J. Plasma Phys. (2017), *vol.* 83, 705830302 © Cambridge University Press 2017
doi:10.1017/S0022377817000393

A 'metric' semi-Lagrangian Vlasov–Poisson solver

Stéphane Colombi[1,2,†] **and Christophe Alard**[1]

[1]Institut d'Astrophysique de Paris, CNRS UMR 7095 and UPMC, 98bis, bd Arago, F-75014 Paris, France

[2]Center for Gravitational Physics, Yukawa Institute for Theoretical Physics, Kyoto University, Kyoto 606-8502, Japan

(Received 19 December 2016; revised 9 May 2017; accepted 10 May 2017)

We propose a new semi-Lagrangian Vlasov–Poisson solver. It employs metric elements to follow locally the flow and its deformation, allowing one to find quickly and accurately the initial phase-space position $\boldsymbol{Q}(\boldsymbol{P})$ of any test particle $\boldsymbol{P}$, by expanding at second order the geometry of the motion in the vicinity of the closest element. It is thus possible to reconstruct accurately the phase-space distribution function at any time t and position $\boldsymbol{P}$ by proper interpolation of initial conditions, following Liouville theorem. When distortion of the elements of metric becomes too large, it is necessary to create new initial conditions along with isotropic elements and repeat the procedure again until next resampling. To speed up the process, interpolation of the phase-space distribution is performed at second order during the transport phase, while third-order splines are used at the moments of remapping. We also show how to compute accurately the region of influence of each element of metric with the proper percolation scheme. The algorithm is tested here in the framework of one-dimensional gravitational dynamics but is implemented in such a way that it can be extended easily to four- or six-dimensional phase space. It can also be trivially generalised to plasmas.

Key words: astrophysical plasmas, plasma simulation

1. Introduction

The dynamics of dark matter in the Universe and stars in galaxies is governed by Vlasov–Poisson equations

$$\frac{\partial f}{\partial t} + \boldsymbol{v}\boldsymbol{\cdot}\nabla_x f - \nabla_x \phi \boldsymbol{\cdot} \nabla_v f = 0, \tag{1.1}$$

$$\Delta_x \phi = 4\pi G \rho, \quad \rho(\boldsymbol{x}, t) \equiv \int f(\boldsymbol{x}, \boldsymbol{v}, t)\,\mathrm{d}\boldsymbol{v}, \tag{1.2a,b}$$

where $f(\boldsymbol{x}, \boldsymbol{v}, t)$ is the phase-space distribution function at position $\boldsymbol{x}$, velocity $\boldsymbol{v}$ and time t, ϕ the gravitational potential, G the gravitational constant and ρ the projected

†Email address for correspondence: colombi@iap.fr

density. Usually, these equations are solved numerically with a N-body approach, where the phase-space distribution function is represented by an ensemble of Dirac functions interacting gravitationally with a softened potential (see, e.g. Hockney & Eastwood 1988; Bertschinger 1998; Colombi 2001; Dolag *et al.* 2008; Dehnen & Read 2011, and references therein). It is known that such a rough representation of the phase-space fluid is not free of biases nor systematic effects (see, e.g. Aarseth, Lin & Papaloizou 1988; Kandrup & Smith 1991; Boily, Athanassoula & Kroupa 2002; Binney 2004; Melott 2007; Joyce, Marcos & Sylos Labini 2009; Colombi *et al.* 2015). With the computational power available today, an alternative approach is becoming possible, consisting in solving directly Vlasov dynamics in phase space (see, e.g. Yoshikawa, Yoshida & Umemura 2013). In fact, this point of view is already commonly adopted in plasma physics (see, e.g. Grandgirard *et al.* 2016 and references therein), but seldom used in the gravitational case.

In this paper, we consider the case where the phase-space distribution is a smooth function of the vector

$$\boldsymbol{P} \equiv (\boldsymbol{x}, \boldsymbol{v}). \tag{1.3}$$

This means that initial conditions have non-zero initial velocity dispersion, which is relevant to the analysis of e.g. galaxies or relaxed dark matter halos. On the other hand, if the aim would be to follow accurately the dynamics of dark matter in the Cold Dark Matter paradigm, the initial velocity would be null. In this case, the problem is different since one has to describe the foldings of a D-dimensional hypersurface evolving in $2D$-dimensional phase space (see, e.g. Hahn & Angulo 2016; Sousbie & Colombi 2016).

In the warm case considered here, it is possible to represent $f(\boldsymbol{P}, t)$ on an Eulerian grid. Most of the numerical methods trying to resolve directly the dynamics of function $f(\boldsymbol{P}, t)$ have been developed in plasma physics and are of semi-Lagrangian nature. They usually exploit directly Liouville's theorem, namely that f is conserved along characteristics. In practice, a test particle is associated with each grid site where the value f_T of the phase-space distribution function has to be evaluated. This test particle is followed backwards in time to find its position at previous time step. Then, function f from the previous time step is interpolated at the root of the characteristic to obtain the searched value of f_T, following Liouville's theorem. This approach was pioneered by Cheng & Knorr (1976) in a split fashion and first applied to astrophysical systems by Fujiwara (1981), Nishida *et al.* (1981) and Watanabe *et al.* (1981). In the classical implementation, interpolation of the phase-space distribution function is performed using a third-order spline.

There exist many subsequent improvements over the splitting algorithm of Cheng & Knorr (see, e.g. Shoucri & Gagne 1978; Sonnendrücker *et al.* 1999; Filbet, Sonnendrücker & Bertrand 2001; Besse & Sonnendrücker 2003; Crouseilles, Mehrenberger & Sonnendrücker 2010; Rossmanith & Seal 2011; Qiu & Shu 2011; Güçlü, Christlieb & Hitchon 2014, but this list is far from comprehensive). The main problem common to all these algorithms is the unavoidable diffusion due to repeated resampling of the phase-space distribution function. Indeed, because of the finite grid resolution, repeated interpolation, as sophisticated as it can be, induces information loss and augmentation of entropy. One way to fix this problem is to employ adaptive mesh refinement (see, e.g. Gutnic *et al.* 2004; Besse *et al.* 2008; Besse, Deriaz & Madaule 2017), i.e. to augment local resolution when the level of detail asks for it. However, diffusion can also be significantly dampened if remapping of the phase-space distribution function is performed as seldom as possible. One way to achieve this is to represent $f(\boldsymbol{P}, t)$ with an ensemble of elements of which the shape is allowed to change with time according to the Lagrangian equations of motion (see, e.g. Alard & Colombi 2005; Campos Pinto *et al.* 2014; Larson & Young 2015). In the 'cloudy' method that we proposed in Alard & Colombi (2005), the phase-space distribution is sampled on an ensemble of overlapping round Gaussian functions of which the contours transform into ellipsoids following the phase-space flow. When the ellipsoids become too elongated, the phase-space distribution function is resampled with new round clouds. Because resampling is seldom performed, say at fractions of dynamical times, diffusion effects are much less prominent, even though coarse graining is still performed at the resolution scale given by the inter-cloud centre spacing. The problem with this remapping approach is the computational cost: to have a smooth and accurate representation of the phase-space distribution function, the clouds need to overlap significantly and the extra cost due to this overlap becomes prohibitive in a high number of dimensions.

However, one can realise in fact that it is not necessary to represent the phase-space distribution on a set of clouds: the only important piece of information is the metric intrinsically associated with each cloud of the previous approach, i.e. the information on how function $f(\boldsymbol{P}, t)$ is deformed by the motion. If one is indeed able to follow the properties of the motion up to some order in the neighbourhood of test particles, that we call metric elements, reconstructing the initial position $\boldsymbol{Q}(\boldsymbol{P}, t)$ of any point nearby is straightforward, and so is the calculation of $f(\boldsymbol{P}, t)$ from interpolation of initial conditions following Liouville's theorem.

Our algorithm, `Vlamet`, described in details in § 2, can thus be summarised as follows. It consists in sampling at all times the phase-space distribution function $f(\boldsymbol{P}, t)$ on a grid of fixed resolution and following f at second order in space and time using a predictor–corrector time integration scheme. The calculation of $f(\boldsymbol{P}, t)$ exploits directly Liouville's theorem, namely that f is conserved along the characteristics, i.e. is equal to its initial value $f_{ini} \equiv f(\boldsymbol{Q}, t_{ini})$ at Lagrangian position $\boldsymbol{Q}(\boldsymbol{P}, t)$. To follow these characteristics accurately, we define a set of elements of metric that are sparse sampling the grid. These elements of metric follow the Lagrangian equations of motion and carry during time local information about the deformation of phase space up to second order, to account for the curvature of phase-space structures building up during the course of the dynamics. To resample f at time t and at a given point $\boldsymbol{P}$ of phase space, we consider the closest elements of metric in Lagrangian space and reconstruct the initial position $\boldsymbol{Q}$ of a test particle coinciding with $\boldsymbol{P}$ using a second-order expansion of the geometry of the flow around the metric elements. Once the initial position $\boldsymbol{Q}$ is found, we just need to interpolate f_{ini} at position $\boldsymbol{Q}$ and attribute it to $f(\boldsymbol{P}, t)$.

At some time $t_{resample}$, the distortion of phase space is too strong to be simply represented at second order: $f(\boldsymbol{P}, t_{resample})$ is considered as a new initial condition and new isotropic elements of metric are created. Between initial time and $t_{resample}$, we use a cheap, second-order interpolation to reconstruct $f(\boldsymbol{P}, t)$, while resampling at $t = t_{resample}$ is more accurate using third-order splines. Indeed, although our algorithm is meant to be globally precise only at second order, a third-order description of the phase-space distribution function is essential at the moment of resampling to be able to conserve quantities such total energy and total mass in the long term. Furthermore, to preserve smoothness of the resampling procedure, we perform some interpolation in Lagrangian space between the initial positions $\boldsymbol{Q}$ proposed by each element of metric close to the point of interest. Indeed, different neighbouring metric elements can provide different answers. The procedure is illustrated by figure 1.

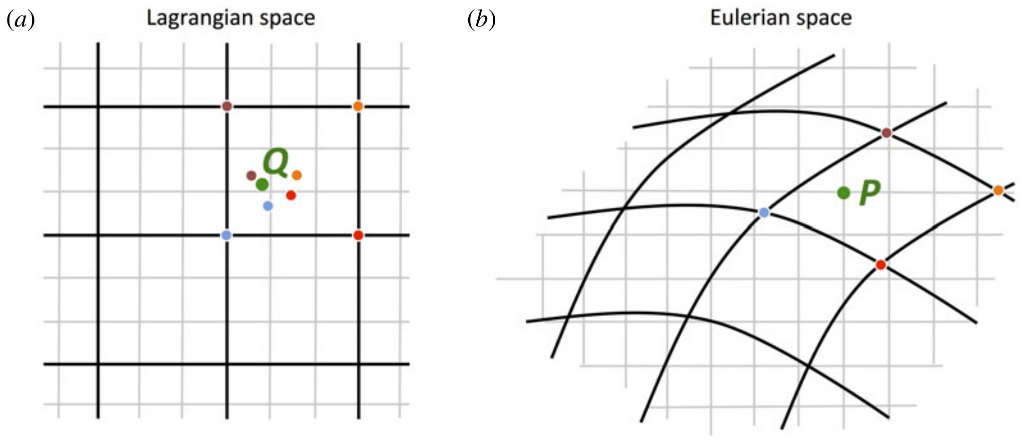


FIGURE 1. Schematic representation of the algorithm. At all times we would like to follow the phase-space distribution function $f(\boldsymbol{P}, t)$ sampled on a fine Eulerian grid, represented here in grey. To achieve this, we define local elements of metric initially sampling a sparser grid, as represented in black in (*a*). These elements of metric follow the motion and its distortions, as illustrated by the deformation of the black grid in (*b*). To reconstruct f at all times on the Eulerian grid, e.g. the green point in (*b*), one traces back in time the trajectory of a test particle associated with the green point. To compute the position of this particle during the transport phase, we just use the closest element of metric in Lagrangian space, here the blue point on (*a*). Once $\boldsymbol{Q}$ is calculated, we apply Liouville's theorem i.e. $f(\boldsymbol{P}, t) = f_{ini} \equiv f(\boldsymbol{Q}, t_{ini})$ where t_{ini} is the initial time, computing $f(\boldsymbol{Q}, t_{ini})$ with a second-order interpolation over the Lagrangian grid. Of course, at some point, $t = t_{resample}$, distortion of phase-space structures is too large to be only locally represented at second order, so one has to start again the process by considering $f(\boldsymbol{P}, t_{resample})$ as new initial conditions. At time $t_{resample}$, we proceed slightly differently to compute $f(\boldsymbol{P}, t)$ more accurately. Indeed, because our local representation of phase space is only valid up to second order, the reconstructed initial position proposed by each element of metric is slightly different, as illustrated by (*a*). To avoid an unsmooth representation of the reconstruction of the initial position $\boldsymbol{Q}$, we perform at $t = t_{resample}$ a more accurate interpolation of the initial position, here between the purple, the orange, the red and the blue points and a more accurate interpolation using third-order splines on the Lagrangian grid to compute $f(\boldsymbol{Q}, t_{ini})$.

In § 3, we thoroughly test the performance of our algorithm, by checking how a stationary solution is preserved and by simulating systems with Gaussian initial condition as well as random sets of halos. To demonstrate the potential improvements brought by our code `Vlamet` compared to traditional semi-Lagrangian approaches, we compare our algorithm to the traditional splitting method of Cheng & Knorr (1976) with third-order spline interpolation. Results are also tested against simulations performed with the entropy conserving waterbag code of Colombi & Touma (2014). The waterbag scheme, extremely accurate but very costly, is meant to provide a supposedly 'exact' solution. It is mainly used to test the convergence of `Vlamet` and of the splitting algorithm for the Gaussian initial condition. From the pure mathematical side, which is not really approached in this article, it is worth mentioning the recent work of Campos Pinto & Charles (2016), who discuss some convergence properties of a simplified version of our algorithm.

2. Algorithm

Now we describe the main steps of the algorithm, namely, (i) our second-order representation of the metric (§ 2.1), (ii) our interpolation schemes to reconstruct f (§ 2.2), (iii) our percolation algorithm to find the nearest element of metric to an Eulerian pixel $\boldsymbol{P}$ and the reconstruction of its Lagrangian counterpart $\boldsymbol{Q}$ using an interpolation between the alternatives proposed by the neighbouring elements of metric (§ 2.3), (iv) the equations of motion as well as their numerical implementation (§ 2.4) and finally, (v) the calculation of the force, its first and second derivatives, required to follow the second-order representation of the local metric during motion (§ 2.5).

While the actual implementation of the algorithm is performed in two-dimensional (2-D) phase space, generalisation to higher number of dimensions will be discussed in detail. Indeed, our algorithm is meant to improve other standard semi-Lagrangian solvers in terms of diffusion while being of comparable cost, and is designed in such a way that it can be easily generalised to four- and six-dimensional phase space.

2.1. *Second-order representation of the metric*

The Lagrangian position of each fluid element in phase space is defined by

$$\boldsymbol{Q} \equiv (q_x, q_v), \tag{2.1}$$

where q_x and q_v are its initial position and velocity, respectively. Here we consider 2-D phase space, so these quantities are scalar, but in general, they are vectors. In what follows, unless specified otherwise, all the calculations apply to the general case: one just has to replace q_x and q_v with $\boldsymbol{q}_x$ and $\boldsymbol{q}_v$. The Eulerian position at time t of an element of fluid in phase space is given by

$$\boldsymbol{P}(\boldsymbol{Q}, t) \equiv [x(q_x, q_v, t), v(q_x, q_v, t)], \tag{2.2}$$

where $x(q_x, q_v, t)$ and $v(q_x, q_v, t) = \partial x/\partial t$ are its position and velocity at time t. Initially

$$\boldsymbol{P}(\boldsymbol{Q}, t = 0) = \boldsymbol{Q}. \tag{2.3}$$

In our second-order approach, we need the deformation tensor,

$$\boldsymbol{T}(\boldsymbol{Q}, t) \equiv \frac{\partial \boldsymbol{P}}{\partial \boldsymbol{Q}}, \tag{2.4}$$

as well as the Hessian of the characteristics,

$$\boldsymbol{H}(\boldsymbol{Q}, t) \equiv \frac{\partial^2 \boldsymbol{P}}{\partial \boldsymbol{Q}^2}. \tag{2.5}$$

With the knowledge at all times of the position $\boldsymbol{P}_m$, the deformation tensor $\boldsymbol{T}_m$ and the Hessian $\boldsymbol{H}_m$ for each element of metric, we can predict, at second order, the position of an element of fluid initially sufficiently close to an element of metric:

$$\boldsymbol{P} \simeq \boldsymbol{P}_m + \boldsymbol{T}_m(\boldsymbol{Q} - \boldsymbol{Q}_m) + \tfrac{1}{2}(\boldsymbol{Q} - \boldsymbol{Q}_m)^{\mathrm{T}} \boldsymbol{H}_m (\boldsymbol{Q} - \boldsymbol{Q}_m). \tag{2.6}$$

Of course the series expansion could be developed further, but we restrict ourselves here to an algorithm at second order in space and time. Indeed, going beyond second order would be very costly in a higher number of dimensions, particularly six-dimensional phase space. The interesting bit, as we shall see later, is the inverse relation between Eulerian position and Lagrangian position, i.e. at second order

$$\boldsymbol{Q} \simeq \boldsymbol{Q}_m + \boldsymbol{T}_m^{-1}(\boldsymbol{P}-\boldsymbol{P}_m) - \tfrac{1}{2}\boldsymbol{T}_m^{-1}(\boldsymbol{P}-\boldsymbol{P}_m)^{\mathrm{T}}[\boldsymbol{T}_m^{-1}]^{\mathrm{T}}\boldsymbol{H}_m\boldsymbol{T}_m^{-1}(\boldsymbol{P}-\boldsymbol{P}_m). \tag{2.7}$$

In the 2-D phase-space case considered here, the deformation tensor $\boldsymbol{T}$ is a 4 element matrix with no special symmetry except that according to the Hamiltonian nature of the system its determinant should be conserved with time and equal to the initial value:

$$|\boldsymbol{T}| = 1, \tag{2.8}$$

at all times. Indeed

$$\boldsymbol{T}(\boldsymbol{Q}, t=0) = \boldsymbol{I}, \tag{2.9}$$

where $\boldsymbol{I}$ is the identity matrix. Note that this property is general, as volume is conserved in phase space, and is not specific to the 1-D dynamical case. We have,

$$J \equiv |\boldsymbol{T}| = \frac{\partial x}{\partial q_x}\frac{\partial v}{\partial q_v} - \frac{\partial x}{\partial q_v}\frac{\partial v}{\partial q_x} = 1, \tag{2.10}$$

hence $\boldsymbol{T}$ has only 3 independent elements. In practice we shall not impose equation (2.10) but checking to which extent it is verified can be used as a test of the good numerical behaviour of the code, or equivalently, as a test of its symplecticity.

In higher number of dimensions, phase-space volume conservation is not the only constraint. In fact it is a consequence of the symplectic nature of the system, which hence conserves Poincaré invariants, in particular symplectic forms of the kind

$$\mathcal{I} = \tfrac{1}{2}[\mathrm{d}\boldsymbol{P}_1]^{\mathrm{T}}\boldsymbol{S}\,\mathrm{d}\boldsymbol{P}_2, \tag{2.11}$$

where

$$\boldsymbol{S} \equiv \begin{pmatrix} \boldsymbol{0} & -\boldsymbol{I} \\ \boldsymbol{I} & \boldsymbol{0} \end{pmatrix} \tag{2.12}$$

is the symplectic matrix, while $\mathrm{d}\boldsymbol{P}_1$ and $\mathrm{d}\boldsymbol{P}_2$ correspond to 2 sides of an elementary triangle composed of three particles following the equations of motion in phase space. In particular,

$$\mathrm{d}\boldsymbol{P}_j = \frac{\partial \boldsymbol{P}}{\partial \boldsymbol{Q}}\,\mathrm{d}\boldsymbol{Q}_j. \tag{2.13}$$

So the generalisation of equation (2.10) to a higher number of dimensions provides the following constraints:

$$\left[\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{q}_x}\right]^{\mathrm{T}}\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{q}_x} - \left[\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{q}_x}\right]^{\mathrm{T}}\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{q}_x} = 0, \tag{2.14}$$

$$\left[\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{q}_v}\right]^{\mathrm{T}} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{q}_v} - \left[\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{q}_v}\right]^{\mathrm{T}} \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{q}_v} = 0, \tag{2.15}$$

$$\left[\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{q}_x}\right]^{\mathrm{T}} \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{q}_v} - \left[\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{q}_x}\right]^{\mathrm{T}} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{q}_v} = \boldsymbol{I}, \tag{2.16}$$

which can be again used to double check, *a posteriori*, the good behaviour of the code in the general case.

In the 2-D phase-space case considered here, the Hessian $\boldsymbol{H}$ is a $2 \times 2 \times 2$ tensor in which only 6 terms are independent if one just takes into account the symmetries in second derivatives, e.g. $\partial^2 x/(\partial q_x \partial q_v) = \partial^2 x/(\partial q_v \partial q_x)$. In four- and six- dimensional phase space, the same symmetries imply that among its $4 \times 4 \times 4 = 64$ and $6 \times 6 \times 6 = 216$ elements, the Hessian includes 40 and 126 independent terms, respectively.

While exploiting the Hamiltonian nature of the system would allow us to carry less information, we chose for each element of metric to store the phase-space coordinates, the deformation tensor and the Hessian. The calculations above show that this ends up in carrying $2+4+6=12$, $4+16+40=60$ and $6+36+126=168$ numbers for each element of metric in two-, four- and six-dimensional phase space, respectively. This might look a large number, particularly in four- and six-dimensional phase space, but one has to keep in mind the fact that elements of metric are much sparser than the grid used to compute the phase-space distribution function. For instance, a factor 3 in each direction of the spacing between elements of metric compared to the grid cell size implies that there is one element of metric for $3^6 = 729$ grid sites. Another issue is the cost of applying the inverse mapping (2.7), but it can be reduced considerably by precomputing for each element of metric the matrixes $\boldsymbol{T}_m^{-1}$ and $[\boldsymbol{T}_m^{-1}]^{\mathrm{T}} \boldsymbol{H}_m \boldsymbol{T}_m^{-1}$ prior to any interpolation of the phase-space distribution function using the inverse mapping.

2.2. *Interpolation schemes*

We use two interpolation schemes. Between two coarse steps, a period of time during which the phase-space distribution function is transported along motion using the same elements of metric at each time step, we use a simple interpolation of f based on local fitting of a quadratic hypersurface. Then, during remapping, i.e. when new elements of metric are set, f is interpolated with a third-order spline in order to have a more accurate reconstruction of phase space. The advantage of using second-order interpolation during the transport phase is that generalisation to six-dimensional phase space is relatively cheap, since only 28 terms contribute in this case to such an interpolation. The loss of accuracy due to the relative roughness of the reconstruction, which is not even continuous, has little consequence for the dynamics because most of the information is recovered during the resampling phase. This will be tested and demonstrated in § 3. On the other hand, because our method does not allow for splitting, $4^6 = 4096$ terms contribute to each grid site when resampling f in six dimensions with the third-order spline interpolation, if one assumes that this latter takes approximately 4 operations per grid site in one dimension. This rather costly step is thus approximately $4096/24 \simeq 171$ times more expensive than what is expected when performing a full time step in the standard splitting algorithm*. It is however performed rarely, hence making the algorithm of potentially comparable cost to the

*This calculation assumes that third-order spline interpolation is used in the splitting algorithm. In a split fashion, this interpolation is performed individually on each axis, which corresponds, in six dimension, to $6 \times 4 = 24$ operations per grid site for a full time step, or $9 \times 4 = 36$ operations if synchronisation between velocities and positions is performed.

standard semi-Lagrangian approach, with the advantage of being, in principle, less diffusive.

Now we provide the details for the second-order interpolation we use, followed by the standard third-order spline interpolation.

2.2.1. *Second-order interpolation*

The general procedure for second-order interpolation consists in understanding that a second-order hypersurface requires respectively 6, 15 and 28 points in 2-D, 4-D and 6-D phase space. What we have access to are the values $F(\boldsymbol{G})$ of the phase-space distribution function on a grid $\boldsymbol{G}$, e.g. in two dimensions $\boldsymbol{G}=(i,j)$ where i and j are integers, corresponding to the following Lagrangian positions in phase space,

$$\boldsymbol{Q}_G=(i\Delta x+q_{x,mid}, j\Delta v+q_{v,mid}), \tag{2.17}$$

where Δx and Δv are the steps chosen for the grid (Eulerian or Lagrangian) that remain, in the current version of the code, fixed during runtime and $(q_{x,mid}, q_{v,mid})$ corresponds to the middle of the Lagrangian computational domain. The bounds of the computational domain are then related to maximum values n_x and n_v of $|i|$ and $|j|$.

This is the way we proceed in 2-D phase space: given a point $\boldsymbol{Q}=(q_x, q_v)$ in phase space, we first find the nearest grid point (i_n, j_n) to it, then consider the cross composed of (i_n+1, j_n), (i_n, j_n+1), (i_n-1, j_n), (i_n, j_n-1) and (i_n, j_n) on which the interpolation must coincide with F. The last point (i,j) needed is taken to be the one for which a square composed of 3 points of the cross and (i,j) contains (q_x, q_v) (see figure 2). With this choice of the points of the grid for the interpolation, we have, at second order,

$$\begin{aligned}f(q_x, q_v)=&F(i_n, j_n)(1-\mathrm{d}q_x^2-\mathrm{d}q_v^2)\\&+\tfrac{1}{2}F(i_n+1, j_n)\,\mathrm{d}q_x(1+\mathrm{d}q_x)+\tfrac{1}{2}F(i_n-1, j_n)\,\mathrm{d}q_x(1-\mathrm{d}q_x)\\&+\tfrac{1}{2}F(i_n, j_n+1)\,\mathrm{d}q_v(1+\mathrm{d}q_v)+\tfrac{1}{2}F(i_n-1, j_n)\,\mathrm{d}q_v(1-\mathrm{d}q_v)\\&+[F(i_c+1, j_c+1)+F(i_c, j_c)-F(i_c+1, j_c)-F(i_c, j_c+1)]\,\mathrm{d}q_x\,\mathrm{d}q_v,\end{aligned} \tag{2.18}$$

with

$$(i_n, j_n)=(\lfloor(q_x-q_{x,mid})/\Delta x\rceil, \lfloor(q_v-q_{v,mid})/\Delta v\rceil), \tag{2.19}$$

$$(i_c, j_c)=(\lfloor(q_x-q_{x,mid})/\Delta x\rfloor, \lfloor(q_v-q_{v,mid})/\Delta v\rfloor), \tag{2.20}$$

$$\mathrm{d}q_x=(q_x-q_{x,mid})/\Delta x-i_n, \tag{2.21}$$

$$\mathrm{d}q_v=(q_v-q_{v,mid})/\Delta v-j_n, \tag{2.22}$$

where $\lfloor\cdot\rceil$ and $\lfloor\cdot\rfloor$ are respectively the nearest integer and the integer part functions. Note obviously that when defining the edges of the computing domain, one makes sure that the interpolation is defined everywhere. In the present implementation, f is supposed to be different from zero on a compact domain in phase space. To avoid artificial enlarging of the computing domain during runtime due to diffusion and aliasing, the computing domain is restricted to a rectangular region containing all the points where

$$|f|>f_{th}, \tag{2.23}$$

with f_{th} a small number, typically $f_{th}=10^{-6}$. Note the absolute value in equation (2.23), because our algorithm does not enforce positivity of f and better total mass conservation is obtained by keeping the regions where $f<0$.

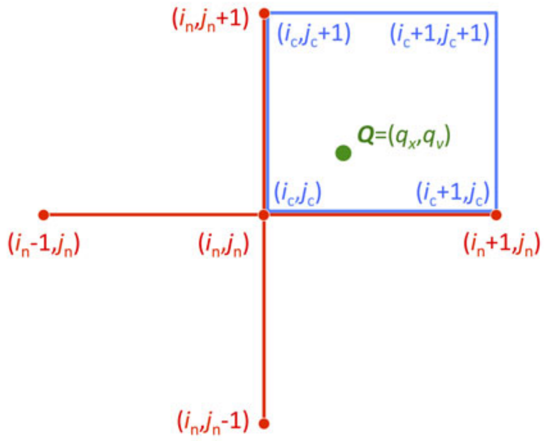


FIGURE 2. Procedure employed to select the points of the grid used to perform second-order interpolation of the phase-space distribution function at position $\boldsymbol{Q}$ in $2D$-dimensional phase space. One first finds the nearest grid point $\boldsymbol{G}_n$ to $\boldsymbol{Q}$, $\boldsymbol{G}_n = (i_n, j_n)$ for $D = 1$, and select all the segments of the grid composed of 3 grid points and centred on $\boldsymbol{G}_n$: one obtains the red cross for one dimension. Then one selects, for each plane passing through a pair of such segments the four elements of the grid so that the square composed of these grid elements contains the projection of $\boldsymbol{Q}$: one of them, $(i_c + 1, j_c + 1)$ on the figure, provides one additional point for the interpolation.

The generalisation of this second-order interpolation to higher number of dimensions is straightforward: the most difficult is to choose the grid points through which the hypersurface must pass. To achieve this, similarly as in two dimensions, we first find the nearest grid point $\boldsymbol{G}_n$ to $\boldsymbol{Q}$. Then take the collection of all segments centred on $\boldsymbol{G}_n$ and composed of 3 points of the grid (the cross defined above in two dimensions): this provides us in total 9 and 13 points respectively in four and six dimensions. Then, project $\boldsymbol{Q}$ onto the planes defined by each pair of such segments and find the corresponding grid site in this plane such that the square composed of this grid site plus 3 contained in the 2 segments contains the projection of $\boldsymbol{Q}$ and add it to our list of points defining the second-order hypersurface. There are 6 and 15 such points in four and six dimensions, respectively, and we thus end up using this procedure with respectively 15 and 28 points as needed to interpolate our hypersurface, with straightforward generalisation of equation (2.18) to perform the interpolation.

2.2.2. *Standard third-order spline*

The third-order interpolation we use employs standard cubic B-splines as very clearly described in e.g. Crouseilles, Latu & Sonnendrücker (2009), so we do not provide the detail here, except the edge conditions, which are

$$f(q_x, q_v) = 0, \quad \frac{\partial f}{\partial q_x} = 0, \quad |q_x| = q_{x,mid} + n_x \Delta x, \tag{2.24a,b}$$

$$f(q_x, q_v) = 0, \quad \frac{\partial f}{\partial q_v} = 0, \quad |q_v| = q_{v,mid} + n_v \Delta v, \tag{2.25a,b}$$

and this can be trivially generalised to four- or six-dimensional phase space.

2.3. *Tracing back the characteristics in Lagrangian space: percolation algorithm*

To determine the value of $f(\boldsymbol{P}, t)$, we have to trace the trajectory of a test particle coinciding with $\boldsymbol{P}$ back to its Lagrangian, initial position $\boldsymbol{Q}$ (the position at time of remapping, in general). To do this, we need to find the closest element of metric to this test particle and use equation (2.7) to compute $\boldsymbol{Q}$. However, the closest element of metric, from the dynamical point of view, has to be found in Lagrangian space, which complicates the procedure. Furthermore, two close test particles might end up associated with different elements of metric that will propose slightly different mappings, implying that function $\boldsymbol{Q}(\boldsymbol{P})$ is discontinuous if no supplementary treatment is performed. While this is not too problematic during the transport phase, it can have dramatic consequences on the long term during the remapping step which requires a high level of precision when interpolating the phase-space distribution function with third-order splines. Hence, each time a full remapping of f is performed, one needs to find not one but several elements of metric in the neighbourhood of a test particle to be able to interpolate between the proposed Lagrangian positions provided by each of these metric elements so that function $\boldsymbol{Q}(\boldsymbol{P})$ stays smooth: this is the most complex part of the algorithm, because enforcing full continuity of the interpolation is actually not trivial since it has to be performed in Lagrangian space. The percolation algorithm however allows us to define in an unambiguous way regions of influence of each element of metric which in turn will allow us to make function $\boldsymbol{Q}(\boldsymbol{P})$ smooth.

We assume that elements of metric are initially set on a Lagrangian grid covering the computing domain, $\boldsymbol{M} = (i_m, j_m)$, corresponding to Lagrangian positions

$$\boldsymbol{Q}_m(i_m, j_m) = (\Delta x_m i_m + q_{x,mid}, \Delta v_m j_m + q_{v,mid}), \tag{2.26}$$

with

$$\Delta x_m \gtrsim \Delta x, \quad \Delta v_m \gtrsim \Delta v. \tag{2.27a,b}$$

The actual computing domain changes subsequently with time. To estimate it at each time step we use the positions of the metric elements and we define a rectangle with lower left and upper right corners respectively given by $[\min_{i_m,j_m}(x_m), \min_{i_m,j_m}(v_m)]$ and $[\max_{i_m,j_m}(x_m), \max_{i_m,j_m}(v_m)]$. This means that the element of metric coverage should be wide enough to avoid missing regions where f should be not equal to zero. We therefore allow for an extra layer of metric elements on all the sides of the Lagrangian computing domain. Note thus that the calculation of the computing domain is far from optimal, which can have dramatic consequences for the cost of the algorithm in six dimensions. In order to fix this problem, one would need to define a more complex shape-dependent computational domain by e.g. constructing a k-d tree, i.e. a spatial decomposition of phase space with nested cubes. Such a k-d tree could also constitute the basis for an adaptive mesh refinement algorithm following in an optimal way not only the computational domain, but also all the details of the phase-space distribution function when needed. However such a sophistication is not needed in our 2-D implementation so we stick to the rectangular computing domain for now.

2.3.1. *Percolation*

To compute the region of influence of each element of metric (i_m, j_m) with coordinates (x_m, v_m), we percolate around it on the Eulerian computational grid starting from its nearest grid point

$$(i, j) = (\lfloor (x_m - x_{mid})/\Delta x \rceil, \lfloor (v_m - v_{mid})/\Delta v \rceil), \tag{2.28}$$

with (x_{mid}, v_{mid}) being the time-dependent position of the middle of the Eulerian computational domain. The initial Lagrangian position $\boldsymbol{Q}(i, j, i_m, j_m)$ of this point is estimated with equation (2.7) using the element of metric we started from as a reference. Note that another neighbouring element of metric (i'_m, j'_m) would propose a slightly different value for the initial position of the test particle associated with Eulerian grid site (i, j), hence the dependence on (i_m, j_m) of $\boldsymbol{Q}$. Then one defines the function

$$\begin{aligned} \delta\boldsymbol{Q}(i, j, i_m, j_m) &\equiv [\delta q_x(i, j, i_m, j_m), \delta q_v(i, j, i_m, j_m)] && (2.29)\\ &\equiv \boldsymbol{Q}(i, j, i_m, j_m) - \boldsymbol{Q}_m(i_m, j_m), && (2.30) \end{aligned}$$

where $\boldsymbol{Q}_m(i_m, j_m)$ is the Lagrangian position of the element of metric, to find out which neighbours (i'_m, j'_m) of the element of metric are susceptible to be closer to the true initial $\boldsymbol{Q}(i, j)$ in Lagrangian space. Because the calculation of $\boldsymbol{Q}(i, j)$ is not exact we need to introduce the concept of uncertainty. Considering all the neighbours of the current element of metric under consideration

$$(i'_m, j'_m) : \begin{cases} i'_m \in [i_m - 1, i_m + 1], \\ j'_m \in [j_m - 1, j_m + 1], \\ (i'_m, j'_m) \neq (i_m, j_m), \end{cases} \tag{2.31}$$

we are able to chose the one which proposes the smallest value of

$$d[\delta\boldsymbol{Q}(i, j, i_m, j_m)] \equiv \sqrt{(\delta q_x/\Delta x)^2 + (\delta q_v/\Delta v)^2}, \tag{2.32}$$

where the dependence on (i, j, i_m, j_m) of δq_x and δq_v has been made implicit. If there is a candidate element of metric (i'_m, j'_m) closer than (i_m, j_m) to the Eulerian grid element (i, j) using this Lagrangian distance, the percolation process is stopped. Otherwise, one continues (bond) percolation i.e. proceeds with neighbouring grid sites $(i + 1, j)$, $(i - 1, j)$, $(i, j + 1)$, $(i, j - 1)$ unless they have been already processed.

Note that the percolation procedure just defined above is not always free of aliasing: it can be successful only if the separation between elements of metric is sufficiently large compared to the grid element size and if their zone of influence is not too distorted. Testing the validity of the percolation process is relatively simple, by checking if some points inside the computing domain have been left unexplored. If it is the case, percolation is performed again locally until all the unexplored points are processed, which induces a small additional cost in the algorithm.

To speed up the process, which is not really relevant here in 2-D phase space but crucial in a higher number of dimensions, we reduce the range of investigation of values of i'_m to $[i_{inf}, i_{sup}]$ where

$$\delta q_x < -\alpha \Delta x_m : i_{inf} = -1,\ i_{sup} = 0, \tag{2.33}$$

$$-\alpha \Delta x_m \leqslant \delta q_x < \alpha \Delta x_m : i_{inf} = 0,\ i_{sup} = 0, \tag{2.34}$$

$$\alpha \Delta x_m \leqslant \delta q_x : i_{inf} = 0,\ i_{sup} = 1, \tag{2.35}$$

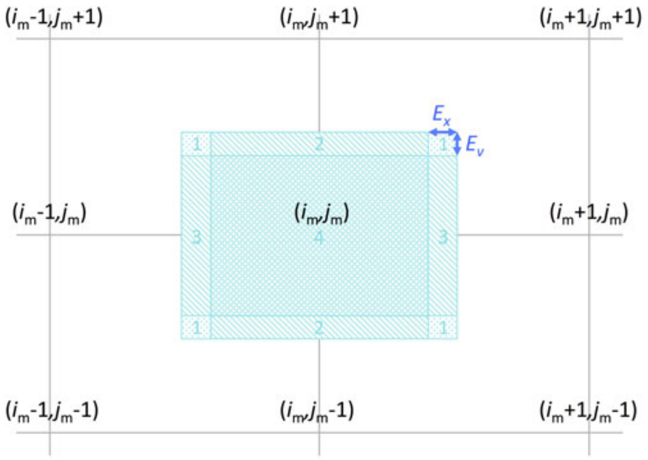


FIGURE 3. Scheme determining the range of investigation of neighbouring elements of metric during the percolation process in Lagrangian space. We consider a pixel $\boldsymbol{P}$ in Eulerian space that we know to be already close to an element of metric (i_m, j_m) because it is a direct neighbour of a pixel known to be in the region of influence of (i_m, j_m). We remap it to its Lagrangian counterpart $\boldsymbol{Q}$ using the element of metric (i_m, j_m). Then we aim to find the closest element of metric to it. To do so and speed up the process, four cases are considered, to take into account the small errors on $\boldsymbol{Q}$ (in particular the fact that the reconstructed $\boldsymbol{Q}$ depends on the element of metric under consideration, see figure 1), that we write $\boldsymbol{E}_{map} = (E_x, E_v)$. In case 4, $\boldsymbol{Q}$ is close enough to the element of metric: we do not need to investigate any neighbour and decide it to belong to the region of influence of (i_m, j_m). In cases 2 and 3, we need only to investigate another element of metric, e.g. $(i_m, j_m + 1)$ in the upper part of the figure to decide if it closer to its own estimate of $\boldsymbol{Q}$ using the distance defined in equation (2.32). Finally case 1 is the most costly since we have to investigate three neighbouring element of metric, but it is seldom considered.

with the dependence on (i, j, i_m, j_m) of δq_x still implicit and α is a parameter close to 1/2. Similar conditions are set to define an interval $[j_{inf}, j_{sup}]$. The parameter α is introduced to take into account the uncertainty $\boldsymbol{E}_{map} = (E_x, E_v)$ on the remapping (2.7). This is illustrated by figure 3. However, even though we can actually estimate $\boldsymbol{E}_{map}$ as described below in § 2.3.3, we do not use it to calculate α which is fixed from the beginning. Indeed, our implementation favours performance over perfect accuracy, to make a future extension to 6-D phase space possible: while it is better to find in Lagrangian space the actual nearest element of metric to a test particle, this is not a must and the algorithm can still work without it, at the cost of some loss of accuracy on the long run.

To illustrate the result of the percolation process, figure 4 displays, at $t = 25$, the region of influence of each element of metric at the end of a cycle (i.e. just before resampling with new elements of metric) for the simulation II with Gaussian initial condition described in § 3.

2.3.2. *Interpolation of the Lagrangian position of a test particle*

We now explain how the Lagrangian position of a test particle can be calculated in a smooth way with proper interpolation in Lagrangian space between predicates given by each of neighbouring elements of metric. We define the following mapping

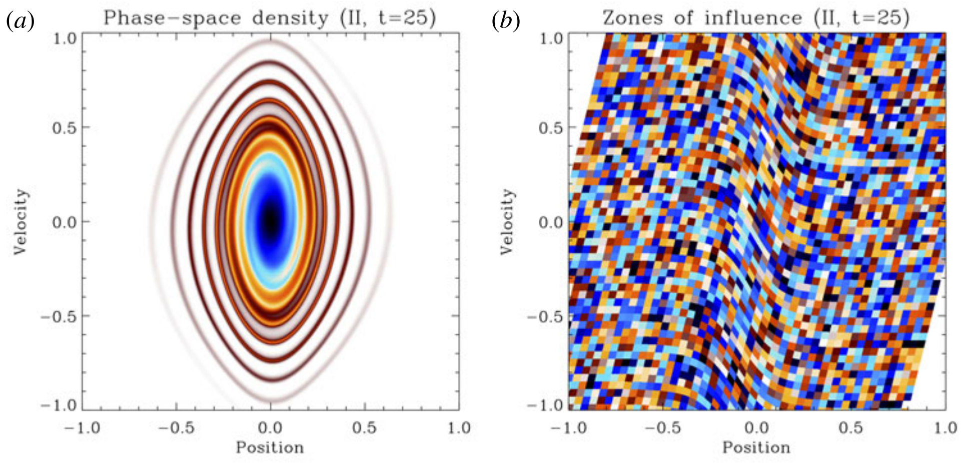


FIGURE 4. Gaussian run and percolation. (*a*) The phase-space distribution function is shown at $t = 25$ for a simulation with Gaussian initial conditions and parameters corresponding to item II of table 1 in § 3. (*b*) The corresponding Lagrangian zone of influence of each element of metric obtained from the percolation process is drawn at the end of a cycle (just before full resampling with new metric elements). To visualise the results more clearly, a random colour has been associated with each element of metric.

function $\boldsymbol{Q}_r(\boldsymbol{P})$ from Eulerian to Lagrangian space

$$\boldsymbol{Q}_r(\boldsymbol{P}) \equiv \frac{\displaystyle\sum_{i'_m,j'_m} W\{2d[\delta\boldsymbol{Q}(\boldsymbol{P}, i'_m, j'_m)]\}\boldsymbol{Q}(\boldsymbol{P}, i'_m, j'_m)}{\displaystyle\sum_{i'_m,j'_m} W\{2d(\delta\boldsymbol{Q}(\boldsymbol{P}, i'_m, j'_m)]\}}, \tag{2.36}$$

where function $\boldsymbol{Q}(\boldsymbol{P}, i_m, j_m)$ is the obvious extension to the continuum of its discrete counterpart $\boldsymbol{Q}(i, j, i_m, j_m)$ defined previously, likewise for $\delta\boldsymbol{Q}(\boldsymbol{P}, i_m, j_m)$, and $W(x)$ is a smooth isotropic kernel function compactly supported. Clearly, function $\boldsymbol{Q}_r(\boldsymbol{P})$ as defined above is at least continuous. Our choice for $W(x)$ is the spline of Monaghan (1992):

$$W(x) \equiv \begin{cases} 1 - \frac{3}{2}x^2 + \frac{3}{4}x^3, & x \leqslant 1, \\ \frac{1}{4}(2-x)^3, & 1 < x \leqslant 2, \\ 0, & x > 2, \end{cases} \tag{2.37}$$

which preserves the second-order nature of the Taylor expansion around each metric element position. Our choice of $W(x)$ allows us to restrict the range of investigation of values of i'_m and j'_m to the respective intervals $[i_m - 1, i_m + 1]$ and $[j_m - 1, j_m + 1]$ where (i_m, j_m) corresponds to the closest element of metric to point $\boldsymbol{P}$ as computed in the previous paragraph, § 2.3.1, for sites (i, j) of a grid†. Here, we indeed only need to consider points $\boldsymbol{P} = (\Delta x i + x_{mid}, \Delta v j + v_{mid})$ associated with such an Eulerian grid. To reduce furthermore the number of elements of metric investigated to perform

†Obviously, we assume here that the errors in the remapping (2.7) are smaller than the inter-element of metric separation.

the interpolation (2.36) while still taking into account small variations related to the previously discussed uncertainty $\boldsymbol{E}_{map}$ on the calculation of $\boldsymbol{Q}$, we restrict the search to the intervals $[i_{min}, i_{max}]$ and $[j_{min}, j_{max}]$ defined as follows, with the same notations as in equations (2.33), (2.34) and (2.35):

$$\delta q_x < -\beta \Delta x_m : i_{min} = -1,\ i_{max} = 0, \tag{2.38}$$

$$-\beta \Delta x_m \leqslant \delta q_x < \beta \Delta x_m : i_{min} = -1,\ i_{max} = 1, \tag{2.39}$$

$$\beta \Delta x_m \leqslant \delta q_x : i_{min} = 0,\ i_{max} = 1, \tag{2.40}$$

where β is a small parameter accounting for the error $\boldsymbol{E}_{map}$, similarly for j_{min} and j_{max}, with the additional constraint $i'_m j'_m = 0$ if $i_{min} i_{max} = -1$ or $j_{min} j_{max} = -1$ to gain even more time.

Again, similarly as stated at the end of § 2.3.1, the above algorithm is not guaranteed to provide a smooth interpolation if the error on the remapping associated with the metric elements becomes larger than some threshold. However, this should have little consequence for the algorithm accuracy and the gain of speed in high-dimensional phase space is considerable with such a procedure. Indeed, if $\boldsymbol{E}_{map}$ becomes too large, the loss of accuracy on the characteristics themselves is much more dramatic than small discontinuities introduced on their interpolation.

2.3.3. *Estimates of errors on the Lagrangian position and choice of the margin parameters α and β*

In the two subsections above, we defined two parameters α and β but did not provide any numerical values. In principle these parameters could be user defined, but we choose to fix them:

$$\alpha = \tfrac{\sqrt{3}}{4} \simeq 0.43, \tag{2.41}$$

$$\beta = \tfrac{1}{2}\left(1 - \tfrac{\sqrt{3}}{2}\right) \simeq 0.067. \tag{2.42}$$

The idea behind this choice is the following. Firstly, according to figure 5, since we do not want to consider any diagonal element of metric if $i_{min} i_{max} = -1$ or $j_{min} j_{max} = 1$, we must have for full consistency,

$$\beta \leqslant 1 - \tfrac{\sqrt{3}}{2} \simeq 0.13. \tag{2.43}$$

Because there is some error on the reconstructed value of $\boldsymbol{Q}$, we take half this value, assuming that

$$E_x/\Delta x_m,\ E_y/\Delta v_m \leqslant E_{max} \equiv \tfrac{1}{2}\left(1 - \tfrac{\sqrt{3}}{2}\right) \simeq 0.067. \tag{2.44}$$

Hence $\alpha = 1/2 - E_{max}$ is given by equation (2.41) to take into account this upper bound for the errors. With this setting, the algorithm is fully consistent only if the uncertainty on the remapped trajectories remains small enough, but even if $E_x/\Delta x_m$ or $E_v/\Delta v_m$ are larger than E_{max}, it still works. It is simply the case that function $\boldsymbol{Q}_r(\boldsymbol{P})$ is not guaranteed to be completely smooth anymore.

Note that in our algorithm, a global error $E_{map} = \sqrt{(E_x/\Delta x_m)^2 + (E_v/\Delta v_m)^2}$ is estimated during the percolation process by comparing the value of $\boldsymbol{Q}$ proposed by a neighbouring element of metric to the value of $\boldsymbol{Q}$ proposed by the element of metric under consideration using the Lagrangian distance d defined in equation (2.32). Such

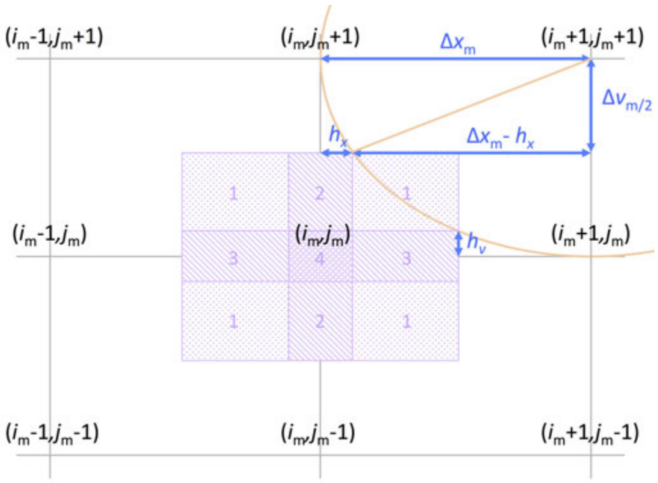


FIGURE 5. Scheme determining the number of neighbouring elements of metric to consider for the interpolation (2.36) used to reconstruct the Lagrangian coordinate $\boldsymbol{Q}$ of a test particle $\boldsymbol{P}$ in the region of influence of metric element (i_m, j_m). This figure also justifies equation (2.43). Each element of metric has an ellipsoid region of influence (ocher ellipse) for computing the weights in equation (2.36). Outside this area, the element of metric does not contribute. Therefore, in the absence of uncertainty on the reconstruction of the Lagrangian coordinate, one can safely define 4 kinds of regions, as shown on the figure, with $h_x = (1 - \sqrt{3}/2)\Delta x_m$ and $h_v = (1 - \sqrt{3}/2)\Delta v_m$, steaming from the ellipse equations $(1 - h_x/\Delta x_m)^2 + 1/4 = 1$ and $1/4 + (1 - h_v/\Delta v_m)^2 = 1$. In region 4, only the cross composed of $(i_m - 1, j_m)$, $(i_m + 1, j_m)$, $(i_m, j_m - 1)$, $(i_m, j_m + 1)$ and (i_m, j_m) contributes to the interpolation (2.36). In regions 2 and 3, only 4 elements of metric contribute, namely $(i_m - 1, j_m)$, $(i_m + 1, j_m)$, $(i_m, j_m + 1)$ and (i_m, j_m) for the upper area 2. In regions labelled by 1, four elements of metric contribute, e.g. (i_m, j_m), $(i_m + 1, j_m)$, $(i_m, j_m + 1)$, $(i_m + 1, j_m + 1)$ in the upper right quadrant. In the presence of uncertainty, we need to change the extensions of these regions to take it into account, as discussed further in the main text.

an error can in principle be used to force global remapping if E_{map} exceeds some threshold, for instance E_{max} to preserve full smoothness of function $\boldsymbol{Q}_r(\boldsymbol{P})$. While this is an option in our code, we do not use it for the tests performed in § 3. Indeed, we found difficult to find some reliable criterion on E_{map} and preferred to keep the number n_s of time steps between full resampling fixed and see how the results depend on the choice of n_s, being aware of the fact that this is potentially suboptimal. Note also that E_x and E_v could be estimated as the residuals from the third-order description of the local metric elements, but this is very costly in highly dimensional phase space.

2.4. *Equations of motion*

In 2-D phase space, the equations characterising the evolution of an element of metric with time are given by the following expressions with some obvious new notations:

$$\frac{\partial \boldsymbol{x}}{\partial t} = \boldsymbol{v}, \tag{2.45}$$

$$\frac{\partial \boldsymbol{v}}{\partial t} = \boldsymbol{a}, \tag{2.46}$$

$$\frac{\partial T_{i,j}}{\partial t} = T_{i+D,j}, \quad i \leqslant D, \tag{2.47}$$

$$\frac{\partial T_{i+D,j}}{\partial t} = \sum_{r=1}^{D} \frac{\partial a_i}{\partial x_r} T_{r,j}, \tag{2.48}$$

$$\frac{\partial H_{i,j,k}}{\partial t} = H_{i+D,j,k}, \quad i \leqslant D, \tag{2.49}$$

$$\frac{\partial H_{i+D,j,k}}{\partial t} = \sum_{r=1}^{D} \frac{\partial a_i}{\partial x_r} H_{r,j,k} + \sum_{r,s=1}^{D} \frac{\partial^2 a_i}{\partial x_r \partial x_s} T_{r,j} T_{s,k}, \tag{2.50}$$

where $\boldsymbol{a}$ is the acceleration calculated by solving Poisson's equation. Note thus that we need to have access at all times to the gradient and the Hessian of the gravitational force. In these equations we have dropped the 'm' subscript for simplicity.

The actual time step implementation uses a splitting algorithm with a drift phase during half a time step where only spatial positions are modified, followed with a kick phase where only velocities are modified during a full time step using the acceleration computed after the drift, followed again by a drift during half a time step, where spatial positions are modified using the new velocities.

The first drift step or predictor is thus written

$$\boldsymbol{x}(t + \Delta t/2) = \boldsymbol{x}(t) + \tfrac{1}{2}\boldsymbol{v}(t)\Delta t, \tag{2.51}$$

$$T_{i,j}(t + \Delta t/2) = T_{i,j}(t) + \tfrac{1}{2} T_{i+D,j}(t)\Delta t, \quad i \leqslant D, \tag{2.52}$$

$$H_{i,j,k}(t + \Delta t/2) = H_{i,j,k}(t) + \tfrac{1}{2} H_{i+D,j,k}\Delta t, \quad i \leqslant D, \tag{2.53}$$

after which Poisson's equation is solved to compute the acceleration $\boldsymbol{a}(t + \Delta t/2)$, its gradient and its Hessian. To compute the acceleration, we first need to estimate the phase-space distribution function $f(\boldsymbol{x}, \boldsymbol{v}, t + \Delta t/2)$ on the Eulerian grid. To map f back to initial conditions (or previous stage after full remapping), we use the elements of metric as transformed by the predictor step.

The kick phase reads

$$\boldsymbol{v}(t + \Delta t) = \boldsymbol{v}(t) + \boldsymbol{a}(t + \Delta t/2)\Delta t, \tag{2.54}$$

$$T_{i+D,j}(t + \Delta t) = T_{i,j}(t) + \sum_{r=1}^{D} \frac{\partial a_i}{\partial x_r} T_{r,j}(t + \Delta t/2)\Delta t, \tag{2.55}$$

$$\begin{aligned} H_{i+D,j,k}(t + \Delta t) &= H_{i+D,j,k}(t) + \sum_{r=1}^{D} \frac{\partial a_i}{\partial x_r} H_{r,j,k}(t + \Delta t/2)\Delta t \\ &\quad + \sum_{r,s=1}^{D} \frac{\partial^2 a_i}{\partial x_r \partial x_s} T_{r,j}(t + \Delta t/2) T_{s,k}(t + \Delta t/2)\Delta t. \end{aligned} \tag{2.56}$$

Finally the second drift phase, or corrector, is expressed as follows:

$$\boldsymbol{x}(t + \Delta t) = \boldsymbol{x}(t + \Delta t/2) + \tfrac{1}{2}\boldsymbol{v}(t + \Delta t)\Delta t, \tag{2.57}$$

$$T_{i,j}(t + \Delta t) = T_{i,j}(t + \Delta t/2) + \tfrac{1}{2} T_{i+D,j}(t + \Delta t)\Delta t, \quad i \leqslant D, \tag{2.58}$$

$$H_{i,j,k}(t+\Delta t) = H_{i,j,k}(t+\Delta t/2) + \tfrac{1}{2} H_{i+D,j,k}(t+\Delta t)\Delta t, \quad i \leqslant D. \tag{2.59}$$

With our predictor–corrector scheme, it is possible to adopt a slowly variable time step. In this case, we use the following dynamical constraint (see, e.g. Alard & Colombi 2005; Colombi & Touma 2014),

$$\Delta t \leqslant \frac{C_{dyn}}{\sqrt{\rho_{max}}}, \quad C_{dyn} \ll 1, \tag{2.60}$$

where $\rho_{max} = \max_x \rho(x)$ is the maximum value of the projected density, $\rho(x) \equiv \int f(x, v)\, \mathrm{d}v$. Note that for the tests performed in § 3, we used a constant time step, but this should not have critical effects on the numerical results.

2.5. *Calculation of the force and its derivatives*

In practice, the force is calculated on a grid and its gradient and Hessian estimated by simple finite difference methods.

In our 2-D phase-space implementation, the acceleration is calculated at discrete points i with coordinate $x_i = x_{mid} + \Delta x i$ corresponding to the projection of the Eulerian phase-space grid. To estimate it, we first calculate the projected mass $M(i)$ inside each pixel i by summing up the phase-space density calculated at each point (i, j) of the Eulerian phase-space grid. In the present work we consider isolated systems in an empty space, which means that the acceleration at a given point x is simply proportional to the total mass at the right of point x minus the total mass at the left of x. This implies, within some trivial factor depending on code units

$$a(i) = M_{total} - M_{left}(i) - M_{left}(i-1), \tag{2.61}$$

with

$$M_{left}(i) = \sum_{m \leqslant i} M(m), \tag{2.62}$$

and where M_{total} is the total mass of the system. Note thus that the mass in pixel i does not contribute to $a(i)$ as it is supposed to be distributed symmetrically inside it[‡]. In a larger number of dimensions, Poisson's equation is more complex to solve but this can be done on a fixed grid resolution with Fast Fourier Transform (FFT) methods, whether the system is spatially periodic or isolated as supposed here.

Our estimates of the first and second derivatives of the acceleration at the grid points are written

$$\frac{\partial a}{\partial x_i} = \frac{a(i+1) - a(i-1)}{\Delta x}, \tag{2.63}$$

$$\frac{\partial^2 a}{\partial x_i^2} = \frac{a(i+1) + a(i-1) - 2a(i)}{(\Delta x)^2}, \tag{2.64}$$

which is equivalent to fitting the acceleration with a second-order polynomial on the three successive points corresponding to $i-1$, i and $i+1$.

[‡]Note that a potentially better procedure could consist in using a staggered mesh for computing the acceleration, in which the nodes would correspond to boundaries of each pixel of the grid.

Designation	$\varDelta$	$\varDelta_m$	n_s	t_{CPU}
I	0.002	0.02	25	1.0
II	0.002	0.04	25	1.0
III	0.002	0.01	25	1.2
IV	0.002	0.005	25	1.4
V	0.002	0.02	100	1.1
VI	0.002	0.02	50	1.1
VII	0.002	0.02	10	1.2
VIII	0.002	0.02	5	1.5
IX	0.002	0.02	1	3.2
X	0.001	0.01	25	4.0
XI	0.004	0.04	25	0.4
XII	0.0005	—	—	8.8
XIII	0.00707	—	—	4.5
XIV	0.001	—	—	2.3
XV	0.00141	—	—	1.2
XVI	0.002	—	—	0.7

TABLE 1. Parameters of the simulations performed in this work, namely the grid cell size, $\varDelta = \Delta x = \Delta y$, the initial separation between each metric element, $\varDelta_m = \Delta x_m = \Delta y_m$, the number of subcycles, n_s, between resamplings and the approximate total Central Processing Unit (CPU) time expressed in units of the time spent for simulation I. The calculation of t_{CPU} was performed for the simulations with initial conditions corresponding to the random set of stationary clouds as described in § 3.2, but the results do not change drastically for other initial conditions. A space separates the runs performed with `Vlamet` (I–XI) from those performed with the standard splitting algorithm (XII–XVI). Additionally, the time step Δt was chosen constant, given by $\Delta t = 0.01$, 0.01 and 0.0025 for the stationary, Gaussian and random set of halos, respectively. The two last choices come from the constraints on the time step obtained in the waterbag runs by Colombi & Touma (2014).

To compute the force at an arbitrary point of the computing domain, we perform dual TSC (triangular shaped cloud, see e.g. Hockney & Eastwood 1988) interpolation from the grid, i.e.

$$a(x) = \tfrac{1}{2}\left(\tfrac{1}{2} - w\right)^2 a(i-1) + \left(\tfrac{3}{4} - w^2\right) a(i) + \tfrac{1}{2}\left(\tfrac{1}{2} + w\right)^2 a(i+1), \tag{2.65}$$

$$i = \lfloor (x - x_{mid})/\Delta x \rceil, \tag{2.66}$$

$$w = (x - x_{mid})/\Delta x - i, \tag{2.67}$$

and likewise for its gradient and its second derivative. Note that we also tested linear interpolation, which leads in practice to nearly the same results as TSC for all the numerical tests we discuss in next section.

Clearly, the way we estimate the force and its successive derivatives is probably too naive but actually works pretty well for all the experiments we did so we did not attempt to improve this part of our algorithm. However, this should be investigated in a future work.

3. Numerical experiments

To test the method, we performed an ensemble of simulations of which the parameters are listed in table 1. In § 3.2, we compare the results obtained with `Vlamet`

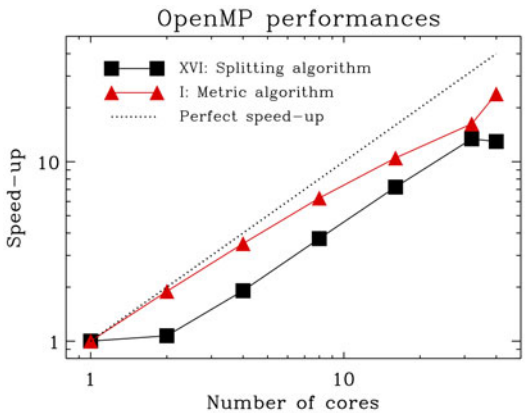


FIGURE 6. Accelerations of our implementations of `Vlamet` (in red) and of the splitting algorithm (in black) as functions of the number of cores on a shared memory architecture with OpenMP library. The plot is performed for the parameters corresponding to runs I and XVI in table 1, starting from an ensemble of stationary clouds. Perfect speed up is represented as a dotted line.

to the standard splitting algorithm of Cheng & Knorr (1976) with cubic B-spline interpolation as well as the extremely accurate entropy conserving waterbag simulations of Colombi & Touma (2014, hereafter CT14). Before going into the details of these comparisons, we first provide in § 3.1 additional technical information on the performances of `Vlamet` and of the splitting algorithm[§]. In particular, we discuss parallelisation issues as well as extension to a higher number of dimensions.

3.1. *Performance: parallel scaling and computing cost*

Both `Vlamet` and the splitting algorithm are parallelised on shared memory architecture using OpenMP library, but no attempt has been made to optimise cash memory, so there is still room for improvement. Parallel scaling is displayed in figure 6, which shows that the two codes present a reasonable speed up, greater than 10 on 32 cores. The scaling is slightly better for the metric code than for the splitting algorithm but this comparison is irrelevant without further optimisation. The main point here is to show that OpenMP parallelisation is rather straightforward for `Vlamet`, while it is already well known that both distributed and shared memory parallelisations are not a problem for the splitting algorithm (see, e.g. Crouseilles *et al.* 2009).

As an additional note, but this is something we leave for future work, there is in principle no particular difficulty in parallelising `Vlamet` on a distributed memory architecture with the Message Passing Interface (MPI) library. Except for solving Poisson's equation and for cubic B-spline interpolation that we will discuss, all the operations performed in the algorithm, including the percolation process, are of local nature, hence trivially compatible with domain decomposition. In four- and six-dimensional phase space, introducing a k-d tree structure to focus on important

[§]Comparisons to the waterbag code in terms of performances are irrelevant here, since the future goal is to generalise `Vlamet` to higher number of dimensions in phase space. The waterbag method, extremely accurate but already costly in 2-D phase space, becomes prohibitive in a higher number of dimensions. For instance, in 4-D phase space, it consists in following, in an adaptive fashion, ensembles of hypersurfaces sampled with tetrahedra (see, e.g. Sousbie & Colombi 2016).

regions of phase space is also quite standard in parallel programming. The solution to Poisson equation is well known in many parallel implementations. For instance there exist parallel fast Fourier transform libraries such as FFTW to compute the gravitational potential on a fixed resolution grid. As for spline interpolation, a domain decomposition implementation is possible using the method proposed by Crouseilles *et al.* (2009). In this case, interpolation is localised to each domain by using Hermite boundary conditions between the domains with an *ad hoc* reconstruction of the derivatives. In this implementation, each domain would for instance correspond to a leaf of the k-d tree mentioned above. Of course, as long as we did not actually implement the algorithm on distributed memory architecture, this discussion stays at the speculative level.

In terms of total computational time, interesting conclusions can be derived from examination of the fifth column of table 1, which indicates, for each run, the approximate total CPU cost in units of simulation I. Neglecting the Poisson equation resolution step, both `Vlamet` and the splitting algorithm are expected in general to scale approximately like the number $n_{cells} \propto \varDelta^{-2D}$ of cells in the grid representing phase space, where D (here equal to unity) is the number of dimensions of space. This is approximately the case in table 1 if $\varDelta$ is small enough. The cost also increases for `Vlamet` when resamplings of the phase-space distribution function are more frequent (in practice, if $n_s \lesssim 10$ for the runs we did) or when the element of metric density increases (in practice, if $\varDelta_m \lesssim 10\varDelta$ for the runs we did).

When it comes to really compare `Vlamet` to the standard splitting algorithm, we focus on single thread runs, assuming that full parallelisation is equally optimal on both algorithms. We noticed that with the same spatial resolution $\varDelta$, `Vlamet` is approximately $\alpha_{slower} = 1.4$–2 times slower than the splitting algorithm for $n_s \gtrsim 25$ and $\varDelta_m/\varDelta \gtrsim 2.5$. How these estimates generalise to four and six dimensions remains an open question, but we can try to infer some rough estimates. From now on we restrict to 6-D phase space. In this case, the critical step in `Vlamet` is the cubic spline interpolation during the resampling phase, which is expected to be approximately 171 times more costly than a step of the standard splitting algorithm, as discussed in introduction of § 2.2. The rest of the algorithm, on the other hand, should perform comparably well to the splitting scheme, i.e. should be α_{loss} times slower with α_{loss} of the order of unity. Hence we expect, in six dimensions, a factor $\alpha^{6D}_{slower} \simeq (n_s \times \alpha_{loss} + 171)/n_s$ between `Vlamet` and the standard splitting algorithm, that is α^{6D}_{slower} ranging from about 2 in the optimistic cases (e.g. $\alpha_{loss} \simeq 1$ and $n_s = 200$) to approximately 20 in the pessimistic ones (e.g. $\alpha_{loss} = 10$ and $n_s = 15$). While a factor as large as 20 might seem potentially prohibitive, we remind the reader that our code is supposed to be less diffusive than the splitting algorithm, as we shall illustrate next, hence should require less spatial resolution. A simple factor 2 in resolution corresponding to a gain of approximately $2^6 = 64$ in speed and memory would make `Vlamet` extremely competitive.

3.2. *Test runs: comparisons between codes*

Before going further, it is important here to specify the units in which we are solving the Vlasov–Poisson equations. Our choice is the same as in CT14, i.e. we are studying the following system of equations:

$$\frac{\partial f}{\partial t} + v\frac{\partial f}{\partial x} - \frac{\partial \phi}{\partial x}\frac{\partial f}{\partial v} = 0, \tag{3.1}$$

$$\frac{\partial^2 \phi}{\partial x^2} = 2\rho = 2\int f(x, v, t)\,\mathrm{d}v. \tag{3.2}$$

To test `Vlamet`, three kinds of initial conditions were considered, namely the thermal equilibrium solution, Gaussian initial conditions and a random set of stationary clouds, as detailed below.

(i) The thermal equilibrium solution (Spitzer 1942; Camm 1950; Rybicki 1971) reads

$$f_S(x, v) = \frac{\rho_S}{\left[\mathrm{ch}\left(\sqrt{\sqrt{2\pi}\rho_S/\sigma_S x}\right)\right]^2} \exp\left[-\frac{1}{2}\left(\frac{v}{\sigma_S}\right)^2\right]. \tag{3.3}$$

In practice, we set

$$f_{ini}(x, v) = f_S(x, v) g_\eta[f_S(x, v)], \tag{3.4}$$

with the apodising function g_η given by

$$g_\eta(y) = \eta \max\left[1 + 2\ \mathrm{th}\left(\frac{y-\eta}{\eta}\right), 0\right]. \tag{3.5}$$

For the runs performed in this paper, we took $\rho_S = 4$, $\sigma_S = 0.2$ and $\eta = 0.02$, which means that the maximum projected density of the system is $\rho_{max} \simeq 2$. The harmonic dynamical time corresponding to such a density is, in our units,

$$T_{dyn} = \frac{2\pi}{\sqrt{2\rho_{max}}}. \tag{3.6}$$

The simulation was run up to $t = 100$, which corresponds to $100/T_{dyn} \simeq 32$ dynamical times. Thermal equilibrium is stable and hence represents a crucial test for a code, which has to be able to maintain it accurately. Figure 7 shows how `Vlamet` performs for runs I, II, III and IV, when considering the difference $\delta f = f(x, v, t) - f_{ini}(x, v)$ at $t = 100$, i.e. checks the behaviour of the code when changing the distance $\varDelta_m$ between metric elements while keeping resolution fixed to $\varDelta = 0.002$. For comparison, a curve corresponding to the splitting algorithm run XVI is also shown at same resolution $\varDelta$. As expected, `Vlamet` performs very well, as long as $\varDelta_m/\varDelta$ remains sufficiently small, $\varDelta_m/\varDelta \lesssim 10$, to avoid too large errors on the reconstruction of Lagrangian positions from the metric elements. We also checked that $|\delta f|$ decreases when augmenting the spatial resolution of the simulation. Note that the splitting algorithm seems to perform even better than `Vlamet`, as suggested by the turquoise curve in figure 7(*b*). A residual error remains in all cases, since there is a clear convergence to a non-zero δf. From performing a run similar to I but with a twice larger time step and another one with a truncation parameter η twenty times smaller, we see that this small residual can be attributed to the finite value of the time step and especially the (rather strong) truncation of f with apodising function g_η, which implies that the initial conditions constructed this way do not correspond exactly to thermal equilibrium. Although we do not have any rigorous mathematical argument to support the claim that follows, decreasing Δt and the value of η should make this residual tend to zero, both for `Vlamet` and the splitting algorithm, provided that resolution $1/\varDelta$ becomes also arbitrarily large and the ratio $\varDelta_m/\varDelta$ is kept sufficiently small.

(ii) Gaussian initial conditions are given by

$$f_{ini}(x,\, v) = G(x,\, v), \quad x^2+v^2 \leqslant \mathcal{R}^2, \tag{3.7}$$

$$= G(x,\, v)\max\left[1+2\ \mathrm{th}\left(\frac{\mathcal{R}-\sqrt{x^2+v^2}}{\eta_G}\right),0\right], \quad x^2+v^2 > \mathcal{R}^2, \tag{3.8}$$

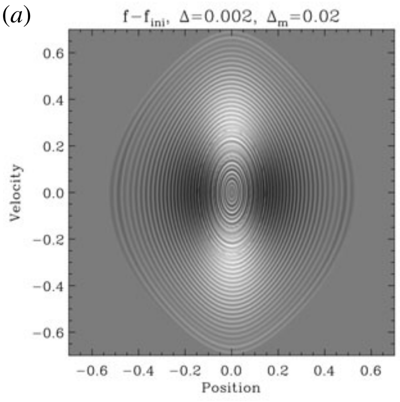
with

$$G(x,\, v) \equiv \rho_G \exp\left(-\frac{1}{2}\frac{x^2+v^2}{\sigma_G^2}\right). \tag{3.9}$$

In practice, to compare the output of `Vlamet` to the waterbag result of CT14, we take $\mathcal{R}=1$, $\rho_G=4$, $\sigma_G=0.2$ and $\eta_G=0.02$, which makes the total mass of the system approximately equal to unity for a Gaussian truncated at 5 sigma. The measured maximum projected density of this system, is, after relaxation, $\rho_{max}=2.95$, so simulating this system up to $t=100$ corresponds to following it during approximately 39 dynamical times according to equation (3.6). Figure 8 shows, at various times, the phase-space distribution function obtained from run I and the waterbag run `Gaussian84` of CT14. The agreement between both simulations seems visually close to perfect. A more careful examination of the details of the phase-space distribution function shows small deviations which can be reduced by augmenting the resolution of the `Vlamet` run. This is illustrated by figure 9, which displays on a small interval, the projected density for runs I, X, XI (different resolutions $\varDelta$ but fixed $n_s=25$ and fixed ratio $\varDelta_m/\varDelta=10$) and `Gaussian84`. What is particularly interesting in this figure are the turquoise curves corresponding to runs performed with the splitting algorithm, namely XII, XIV and XVI. We notice a very good match between `Vlamet` and the splitting algorithm but only if this latter is run at twice the resolution of `Vlamet`! This figure already demonstrates the significantly weaker level of diffusion of `Vlamet` compared to the splitting algorithm.

(iii) The ensemble of stationary clouds corresponds to a set of halos following the stationary solution (3.3), created exactly as in CT14. In particular, their profile follows equation (3.3) with $\rho_S=6$ and their velocity dispersion, σ_S, ranges in the interval $[0.005, 0.1]$. The components are added on the top of each other in phase space, to obtain the desired initial distribution function $f_{ini}(x,\, v)$, which is apodised as in equation (3.4) with $\eta=0.05$. The interesting fact about this system is that it was found experimentally by CT14 to be chaotic. In figure 10, the phase-space distribution function is shown at various times for simulation I and run `Random` of CT14. At $t=25$, the agreement between both simulations is impressive, despite the fact that the phase-space distribution function is sampled with only three waterbags in the `Random` run of CT14. At $t=80$, the system presents remarkable structures: some halos have been destroyed by tidal effects and a complex filamentary structure has grown. At this time, the `Vlamet` and the waterbag runs diverge from each other, but remain still in agreement at the macroscopic level. The fact that the simulations diverge from each other is not surprising since the run performed by CT14 samples the phase-space distribution function with only 3 waterbags. The differences are also amplified by the intrinsic chaotic nature of the system. Note that the `Vlamet` run is subject, at this point, to significant aliasing effects. For instance, the minimum value of the measured phase-space distribution function on figure 10(e) is equal to $f_{min}=-0.38$, which

(*a*)



(*b*)

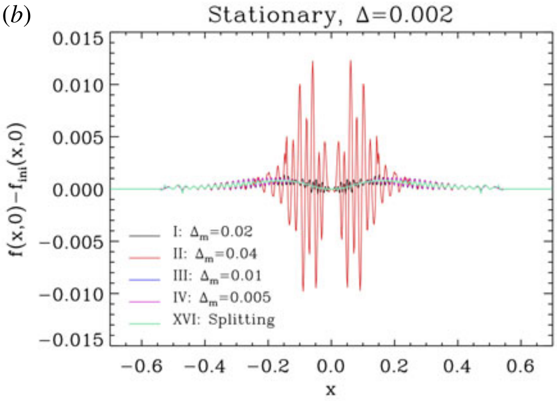


FIGURE 7. Deviation from stationarity for simulations with $\varDelta = 0.002$. (*a*) The difference δf between the phase-space density measured at $t = 100$ and the initial one for simulation I. Darker regions correspond to larger values of δf, which ranges in the interval $\delta f \in [-0.0014, 0.0014]$. (*b*) The value of $\delta f(x, 0)$ is plotted for simulations I, II, III, IV (i.e. varying inter-element of metric distance $\varDelta_m$ while keeping spatial resolution $\varDelta = 0.002$ fixed) and XVI. Note that the purple and blue curves nearly superpose on each other.

is of rather large magnitude given the maximum theoretical value of f, $f_{max} = 6$. These aliasing effects are inherent to the spline interpolation procedure we are using. Importantly, aliasing effects do not affect the dynamical properties of the system: increasing the resolution or decreasing it (runs X and XI) does not change the results significantly, despite the chaotic nature of the system, except at very small scales close to spatial resolution, of course. As illustrated below quantitatively, similarly to the Gaussian case discussed above, simulations with the splitting algorithm provide comparable results to `Vlamet` but only if performed at better resolution.

To perform more quantitative tests, we checked for conservation of total energy, which can be conveniently written

$$E = \frac{1}{2}\int v^2 f(x, v, t)\,\mathrm{d}x\,\mathrm{d}v - \frac{1}{4}\int [a(x, t)^2 - M_{total}^2]\,\mathrm{d}x, \tag{3.10}$$

where $a(x, t)$ is the gravitational force and M_{total} the total mass. These integral are computed directly as sums on the grid used to sample the phase-space distribution function and the force. In addition, we check for conservation of the following Casimir

$$S = -\int f(x, v)\log|f(x, v)|\,\mathrm{d}x\,\mathrm{d}v, \tag{3.11}$$

which is numerically estimated the same way as total energy. This quantity would reduce to Gibbs entropy if positivity of the phase-space distribution function was preserved. While we know it is not rigorously correct, we shall still designate S by entropy. We also tested for total mass conservation, as well as L^1 and L^2 norms, but do not show the results here to avoid superfluous content. Indeed, energy and entropy were found in practice to be sufficiently representative of the system when it comes to test the accuracy of `Vlamet`.

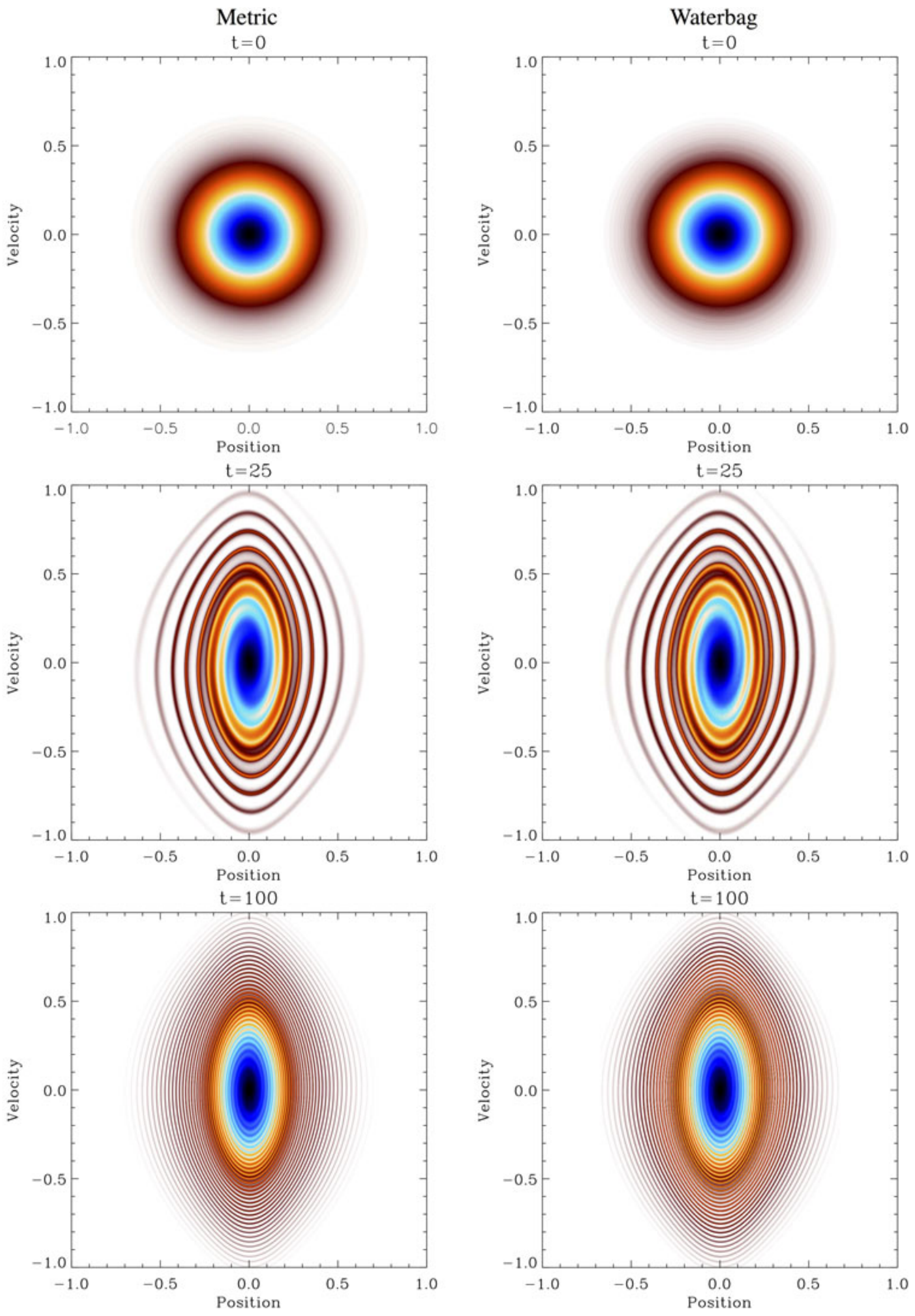


FIGURE 8. Phase-space density in the simulations with Gaussian initial conditions. The simulation I with the metric method is compared to a waterbag run of Colombi & Touma (2014). In the latter case, the phase-space distribution function is represented with 84 waterbags.

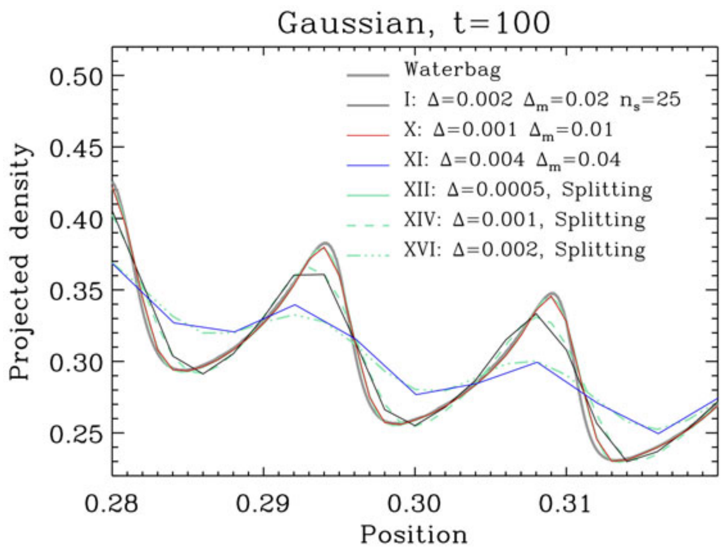


FIGURE 9. Projected density in the simulations with Gaussian initial conditions: detail. The simulations performed with `Vlamet`, I, X and XI are compared to the results obtained with the splitting algorithm, XII, XIV and XVI, as well as a waterbag run of Colombi & Touma (2014), which is supposed to represent the 'exact' solution. Note that while the waterbag measurements are displayed at the fine grained level, the results from the `Vlamet` and the splitting algorithm runs are shown at the grid resolution, $\varDelta$.

Figures 11–13 show E and S in the simulations we performed for the stationary, the Gaussian and the random set of halos, respectively. These figures allow us to test thoroughly the performance of `Vlamet` and to compare it to the splitting algorithm. They are supplemented with figure 14, which displays the corresponding maximum mismatch between the Lagrangian positions predicates from neighbouring metric elements, as defined in § 2.3.3.

The most striking result when examining figures 12 and 13 is the superior behaviour of `Vlamet` compared to the standard splitting method with respect to entropy, in agreement with the preliminary conclusions driven from the analysis of figure 9. Indeed, to obtain comparable entropy conservation in `Vlamet` and the splitting algorithm, it is required to have a higher spatial resolution in the latter. Of course, the results depend on the choice of n_s and of the ratio $\Delta x_m/\varDelta$. For instance, `Vlamet` runs with $n_s = 25$ and $\Delta x_m/\varDelta = 10$ perform as well as the splitting algorithm with twice the spatial resolution. But if n_s is increased to 100, it is required to increase the resolution by a factor as large as 2.8 (figure 13*d*) or 4 (figure 12*d*) in the splitting algorithm to perform as well as `Vlamet`. The price to pay might be energy conservation, which is in general not as good in `Vlamet` as in the splitting algorithm, that preserves energy at a level better than $\sim 10^{-5}$ regardless of initial conditions and spatial resolution. However, energy conservation violation in `Vlamet` can be kept under control with the appropriate combination of parameters n_s and ratio $\varDelta_m/\varDelta$, as discussed further below. On the same ground, we notice, when examining figure 11, that the splitting algorithm seems to preserve better the thermal equilibrium solution than `Vlamet`, as already noticed in figure 7(*b*), unless n_s or/and $\varDelta_m/\varDelta$ are chosen small enough. However, one has to be aware that the smoothness of the thermal equilibrium solution provides a very good ground for the splitting algorithm.

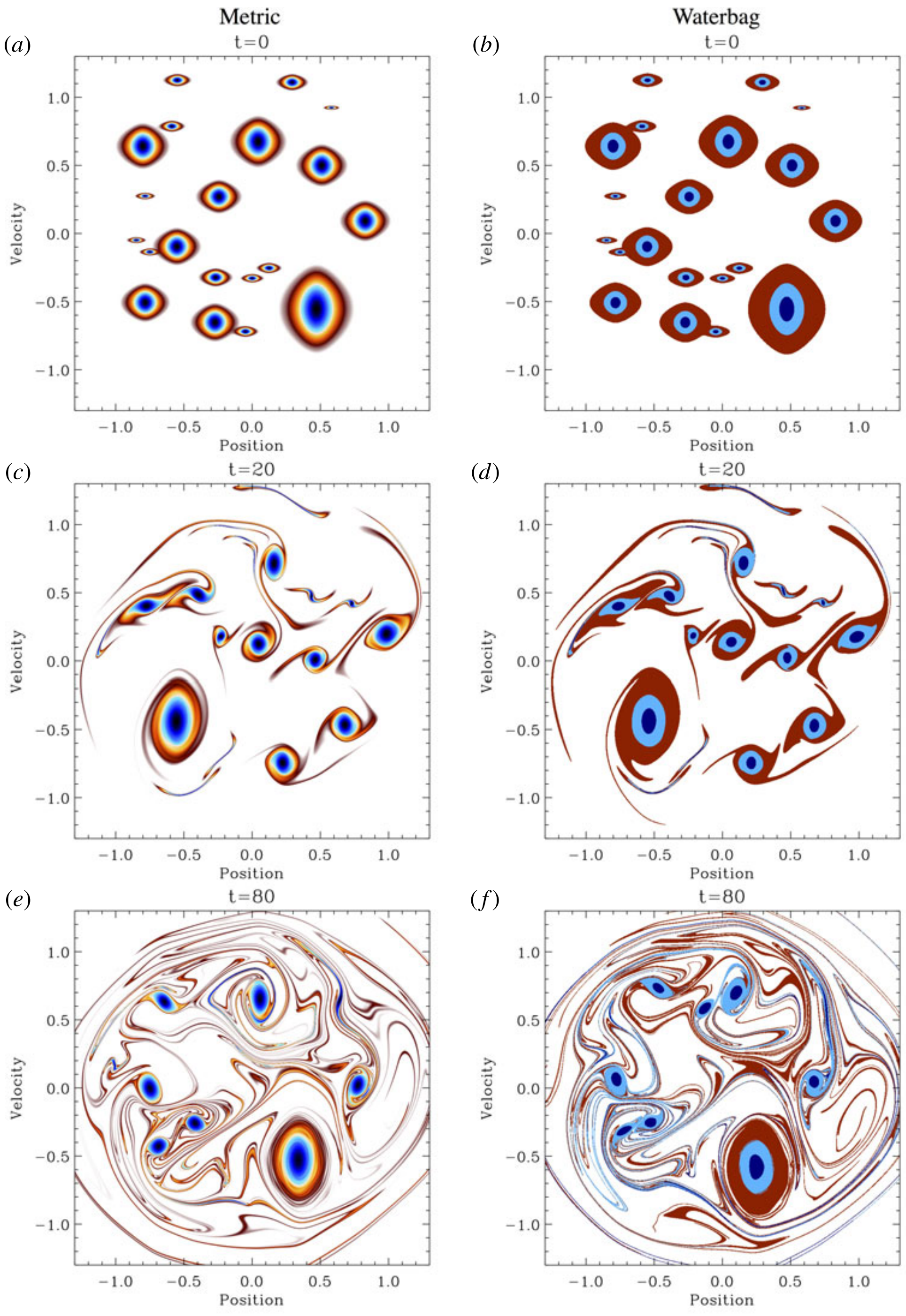


FIGURE 10. Phase-space density in the simulations with random initial conditions. The simulation I with the metric method is compared to a waterbag run of Colombi & Touma (2014). In the latter case, the phase-space distribution function is represented with only with three waterbag levels.

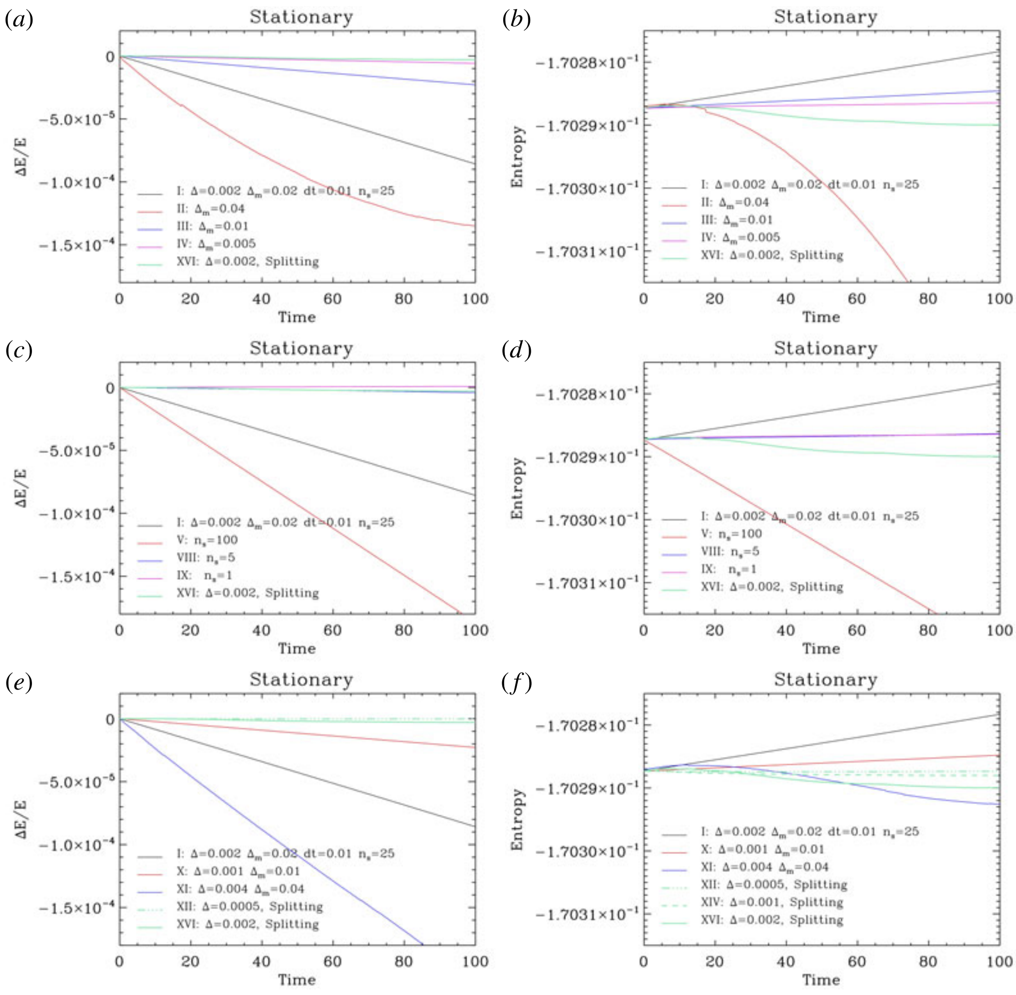


FIGURE 11. Conservation of energy (*a*,*c*,*e*) and entropy (*b*,*d*,*f*) in the stationary runs. Each line of panels corresponds to varying one parameter of the simulations as listed in table 1. In the first line of panels, we vary the elements of metric sampling, $\varDelta_m = \Delta x_m = \Delta v_m$, while keeping other parameters, in particular spatial resolution $\varDelta = \Delta x = \Delta v$, fixed. In the second line of panels, it is the number of time steps n_s between each resampling which changes. Finally, in the last line of panels, overall resolution is changed, i.e. both $\varDelta$ and $\varDelta_m$ while keeping the ratio $\varDelta_m/\varDelta$ fixed. For comparison, results from the splitting algorithm are also displayed as turquoise curves as indicated on each panel. In the latter case, the only parameter of interest is spatial resolution $\varDelta$.

Further investigation of figures 11–14 allows us to study more in detail how `Vlamet` behaves with the various control parameters:

(1) Increasing overall resolution, i.e. decreasing $\varDelta = \Delta x = \Delta y$ and $\varDelta_m = \Delta x_m = \Delta y_m$ by some factor improves both energy conservation and entropy conservation, as can be seen from last line of panels of all the figures, as expected. Note that adopting $\Delta x = \Delta y$ and $\Delta x_m = \Delta y_m$ as we did in this work is not necessarily optimal, but is a natural choice in the code units if the total mass of the system is of the order of unity.

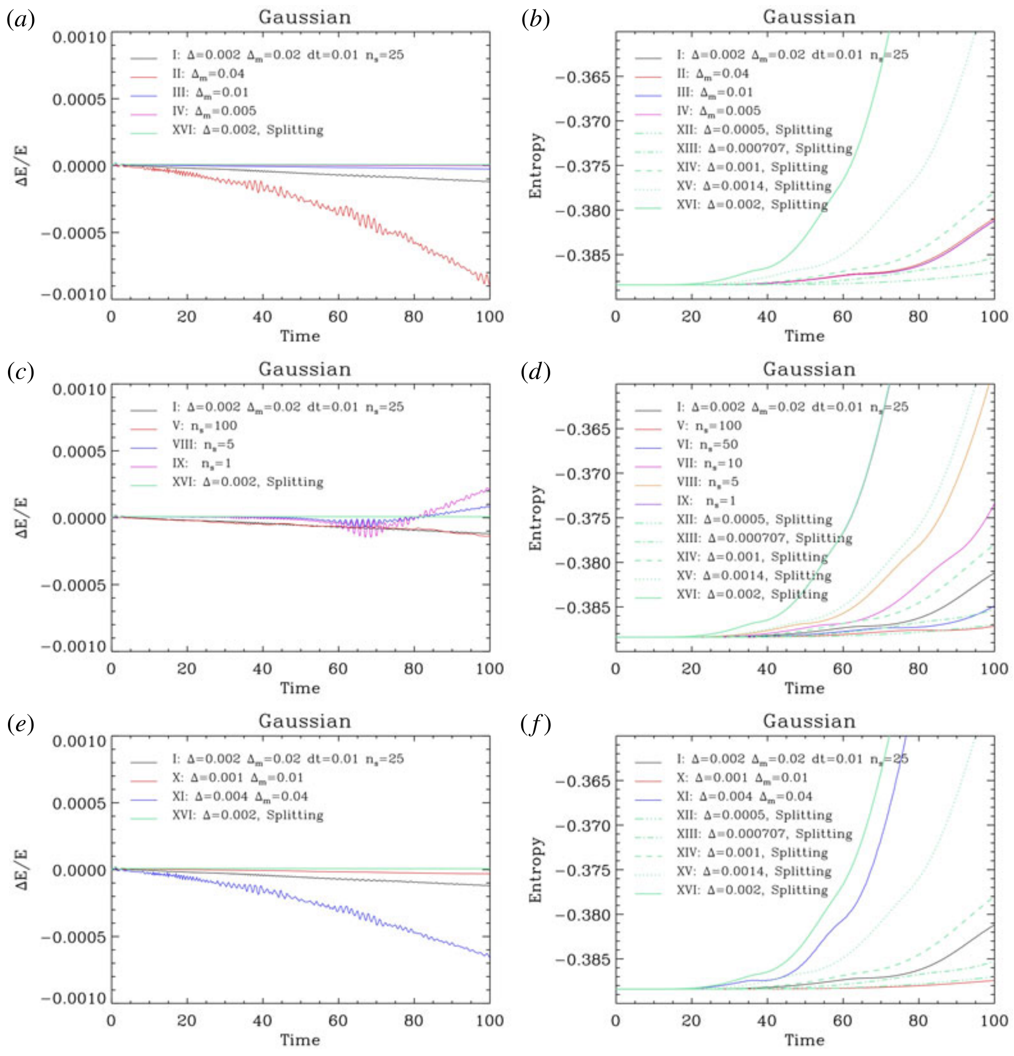


FIGURE 12. Conservation of energy (*a*,*c*,*e*) and entropy (*b*,*d*,*f*) in the Gaussian runs, similarly as in figure 11. In (*a*,*c*,*e*), energy conservation of the splitting algorithm is plotted only for run XVI, but other runs (XII, XIII, XIV and XV) would provide nearly indistinguishable results given the interval of values investigated.

(2) Changing the resampling frequency parameter n_s affects both entropy and energy, but to a much larger extent the former than the latter. In particular, increasing n_s up to 100 clearly improves entropy at an acceptable cost for energy conservation, of which the violation remains below the $\sim 5 \times 10^{-4}$ level, as can be seen from the second line of panels of figures 11–13. This is not surprising: performing less frequent resamplings of the phase-space distribution function decreases diffusion and aliasing, which is good for entropy conservation, but it worsens energy conservation, because (i) the error on the reconstruction of Lagrangian coordinates increases (middle line of panels in figure 14) and (ii) the phase-space distribution function is sampled more approximately during subcycles. While

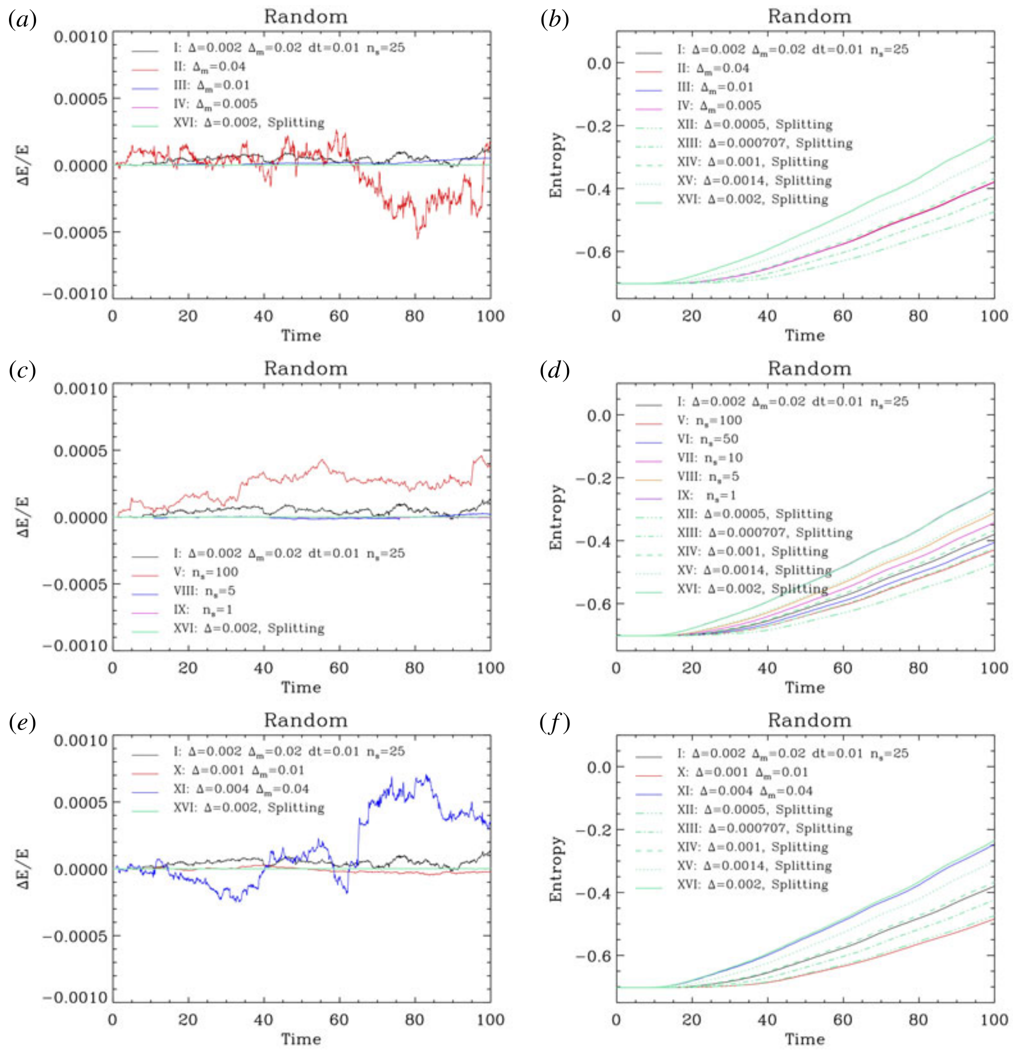


FIGURE 13. Conservation of energy (*a*,*c*,*e*) and entropy (*b*,*d*,*f*) in the random set of halos runs, similarly as in figure 11. Again, energy conservation of the splitting algorithm is plotted only for run XVI, because other runs would provide nearly indistinguishable results.

effect (i) can be reduced by augmenting the metric elements density as discussed below, effect (ii) always remains, which prevents arbitrarily large values of n_s.

(3) Sparse sampling the elements of metric, i.e. increasing Δ_m, augments the error in the Lagrangian coordinates reconstruction (first line of panels in figure 14). This affects energy conservation but not so much the entropy (except for the stationary simulations, which nevertheless preserve entropy at a level better than 10^{-3}). Entropy is indeed mainly sensitive to spatial resolution, i.e. the value of Δ, and as stated just above, to the frequency of resamplings. For the experiments we did, we find that the ratio Δ_m/Δ should not exceed a factor 10, and the smaller it is, the better the results, obviously. For instance, using $\Delta_m/\Delta \lesssim 5$ leads to energy

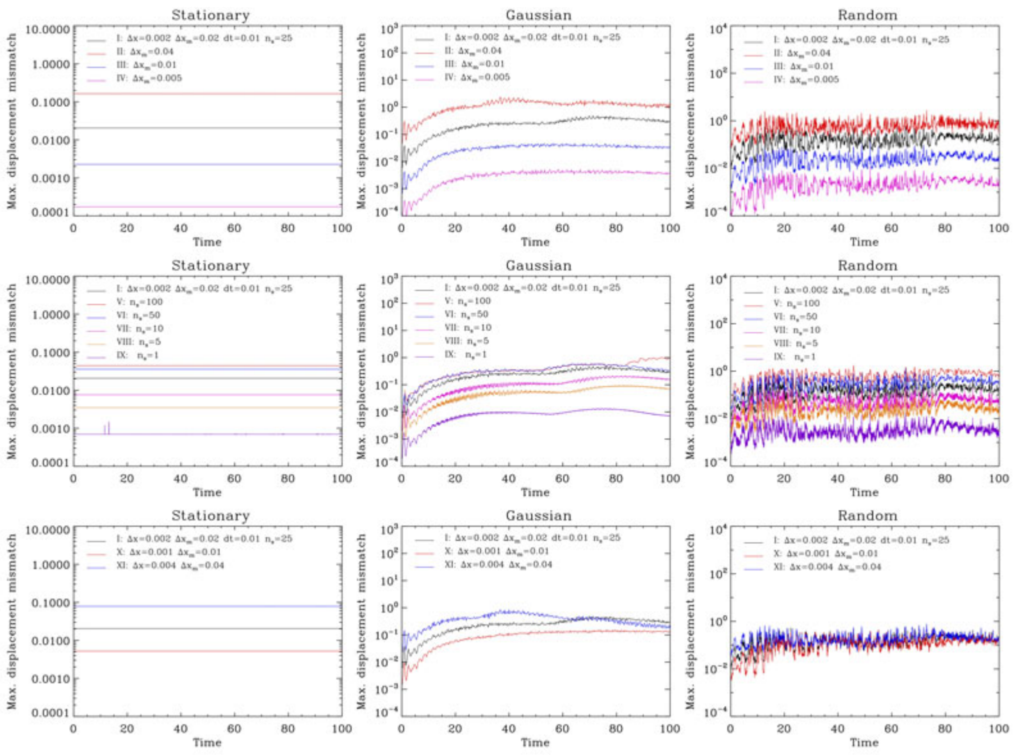


FIGURE 14. The maximum reconstructed displacement mismatch between neighbouring elements of metric, E_{map}, as defined in § 2.3.3, as a function of time for the `Vlamet` runs considered in the energy and entropy plots shown in previous figures. E_{map} is expressed in units of spatial resolution, Δ.

conserved better than $\sim 5 \times 10^{-5}$ for $\Delta = 0.002$ and $n_s = 25$. Being aware that the gain in memory is already a factor $2.5^6 = 244$ in six dimensions when sparse sampling the elements of metric by a factor 2.5 in each direction compared to the computational grid, it is obviously wise to keep the elements of metric sampling sufficiently dense at the price of a small increase of the computational cost (a factor of the order of 1.4 in 2-D phase space for $\Delta_m/\Delta = 2.5$, according to fifth column of table 1). Furthermore, a higher density of metric elements allows for a larger value of n_s.

4. Conclusion

We have proposed and tested a new Vlasov solver, `Vlamet`, which uses small elements of metric to carry information about the flow and its deformation. While slightly more costly in 2-D phase space than standard semi-Lagrangian solvers such as the splitting algorithm at the same resolution, this code is intrinsically less diffusive due to seldom actual resampling of the phase-space distribution function, as explicitly demonstrated by our numerical experiments. For the tests we performed, we indeed found that to have comparable level of details in the splitting algorithm and in `Vlamet`, it was needed to increase the spatial resolution by a factor 2–4 in the splitting algorithm runs, and this factor might turn to become even larger after additional optimisation: some work is indeed still needed to understand how to use `Vlamet` in the best way, while keeping errors, in particular deviations from energy conservation, under control. Indeed, despite the numerous runs we did, we did not span all the possible range of control parameters of `Vlamet`. In particular, we did not explore thoroughly the case when the separation between elements of metric approaches the spatial resolution, e.g. $2.5 \lesssim \Delta_m/\Delta \lesssim 5$, which should allow for even more seldom resamplings than what we achieved.

The gain in effective spatial resolution of `Vlamet` compared to the splitting algorithm compensates largely for the cost of the code and makes our method potentially extremely competitive in four- and six-dimensional phase space. Our current implementation treats isolated self-gravitating systems in 2-D phase space and uses parallel acceleration on shared memory architecture with the OpenMP library. The next step, quite relevant to galactic dynamics consists in generalising the code to four- and six-dimensional phase space. This extension will require the introduction of adaptive mesh refinement with e.g. a k-d tree structure to focus on regions of phase space of interest as well as parallel programming on distributed memory architecture. This should not present, in principle, any major difficulty, as discussed in § 3.1.

Note also that generalisation to plasmas is obvious and could represent a way to improve the accuracy of current codes.

Still, our implementation remains quite naive and some more work is needed for full optimisation. In particular, we use simple finite difference method to compute gradient and Hessian of the force, which calls for improvements. Also, at the time of resampling, we employ the traditional third-order spline interpolation which is clearly not free of defects, in particular aliasing effects that induce strong deviations from positivity of the phase-space distribution function (while, nevertheless not modifying the good dynamical behaviour of the system). One possible way to improve over cubic B-splines could consist in employing a discontinuous Galerkin representation (see, e.g. Cockburn, Karniadakis & Shu 2000; Qiu & Shu 2011; Besse *et al.* 2017), where the phase-space distribution function is described in each computational cell by some polynomial of some order. Continuity is not necessarily enforced between neighbouring cells, which provides more flexibility. However, while the splitting algorithm allows one to decompose the dynamical set-up in individual 1-D problems, this is not possible in `Vlamet`. We have to find a way, at the moments of remapping, to reconstruct a new discontinuous Galerkin representation by composing these local polynomials defined in a four- or six-dimensional space with the nonlinear displacement field provided by the metric elements, which remains a challenging task.

Acknowledgements

S.C. greatly acknowledges hospitality of Yukawa Institute for Theoretical Physics (YITP), where part of this work has been carried out. We thank N. Besse for fruitful discussions. We also acknowledge the support of YITP in organising the workshop 'Vlasov–Poisson: towards numerical methods without particles' in Kyoto, funded by grant YITP-T-15-02, ANR grant ANR-13-MONU-0003 and by Institut Lagrange de Paris (ANR-10-LABX-63 and ANR-11-IDEX-0004-02). This work was supported in part by ANR grant ANR-13-MONU-0003.

REFERENCES

AARSETH, S. J., LIN, D. N. C. & PAPALOIZOU, J. C. B. 1988 On the collapse and violent relaxation of protoglobular clusters. *Astrophys. J.* **324**, 288–310.

ALARD, C. & COLOMBI, S. 2005 A cloudy Vlasov solution. *Mon. Not. R. Astron. Soc.* **359**, 123–163.

BERTSCHINGER, E. 1998 Simulations of structure formation in the universe. *Annu. Rev. Astron. Astrophys.* **36**, 599–654.

BESSE, N., DERIAZ, E. & MADAULE, É. 2017 Adaptive multiresolution semi-Lagrangian discontinuous Galerkin methods for the Vlasov equations. *J. Comput. Phys.* **332**, 376–417.

BESSE, N., LATU, G., GHIZZO, A., SONNENDRÜCKER, E. & BERTRAND, P. 2008 A wavelet-MRA-based adaptive semi-Lagrangian method for the relativistic Vlasov Maxwell system. *J. Comput. Phys.* **227**, 7889–7916.

BESSE, N. & SONNENDRÜCKER, E. 2003 Semi-Lagrangian schemes for the Vlasov equation on an unstructured mesh of phase space. *J. Comput. Phys.* **191**, 341–376.

BINNEY, J. 2004 Discreteness effects in cosmological N-body simulations. *Mon. Not. R. Astron. Soc.* **350**, 939–948.

BOILY, C. M., ATHANASSOULA, E. & KROUPA, P. 2002 Scaling up tides in numerical models of galaxy and halo formation. *Mon. Not. R. Astron. Soc.* **332**, 971–984.

CAMM, G. L. 1950 Self-gravitating star systems. *Mon. Not. R. Astron. Soc.* **110**, 305–324.

CAMPOS PINTO, M. & CHARLES, F. 2016 From particle methods to forward-backward Lagrangian schemes, Preprint, hal-01385676.

CAMPOS PINTO, M., SONNENDRÜCKER, E., FRIEDMAN, A., GROTE, D. P. & LUND, S. M. 2014 Noiseless Vlasov–Poisson simulations with linearly transformed particles. *J. Comput. Phys.* **275**, 236–256.

CHENG, C. Z. & KNORR, G. 1976 The integration of the Vlasov equation in configuration space. *J. Comput. Phys.* **22**, 330–351.

COCKBURN, B., KARNIADAKIS, G. E. & SHU, C. W. 2000 The development of discontinuous Galerkin methods. In *Discontinuous Galerkin Methods* (ed. B. Cockburn, G. E. Karniadakis & C. W. Shu), Lecture Notes in Computational Science and Engineering, vol. 11, pp. 3–50. Springer.

COLOMBI, S. 2001 Dynamics of the large-scale structure of the universe: N-body techniques. *New Astronomy Rev.* **45**, 373–377.

COLOMBI, S., SOUSBIE, T., PEIRANI, S., PLUM, G. & SUTO, Y. 2015 Vlasov versus N-body: the Hénon sphere. *Mon. Not. R. Astron. Soc.* **450**, 3724–3741.

COLOMBI, S. & TOUMA, J. 2014 Vlasov–Poisson in 1D: waterbag. *Mon. Not. R. Astron. Soc.* **441**, 2414–2432; (CT14).

CROUSEILLES, N., LATU, G. & SONNENDRÜCKER, E. 2009 A parallel Vlasov solver based on local cubic spline interpolation on patches. *J. Comput. Phys.* **228**, 1429–1446.

CROUSEILLES, N., MEHRENBERGER, M. & SONNENDRÜCKER, E. 2010 Conservative semi-Lagrangian schemes for Vlasov equations. *J. Comput. Phys.* **229**, 1927–1953.

DEHNEN, W. & READ, J. I. 2011 N-body simulations of gravitational dynamics. *Eur. Phys. J. Plus* **126**, 55.

DOLAG, K., BORGANI, S., SCHINDLER, S., DIAFERIO, A. & BYKOV, A. M. 2008 Simulation techniques for cosmological simulations. *Space Sci. Rev.* **134**, 229–268.

FILBET, F., SONNENDRÜCKER, E. & BERTRAND, P. 2001 Conservative numerical schemes for the Vlasov equation. *J. Comput. Phys.* **172**, 166–187.

FUJIWARA, T. 1981 Vlasov simulations of stellar systems – infinite homogeneous case. *Publ. Astronom. Soc. Japan* **33**, 531.

GÜÇLÜ, Y., CHRISTLIEB, A. J. & HITCHON, W. N. G. 2014 Arbitrarily high order convected scheme solution of the Vlasov–Poisson system. *J. Comput. Phys.* **270**, 711–752.

GUTNIC, M., HAEFELE, M., PAUN, I. & SONNENDRÜCKER, E. 2004 Vlasov simulations on an adaptive phase-space grid. *Comput. Phys. Commun.* **164**, 214–219.

GRANDGIRARD, V., ABITEBOUL, J., BIGOT, J., CARTIER-MICHAUD, T., CROUSEILLES, N., DIF-PRADALIER, G., EHRLACHER, C., ESTEVE, D., GARBET, X., GHENDRIH, P. *et al.* 2016 A 5D gyrokinetic full-f global semi-Lagrangian code for flux-driven ion turbulence simulations. *Comput. Phys. Commun.* **207**, 35–68.

HAHN, O. & ANGULO, R. E. 2016 An adaptively refined phase-space element method for cosmological simulations and collisionless dynamics. *Mon. Not. R. Astron. Soc.* **455**, 1115–1133.

HOCKNEY, R. W. & EASTWOOD, J. W. 1988 *Computer Simulation Using Particles*. Hilger.

JOYCE, M., MARCOS, B. & SYLOS LABINI, F. 2009 Energy ejection in the collapse of a cold spherical self-gravitating cloud. *Mon. Not. R. Astron. Soc.* **397**, 775–792.

KANDRUP, H. E. & SMITH, H. JR. 1991 On the sensitivity of the N-body problem to small changes in initial conditions. *Astrophys. J.* **374**, 255–265.

LARSON, D. J. & YOUNG, C. V. 2015 A finite mass based method for Vlasov–Poisson simulations. *J. Comput. Phys.* **284**, 171–185.

MELOTT, A. L. 2007 Comment on 'Discreteness Effects in Simulations of Hot/Warm Dark Matter' by J. Wang & S. D. M. White, ArXiv e-prints arXiv:0709.0745.

MONAGHAN, J. J. 1992 Smoothed particle hydrodynamics. *Annu. Rev. Astron. Astrophys.* **30**, 543–574.

NISHIDA, M. T., YOSHIZAWA, M., WATANABE, Y., INAGAKI, S. & KATO, S. 1981 Vlasov simulations of stellar disks – part two – nonaxisymmetric case. *Publ. Astronom. Soc. Japan* **33**, 567.

QIU, J.-M. & SHU, C.-W. 2011 Positivity preserving semi-Lagrangian discontinuous Galerkin formulation: theoretical analysis and application to the Vlasov–Poisson system. *J. Comput. Phys.* **230**, 8386–8409.

ROSSMANITH, J. A. & SEAL, D. C. 2011 A positivity-preserving high-order semi-Lagrangian discontinuous Galerkin scheme for the Vlasov–Poisson equations. *J. Comput. Phys.* **230**, 6203–6232.

SHOUCRI, M. M. & GAGNE, R. R. J. 1978 Splitting schemes for the numerical solution of a two-dimensional Vlasov equation. *J. Comput. Phys.* **27**, 315–322.

SPITZER, L. JR. 1942 The dynamics of the interstellar medium. III. Galactic distribution. *Astrophys. J.* **95**, 329.

RYBICKI, G. B. 1971 Exact statistical mechanics of a one-dimensional self-gravitating system. *Astrophys. Space Sci.* **14**, 56–72; (Papers appear in the Proceedings of IAU Colloquium No. 10 Gravitational N-Body Problem (ed. by Myron Lecar), R. Reidel Publ. Co., Dordrecht-Holland).

SONNENDRÜCKER, E., ROCHE, J., BERTRAND, P. & GHIZZO, A. 1999 The semi-Lagrangian method for the numerical resolution of the Vlasov equation. *J. Comput. Phys.* **149**, 201–220.

SOUSBIE, T. & COLOMBI, S. 2016 ColDICE: A parallel Vlasov–Poisson solver using moving adaptive simplicial tessellation. *J. Comput. Phys.* **321**, 644–697.

WATANABE, Y., INAGAKI, S., NISHIDA, M. T., TANAKA, Y. D. & KATO, S. 1981 Vlasov simulations of stellar disks – part one – axisymmetric case. *Publ. Astronom. Soc. Japan* **33**, 541.

YOSHIKAWA, K., YOSHIDA, N. & UMEMURA, M. 2013 Direct integration of the collisionless Boltzmann equation in six-dimensional phase space: self-gravitating systems. *Astrophys. J.* **762**, 116.