\pi} \int_{-1}^{1} f(\phi, \cos^{-1} y) dy d\phi. \tag{8}$$

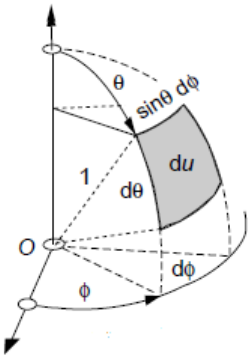

Figure 2: Representation of the area element du on the unit hemisphere.

To estimate Q we adopt a sampling design on the unit sphere involving n_1 systematic values of ϕ with period $T_1 = 2\pi/n_1$ and n_2 systematic values of $y = \cos(\theta)$ with period $T_2 = 2/n_2$; n_1. The $n_1 n_2$ sampling sites $\{(\phi_i, \theta_j)\}$ have the following spherical polar coordinates:

$$\begin{aligned} \phi_i &= z_1 + iT_1, \\ i &= 0, 1, .., n_1 - 1, \end{aligned} \tag{9}$$

$$\begin{aligned} y_j &= z_2 - jT_2, \\ j &= 0, 1, .., n_2 - 1. \end{aligned} \tag{10}$$

where $z = (z_1, z_2)$ is a pair of independent uniform random variables $z_1 \sim UR([0, T_1))$, $z_2 \sim UR([1 - T_2, 1))$. Each sampling site lies in a cell of a grid on the unit sphere formed by meridians a constant angle T_1 apart, and parallel circles a constant distance T_2 apart. This grid consists of $n_1 n_2$ cells of equal area $4\pi/(n_1 n_2)$, but not of equal shape – hence the design is called pseudosystematic.

To predict the variance of the corresponding estimator is difficult unless the measurement function is periodic in both angles ϕ and θ. To this end we redefine the measurement function as the average of antipodal observations of f namely

$$F(\phi, \theta) = \frac{1}{2}[f(\phi, \theta) + f(\phi + \pi, \pi - \theta)], \tag{11}$$

whereby $F(\phi, \theta) = F(\phi + 2k\pi, \theta + l\pi)$, $(k, l \in \mathbb{Z})$, as required. The target parameter has essentially the same form as before with f replaced by F, that is,

$$Q = \int_0^{2\pi} \int_0^{\pi} F(\phi, \theta) \sin\theta d\theta d\phi. \tag{12}$$

Now an unbiased estimator of Q based on the aforementioned design is:

$$\widehat{Q} = T_1 T_2 \sum_{i=0}^{n_1 - 1} \sum_{j=0}^{n_2 - 1} F(z_1 + iT_1, \cos^{-1}(z_2 - jT_2)), \tag{13}$$

which depends on the random pair (z_1, z_2).

3.2 Example: Applying the nucleator method

As a way to illustrate the nucleator method, the parameters selected are $n_1 = 3$, $n_2 = 3$ to generate a sample. The first step is calculate $z = (z_1, z_2)$, which is defined as a couple of random values with uniform distributions as we define previously. Hence, these random values are defined by periods $T_1 = 2\pi/3$ and $T_2 = 2/3$, which define the directions of the rays.

These directions are used to generate the systematic sample Λ_z, which is a set of 9 values $\{(z_1, z_2), (z_1+2\pi/3, z_2), (z_1+4\pi/3, z_2), (z_1, z_2+2/3), (z_1, z_2+4/3), (z_1+2\pi/3, z_2+2/3), (z_1+4\pi/3, z_2+2/3), (z_1+2\pi/3, z_2+4/3), (z_1+4\pi/3, z_2+4/3)\}$. Fig. 3 represents the isotropic rays, which allow us to calculate the radius r from origin to the particle surface for each ray and its antipode.

Therefore the set of r values, which is formed by 18 values (9 radii of the rays and 9 radii of the antipodes), are used to calculate $f(\phi, \theta)$ measurement function. The results $f(\phi, \theta)$ are employed to calculate the reformulated measurement function $F(\phi, \theta)$ and a volume estimation $\widehat{Q} = 8.4486$. The real volume is $Q = 9.0194$.

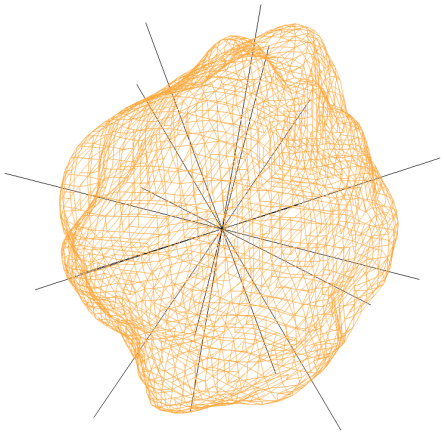

Figure 3: Representation of the rays (rays and antipodes) contained in the systematic sample Λ_z, on Particle 1 defined below.

In the next subsection we analyse the error variance estimation from one single sample.

3.3 Variance prediction of volume estimations

Since the nucleator volume estimation is unbiased the quality of the estimator is given by the variance. The variance estimation in systematic sampling is not an easy matter. In general no unbiased variance estimators are known. Here we

use the nucleator variance estimators derived in Gual-Arnau and Cruz-Orive (2002) and shown in (Eqs. 14, 15).

These equations were deduced by a specific covariogram model and Fourier expansions and that is why we need to work with a periodical measurement function on the sphere. True variances are denoted by $\mathrm{Var}(\cdot)$ whereas variance estimators are denoted by $\mathrm{var}(\cdot)$.

$$\begin{aligned}\mathrm{var}_{0,0}\left(\widehat{Q}\right) = \\ \frac{4\pi^2}{9n_1n_2(n_1-1)(n_2-1)} \\ \times[6(n_1-1)(C_{00}-C_{01}) \\ +6(n_2-1)(C_{00}-C_{10}) \\ -(n_1^2+n_2^2-1)(C_{00}-C_{01}-C_{10}C_{11})] \quad .\end{aligned} \tag{14}$$

$$\begin{aligned}\mathrm{var}_{1,1}\left(\widehat{Q}\right) = \\ \frac{4\pi^2}{225n_1n_2(n_1-1)^2(n_2-1)^2} \\ \times[30(n_1-1)^2(C_{00}-C_{01}) \\ +30(n_2-1)^2(C_{00}-C_{10}) \\ -(n_1^4+n_2^4-1)(C_{00}-C_{01}-C_{10}C_{11})] \quad .\end{aligned} \tag{15}$$

Eqs. 14, 15, give us a prediction of the variance from a single volume estimation. These equations depend on sampling sizes n_1 and n_2. Moreover, both equations are determined by C_{kl} coefficients, which represent the relationship between some adjacent measurements depending on the values $k = 0, 1$ and $l = 0, 1$. As a way to simplify, we denote $F_{ij} = F(z_1 + iT_1, \cos^{-1}(z_2 - jT_2))$, thus:

$$C_{kl} = \sum_{i=0}^{n_1-1}\sum_{j=0}^{n_2-1} F_{ij}F_{i+k,j+l}, \quad F_{ij} = F_{i+n_1,j} = F_{i,j+n_2} = F_{i+n_1,j+n_2}. \tag{16}$$

3.4 Particles

In this subsection we define the particles considered for volume and variance estimation and implement them in Blender (http://www.blender.org/).

- **Particle 0:** First, we use the Gaussian-like particle, $g(x, y, z)$ defined in Gual-Arnau and Cruz-Orive (2002). This particle is generated from the unit sphere $\mathbb{S}^2 = \{(x, y, z) : x^2 + y^2 + z^2 = 1\}$:

$$g(x, y, z) = 1 + t(x, y, z), \quad (x, y, z) \in \mathbb{S}^2, \tag{17}$$

  where the bounded test function $t : \mathbb{R}^3 \to (-1, \infty)$ is defined as:

$$
\begin{aligned}
t(x,y,z) = \\
\exp\{-2[(x+0.2)^2+(y+0.2)^2+(z-1)^2]\} \\
+0.5\exp\{-4[(x-0.7)^2+(y-0.7)^2+z^2]\} \\
-0.25\exp\{-4[(x+0.7)^2+(y+0.7)^2+z^2]\}.
\end{aligned} \tag{18}
$$

This particle allows us to check for consistency and to test the method under optimal conditions, since it is star convex. The particle has a well defined symmetry and is completely smooth but is oval rather than spherical as can be seen in Fig. 4.

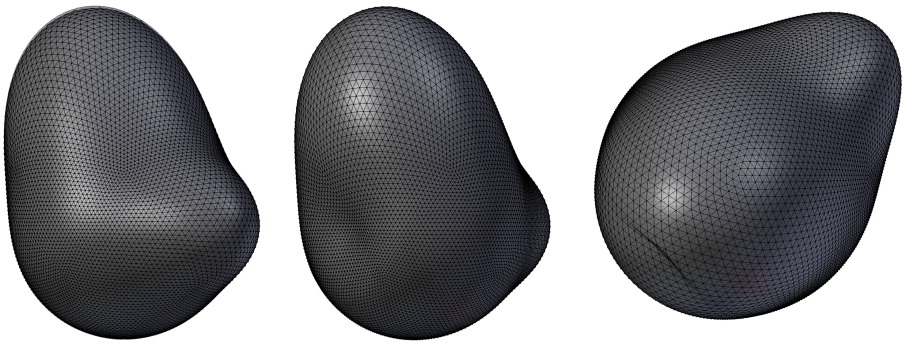

Figure 4: Gaussian-like simulated particle; right side view, front view and top view.

- **Particle 1:** This particle is an irregular, asteroid-like particle, which we simulate as follows: Based on a simple icosahedron, we select a uniform random subset containing 50% of the total vertices. Each selected vertex in the subset is scaled in the j direction, $j = (x, y, z)$, by a random uniform factor $c_{ij} \sim U([u_{ij}, v_{ij}])$. The whole process is repeated $n_{steps} = 4$ times with different sets of selected vertices and particular factors c_{ij}. The factors defined by the i index referring to each of the steps, $i = 1, 2, ..., n_{steps}$. The sampling intervals $[u_{ij}, v_{ij}]$ depending on i, j are given in Table 1(Ronan (2014)).

  Between each of the four scaling steps, the particle edges are subdivided to add new vertices by interpolating smoothly across the deformed object. Finally, we rotate the particle an angle α, $\alpha \sim U([-\pi, \pi])$ rad. The resulting asteroid-like particle can be seen in Fig. 5. The particle is star convex with a slightly irregular surface but keeping some spherical symmetry.

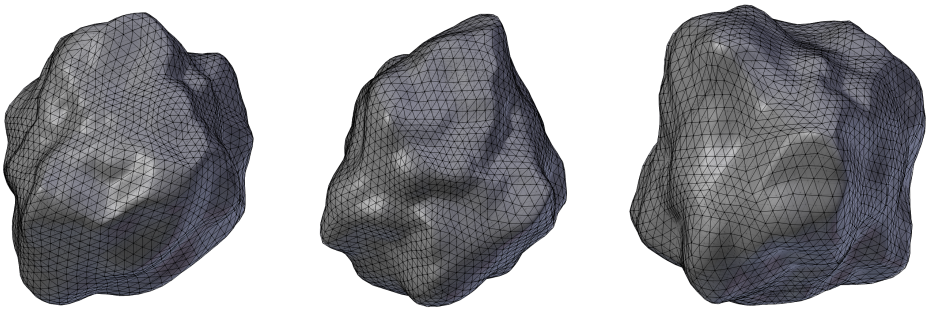

Figure 5: Simulated particle 1; right side view, front view and top view.

Table 1: Scaling intervals which define the interval $[u_{ij}, v_{ij}]$ to obtain the random uniform scale factor. i represents the scaling step and j stands for the target Cartesian coordinate.

	$[u_{ij}, v_{ij}]$		
i \ j	x	y	z
1	[1.1,1.4]	[1.1,1.4]	[1.1,1.4]
2	[1.05,1.15]	[1.05,1.15]	[1.05,1.15]
3	[0.92,1.05]	[0.92,1.05]	[0.92,1.05]
4	[0.9,1.5]	[1,1]	[1,1]

- **Particle 2:** This particle is obtained by the same method as used for particle 1. However the particle is manually modified to obtain a non-star-convex particle with the origin outside the surface (Fig. 6). However it is somewhat spherical but the surface is irregular. This allows us to test the nucleator method under these particular conditions where each ray may cross the particle surface multiple times (Fig. 6).

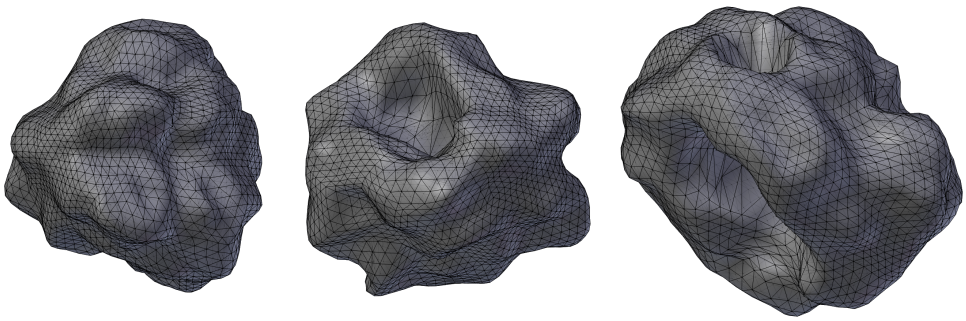

Figure 6: Simulated particle 2; right side view, front view and top view.

3.5 Pseudosystematic sampling to calculate empirical variance

In order to test the performance of the variance estimators, $\mathrm{var}_{0,0}(\widehat{Q})$, $\mathrm{var}_{1,1}(\widehat{Q})$ defined above (Eqs. 14 and 15), we calculate the empirical variance, through Monte Carlo replications of the nucleator estimator on the particles described in the previous section.

As a prior consideration to carry out the replication process, we must bear in mind that the sampling method is periodic, $\phi + 2\pi = \phi$, $\theta + \pi = \theta$ and $y + 2 = y$. The unit sphere surface can be represented as a periodical grid on which the sampling process defined above is replicated. Each replication involves taking a sample $\Lambda_z^{k,l}$, where the nucleator method is applied to obtain a volume estimation $\widehat{Q}^{k,l}(z)$, as follows:

$$\Lambda_z^{k,l} = \{(\phi_{ik}, y_{jl})\}, \tag{19}$$

$$\begin{aligned} \phi_{ik} &= z_1 + iT_1 + kT_{N1}, \\ i = 0, 1, .., n_1 - 1, \quad k &= 0, 1, .., N_1 - 1, \end{aligned} \tag{20}$$

$$\begin{aligned} y_{jl} &= z_2 - jT_2 - lT_{N2}, \\ j = 0, 1, .., n_2 - 1, \quad l &= 0, 1, .., N_2 - 1, \end{aligned} \tag{21}$$

where k and l are the indexes, which determine the position of a particular sample and its corresponding volume estimation in the replications set. This replication process is systematically repeated $N_1 \times N_2$ times (Fig. 7), where N_1 is the number of systematic replications on the azimuthal angle and N_2 on the polar angle with periods between samples $T_{N1} = 2\pi/N_1$ and $T_{N2} = 2/N_2$. The entire replication process requires a single starting random ray z. The indexes i and j determine the position of a particular ray in a sample as in Eq. 13.

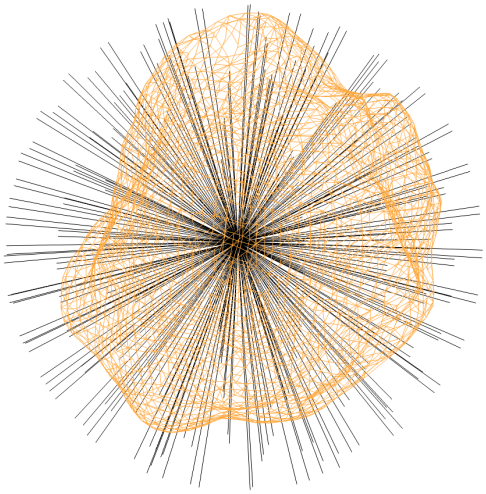

Figure 7: Pseudosystematic sampling method applied to Particle 1, with $n_1 = 3$, $n_2 = 3$, $N_1 = 4$ and $N_2 = 4$.

The empirical variance, $\mathrm{Var_e}(\widehat{Q})$, and coefficient of error, $\mathrm{CE}_e^2(\widehat{Q})$, can be calculated from the set of volume estimations $\{\widehat{Q}^{k,l}(z)\}$ obtained from the Monte Carlo replications:

$$\mathrm{Var_e}(\widehat{Q}) = \frac{1}{N_1 N_2} \sum_{k=0}^{N_1-1} \sum_{l=0}^{N_2-1} (\widehat{Q}^{k,l}(z) - \mathbb{E}_e(\widehat{Q}))^2, \tag{22}$$

$$\mathrm{CE}_e^2(\widehat{Q}) = \frac{\mathrm{Var_e}(\widehat{Q})}{Q^2}, \tag{23}$$

$$\mathbb{E}_e(\widehat{Q}) = \frac{1}{N_1 N_2} \sum_{k=0}^{N_1-1} \sum_{l=0}^{N_2-1} (\widehat{Q}^{k,l}(z)). \tag{24}$$

4 Results

We test the performance of the theoretical variance estimators against the empirical variance. Two sets of $N_1 \times N_2$ variance estimations are obtained applying Eqs. 14, 15 for each of the $N_1 \times N_2$ replications.

The parameters used for the Monte Carlo replications are all possible combinations of $n_1 = 2, 3, .., 10$ $n_2 = 2, 3, .., 10$ with $N_1 = 180/n_1$, $N_2 = 90/n_2$ yielding a total of $180 \times 90 = 16200$ generated rays .

Figs. 8, 9, 10 show the replicated volume estimations for particles $0, 1$ and 2 respectively. The values of n_1 and n_2 considered for the graphs are $n_1 = 3, 6, 9$ and $n_2 = 2, 3, 4, .., 10$.

Figs. 11, 12 and 13 show the empirical coefficient of error defined above, and the theoretical coefficients of error:

$$\mathrm{ce}_{0,0}^2(\widehat{Q}) = \frac{\mathrm{var}_{0,0}\left(\widehat{Q}\right)}{Q^2}, \tag{25}$$

$$\mathrm{ce}_{1,1}^2(\widehat{Q}) = \frac{\mathrm{var}_{1,1}\left(\widehat{Q}\right)}{Q^2}. \tag{26}$$

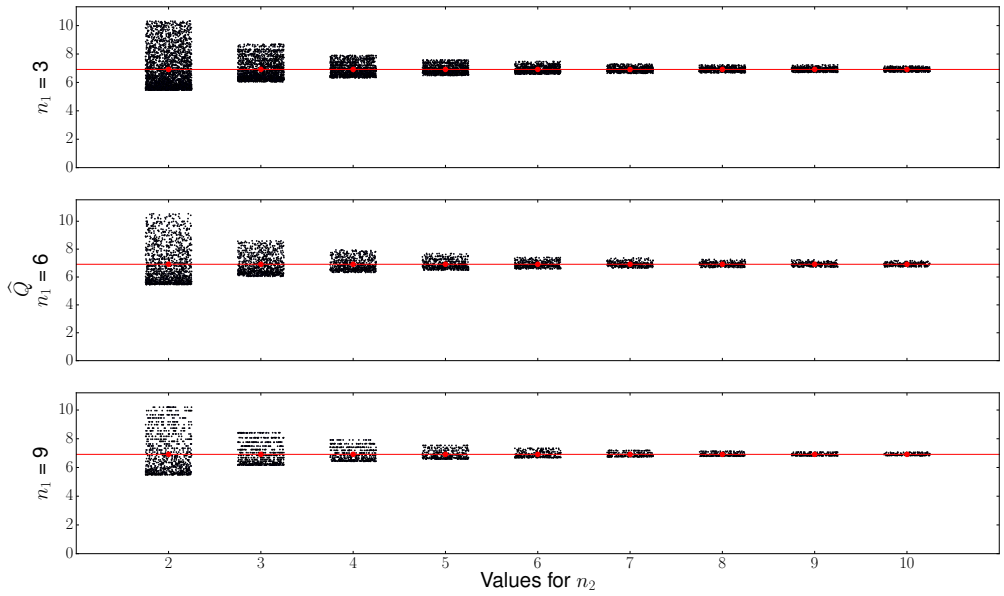

Figure 8: Particle 0 Monte Carlo replications (black dots) represent all the replicated volume estimations, whose average is illustrated by a red dot, and the red line represents the real volume.

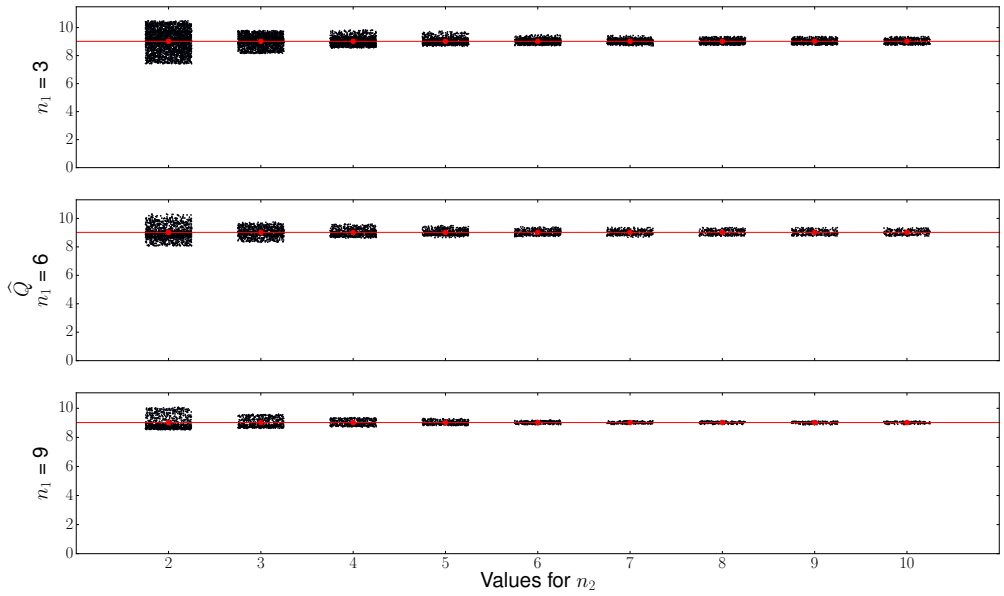

Figure 9: Particle 1 Monte Carlo replications (black dots) represent all the replicated volume estimations, whose average is illustrated by a red dot, and the red line represents the real volume.

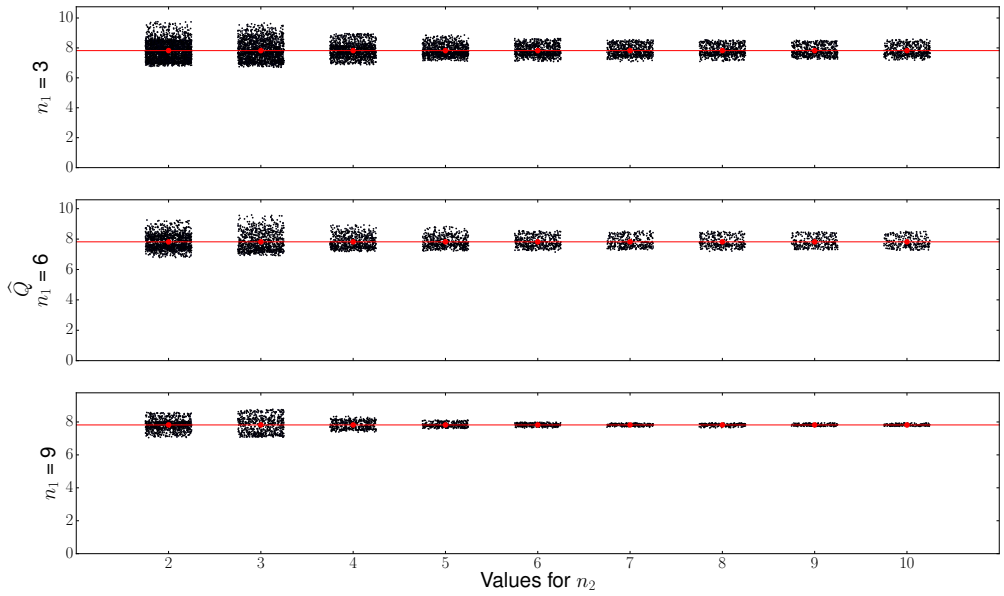

Figure 10: Particle 2 Monte Carlo replications (black dots) represent all the replicated volume estimations, whose average is illustrated by a red dot, and the red line represents the real volume.

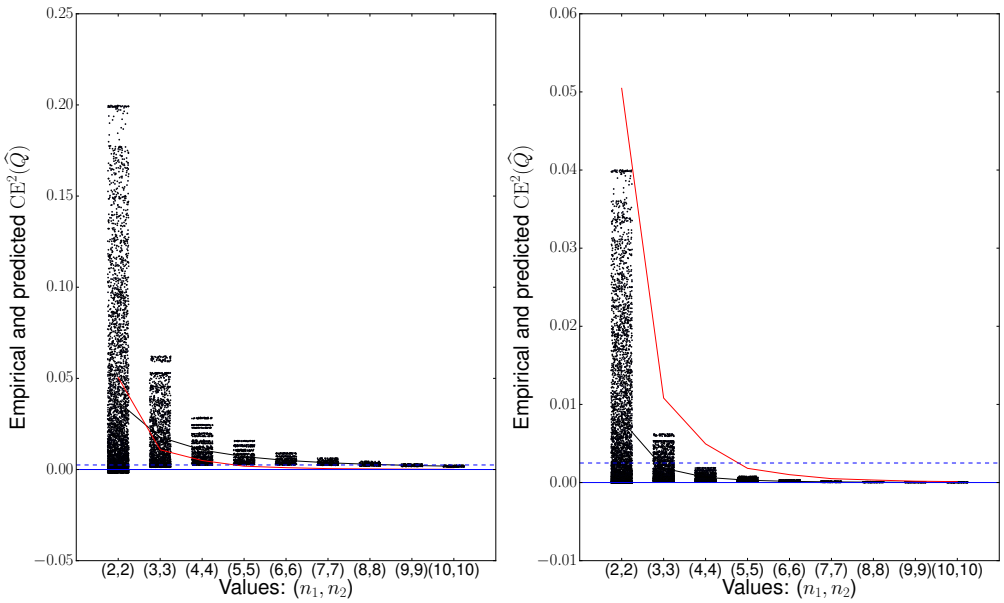

Figure 11: Particle 0 Monte Carlo replications for estimators $\mathrm{ce}^2_{0,0}(\widehat{Q})$ (left) and $\mathrm{ce}^2_{1,1}(\widehat{Q})$ (right). The black line represents their mean and the red line the empirical coefficient of error. The dashed blue line corresponds to 5% coefficient of error.

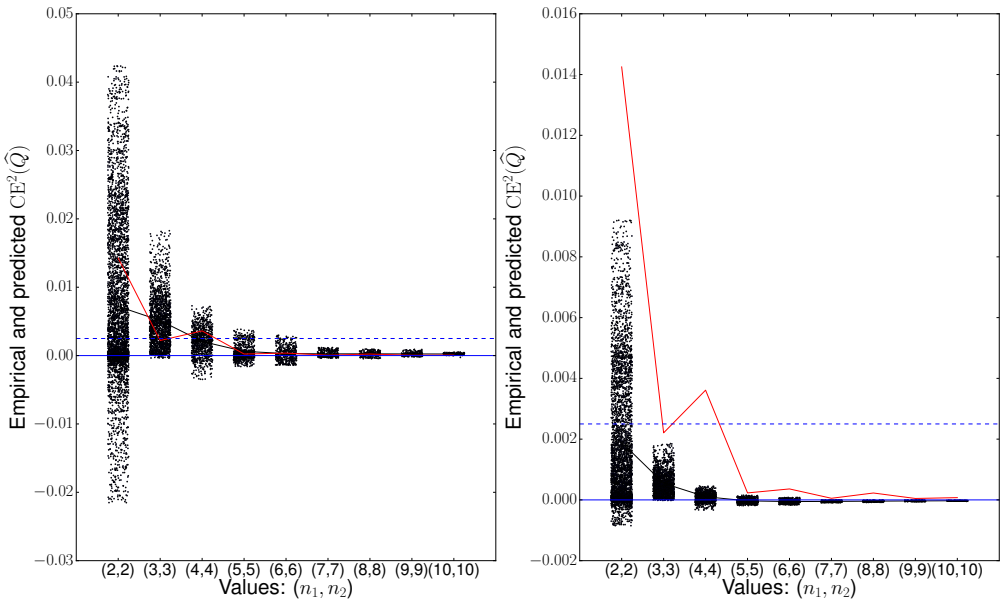

Figure 12: Particle 1 Monte Carlo replications for estimators $\mathrm{ce}^2_{0,0}(\widehat{Q})$ (left) and $\mathrm{ce}^2_{1,1}(\widehat{Q})$ (right). The black line represents their mean and the red line the empirical coefficient of error. The dashed blue line corresponds to 5% coefficient of error.

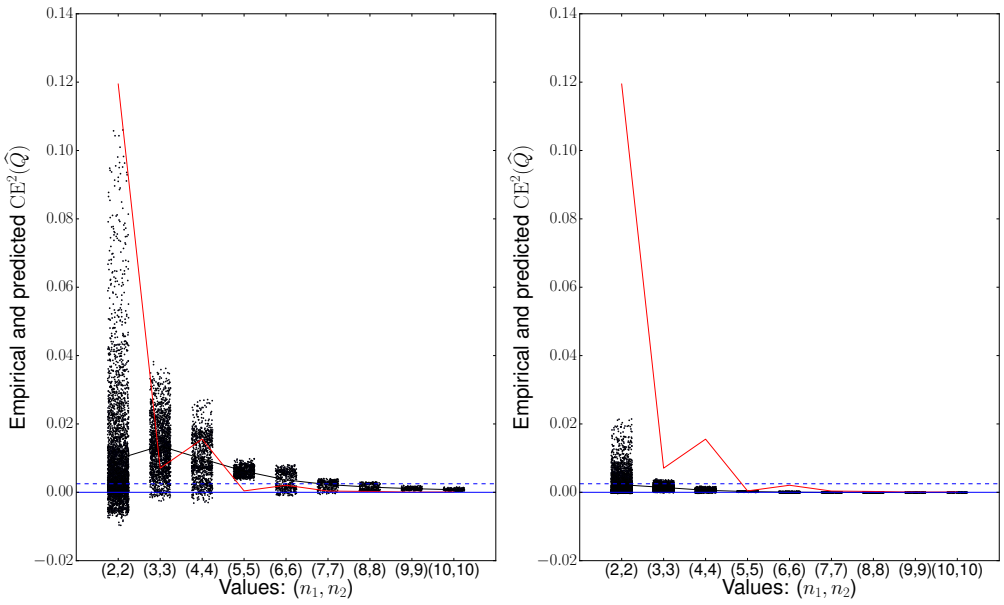

Figure 13: Particle 2 Monte Carlo replications for estimators $\mathrm{ce}^2_{0,0}(\widehat{Q})$ (left) and $\mathrm{ce}^2_{1,1}(\widehat{Q})$ (right). The black line represents their mean and the red line the empirical coefficient of error. The dashed blue line corresponds to 5% coefficient of error.

5 Discussion

The results shown above, allow us to analyse the behaviour of the volume and variance estimators. First, we verify that the results corresponding to the Gaussian-like particle in the ranges $1 < n_1 \leq 4$, $1 < n_2 \leq 4$ match those in Gual-Arnau and Cruz-Orive (2002) within rounding errors, confirming that our method is properly implemented. Figs. 8, 9 and 10, show that the volume estimations are unbiased regardless from the origin position and type of particle, as expected from Gundersen (1988). However the variance may increase if the origin is located far from the center of mass or if the particle shape is non-spherical.

When computing the estimators $\mathrm{var}_{0,0}(\widehat{Q})$,$\mathrm{var}_{1,1}(\widehat{Q})$ we remark that sometimes they yield negative variance estimations. This happens when the particle shows very abrupt changes and an irregular surface or when $|n_2 - n_1|$ is large as we can see in Figs. 12 and 13.

Estimator $\mathrm{var}_{1,1}(\widehat{Q})$ shows very poor performance. In many cases the variance estimation is negative as we can see in Figs. 11, Fig. 12 and Fig. 13. Therefore we focus on estimator $\mathrm{var}_{0,0}(\widehat{Q})$ that shows a better performance for all the considered particles. The estimation for particle 0 looks worse since for $n_1 \geq 6$ and $n_2 \geq 6$ the empirical coefficient of error is almost zero and therefore difficult to estimate.

Therefore we do not recommend using $\mathrm{var}_{1,1}(\widehat{Q})$, whereas $\mathrm{var}_{0,0}(\widehat{Q})$ can be useful in some cases.

We conclude that the theoretical variance estimators are subject to improvement. The symmetry restrictions of the covariogram model used in Gual-Arnau and Cruz-Orive (2002) are probably not accurate enough for realistic particles.

References

Gundersen, H. J. G. (1988). The nucleator. *Journal of microscopy, 151(1), 3-21.*

Howard, V., Reed, M. (2004). Unbiased stereology: three-dimensional measurement in microscopy. *Garland Science.*

Gual-Arnau, X., Cruz-Orive, L. M. (2002). Variance prediction for pseudosystematic sampling on the sphere. *Advances in Applied Probability*, 469-483.

Gual-Arnau, X., Cruz-Orive, L. M. (2000). Systematic sampling on the circle and on the sphere. *Advances in Applied Probability*, 628-647.

Anders, M. (2010). Blender 2.49 Scripting: Extend the Power and Flexibility of Blender with the Help of Python, a High-level, Easy-to-learn Scripting Language. *Packt Publishing Ltd.*

Cruz-Orive, L. M. (1987). Particle number can be estimated using a disector of unknown thickness: the selector. *Journal of microscopy*, 145(Pt 2), 121-142.

Arfken, George B. (2013). Mathematical methods for physicists. *Academic press.*

Baddeley, Adrian, and Eva B. Vedel Jensen. (2004). Stereology for statisticians. *CRC Press.*

Leobacher, G., and Pillichshammer, F. (2014). Introduction to quasi-Monte Carlo integration and applications. Springer International Publishing.

Spherical Coordinate System, `http://en.wikipedia.org/wiki/Spherical_coordinate_system`, 09-06-2015.

Ronan, Philip. (2014). Create a random 3D asteroid, `http://codegolf.stackexchange.com/questions/23776/create-a-random-3d-asteroid`.

APPENDIX A: Code

All the code was developed in Python 3.3.2 on the free open-source 3D computer graphics software Blender 2.68, see Anders (2010). The code responsible to generate the particles is presented below.

Listing 1: Code used to generate particles 1 and 2.

```
def asteroid(size=1,x=0,y=0,z=0,sub=1):
    bpy.ops.mesh.primitive_ico_sphere_add(size=size,
        location=(x,y,z),rotation=(0,0,0))
    bpy.ops.object.mode_set(mode='EDIT')
    bpy.ops.mesh.faces_shade_smooth()
    bpy.ops.mesh.select_mode(type='VERT')
    bpy.ops.mesh.select_all(action='DESELECT')
    bpy.ops.mesh.select_random()
    bpy.ops.transform.resize(value=(random.uniform
        (1.1,1.4),random.uniform(1.1,1.4),random.uniform
        (1.1,1.4)))
    bpy.ops.mesh.select_all(action='SELECT')
    bpy.ops.mesh.subdivide(smoothness=1)
    bpy.ops.mesh.select_all(action='DESELECT')
    bpy.ops.mesh.select_random()
    bpy.ops.transform.resize(value=(random.uniform
        (1.05,1.15),random.uniform(1.05,1.15),random.
        uniform(1.05,1.15)))
    bpy.ops.mesh.select_all(action='SELECT')
    if sub == 1:
        bpy.ops.mesh.subdivide(smoothness=1)
    bpy.ops.mesh.select_random()
    bpy.ops.transform.resize(value=(random.uniform
        (0.92,1.05),random.uniform(0.92,1.05),random.
        uniform(0.92,1.05)))
    bpy.ops.mesh.select_all(action='SELECT')
    if sub == 1:
        bpy.ops.mesh.subdivide(smoothness=1)
    stretch = random.uniform(0.9,1.5)
    bpy.ops.transform.resize(value=(stretch,1,1))
    bpy.ops.transform.rotate(value=(random.uniform
        (-1.57,1.57)), axis=(random.uniform(-1.57,1.57),
        random.uniform(-1.57,1.57),random.uniform
        (-1.57,1.57)))
    bpy.ops.object.mode_set(mode='OBJECT')
```

Listing 2: Code used to generate particle 0.

```
def gaussian():
```

```
2       bpy.ops.mesh.primitive_ico_sphere_add(subdivisions=6,
            size=1,location=(0,0,0),rotation=(0,0,0))
3       me = bpy.context.object.data
4       O = 0
5       for v in me.vertices:
6           if (v.co[0]==0) and (v.co[1]==0) and (v.co[2]==0)
                :
7               O = v
8               bpy.ops.object.mode_set(mode='EDIT')
9               bpy.ops.mesh.select_mode(type='VERT')
10              bpy.ops.mesh.select_all(action="DESELECT")
11              O.select =True
12      for v in me.vertices:
13          if v != O:
14              x=v.co[0]
15              y=v.co[1]
16              z=v.co[2]
17              t=math.exp(-2*(math.pow(x+0.2,2)+math.pow(y
                    +0.2,2)+math.pow(z-1,2)))+0.5*math.exp
                    (-4*(math.pow(x-0.7,2)+math.pow(y-0.7,2)+
                    math.pow(z,2)))-0.25*math.exp(-4*(math.pow
                    (x+0.7,2)+math.pow(y+0.7,2)+math.pow(z,2))
                    )
18              v.co[0] = x*(1+t)
19              v.co[1] = y*(1+t)
20              v.co[2] = z*(1+t)
21      bpy.ops.object.mode_set(mode='EDIT')
22      bpy.ops.mesh.faces_shade_smooth()
23      bpy.ops.object.mode_set(mode='OBJECT')
```

Moreover, there are three algorithms involved in the nucleator method implementation. The first and most basic is the next one:

Listing 3: Method used to obtain the particle mesh.

```
1   def get_polygons(obj_name):
2       scene = bpy.context.scene
3       bpy.ops.object.mode_set(mode='OBJECT', toggle =False)
4       obj = bpy.data.objects[obj_name]
5       scene.objects.active = obj
6       polygons = []
7       verts = []
8       polygons_sh_tmp = []
9       me = bpy.context.object.data
10      for poly in me.polygons:
11          pol = []
12          pol.append(poly.index)
```

```
13          pol.append(poly.loop_total)
14          tmp_pol = []
15          for loop_index in range(poly.loop_start, poly.
                loop_start + poly.loop_total):
16              pol.append(me.loops[loop_index].vertex_index)
17              tmp_pol.append(me.vertices[me.loops[
                    loop_index].vertex_index].co+bpy.context.
                    object.location)
18          polygons.append(pol)
19          polygons_sh_tmp.append(Polygon(tmp_pol))
20      for ver in me.vertices:
21          verts.append(ver.co+bpy.context.object.location)
22      plain_verts = [vert.to_tuple() for vert in verts]
23      points_sh = MultiPoint(plain_verts)
24      polygons_sh = MultiPolygon(polygons_sh_tmp)
25      pol_verts = []
26      for pol in polygons_sh:
27          verts = []
28          P1_T=mathutils.Vector(pol.exterior.coords[0])
29          P2_T=mathutils.Vector(pol.exterior.coords[1])
30          P3_T=mathutils.Vector(pol.exterior.coords[2])
31          verts.append(P1_T)
32          verts.append(P2_T)
33          verts.append(P3_T)
34          pol_verts.append(verts)
35      return polygons,plain_verts,polygons_sh,points_sh,
            pol_verts
```

This method is used to obtain five sets of polygons and vertices, which represent the particle mesh in different ways. We only need to provide it the object name to obtain the data sets.

These data sets are used by the next method to obtain the collisions between a provided ray (ray and its antipode) and the particle surface. Hence, we need to establish the ray parameters (ϕ, θ) and the measurement origin to obtain the intersections coordinates and the distances between the origin and the particle surface.

Listing 4: Method which calculate the intersections and distances between a ray and a particle surface given the measurement origin.

```
1  def get_collisions_line(fi,theta,r_o,fi_o,theta_o,
       polygons,polygons_sh,obj_name,graphics = 1,max_r = 0,
       pol_verts=[]):
2      if max_r == 0:
3          max_r = max_radius_object(obj_name,graphics)
4      if graphics == 1:
```

```
5           create_line(-max_r*2,max_r*2,fi,theta,r_o,fi_o,
                theta_o,'Line')
6           scene = bpy.context.scene
7           bpy.ops.object.mode_set(mode='OBJECT', toggle =
                False)
8           obj = bpy.data.objects[obj_name]
9           scene.objects.active = obj
10          scene.update()
11      intersection_polygons = []
12      intersection_points = []
13      intersection_distances = []
14      number_intersections = 0
15      O = mathutils.Vector(coord_conversion_spher_to_cart(
            r_o,fi_o,theta_o))
16      P1 = mathutils.Vector(coord_conversion_spher_to_cart
            (-max_r*1000,fi,theta))+O
17      P2 = mathutils.Vector(coord_conversion_spher_to_cart(
            max_r*1000,fi,theta))+O
18      indice = 0
19      for pol in polygons_sh:
20          if len(pol_verts)==0:
21              P1_T=mathutils.Vector(pol.exterior.coords[0])
22              P2_T=mathutils.Vector(pol.exterior.coords[1])
23              P3_T=mathutils.Vector(pol.exterior.coords[2])
24              intersection = mathutils.geometry.
                    intersect_ray_tri(P1_T,P2_T,P3_T,P2-P1,O)
25          else:
26              intersection = mathutils.geometry.
                    intersect_ray_tri(pol_verts[indice][0],
                    pol_verts[indice][1],pol_verts[indice][2],
                    P2-P1,O)
27          if (intersection != None):
28              number_intersections = number_intersections +
                     1
29              intersection_polygons.append(polygons[indice
                    ])
30              intersection_points.append(intersection)
31              P = intersection
32              intersection_distances.append(math.sqrt(math.
                    pow(P[0]-O[0],2)+math.pow(P[1]-O[1],2)+
                    math.pow(P[2]-O[2],2)))
33          indice = indice + 1
34      if graphics == 1:
35          select_some_polys(intersection_polygons,obj_name)
36      return number_intersections, intersection_polygons,
            intersection_points, intersection_distances
```

Hence these methods are the base to implements the nucleator method.

Listing 5: Nucleator method implementation.

```
1  def nucleator(r_max, n_fi, n_theta, obj_name, graphics = 1,
       i_fi = -1,i_y = -2,polygons = [], polygons_sh = [],
       pol_verts = [], O_p=[0,0,0]):
2      fi_step = (math.pi*2)/n_fi
3      theta_step = 2/n_theta
4      Fs = []
5      if i_fi != -1:
6          init_fi = i_fi
7      else:
8          init_fi = random.uniform(0,2*math.pi)
9      if i_y != -2:
10         init_y = i_y
11     else:
12         init_y = random.uniform(-1,1)
13     fis = [init_fi+(i*fi_step) for i in range(n_fi)]
14     thetas = [(math.acos(init_y-(i*theta_step))) for i in
           range(n_theta)]
15     if (len(polygons)==0) or (len(polygons_sh)==0) or (
          len(pol_verts)==0):
16         polygons, plain_verts, polygons_sh, points_sh,
              pol_verts = get_polygons(obj_name)
17     volume_estimations = []
18     total_estimation = 0
19     volume_estimation = 0
20     cont = 1
21     suma = 0
22     for fi in fis:
23         Fs_temp = []
24         for theta in thetas:
25             if graphics == 1:
26                 create_line(-r_max,r_max,fi,theta,O_p[0],
                      O_p[1],O_p[2],str(fi)+str(theta))
27             n_inter, inter_polys, inter_points,
                  inter_dist = get_collisions_line(fi,theta,
                  O_p[0],O_p[1],O_p[2],polygons,polygons_sh,
                  obj_name,0,r_max,pol_verts)
28             estimation,F = estimate_volume(inter_points,
                  inter_dist, O_p)
29             Fs_temp.append(F)
30             if (n_fi > 1) and (n_theta > 1):
31                 total_estimation = total_estimation +
                      estimation
32                 volume_estimations.append(estimation)
```

```
33          if n_fi == 1:
34              estimation = ((math.pi*4)/n_theta)*sum(
                    Fs_temp)
35              volume_estimations.append(estimation)
36              total_estimation = total_estimation +
                    estimation
37          if n_theta == 1:
38              suma = suma + Fs_temp[0]
39              if (cont % n_fi) == 0:
40                  estimation = ((math.pi*4)/n_fi)*suma
41                  volume_estimations.append(estimation)
42                  total_estimation = total_estimation +
                        estimation
43                  suma = 0
44              cont = cont + 1
45          Fs.append(Fs_temp)
46      if (n_fi > 1) and (n_theta > 1):
47          volume_estimation = total_estimation/(n_fi*
                n_theta)
48      elif n_fi == 1:
49          volume_estimation = total_estimation
50      elif n_theta == 1:
51          volume_estimation = total_estimation
52      return volume_estimations, volume_estimation, Fs
```

Listing 6: Auxiliar method to calculate the volume estimation.

```
1   def estimate_volume(inter_points, inter_dist, O_p=[0,0,0]):
2       O = coord_conversion_spher_to_cart(O_p[0], O_p[1], O_p
            [2])
3       estimation = -1
4       F = -1
5       rs1 = []
6       rs2 = []
7       signs = []
8       index = 0
9       for p in inter_points:
10          if (len(rs1) == 0) and (len(rs2) == 0):
11              if (p[0]-O[0])+math.copysign((p[0]-O[0]),-1)
                    == 0:
12                  signs.append(1)
13              else:
14                  signs.append(-1)
15              if (p[1]-O[1])+math.copysign((p[1]-O[1]),-1)
                    == 0:
16                  signs.append(1)
17              else:
```

```
18                      signs.append(-1)
19                  if (p[2]-O[2])+math.copysign((p[2]-O[2]),-1)
                        == 0:
20                      signs.append(1)
21                  else:
22                      signs.append(-1)
23                  rs1.append(inter_dist[index])
24              else:
25                  if ((p[0]-O[0])-math.copysign((p[0]-O[0]),
                        signs[0]) == 0) and ((p[1]-O[1])-math.
                        copysign((p[1]-O[1]),signs[1]) == 0) and
                        ((p[2]-O[2])-math.copysign((p[2]-O[2]),
                        signs[2]) == 0):
26                      rs1.append(inter_dist[index])
27                  else:
28                      rs2.append(inter_dist[index])
29              index = index + 1
30      rs1.sort()
31      rs2.sort()
32      r1 = 0
33      r2 = 0
34      if (len(rs1) % 2) == 0:
35          signor1 = -1
36      else:
37          signor1 = 1
38      if (len(rs2) % 2) == 0:
39          signor2 = -1
40      else:
41          signor2 = 1
42      for r in rs1:
43          r1 = r1 + (signor1*math.pow(r,3))
44          signor1 = signor1*(-1)
45      for r in rs2:
46          r2 = r2 + (signor2*math.pow(r,3))
47          signor2 = signor2*(-1)
48      estimation = (4/3)*math.pi*(1/2)*((r1)+(r2))
49      F = (1/3)*(1/2)*((r1)+(r2))
50      return estimation, F
```

These methods generate a systematic sample Λ_z for a given z value. We need to provide it also the measurement origin to obtain a volume estimation and the reformulated measurement function $F(\phi, \theta)$ values, which are used below to obtain the variance estimations.

Finally, the main method that implements the pseudosystematic sampling to calculate the empirical variance and to obtain the variance estimations is defined

below:

Listing 7: Main algorithm, which implements the pseudosystematic sampling method.

```
def pseudosystematic_sampling(n1,n2,max_N1,max_N2,
    obj_name,graphics = 1,O_p=[0,0,0]):
     r_max = max_radius_object(obj_name,graphics)
     N1 = int(max_N1/n1)
     N2 = int(max_N2/n2)
     T1 = (math.pi*2)/n1
     T2 = 2/n2
     dT1 = T1/N1
     dT2 = T2/N2
     init_random_sample_fi = random.uniform(0,dT1)
     init_random_sample_y = random.uniform(1,1-dT2)
     volume_estimations = []
     variances_th_00 = []
     variances_th_11 = []
     polygons, plain_verts, polygons_sh, points_sh,
        pol_verts = get_polygons(obj_name)
     for k in range(N1):
          for l in range(N2):
               fi0 = init_random_sample_fi + (k*dT1)
               y0 = init_random_sample_y - (l*dT2)
               vols,estimation,Fs = nucleator(r_max,n1,n2,
                  obj_name,graphics,fi0,y0,polygons,
                  polygons_sh,pol_verts,O_p)
               if (n1 > 1) and (n2 > 1):
                    C00=C_value(0,0,n1,n2,Fs)
                    C01=C_value(0,1,n1,n2,Fs)
                    C10=C_value(1,0,n1,n2,Fs)
                    C11=C_value(1,1,n1,n2,Fs)
                    var00 = ((math.pow(math.pi,2)*4)/((9*n1*
                       n2)*(n1-1)*(n2-1)))*((6*(n1-1)*(C00-
                       C01))+(6*(n2-1)*(C00-C10))-(math.pow(
                       n1,2)+math.pow(n2,2)-1)*(C00-C01-C10+
                       C11))
                    var11 = ((math.pow(math.pi,2)*4)/(225*n1*
                       n2*math.pow((n1-1),2)*math.pow((n2-1)
                       ,2)))*((30*math.pow((n1-1),2)*(C00-C01
                       ))+(30*math.pow((n2-1),2)*(C00-C10))-(
                       math.pow(n1,4)+math.pow(n2,4)-1)*(C00-
                       C01-C10+C11))
                    variances_th_00.append(var00)
                    variances_th_11.append(var11)
               elif n1 == 1:
```

```
30                      var0 = (16*math.pow(math.pi,2)*((C_value
                            (0,0,n1,n2,Fs)-C_value(0,1,n1,n2,Fs)))
                            )/((6*math.pow(n2,2))-(6*n2))
31                      var1 = (16*math.pow(math.pi,2)*((C_value
                            (0,0,n1,n2,Fs)-C_value(0,1,n1,n2,Fs)))
                            )/((30*math.pow(n2,3))-(60*math.pow(n2
                            ,2))+(30*n2))
32                      variances_th_00.append(var0)
33                      variances_th_11.append(var1)
34                  elif n2 == 1:
35                      var0 = (16*math.pow(math.pi,2)*((C_value
                            (0,0,n1,n2,Fs)-C_value(1,0,n1,n2,Fs)))
                            )/((6*math.pow(n1,2))-(6*n1))
36                      var1 = (16*math.pow(math.pi,2)*((C_value
                            (0,0,n1,n2,Fs)-C_value(0,1,n1,n2,Fs)))
                            )/((30*math.pow(n1,3))-(60*math.pow(n1
                            ,2))+(30*n1))
37                      variances_th_00.append(var0)
38                      variances_th_11.append(var1)
39                  volume_estimations.append(estimation)
40      return volume_estimations,variances_th_00,
            variances_th_11
```

This method use a set of parameters which define the pseudosystematic sampling (n_1,n_2,N_1 and N_2). These parameters and other auxiliary methods, which implement basic calculations, are used to obtain three sets of results (volume estimations, $\mathrm{var}_{0,0}\left(\widehat{Q}\right)$ theoretical variances and $\mathrm{var}_{1,1}\left(\widehat{Q}\right)$ theoretical variances).

APPENDIX B: Libraries

The libraries used to implement the methods presented previously are as follows:

- **Shapely**: Shapely is a BSD-licensed Python package for manipulation and analysis of planar geometric objects. This library is used to manipulate the elements of the particle mesh as sets of vectors and matrices with which to work.
- **Mathutils**: More specifically the module geometry is the default Blender geometry module. This module is used to obtain the intersections between a ray and a triangle.
- **Sympy**: Sympy is a Python library for symbolic mathematics. It aims to become a full-featured computer algebra system. In concrete the geometric modules are used to create geometrical entities, such as lines and circles, and query for information about these entities.
- **Bpy**: This module is provided by Blender to the Python interpreter. The module can be imported in a Python script and gives access to blender data, classes and functions. Therefore, we need these functions to manipulate the Blender objects (particles).
- **Bmesh**: Bmesh is the Blender mesh system, which is used to remove, modify or create new meshes. This module allow us to manipulate simple meshes to create complex objects.
- **Other default Python libraries**: Other default libraries as math, random or time are used to implement basic methods, which are the tools to create the main methods explained above.