ARTICLE IN PRESS

Appl. Numer. Math. xxx (xxxx) xxx

Contents lists available at ScienceDirect

Applied Numerical Mathematics

journal homepage: www.elsevier.com/locate/apnum

Research Paper

A GPU-accelerated Lagrangian method for solving the Liouville equation in random differential equation systems

V.J. Bevia, S. Blanes, J.C. Cortés *, N. Kopylov, R.J. Villanueva

Instituto Universitario de Matemática Multidisciplinar, Universitat Politècnica de València, Camino de Vera s/n, 46022, Valencia, Spain

ARTICLE INFO

Keywords:
Random differential equations
Liouville equation
Lagrangian particle methods
Wavelet adaptive mesh refinement
Radial basis function interpolation

ABSTRACT

This work presents and analyzes a numerical approach to efficiently solve the Liouville equation in the context of random ODEs using GPGPUs. Our method combines wavelet compression-based adaptive mesh refinement, Lagrangian particle methods, and radial basis function interpolation to create a versatile algorithm applicable in multiple dimensions. We discuss the advantages and limitations of this algorithm. To demonstrate its effectiveness, we compute the probability density function for various 2D and 3D random ODE systems with applications in physics and epidemiology.

1. Introduction and related work

Uncertainty quantification plays a relevant role in studying and analyzing differential equations by providing a framework to account for the inherent uncertainties present in real-world systems and their modeling. Differential equations are often used to describe the dynamic behavior of complex systems across various scientific and engineering domains. However, these models involve parameters, initial conditions, and inputs subject to uncertainties arising from measurement errors, incomplete knowledge, or natural variability. Ignoring these uncertainties can lead to unreliable predictions and a limited understanding of the system's behavior. This is particularly crucial in decision-making processes where accurate predictions and risk assessment are essential, such as physics, mechanics, climate modeling, financial forecasting, or engineering design.

One of the most versatile approaches to uncertainty quantification is considering Random Ordinary Differential Equations (RODEs) [50,49,56,51,22]. Under this approach, we consider the model parameters and initial or boundary conditions to be random variables following a certain distribution, which can be known *a priori* or not. Monte Carlo simulation [27], generalized Polynomial Chaos (gPC) [44], and perturbation methods [2] are powerful techniques extensively used for uncertainty quantification in RODEs across multiple scientific and engineering applications. Monte Carlo simulation employs random sampling to propagate uncertainties through models, generating many simulations to estimate statistical quantities of interest. This approach is widely employed when dealing with complex, high-dimensional systems where analytical solutions are impractical or unachievable. On the other hand, gPC is a deterministic technique that approximates uncertain variables using orthogonal polynomials. The gPC approach enables efficient (but limited) uncertainty propagation and quantification by representing the uncertainties in terms of polynomial expansions. It offers advantages such as faster convergence and reduced computational cost compared to traditional Monte Carlo methods, making it especially useful for analyzing systems with moderate or low-dimensional uncertainties.

* Corresponding author.
E-mail addresses: vibees@upv.es (V.J. Bevia), serblaza@imm.upv.es (S. Blanes), jccortes@imm.upv.es (J.C. Cortés), nikop1@upv.es (N. Kopylov), rjvillan@imm.upv.es (R.J. Villanueva).

https://doi.org/10.1016/j.apnum.2024.09.021
Received 27 June 2024; Received in revised form 10 September 2024; Accepted 20 September 2024

Available online 26 September 2024

Please cite this article as: V.J. Bevia et al., *Appl. Numer. Math.*, https://doi.org/10.1016/j.apnum.2024.09.021

Nevertheless, the main limitation of these approaches is that these methods were designed with the main objective of obtaining the moments of the solution stochastic process, such as the mean and variance. A greater goal would be to obtain the first Probability Density Function (1-PDF) [50, Ch. 3]. This function provides a means to propagate uncertainty from the input to the output of RODE models, enabling a more comprehensive analysis of the system's response under various conditions. It allows for estimating prediction regions or probability distributions of the model outputs, providing a measure of the reliability and range of possible outcomes. Indeed, consider a stochastic process, say $\mathbf{X}(t) \equiv \mathbf{X}(t,\omega) = (X_1(t,\omega), \dots, X_d(t,\omega))$, $\omega \in \Omega$, where $(\Omega, \mathcal{F}_\Omega, \mathbb{P})$ is a complete probability space. Its 1-PDF, say $\rho_t(\mathbf{x})$ (also denoted by $\rho(\mathbf{x},t)$ or $\rho_{\mathbf{X}(t)}(\mathbf{x})$), allows the computation of the expectation ($\mathbb{E}[\,]$) of any measurable transformation, g, of the stochastic process,

$$\mathbb{E}[g(\mathbf{X}(t,\omega)] = \int_{\mathbb{R}^d} g(\mathbf{x})\rho_t(\mathbf{x})\mathrm{d}\mathbf{x}.$$

In the particular case that $g(\mathbf{X}(t,\omega)) = (X_i(t,\omega))^k$, $k = 1,2,\dots$, $i = 1,2,\dots,d$, one can calculate any one-dimensional statistical moments of the components of the solution,

$$\mathbb{E}[(X_i(t,\omega))^k] = \int_{\mathbb{R}^d} x_i^k \rho_t(\mathbf{x})\mathrm{d}\mathbf{x}, \quad k = 1,2,\dots, \ i = 1,2,\dots,d.$$

Taking $k = 1$, one can compute every component of the expectation vector

$$\mathbb{E}[\mathbf{X}(t,\omega)] = (\mathbb{E}[X_1(t,\omega)], \dots, \mathbb{E}[X_d(t,\omega)]),$$

while taking $g(\mathbf{X}(t,\omega)) = (X_i(t,\omega) - \mathbb{E}[X_i(t,\omega)])^2$, $i = 1,\dots,d$, one obtains the variance vector

$$\mathbb{V}[\mathbf{X}(t,\omega)] = (\mathbb{V}[X_1(t,\omega)], \dots, \mathbb{V}[X_d(t,\omega)]).$$

The 1-PDF also permits computing the probability that the solution lies within any region of interest,

$$\mathbb{P}[\{\omega \in \Omega : \mathbf{X}(t,\omega) \in B\}] = \int_B \rho_t(\mathbf{x})\mathrm{d}\mathbf{x}, \quad B \in \mathcal{B}(\mathbb{R}^d),$$

where $\mathcal{B}(\mathbb{R}^d)$ denotes the Borel σ-algebra of $\mathbb{R}^d$; as well as the computation of prediction regions at a specific level of prediction $1-\alpha \in (0,1)$, where $\alpha \in (0,1)$ is taken as 0.05 to build a 95% prediction region.

In this regard, there are two main (equivalent) methods for obtaining the 1-PDF of an RODE. The first one is based on the Random Variable Transformation (RVT) theorem [50,45,25] and interesting results in the setting of RODE include [31,21,16]. Another fruitful approach is the Liouville, or Continuity, Equation [50,46,23]. The Liouville equation is a fundamental Partial Differential Equation (PDE) in classical mechanics and statistical physics and plays a crucial role in applied mathematics due to its significance in understanding dynamic systems where certain quantities are conserved through the evolution of the system. This equation describes the time evolution of a density function in the phase space of a dynamical system, which characterizes the probabilistic behavior of a collection of particles or a continuous medium. By providing a mathematical framework to analyze the behavior of complex systems, the Liouville equation is a cornerstone in various disciplines, including fluid dynamics, plasma physics, and quantum mechanics. For example, in fluid dynamics, the Liouville equation describes the conservation of mass and momentum; in plasma physics, the equation is instrumental in understanding the behavior of ionized gases and the dynamics of particles in plasma. Recently, the Liouville equation has proven to be particularly useful for performing Uncertainty Quantification (UQ) in differential-equation-based dynamical systems [11,33,8,10,9] as well as its use in combination with the RVT technique [13,12].

One of the key limiting factors on the applicability of the Liouville equation when studying collective or probabilistic behavior, such as in RODEs, is that every degree of freedom in a physical system or every equation in an RODE system translates into a spatial dimension of the Liouville equation. Considering the computational cost and memory requirements needed to solve PDEs in dimensions higher than 3, it is understandable that the use of the equation was reduced only to the theoretical or academic realm. However, computer science has taken huge leaps in performance in the last decades, especially since General Purpose Graphical Processing Units (GPGPUs, or simply GPUs) have become available for scientific simulation. With their highly parallel architecture, immense processing power, and energy efficiency, GPUs enable tackling complex problems that were previously computationally intractable. GPU simulation has become the cornerstone in fields such as machine learning, computational physics, chemistry, biology, and meteorology, where simulations of intricate phenomena, such as image recognition, fluid dynamics, molecular interactions, cellular processes, or climate modeling, are, respectively, essential. Because of the special nature of the Liouville equation, which will be exploited as shown throughout the current contribution, GPUs can become a powerful ally for the efficient use of the Liouville equation in realistic scenarios.

This contribution is structured as follows: In Section 2, we provide a mathematical discussion of the methods used to design an efficient Liouville equation numerical solver. It is divided into three parts: (1) adaptive mesh refinement, (2) interpolation and remeshing, (3) Lagrangian particles and characteristic curves. In Section 3, we discuss the details of the computational setup of each mathematical tool described in Section 2. Finally, in Section 4, we showcase the applicability of the Liouville equation numerical solver by applying it to several relevant mathematical models formulated with random differential equations.

2. Methods

The following theorem will play a key role in our subsequent development since we aim to efficiently solve the Liouville PDF associated with an RDE.

Theorem 2.1. *[50,46,13] Let $\mathbf{v}_t := \mathbf{v}(\cdot,t) : \mathbb{R}^d \to \mathbb{R}^d$ be a family of Lipschitz-continuous vector fields, continuous in t for all $\mathbf{x} \in \mathbb{R}^d$. Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a complete probability space and $\mathbf{X}(t) = \mathbf{X}(t,\omega)$, $t \geq t_0$, $\omega \in \Omega$ be the stochastic process verifying the following RDE in the almost-surely or mean square sense:*

$$\begin{cases} \mathbf{X}'(t,\omega) &= \mathbf{v}_t(\mathbf{X}(t,\omega)), \quad t > 0,\ \omega \in \Omega, \\ \mathbf{X}(t_0,\omega) &= \mathbf{x}^0(\omega), \qquad\quad \omega \in \Omega, \end{cases} \tag{2.1}$$

with $\mathbf{x}^0 \in \mathcal{L}_2^d(\Omega,\mathbb{P})$. Let $\mathcal{D} \subseteq \mathbb{R}^d$ be a set such that $\{\mathbf{X}([t_0,\infty),\omega)\}_{\omega\in\Omega} \subset \overline{\mathcal{D}}$ almost surely. Then, the 1-PDF of the stochastic process $\mathbf{X}(t)$, denoted by $\rho_t := \rho_{\mathbf{X}(t)}(\cdot,t)$, verifies the Liouville PDE:

$$\partial_t \rho_t(\mathbf{x}) + \nabla_{\mathbf{x}} \cdot [\mathbf{v}_t\, \rho_t](\mathbf{x}) = 0, \qquad (t,\mathbf{x}) \in (t_0,\infty) \times \mathcal{D}, \tag{2.2}$$

$$\rho_0(\mathbf{x}) = f_0(\mathbf{x}), \quad \mathbf{x} \in \overline{\mathcal{D}}, \tag{2.3}$$

$$\rho_t \mathbf{v}_t \cdot \mathbf{n}(\mathbf{x}) = 0, \qquad (t,\mathbf{x}) \in [t_0,\infty) \times \partial\mathcal{D}, \tag{2.4}$$

where f_0 is the PDF of $\mathbf{x}^0$, $\mathbf{n}$ is the normal vector of the boundary of the domain $\mathcal{D}$, denoted by $\partial\mathcal{D}$, and $\nabla_{\mathbf{x}}\cdot$ denotes the divergence operator. Also, $\overline{\mathcal{D}} = \mathcal{D} \cup \partial\mathcal{D}$.

Note that the homogeneous Neumann conditions at the boundary are only considered wherever $\partial\mathcal{D}$ is bounded and it could be exchanged for $\rho_t = 0$ for $\mathbf{x} \in \mathbb{R}^d \backslash \overline{\mathcal{D}}, \forall t \geq t_0$. Unless specified, we will assume that $\partial\mathcal{D}$ is, at least, piecewise C^1, and that f_0 is integrable and C^1 inside its support.

Under certain conditions, we may compute the divergence operator in the conservative form of the Liouville equation (2.2), obtaining its explicit form. Specifically, for all $t > 0$ where $\mathbf{v}_t \in \mathrm{Lip}(\mathcal{D})$, we can rewrite the PDE in (2.2) as follows:

$$\partial_t \rho_t + \mathbf{v}_t \cdot \nabla_{\mathbf{x}} \rho_t = -\rho_t \nabla_{\mathbf{x}} \cdot \mathbf{v}_t, \tag{2.5}$$

for every $\mathbf{x} \in \mathcal{D}$, where $\mathbf{v}_t$ is differentiable. Furthermore, if $\mathbf{v}_t \in \mathrm{C}^1(\mathcal{D})$, this form of the equation is well-posed for every $\mathbf{x} \in \mathcal{D}$ and it can be solved via its characteristic equations. This point will be explored further in the next section.

In most cases, uncertainty is not only located in the initial conditions. Normally, one is interested in analyzing and predicting the evolution of the initial density ρ_0 under the effect of a random vector field; that is, $\mathbf{v}_t = \mathbf{v}_t(\cdot\,; \mathbf{A}(\omega))$, where $\mathbf{A}(\omega) = (A_1(\omega), \ldots, A_m(\omega))$, $\omega \in \Omega$ is the random parameter vector, with a PDF $f_{\mathbf{A}}$. In this case, let us consider a realization $\mathbf{a}$ of $\mathbf{A}$ and the evolution of the corresponding conditional PDF:

$$\partial_t \rho_t(\mathbf{x} \,|\, \mathbf{a}) + \mathbf{v}_t(\mathbf{x}\,;\mathbf{a}) \cdot \nabla_{\mathbf{x}} \rho_t(\mathbf{x} \,|\, \mathbf{a}) = -\rho_t(\mathbf{x} \,|\, \mathbf{a}) \nabla_{\mathbf{x}} \cdot \mathbf{v}_t(\mathbf{x}\,;\mathbf{a}). \tag{2.6}$$

We now have a family of Liouville equations that return the conditional PDF of $\mathbf{X}(t,\omega)$, subject to the deterministic vector field $\mathbf{v}_t(\mathbf{x}\,;\mathbf{a})$. Therefore, to obtain the PDF of $\mathbf{X}(t)$ independently of the realizations of $\mathbf{A}$, we will use the law of total probability:

$$\rho_t(\mathbf{x}) = \int_{\mathbb{R}^m} \rho_t(\mathbf{x} \,|\, \mathbf{a}) f_A(\mathbf{a}) \mathrm{d}\mathbf{a} = \mathbb{E}_{\mathbf{A}}[\rho_t(\mathbf{x} \,|\, \mathbf{A})] \simeq \frac{\sum_{i=1}^{N_{\mathbf{A}}} \rho_t(\mathbf{x} \,|\, \mathbf{a}^i) f_{\mathbf{A}}(\mathbf{a}^i)}{\sum_{i=1}^{N_{\mathbf{A}}} f_{\mathbf{A}}(\mathbf{a}^i)}, \tag{2.7}$$

where the approximation can be obtained by considering the Euler approximations for the first integral and the integral for $f_{\mathbf{A}}$, which has unit mass. It can also be obtained by considering the discretization of the parameter vector's joint PDF:

$$f_{\mathbf{A}} \simeq \sum_{i=1}^{N_{\mathbf{A}}} \frac{f_{\mathbf{A}}(\mathbf{a}^i)}{\sum_{k=1}^{N_{\mathbf{A}}} f_{\mathbf{A}}(\mathbf{a}^k)} \delta_{\mathbf{a}^i},$$

where $\delta_{\mathbf{a}^i}$ is the Dirac delta measure centered at $\mathbf{a}^i$.

Summing up, solving the Liouville equation with a random vector field implies solving it for several deterministic vector fields and then taking the expectation with respect to the parameter vector's distribution.

2.1. Adaptive mesh refinement

The evolution of the PDF of an RDE's solution can undergo large changes in its variance throughout the simulation. For example, it may start as a very narrow function because there is little uncertainty at the system's initial state. Likewise, the asymptotic state

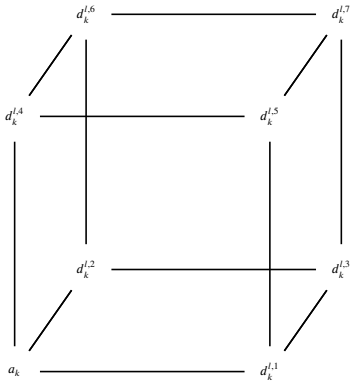


Fig. 1. Illustrative scheme on the location of approximation/detail coefficients for a 3D wavelet transform. Compare with Eq. (2.9).

of the system may be deterministic, so the PDF will converge to a Dirac delta. Evolving the PDF with enough resolution to capture those low-variance situations will allow for better accuracy but at a prohibitive computational cost. In contrast, a lower resolution will give a faster simulation but lower accuracy, especially in low-variance cases.

Adaptive Mesh Refinement (AMR) delivers the best of both worlds: it allows to evolve in greater detail only where needed while keeping the global computational cost as low as possible for faster simulation and lower computational memory requirements. There are many possibilities to perform AMR (patch, block, structured, unstructured, Cartesian, etc. see [52]), but they share the same philosophy: setting a finer grid where the gradient of the solution is larger. One possible way to do this, which we have chosen, is the Wavelet-compression-based AMR in a Cartesian grid [6]. This AMR approach method allows for a fast, easily parallelizable, and memory-efficient technique for the use cases of the numerical procedure described in this contribution.

Wavelet transforms are widely used in audio, image, and video compression [35,1,37]. Mathematically, the wavelet transform allows representing square-integrable functions with an orthonormal $\mathcal{L}_2$ basis, whose spanned space form a set of nested orthogonal spaces:

$$\mathcal{L}_2(\mathbb{R}) = V_{L_0} \oplus W_{L_0} \oplus \ldots \oplus W_L, \quad W_l \subset W_{l+1},$$

where V_{L_0} is the "approximation" space at level L_0, whose base we denote by f^{L_0}, and the collection of nested subspaces $\{W_l\}_{l=L_0}^{L}$ are the "detail" spaces. Each W_l is generated by dilation and translation of a wavelet function g; that is

$$g_k^l(x) = 2^{l/2}\, g(2^l x - k), \quad \forall x \in \mathbb{R}, \tag{2.8}$$

and $\{g_k^l\}_j$ form an $\mathcal{L}_2$-orthonormal basis of W_l (see [1] for further details on the computation of wavelet transforms).

When applied to AMR, some remarks must be made. In one dimension, we have one grid point associated with the wavelet function (detail coefficient) and one with the scaling function (approximation coefficient), which will become the detail coefficient at the next coarser level. In higher dimensions, we will have $2^d - 1$ grid points associated with detail coefficients and one with the approximation coefficient. At the next coarser level, $2^d - 1$ points associated with approximation coefficients will be associated with detail coefficients. Therefore, we will refer to *detail points at level l* when referring to points associated with a detail coefficient at the refinement level l. We use the tensor product of one-dimensional wavelets to handle the multidimensional case.

We can now write out the wavelet compression refinement procedure rigorously. Let $u : \mathbb{R}^d \to \mathbb{R}$ be the function to be compressed. Let $\{u_i := u(\mathbf{x}^i)\}_{i=1}^{2^{dL}}$ be a vector comprised of the finest discretization of u at level L; that is, u evaluated at the equispaced grid $\mathcal{G} := \{\mathbf{x}^i\}_{i=1}^{2^{dL}}$. Then, we can represent each point as:

$$u_i = \sum_{k=1}^{2^{dL_0}} a_k f_k^{L_0}(\mathbf{x}^i) + \sum_{l=L_0+1}^{L} \sum_{k=1}^{2^{dl}} \sum_{\mu=1}^{2^d-1} d_k^{l,\mu} g_k^{l,\mu}(\mathbf{x}^i). \tag{2.9}$$

We will only advect the particles where the corresponding detail coefficient is above a pre-established threshold $\varepsilon > 0$; that is, we will obtain the characteristic curves defined by the solution to (2.16) at all points in $\mathcal{G}_A := \bigcup_{l=L_0+1}^{L} \mathcal{G}_l$, where

$$\mathcal{G}_l := \{\mathbf{x}^i \in \mathcal{G} \,:\, \mathbf{x}^i \in \mathrm{supp}(g_k^{l,\mu}), \quad |d_k^{l,\mu}| \geq \varepsilon, \quad k = 1, \ldots, 2^{dl},\ \mu = 1, \ldots, 2^d - 1\}.$$

Fig. 1 shows the scheme of a wavelet cube in 3D; however, we can also see the structure in lower dimensions. In 1D, the wavelet transform would consist of the vertices $\{a_k, d_k^{l,1}\}$. In 2D, the wavelet transform would have $\{a_k, d_k^{l,1}, d_k^{l,2}, d_k^{l,3}\}$. Note that the nodes that form a certain refinement level are every other node from the previous, finer level. The nodes that do not pass to the next refinement level are associated with detail coefficients; that is, the nodes that belong to $\mathcal{G}_l$ are a subset of those not considered in the following coarser level.

2.2. Interpolation and remeshing

One of the main strengths of Lagrangian methods is that they are meshless methods; particles are not explicitly required to be set on a grid. However, as explained in [19,15], it is advisable to use an underlying mesh because it can greatly improve the simulation's quality. This implies performing "scattered" data interpolation because of the irregular particle spatial distribution after the AMR step. Many kinds of interpolation techniques exist for scattered data in multiple dimensions; we have based our interpolation procedure on Wendland's Compactly Supported Radial Basis Functions (CS-RBF, see [32,54,57,38,30] for the definition and some comments on the advantages of this approach as well as some examples of application).

Let $\mathcal{X} := \{\mathbf{x}^i\}_{i=1}^N \subset \mathcal{D} \subseteq \mathbb{R}^d$ be a set of points and $\mathbf{u} = (u_i := u(\mathbf{x}^i))_{i=1}^N$ the values of a given function $u : \mathcal{D} \to \mathbb{R}$ at the aforementioned locations. CS-RBF interpolation consists of a function $s_{u,\mathcal{X}}$ defined by the linear combination of a compactly supported, radially symmetric kernel, which we denote $\sigma : [0,\infty) \to [0,\infty)$, centered at each particle location $\mathbf{x}^i$, and a vector $\Lambda = (\lambda_1, \dots, \lambda_N)$ called the *interpolation weights*, or just *weights*, such that

$$u(\mathbf{x}) \simeq s_{u,\mathcal{X}}(\mathbf{x}) := \sum_{k=1}^{N} \sigma\left(\frac{\|\mathbf{x} - \mathbf{x}^k\|}{r}\right) \lambda_k = \sigma(\mathbf{x})\Lambda, \tag{2.10}$$

where r is the support radius of the RBF kernel and

$$s_{u,\mathcal{X}}(\mathbf{x}^i) = \sigma(\mathbf{x}^i)\Lambda = u_i, \quad \forall\, i = 1, \dots, N.$$

In matrix terms, this last condition amounts to solving the following linear system

$$\begin{bmatrix} \sigma(\mathbf{x}^1) \\ \sigma(\mathbf{x}^2) \\ \vdots \\ \sigma(\mathbf{x}^N) \end{bmatrix} \Lambda = \underbrace{\begin{bmatrix} \sigma\left(\frac{\|\mathbf{x}^1 - \mathbf{x}^1\|}{r}\right) & \sigma\left(\frac{\|\mathbf{x}^1 - \mathbf{x}^2\|}{r}\right) & \dots & \sigma\left(\frac{\|\mathbf{x}^1 - \mathbf{x}^N\|}{r}\right) \\ \sigma\left(\frac{\|\mathbf{x}^2 - \mathbf{x}^1\|}{r}\right) & \sigma\left(\frac{\|\mathbf{x}^2 - \mathbf{x}^2\|}{r}\right) & \dots & \sigma\left(\frac{\|\mathbf{x}^2 - \mathbf{x}^N\|}{r}\right) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma\left(\frac{\|\mathbf{x}^N - \mathbf{x}^1\|}{r}\right) & \sigma\left(\frac{\|\mathbf{x}^N - \mathbf{x}^2\|}{r}\right) & \dots & \sigma\left(\frac{\|\mathbf{x}^N - \mathbf{x}^N\|}{r}\right) \end{bmatrix}}_{=: A_\sigma} \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \vdots \\ \lambda_N \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_N \end{bmatrix}. \tag{2.11}$$

This matrix has very desirable properties for our purposes. For example, it is symmetric. Also, using any of the functions defined in Iske [32, Table 3.2] returns a positive definite interpolation matrix, guaranteeing that the interpolation problem is well-posed and allowing the use of efficient iterative methods such as the conjugate gradient method. Furthermore, since the RBF kernels are compactly supported, the interpolation matrix will be sparse, which will be helpful when the number of particles is large.

Despite the versatility of this kind of interpolation scheme in a high-dimensional space, there are some drawbacks. The main issue is that the kernel support radius r must be chosen *a priori*. Specifically, the condition number of the interpolation matrix and its error are given by the following relations (see [55]):

$$\operatorname{cond}_2(A_\sigma) \le C q_{\mathcal{X}}^{-d-2k-1}, \quad q_{\mathcal{X}} := \min\{\|\mathbf{x}^i - \mathbf{x}^j\|_2 \,:\, i \ne j\}, \tag{2.12}$$

$$\|u - s_{u,\mathcal{X}}\|_{\mathcal{L}_\infty(\mathcal{D})} \le C(u)\eta^{k+1/2}, \quad \eta = \sup\left\{\min_{i\in\{1,\dots,N\}} \|\mathbf{x} - \mathbf{x}^i\| \,:\, \mathbf{x} \in \mathcal{D}\right\}, \tag{2.13}$$

where d denotes the spatial dimension, and k is related to the smoothness of the interpolation kernel. Also, η, which is the "covering density" of the points with respect to the domain $\mathcal{D}$. On the one hand, the accuracy bound (2.13) is determined by the covering density of the points in space; obviously, more points covering the domain will result in a more accurate interpolation. On the other hand, the matrix condition relation (2.12) shows that particles being very close to each other, will negatively affect the condition of the interpolation matrix, resulting in numerical instabilities and deteriorated convergence.

A crucial goal for our purposes is to have a fast and accurate interpolation scheme, which means trying to find a balance between the matrix condition number and the interpolation accuracy. Therefore, we must balance the number (and closeness) of RBFs and their support radius. Choosing a small radius will help in iteration speed when solving the linear system (2.11); in this case, any given particle affects a small number of neighboring particles. Unfortunately, having a small radius could result in a *bed-of-nails interpolant* (see Fig. 2). Contrarily, a very big radius will negatively affect iteration speed and matrix condition number (see also [48, Pg. 6]). Although there is no definite choice of optimal RBF radius, taking $\sim 5h$ as a starting RBF radius is recommended, subject to further adjustments (see [32]).

RBFs can also be very useful for remeshing a function whose values are known at scattered points in a domain. Indeed, if we know the RBF weights of such a function (vector Λ in (2.10)), we can represent a function at a grid (regular or not). One must find the distance between the grid nodes, denoted by $\{\mathbf{x}^i_{\text{Grid}}\}_{i=1}^{NG}$, and the advected particles and multiply the "interpolation" matrix with the weight vector Λ. Mathematically,

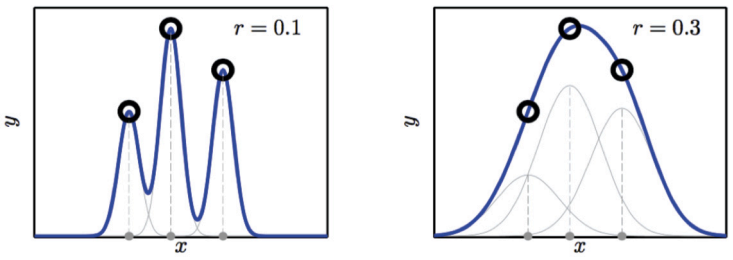


Fig. 2. RBF interpolation in two cases: (left) small radius, also called *bed-of-nails interpolant* and (right) large radius. Image taken from [40].

$$\begin{bmatrix} \sigma\left(\frac{\|\mathbf{x}^1_{\text{Grid}}-\mathbf{x}^1\|}{r}\right) & \sigma\left(\frac{\|\mathbf{x}^1_{\text{Grid}}-\mathbf{x}^2\|}{r}\right) & \dots & \sigma\left(\frac{\|\mathbf{x}^1_{\text{Grid}}-\mathbf{x}^N\|}{r}\right) \\ \sigma\left(\frac{\|\mathbf{x}^2_{\text{Grid}}-\mathbf{x}^1\|}{r}\right) & \sigma\left(\frac{\|\mathbf{x}^2_{\text{Grid}}-\mathbf{x}^2\|}{r}\right) & \dots & \sigma\left(\frac{\|\mathbf{x}^2_{\text{Grid}}-\mathbf{x}^N\|}{r}\right) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma\left(\frac{\|\mathbf{x}^{NG}_{\text{Grid}}-\mathbf{x}^1\|}{r}\right) & \sigma\left(\frac{\|\mathbf{x}^{NG}_{\text{Grid}}-\mathbf{x}^2\|}{r}\right) & \dots & \sigma\left(\frac{\|\mathbf{x}^{NG}_{\text{Grid}}-\mathbf{x}^N\|}{r}\right) \end{bmatrix} \begin{bmatrix} \lambda_1^* \\ \lambda_2^* \\ \vdots \\ \lambda_N^* \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_{NG} \end{bmatrix}, \tag{2.14}$$

where NG denotes the number of nodes at the underlying regular grid.

2.3. Lagrangian particles and characteristic curves

Lagrangian methods refer to a family of numerical methods for solving PDEs that consider the solution field as a family of particles evolved through time in an invariant domain of the ODE, which corresponds to the spatial domain of the PDE [20]. The equations that govern the evolution of each particle are known as *characteristic equations*, and their solutions are called *characteristic curves* [23, Sec. 3.2]. Lagrangian methods focus on each particle's position and its value rather than the information that a given position in space may give of the solution, which is the nature of so-called *Eulerian* methods [4; 53, Sec. 3.3.2].

Mathematically, if we denote the characteristic curve at time t, starting at $(t_0, \mathbf{x}^0)$ by $\Phi(t) := \Phi(t\,; t_0, \mathbf{x}^0)$, then Lagrangian methods study the dynamic behavior of $\rho_t(\Phi(t))$. If we take the time derivative of the composed function, the chain rule gives

$$\frac{\mathrm{d}}{\mathrm{d}t}\rho_t(\Phi(t)) = \partial_t\rho_t(\Phi(t)) + \nabla_{\mathbf{x}}\rho_t(\Phi(t)) \cdot \frac{\mathrm{d}\Phi}{\mathrm{d}t}(t) = L(t, \mathbf{x}, \rho_t, \dots), \tag{2.15}$$

where the terms $\frac{\mathrm{d}\Phi}{\mathrm{d}t}(t)$ and L will vary depending on the specific problem. Some well-known examples are the Boltzmann, Navier-Stokes equations and other general conservation laws. Equation (2.15) shows that the Lagrangian approach transforms a PDE into a family of systems of ODEs. Comparing the Liouville equation (2.5), or (2.6), and (2.15), we can identify the corresponding terms, yielding the following system of characteristic equations:

$$\begin{aligned} \frac{\mathrm{d}}{\mathrm{d}t}\phi_1(t) &= (\mathrm{v}_1)_t(\Phi(t)), \quad \phi_1(0) = \mathrm{x}_1^0, \\ &\vdots \\ \frac{\mathrm{d}}{\mathrm{d}t}\phi_d(t) &= (\mathrm{v}_d)_t(\Phi(t)), \quad \phi_d(0) = \mathrm{x}_d^0, \\ \frac{\mathrm{d}}{\mathrm{d}t}\rho_t(\Phi(t)) &= -\rho_t(\Phi(t))\nabla_{\mathbf{x}} \cdot \mathbf{v}_t(\Phi(t)), \quad \rho_0(\Phi(0)) = f_0(\Phi^0), \end{aligned} \tag{2.16}$$

where $\mathbf{x}^0 = (x_1^0, \dots, x_d^0)$ is a point in $\mathbb{R}^d$, which represents the initial position of the particle to be simulated and $\mathbf{v}_t = ((\mathrm{v}_1)_t, \dots, (\mathrm{v}_d)_t)$ is the vector field from (2.5). Accordingly, the first d equations define the time evolution of the particle, whereas the last equation in (2.16) defines the evolution of the PDF value of the considered particle. This system of ODEs is easily solvable by numerical methods such as the well-known 4th order Runge Kutta integrator [17]. However, note that we can solve the last equation exactly:

$$\rho_t(\Phi(t\,; t_0, \mathbf{x}^0)) = \rho_0(\mathbf{x}^0)\exp\left(-\int_{t_0}^{t} \nabla_{\mathbf{x}} \cdot \mathbf{v}_s(\Phi(s; \mathbf{x}^0))\mathrm{d}s\right). \tag{2.17}$$

In terms of the theory of dynamical systems, the function that solves the system of characteristic equations (2.16) is defined as the *(forward) flow* of the system.

Now, we may consider a function Ψ such that $\Psi(t_0) = \Phi(t) = \mathbf{x}$ and $\Psi(t) = \mathbf{x}^0$. This function, $\Psi(s\,; \mathbf{x}) = \Phi(t - s + t_0\,; t_0, \mathbf{x}^0)$, is known as the *inverse flow* of the system (see, e.g., [14,28]). Using this function, we may rewrite (2.17) as

$$\rho_t(\mathbf{x}) = \rho_0(\Psi(t\,; \mathbf{x}))\exp\left(-\int_{t_0}^{t} \nabla_{\mathbf{x}} \cdot \mathbf{v}_s(\Phi(s\,; \Psi(t\,; \mathbf{x})))\mathrm{d}s\right). \tag{2.18}$$

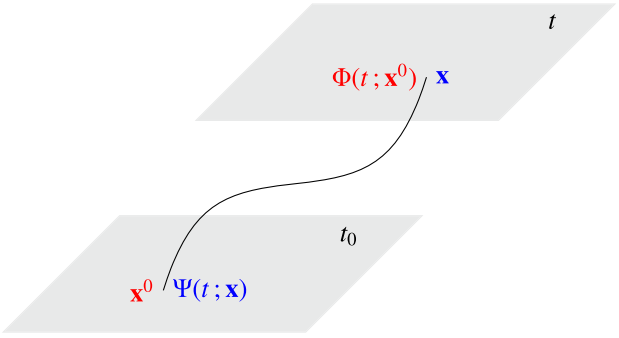


Fig. 3. The flow function (red) gives the position $\mathbf{x}$ at time t of the characteristic curve starting at $\mathbf{x}^0$. The inverse flow function (blue) indicates where the curve must start, $\mathbf{x}^0$, to be located at $\mathbf{x}$ at time t. Image taken from [13] (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.).

Fig. 3 gives a graphical representation about the geometrical meaning of the forward and inverse flow of a dynamical system.

The Liouville equation (2.5) is a quasi-linear PDE, which means that the superposition principle applies: if every element of a family of functions approximates the Liouville equation, then any finite linear combination of these functions will also approximate the equation. Let us consider a set of N particles $\{\mathbf{x}^k\}_{k=1}^N$, also $r>0$, and let us represent the PDF we search for in the following way:

$$\rho_t(\mathbf{x}) \simeq \sum_{k=1}^{N} \underbrace{\lambda_t^k \sigma\left(\frac{\|\mathbf{x}-\Phi(t;\mathbf{x}^k)\|}{r}\right)}_{\zeta_t^k(\mathbf{x})} =: s_t^N(\mathbf{x}). \tag{2.19}$$

Clearly, the ζ_t^k functions do not satisfy the Liouville equation in general (in fact, only when $\nabla_{\mathbf{x}}\cdot\mathbf{v}_t$ is constant). From (2.17), we have that ζ_t^k satisfies the Liouville equation along the characteristics $\Phi(t;\mathbf{x}^k)$ if and only if

$$\overbrace{\lambda_t^k \sigma\left(\frac{\|\Phi(t;\mathbf{x}^k)-\Phi(t;\mathbf{x}^k)\|}{r}\right)}^{=\text{const.}} = \overbrace{\lambda_0^k \sigma\left(\frac{\|\mathbf{x}^k-\mathbf{x}^k\|}{r}\right)}^{=\text{const.}} \exp\left(-\int_0^t \nabla_{\mathbf{x}}\cdot\mathbf{v}_s(\Phi(s;\mathbf{x}^k))\mathrm{d}s\right) \Longleftrightarrow$$

$$\lambda_t^k = \lambda_0^k \exp\left(-\int_0^t \nabla_{\mathbf{x}}\cdot\mathbf{v}_s(\Phi(s;\mathbf{x}^k))\mathrm{d}s\right).$$

Therefore, defining each λ_t^k in this way allows us to track the evolution of ρ_t using the values of $\zeta_t^k(\Phi(t;\mathbf{x}^k))$, as will be detailed in Section 3. Some notes about the accuracy of this method will be given in Remark 2.2.

By trivial evaluation and substitutions using the information from Eq. (2.18), and Eq. (2.19) for $t=0$ (RBF interpolation of the initial condition), we can get a pointwise error function between the exact solution and the RBF approximation ignoring integrator error for both Φ and Ψ. This is also reasonable because we do not consider the implementation of Ψ in our numerical method. Therefore, this function allows for *qualitative* information about the method's behavior. Consider $\mathbf{z}\in D$, then

$$|\rho_t(\mathbf{z}) - s_t^N(\mathbf{z})| \le \exp\left\{-\int_0^t \nabla_{\mathbf{x}}\cdot\mathbf{v}_s(\Phi(s;\Psi(t;\mathbf{z})))\mathrm{d}s\right\}\left|\rho_0(\Psi(t;\mathbf{z})) - \sum_{\mathbf{x}\in\{\mathbf{x}^k\}_{k=1}^N}\lambda_0^{\mathbf{x}}\sigma\left(\frac{\|\Psi(t;\mathbf{z})-\mathbf{x}\|}{r}\right)\right| \tag{2.20}$$

$$+\exp\left\{-\int_0^t \nabla_{\mathbf{x}}\cdot\mathbf{v}_s(\Phi(s;\Psi(t;\mathbf{z})))\mathrm{d}s\right\}\sum_{\mathbf{x}\in\{\mathbf{x}^k\}_{k=1}^N}|\lambda_0^{\mathbf{x}}|\left|\sigma\left(\frac{\|\Psi(t;\mathbf{z})-\mathbf{x}\|}{r}\right)-\sigma\left(\frac{\|\mathbf{z}-\Phi(t;\mathbf{x})\|}{r}\right)\right| \tag{2.21}$$

$$+\sum_{\mathbf{x}\in\{\mathbf{x}^k\}_{k=1}^N}|\lambda_0^{\mathbf{x}}|\sigma\left(\frac{\|\mathbf{z}-\Phi(t;\mathbf{x})\|}{r}\right)\left|\exp\left\{-\int_0^t \nabla_{\mathbf{x}}\cdot\mathbf{v}_s(\Phi(s;\mathbf{x}))\mathrm{d}s\right\}-\exp\left\{-\int_0^t \nabla_{\mathbf{x}}\cdot\mathbf{v}_s(\Phi(s;\Psi(t,\mathbf{z})))\mathrm{d}s\right\}\right|. \tag{2.22}$$

This error function shows three error sources in the RBF approach. The first one, Eq. (2.20), accounts for the interpolation error at the initial condition; that is, the error introduced by Eq. (2.19) at $t=0$, which we use for forward-in-time advection. The second part, seen in Eq. (2.21) accounts for particle displacement rate between 0 and t, which affects the RBF particle coverage. Finally, the third error, in Eq. (2.22), shows the dependency on the divergence of the system, $\nabla_{\mathbf{x}}\cdot\mathbf{v}_t$. Because of the complexity of the error function, we can only analyze simple, specific cases such as $\nabla_{\mathbf{x}}\cdot\mathbf{v}_t = K$, $\forall t\ge t_0$ everywhere in D for some constant K. If $K=0$, both Eq. (2.21)

and Eq. (2.22) disappear. In this case, the interpolation procedure is the only error source. Also, if $K \neq 0$, only the last term Eq. (2.22) disappears since particle distances will vary with constant rate in space and time.

Solving a system of ODEs is (usually) computationally faster than solving full PDEs. However, to use the characteristic equation approach, we must decide what initial points $\{\mathbf{x}^j\}_j$, with their corresponding values $\{\rho_{t_0}(\mathbf{x}^j)\}_j$, will evolve. The AMR procedure determines this part: From the initial PDF ρ_{t_0}, the AMR determines the most relevant points in the PDF domain and their respective values. These points are now considered particles that will evolve over a certain time according to the system of characteristic equations. Now, to approximate the evolution of ρ_t we will consider the value at the center of each ζ_t^j; that is, $\zeta_t^j(\Phi(t;\mathbf{x}^j)) = \lambda_t^j$. Finally, we reinitialize the PDF ρ_t following Eq. (2.14), although we will give a note on that later.

Remark 2.2. Knowledge about how the numerical integrator's time step affects the RBF particle coverage (as seen in Fig. 2) is crucial for a correct simulation. The evolution of the initial PDF could become useless if particles become too separated or too close to each other.

Consider particles $\mathbf{x}_0^i, \mathbf{x}_0^j$ with the initial distance between them $r_0 = \|\mathbf{x}_0^i - \mathbf{x}_0^j\|$. Let us compute some bounds of the separation between them as a function of time. At time t, we denote the squared distance between them as $\alpha_t = \left\|\mathbf{x}_t^i - \mathbf{x}_t^j\right\|^2$, where $\mathbf{x}_t^k := \Phi(t;t_0,\mathbf{x}^k)$. Since we assume that both particles are under the effect of the same vector field, the time derivative of α_t gives:

$$\partial_t \alpha_t = 2\left\langle \mathbf{v}_t(\mathbf{x}_t^i) - \mathbf{v}_t(\mathbf{x}_t^j), \mathbf{x}_t^i - \mathbf{x}_t^j \right\rangle \Rightarrow -2K(\mathbf{v}_t)\alpha_t \leq \partial_t \alpha_t \leq 2K(\mathbf{v}_t)\alpha_t, \tag{2.23}$$

where $K(\mathbf{v}_t) := \sup_{D\times[0,t]} \|J_s\|_2$, and J_t is the Jacobian matrix of the vector field $\mathbf{v}_t$. Assuming $K(\mathbf{v}_t) < \infty$ for all t and using Gronwall's inequality, we obtain:

$$r_0^2 \exp\left(-2\,K(\mathbf{v}_t)t\right) \leq \alpha_t \leq r_0^2 \exp\left(2\,K(\mathbf{v}_t)t\right). \tag{2.24}$$

Now, (2.24) gives bounds on the squared distance. If we want our particles to have a bounded separation $r_{\min} < r_0 < r_{\max}$, solving for t we will give us bounds on the time step length to do so:

$$\alpha_t \leq r_0^2 \exp\left(2K(\mathbf{v}_t)t\right) < r_{\max}^2 \longrightarrow 0 \leq t_1 < \frac{1}{2K(\mathbf{v}_t)} \log\left(\frac{r_{\max}^2}{r_0^2}\right) = \frac{1}{K(\mathbf{v}_t)} \log\left(\frac{r_{\max}}{r_0}\right). \tag{2.25}$$

$$\alpha_t \geq r_0^2 \exp\left(-2K(\mathbf{v}_t)t\right) > r_{\min}^2 \longrightarrow 0 \leq t_2 < -\frac{1}{2K(\mathbf{v}_t)} \log\left(\frac{r_{\min}^2}{r_0^2}\right) = -\frac{1}{K(\mathbf{v}_t)} \log\left(\frac{r_{\min}}{r_0}\right). \tag{2.26}$$

Finally, we choose the maximum time step as $\min\{t_1, t_2\}$. The next section will explain how this idea can be used for a better simulation (see also [6]). We will give more details about the choice of the time step in Section 4 devoted to examples.

Remark 2.3. This section was explained assuming a deterministic vector field because the case of the random vector field amounts to solving the case of the deterministic vector field for several parameter realizations as shown in Eq. (2.7).

3. Complete scheme and computational approach

This section will show how all the concepts defined in Section 2 will be used to compute the time evolution of the PDF of the solution to a system of RODEs. We will see how we have extended these concepts to the complete case (random vector field) and exploited GPUs' parallelism to tackle the complete problem. Fig. 4 shows the full procedure in a single iteration of the numerical method. We will explain each step in full detail.

As stated in the abstract and introduction sections, our numerical approach benefits from massively parallel computational architectures. SIMT (Single Instruction, Multiple Thread) is a computational model that allows for parallel execution of the same instructions across multiple threads. Modern GPUs commonly use it to handle highly parallel workloads efficiently, such as matrix multiplication or particle simulations [18]. Fig. 5 shows a diagram comparing the architecture of a GPU and a CPU. Although a detailed analysis between them is far more complicated than depicted in Fig. 4, we can see that GPUs possess massive parallel computing capabilities.

GPU memory is distributed in several levels and types. In this implementation, we have used two: global memory and registers. On the one hand, global memory is where we store the information that any of the GPU threads can access (GPU DRAM in Fig. 5). This memory is moderately large (several GBs in modern GPUs) but has a high access latency; that is, grabbing values from global memory takes a noticeable amount of time. On the other hand, registers are thread-private variables that have a very low access latency. However, this type of memory can only be accessed by the thread performing the computation, and the total capacity of registers is usually quite low (Purple rectangles/L1 Cache in Fig. 5). In short, to achieve efficient implementation, we must minimize access to the global memory and maximize intra-thread operations using registers.

3.1. Adaptive mesh refinement: finding the relevant particles

As explained in Subsection 2.3, the main building stone of our computational procedure is the system of characteristic equations associated with the Liouville equation. Specifically, we have seen that we can obtain the solution by solving this ODE system for

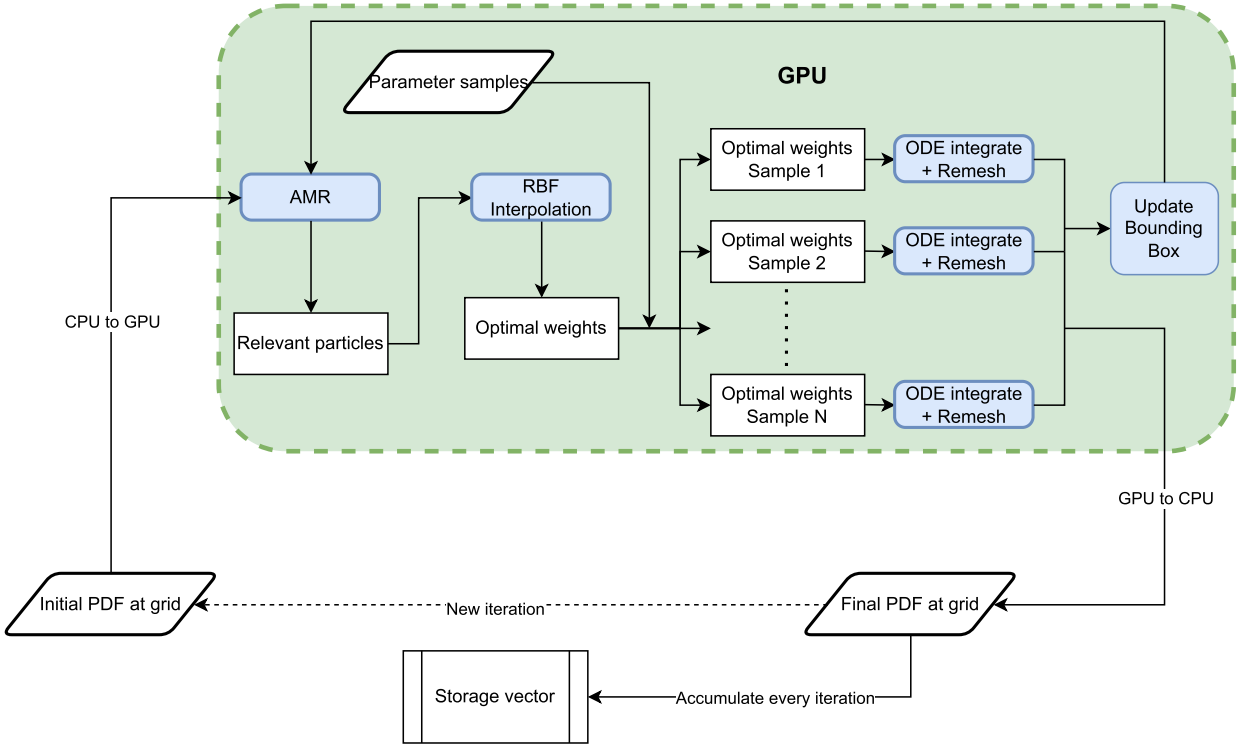


Fig. 4. Flow chart for each iteration in the numerical method for the Liouville equation. The flow starts in `Initial PDF at grid` (bottom left), and ends at `Final PDF at grid`. Although `Parameter samples` appear as initialized in the GPU, it is actually created in the CPU and transferred to the GPU before the simulation's beginning. Also, `Storage vector` is separated from the main flow because this step is done concurrently (while the simulation is running).

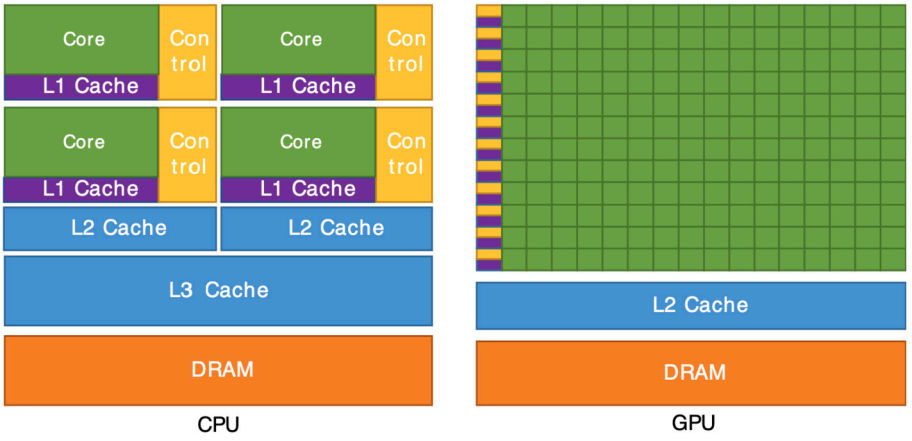


Fig. 5. Comparison of the architecture of a CPU and a GPU. The cores (single green rectangles) are responsible for computations, and each core can make computations in parallel. Modern GPUs have thousands (or even tens of thousands) of cores. The DRAM in the GPU is also called VRAM (Video RAM) to make a distinction with the CPU DRAM. Image taken from [41].

many points in phase space. Now, the choice of the specific particles for which we will solve this system will be chosen by the AMR procedure.

We only use wavelet compression to find the areas with higher gradients; no reconstruction is considered. Therefore, we have used the simplest of wavelets, the Haar wavelet defined as follows:

$$g(x) := \begin{cases} 1, & 0 \le x < 1/2, \\ -1, & 1/2 \le x < 1, \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

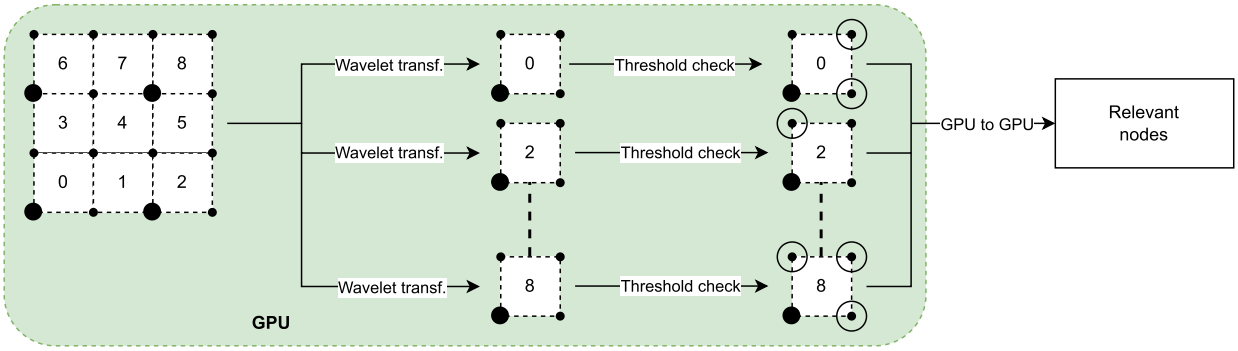


Fig. 6. This flowchart shows the AMR procedure in a 2-dimensional, 16-node mesh for a 1-level refinement. Bigger nodes are the approximation nodes; smaller ones are the detail nodes. Circled nodes are the relevant ones. For more levels, we do the procedure in the GPU for all desired levels, and then we collect the relevant nodes in GPU memory.

This wavelet is not commonly used because there are entire families of wavelets (Daubechies, for example) of a higher order that allows for much better compression and decompression of signals (see [26, Sec.42.3] for a detailed discussion). However, we are not interested in obtaining the optimal compression of the signal, but in finding grid nodes where there is a considerable gradient of the PDF surface; therefore, we can use this very fast wavelet transform together with the translation and dilation equation (2.8). The idea that this wavelet is enough for our purposes has been checked heuristically on several examples in multiple dimensions.

Although computing the discrete wavelet transform with the Haar wavelet is fast, the procedure does not scale well with dimensionality. In fact, the number of computations is $\mathcal{O}(N_{\text{Mesh}}^d)$, where N_{Mesh} is the number of mesh nodes per dimension at the finest level.[1] Regarding memory usage, this method requires two N_{Mesh}^d-size arrays: one for storing the mesh indices and the other for storing the decision variable (whether the node is considered relevant or not).

In an attempt to dampen the computational burden, at each iteration, we store the bounding box of the advected particles, which is a fair approximation of the support region of the PDF. Then, we compute the wavelet transform in this smaller domain at the next iteration. Afterward, we divide the bounding box into blocks with 2^d points per block. We assign each block to a GPU thread to compute a d-dimensional tensor wavelet transform (see Fig. 6). After computing the wavelet transform, each thread will pass over each of the assigned grid nodes of its block and will "activate" each detail-associated node with a value greater than the prescribed threshold (see Fig. 1). This is done recursively, from the finest discretization level up to the coarsest level, as shown in Section 2.1. Finally, we collect the most relevant points for the particle's advection, denoted by $\mathcal{G}_A$. In most practical applications for AMR, we take $\varepsilon \in [10^{-5}, 5 \cdot 10^{-3}]$.

To collect the most relevant nodes, we do two operations. The first one consists in counting the number of relevant nodes. Since they are stored as binary integers (1 or 0), summing the elements of the decision array gives $\#\mathcal{G}_A$. This is done via a parallel reduce operation whose time complexity is $\mathcal{O}(N/N_{\text{Threads}} + \log(N_{\text{Threads}}))$, where N is the number of elements, and N_{Threads} is the number of threads in which we parallelize the computation. The second part is sorting the nodes by setting the selected nodes' indices first. This part has a time complexity of $\mathcal{O}(N)$. Finally, we keep the first $\#\mathcal{G}_A$ elements of the index array.

3.2. Interpolation: finding the weight of each particle

Now, once we have built the set of relevant nodes $\mathcal{G}_A$ from the AMR procedure, we want to initialize the RBF weights to evolve them according to what is shown in the Subsection 2.3. This part is fairly straightforward: we have the relevant points $\{\mathbf{x}^i\}_{i=1}^{N_P} = \mathcal{G}_A$ and the PDF values at those points $\{\rho_t^i = \rho_t(\mathbf{x}^i)\}_{i=1}^{N_P}$. We have to find $\Lambda_t = (\lambda_t^1, \dots, \lambda_t^{N_P})$ such that

$$\begin{bmatrix} \sigma(\mathbf{x}^1) \\ \sigma(\mathbf{x}^2) \\ \vdots \\ \sigma(\mathbf{x}^{N_P}) \end{bmatrix} \Lambda = \underbrace{\begin{bmatrix} \sigma\left(\frac{\|\mathbf{x}^1-\mathbf{x}^1\|}{r}\right) & \sigma\left(\frac{\|\mathbf{x}^1-\mathbf{x}^2\|}{r}\right) & \cdots & \sigma\left(\frac{\|\mathbf{x}^1-\mathbf{x}^N\|}{r}\right) \\ \sigma\left(\frac{\|\mathbf{x}^2-\mathbf{x}^1\|}{r}\right) & \sigma\left(\frac{\|\mathbf{x}^2-\mathbf{x}^2\|}{r}\right) & \cdots & \sigma\left(\frac{\|\mathbf{x}^2-\mathbf{x}^N\|}{r}\right) \\ \vdots & \vdots & \ddots & \vdots \\ \sigma\left(\frac{\|\mathbf{x}^N-\mathbf{x}^1\|}{r}\right) & \sigma\left(\frac{\|\mathbf{x}^N-\mathbf{x}^2\|}{r}\right) & \cdots & \sigma\left(\frac{\|\mathbf{x}^N-\mathbf{x}^N\|}{r}\right) \end{bmatrix}}_{=: A_\sigma} \begin{bmatrix} \lambda_t^1 \\ \lambda_t^2 \\ \vdots \\ \lambda_t^{N_P} \end{bmatrix} = \begin{bmatrix} \rho_t^1 \\ \rho_t^2 \\ \vdots \\ \rho_t^{N_P} \end{bmatrix}.$$

On the computational side, this part consists of two steps. The first one consists of performing fixed-radius nearest neighbors to find particle-to-particle distances. The second one consists of iteratively solving the linear system. Both procedures can be very time and resource-intensive if not done properly.

[1] Actually, one may obtain the exact number of computations: $\frac{2^{d(L-L_0-1)}-1}{2^{d(L-L_0-1)}(2^d-1)} N_{\text{Mesh}}^d$, where L is the finest level, and L_0 is the coarsest level.

To do so, we assign a particle to each GPU thread, which then checks the distance between that particle and all the other particles. This algorithm has a very bad asymptotic limit of $\mathcal{O}(N_P^2/N_{\text{Threads}})$; however, since no trees or new arrays are involved, this algorithm is the fastest and the most memory-efficient in all tested cases.

As for memory usage, at this step we create two $N_P \cdot N_{\text{Max.Neighbors}}$-sized arrays, one for storing the indices of the nearby particles and one for storing the particle-to-particle distances. $N_{\text{Max.Neighbors}}$ is the maximum number of neighbors each particle can have, which is trivially given by the radius of the RBF kernel chosen: $N_{\text{Max.Neighbors}} = (2\lfloor R_{\text{RBF}}/h \rfloor + 1)^d$, where h is the discretization length (assumed equal in all dimensions).

Finally, regarding the system solution, we mentioned in Subsection 2.2 that the interpolation matrix is sparse, symmetric, and positive definite. So we use the sparse conjugate gradient method, using COO-style indexing [29,5,24]. Instead of relying on external libraries, we have tailor-built this algorithm into our code for optimizing resources and computation time.

3.3. Particle advection: evolving the relevant particles

Let us recall that when the vector field $\mathbf{v}_t$ depends on some random parameter $\mathbf{A}(\omega)$, the joint PDF of the solution is given by the law of total probability, as seen in Eq. (2.7). Writing this integral as an expectation shows that we can solve the Liouville equation repeatedly for several realizations of the vector field's parameters and then compute its expectation according to the parameters' random vector joint PDF. Although this may sound straightforward, a naive implementation could leave many GPU threads idle, wasting significant computational resources and making the simulation much slower.

Going deeper into the previous paragraph, we have to integrate the characteristics of the same particles for every realization of a set of samples from the vector field's parameters. To do so, we can concatenate the particles assigned for each vector field parameters' realization and evolve all of them at once, using the full massive parallelization capability of GPUs. That is, we consider a finite number of realizations $\{\mathbf{a}^i\}_{i=1}^{N_\mathbf{A}}$ of the random parameter vector $\mathbf{A}$,[2] then the array of particles to be advected is:

$$\{\mathbf{z}_i\}_{i=1}^{N_\mathbf{A}N_P} = \{\overbrace{\underbrace{\mathbf{x}^1,\dots,\mathbf{x}^{N_P}}_{N_P},\mathbf{x}^1,\dots,\mathbf{x}^{N_P},\dots,\mathbf{x}^1,\dots,\mathbf{x}^{N_P}}^{N_\mathbf{A}N_P}\}, \quad \{\mathbf{x}^j\}_{j=1}^{N_P} = \mathcal{G}_A.$$

The first N_P points, $\{z_i\}_{i=1}^{N_P}$, will be evolved/updated using the vector field with the realization $\mathbf{a}^1$; the next N_P points, $\{z_i\}_{i=N_P+1}^{2N_P}$ will be evolved using $\mathbf{a}^2$ and so on. This step requires solving (numerically) the system of characteristic equations (2.16), whose updated values we define by $\{\mathbf{q}_i\}_{i=1}^{N_\mathbf{A}N_P} := \{\Phi(\Delta t;\mathbf{z}_i)\}_{i=1}^{N_\mathbf{A}N_P}$. The time complexity of this step is $\mathcal{O}(N_P/N_{\text{Threads}})$, where N_P is the number of particles to advect. The error bound depends on the specific numerical integrator used. Regarding memory usage, no new arrays are created because the particle advections are done in-place.

Depending on the nature of the problem at hand, different families of numerical integrators can be used for obtaining $\{\mathbf{q}_k\}_{k=1}^{N_PN_A}$, each one shining in specific applications. For example, there are general numerical methods such as Runge–Kutta methods (e.g., the Runge–Kutta [17] method of order 4 used for numerical experiments in this work) that allow obtaining fairly good approximations for generic systems; numerical integrators that are designed for stiff ODEs where explicit Runge–Kutta methods may fail; symplectic integrators for Hamiltonian systems, known as symplectic integrators, that preserve physical constants such as mechanical energy (see, e.g. [36,14,28]), etc.

In order to obtain the updated RBF weights $\{\lambda_{\Delta t}^j\}_{j=1}^{N_PN_A}$, we use the simple Simpson's rule. To get the midpoint value, we use Hermite interpolation.

Our approach uses only one memory transfer from RAM to VRAM (see Fig. 5), optimizing the available GPU memory bandwidth. Furthermore, the CUDA kernel written for this purpose only uploads a grid node and its corresponding PDF value from global memory. All other auxiliary variables for the numerical integrator particle update are thread-private registers, allowing fast read/write operations. Finally, the particle position and value are rewritten in the global memory. This approach allows each thread to update its position in its reserved memory location, eliminating any possible race condition between threads (two threads trying to access the same memory address) and the need to use shared memory or repeated access to higher-latency global memory [18].

3.4. Reinitialization: projecting particles onto the grid

After the evolution of the AMR-chosen particles has been completed, we re-interpolate the particles back onto the starting grid. As discussed earlier, this consists of a simple sparse matrix-vector multiplication. Let $\{\mathbf{y}_k\}_{k=1}^{N_G}$ be the grid nodes. Then, the updated PDF at each node will finally be given by:

[2] In the examples used in this contribution, we consider $\{\mathbf{a}^i\}_{i=1}^{N_\mathbf{A}} = \{a_1^i\}_{i=1}^{N_{A_1}} \otimes \{a_2^i\}_{i=1}^{N_{A_2}} \otimes \cdots \otimes \{a_M^i\}_{i=1}^{N_{A_M}}$, where $N_\mathbf{A} = N_{A_1}N_{A_2}\cdots N_{A_M}$, and each $\{a_k^i\}_{i=1}^{N_{A_k}}$ is an equidistant partition of the domain of A_k.

$$\rho_{\Delta t}(\mathbf{y}_k) = \mathbb{E}_{\mathbf{A}}[\rho_{\Delta t}(\mathbf{y}_k \mid \mathbf{A})] \simeq \frac{1}{\sum_{i=1}^{N_{\mathbf{A}}} f_{\mathbf{A}}(\mathbf{a}^i)} \left(\sum_{i=1}^{N_{\mathbf{A}}} f_{\mathbf{A}}(\mathbf{a}^i) \rho_{\Delta t}(\mathbf{y}_k \mid \mathbf{a}^i) \right)$$

$$\simeq \sum_{i=1}^{N_{\mathbf{A}}} \underbrace{\frac{f_{\mathbf{A}}(\mathbf{a}^i)}{\sum_{l=1}^{N_{\mathbf{A}}} f_{\mathbf{A}}(\mathbf{a}^l)}}_{:=w_i} \left(\sum_{j=1}^{N_P} \sigma\left(\frac{\|\mathbf{y}_k - \mathbf{q}_{j+N_P(i-1)}\|}{r} \right) \lambda_{\Delta t}^{j+N_P(i-1)} \right). \tag{3.2}$$

Joining the computation for all the grid nodes, we have the following procedure:

$$\begin{bmatrix} \rho_{\Delta t}(\mathbf{y}_1) \\ \rho_{\Delta t}(\mathbf{y}_2) \\ \vdots \\ \rho_{\Delta t}(\mathbf{y}_{N_G}) \end{bmatrix} = \begin{bmatrix} \sigma(\mathbf{y}_1) \\ \sigma(\mathbf{y}_2) \\ \vdots \\ \sigma(\mathbf{y}_{N_G}) \end{bmatrix} \begin{bmatrix} (\lambda_{\Delta t}^{j})_{j=1}^{N_P} w_1 \\ (\lambda_{\Delta t}^{j+N_P})_{j=N_P}^{2\cdot N_P} w_2 \\ \vdots \\ (\lambda_{\Delta t}^{j+N_P(N_A-1)})_{j=1}^{N_P} w_{N_{\mathbf{A}}} \end{bmatrix}.$$

Observe that this part also includes a point search: we have to either find all the grid nodes within r distance from each advected particle or find all the advected particles within the same distance r from a fixed grid node. Following the latter idea, as in the point search at the interpolation step, would be a mistake because now we will have many more particles, and an exhaustive search approach would take too much time. However, following the first idea gives an easy way to find the nearest grid nodes with respect to a given advected particle.

Indeed, we know the finest grid discretization length h and the "bottom" corners of our computational domain. Therefore, for each particle $\mathbf{q}_j \in \{\mathbf{q}_k\}_{k=1}^{N_P N_{\mathbf{A}}}$, we can find the index of the nearest grid node just by dividing each dimensional component of v_i with h and then summing each index accordingly. Once we have its nearest grid node, we can easily know all the grid nodes within R distance from the particle $\mathbf{v}_i$ without having to do any kind of point search, resulting in a very fast procedure.

Regarding the computational aspects, we assign each $\mathbf{q}_j \in \{\mathbf{q}_k\}_{k=1}^{N_P N_{\mathbf{A}}}$ to a GPU thread. We only call $\mathbf{q}_j$ and the relative weight $w_i \in \{w_k\}_{k=1}^{N_{\mathbf{A}}}$ from global memory. We then divide the components, add them correspondingly to find the nearest grid node's index, and write the relative contribution of $\mathbf{q}_j$ to the corresponding grid node as in (3.2). This multiplication is relatively lightweight, computationally speaking. As a downside to this approach, we have to introduce the contribution of each particle to each node's memory address via atomic functions; that is, maybe there is more than one thread trying to write a value at a certain grid node, but this can only be done by serializing the memory access which will degrade the performance of the GPU kernel. However, we have seen experimentally that this does not happen so often; therefore, the performance is not degraded as much as it might look.

With regard to time complexity, it is $\mathcal{O}(N_P(2[R_{\text{RBF}}/h]+1)^d / N_{\text{Threads}})$, where $[\cdot]$ is the rounding function and R_{RBF} is the radius of the chosen RBFs. However, note that the atomic-adding step could slightly degrade performance. Regarding memory usage, no new arrays are created because the results are written back onto the original array containing the values of the PDF on the fine grid.

4. Numerical examples

This section is devoted to showcasing the performance of the numerical method with respect to RODE versions of several relevant mathematical models appearing in physics and epidemiology:

- The simple harmonic oscillator, a 2D linear model whose PDF can be obtained exactly. So, it is presented as a test example;
- The Van der Pol oscillator, a 2D model from physics with nonlinearity;
- The Mathieu equation, a similar 2D nonlinear model from physics exhibiting parametric resonance phenomena;
- The SIR model, a 3D nonlinear model from epidemiology.

In the numerical experiments, we contrast the results of the proposed Liouville equation solver with Monte Carlo simulations. For these problems, we use different types of integrators to demonstrate that the algorithm is integrator-agnostic. However, we do not aim to compare different types of numerical integrators for a specific problem because —when considering the solution per trajectory— the relation between the classes of integrators is well-researched [28,14]. Nonetheless, to limit the scope of this work, we use ODE integrators that are "CUDA-friendly", that is, methods which do not require nonlinear solvers.

Also, regarding the specific choice of RBF used, we use the Wendland function $\phi_{3,1}(r) = \max(1-r,0)^4(4r+1) \in C^2([0,1])$ in all cases. This function is the first smooth CS-RBF for dimensions 2 and 3 (see [32, Tab. 3.2]). Since each problem has different dynamics, we do not want to force any extra smoothness. Also, if the RBF centers become too close to each other in a given system, the using a smoother RBF could negatively impact the matrix condition number (see Equation (2.12)).

All computations were performed in a desktop PC with 40GB of DDR4 RAM, an i9-10900K CPU and an RTX4000 (8GB) GPU. The source code is written in C++ and CUDA and is freely available for download ([7, Version 4.0.0]). The compiler used is `NVCC 12.2` for the CUDA code. In all cases, the operating system is Windows 10 Pro; however, it can also be built and run on Linux. Monte Carlo simulations are carried out using Julia 1.10.3 with the OrdinaryDiffEq.jl 6.78.0 [43], and the density plots are obtained using the kernel smoothing functions of StatsPlots.jl 0.15.7.

Table 1
Parameters of the random harmonic oscillator.

Par.	Distr.	Mean	Variance	Samples
X_0	Normal	1.0	0.3	—
$\dot{X}_0$	Normal	-1.0	0.3	—

(a) Random parameters' statistical information. All parameters are pairwise independent. Initial conditions are truncated in the corresponding problem domain (see right table).

Input	Description
Δt	$2 \cdot 10^{-3}$
Δt_{Reinit}	$2^k,\ k = 1, \dots, 7$
Time span	$[0, 4]$
Computational domain	$[-8, 8] \times [-8, 8]$
Domain points	$2^9 \times 2^9$
AMR threshold	$1 \cdot 10^{-5}$
RBF radius	$0.5h$–$15h$

(b) List of the input parameters of the algorithm.

4.1. Harmonic oscillator: error analysis

Oscillator models are fundamental in science and engineering, representing various periodic motions in natural and artificial systems. The simple harmonic oscillator, characterized by constant amplitude and frequency, serves as the foundational model for such analyses. Though dynamically simple, it is of interest for its closed-form solution to the Liouville equation (denoted by ρ_t^{REF}), providing a benchmark for comparing the Joint Probability Density Function (JPDF) computed by our Liouville solver (denoted by ρ_t).

We consider the following form of the harmonic oscillator with random initial conditions but deterministic frequency [50]:

$$\ddot{X}(t,\omega) = -\nu^2 X(t), \quad X(0,\cdot) = X_0 \in \mathcal{L}_2(\Omega,\mathbb{P}), \quad \dot{X}(0,\cdot) = \dot{X}_0 \in \mathcal{L}_2(\Omega,\mathbb{P}), \tag{4.1}$$

where ν is known as the angular frequency of the system, which we consider to be a deterministic value, and the initial conditions for position and velocity (X_0, $\dot{X}_0$, respectively) are random variables with finite variance.

The related Liouville IBVP is the following:

$$\partial_t \rho_t(x_1,x_2) + x_2 \partial_{x_1} \rho_t(\mathbf{x}) - \nu^2 x_1\, \partial_{x_2} \rho_t(\mathbf{x}) = 0, \qquad (t,x_1,x_2) \in (t_0,t_F) \times \mathcal{D}, \tag{4.2}$$

$$\rho_0(x_1,x_2) = f_0(x_1,x_2), \qquad (x_1,x_2) \in \overline{\mathcal{D}}, \tag{4.3}$$

$$\rho_t \mathbf{v}_t \cdot \mathbf{n}(x_1,x_2) = 0, \qquad (t,x_1,x_2) \in [t_0,t_F] \times \partial\mathcal{D}, \tag{4.4}$$

where $\mathcal{D}$ (computational domain) and t_F (final time) are defined in Table 1b. Its exact solution is:

$$\rho_t^{\text{REF}}(x_1,x_2) = f_0\left(\cos(\nu t)x_1 - \frac{1}{\nu}\sin(\nu t)x_2, \nu \sin(\nu t)x_1 + \cos(\nu t)x_2\right).$$

Our analysis focuses on two key parameters: the RBF radius and Δt_{Reinit}, with corresponding plots provided for each (see Figs. 7a, 7b). We plot the quadratic error of the computed JPDF, ρ_{t_F}, in the integration domain with discretization step h at the end of the integration time interval, that is,

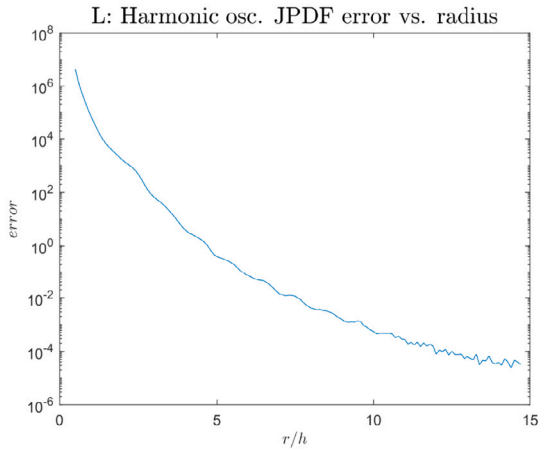
$$\text{Error} := \sum_{i,j} \left(\rho_{t_F}^{\text{REF}}(x_0 + ih, y_0 + jh) - \rho_{t_F}(x_0 + ih, y_0 + jh)\right)^2. \tag{4.5}$$

For the first test, we consider the RBF radius factor r/h between 0.5 and 16, while Δt_{Reinit} is set to the minimal value of Δt (that is, the reinitialization happens at every integration time step). For the second experiment, we fix the best radius to discretization factor, $r/h = 15$, obtained on the previous step, and increase Δt_{Reinit} as powers of 2 (see parameters in Tables 1a and 1b).

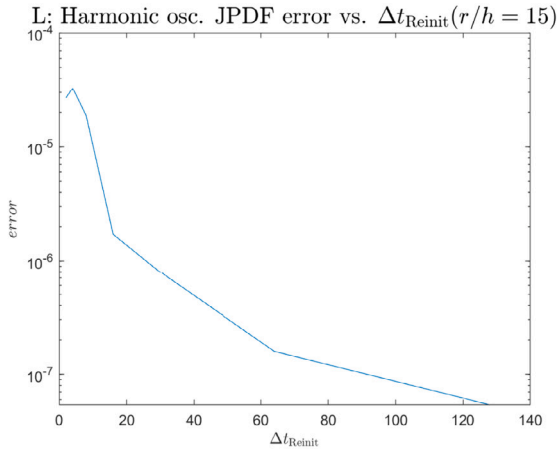
On the one hand, Fig. 7a shows the error curve for varying RBF radius, and we can clearly see a super-linear shape curve, which indicates an $\mathcal{O}((r/h)^q)$ asymptotic error curve, with $q > 1$. In fact, according to theory, the interpolation error is $\mathcal{O}((r/h)^{\frac{3}{2}})$ for the chosen RBF kernel, and this agrees with the error function Eqs. (2.20) to (2.22): since the divergence function of Eq. (4.2) is 0 everywhere, the only error source will be the interpolation procedure. Fig. 7c shows the exact solution to the Liouville IBVP in Eqs. (4.2) to (4.4), while Figs. 7d to 7f shows the JPDFs computed by our Liouville solver at the final time $t_F = 4$.

On the other hand, Fig. 7b shows something that may appear counter-intuitive: a larger reinitialization timestep results in lower error. This is not true in general (see Theorem 2.2), but in cases with zero divergence, such as our simple harmonic oscillator, where error only stems from the interpolation step, a larger reinitialization timestep means that less error will be accumulated because we will reinitialize less often.

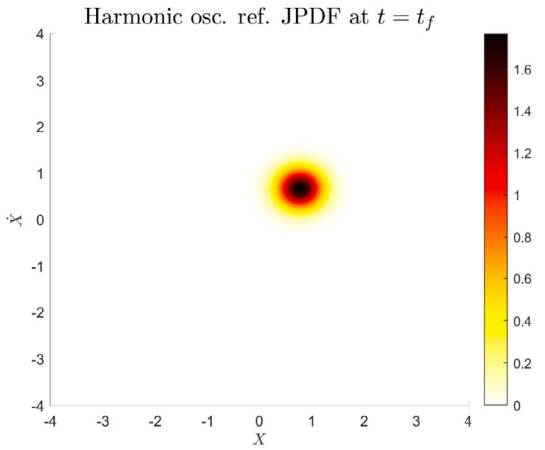
As discussed previously via Eqs. (2.12) and (2.13), we expect both curves to be U-shaped: in the case of the error from the r/h choice, it is expected to decrease as the factor grows; but as discussed in Section 2.2, a very large radius will involve more particle contributions at interpolation, which will result in matrix condition deterioration. Also, regarding the error from the Δt_{Reinit} choice, particle distances will vary when the system has a non-zero divergence, and the RBF coverage (thus, the PDF value approximation) will also deteriorate.



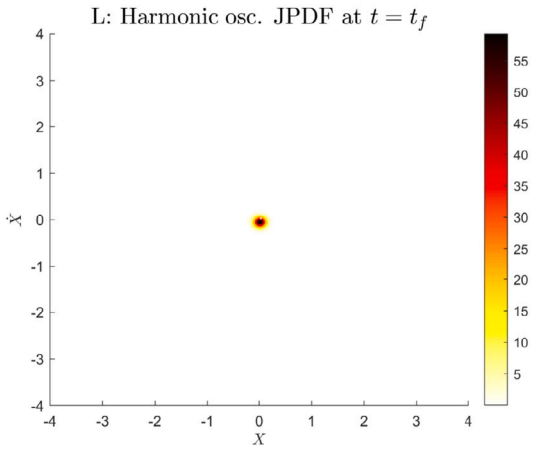
(a) Dependence of the quadratic error in the JPDF vs. the RBF radius for the harmonic oscillator.



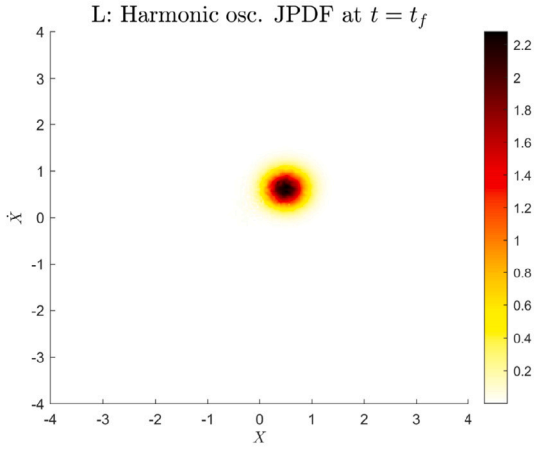
(b) Dependence of the quadratic error in the JPDF vs. the Δt_{Reinit} (the RBF radius $r/h = 15$) for the harmonic oscillator.



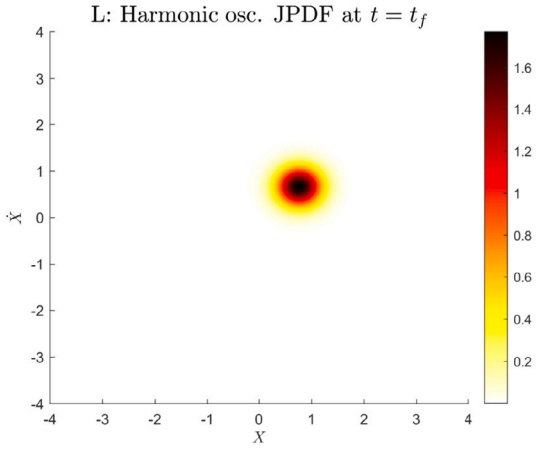
(c) Reference JPDF of the harmonic oscillator.



(d) JPDF of the harmonic oscillator for $r/h = 1$.



(e) JPDF of the harmonic oscillator for $r/h = 2.5$.



(f) JPDF of the harmonic oscillator for $r/h = 5$.

Fig. 7. Comparison of the joint probability density functions (JPDFs) of the solutions to the harmonic oscillator by the Liouville method with different parameters. Here, $t_f = 4$ as defined in Table 1b.

Table 2
Parameters of the random Van der Pol oscillator.

Par.	Distr.	Mean	Variance	Samples
X_0	Normal	1.0	0.3	—
$\dot{X}_0$	Normal	-1.0	0.3	—
μ	Normal	1.0	0.3	64

(a) Random parameters' statistical information. All parameters are pairwise independent. Initial conditions are truncated in the corresponding problem domain (see right table).

Input	Description
Δt	$2 \cdot 10^{-3}$
Δt_{Reinit}	0.01
Time span	$[0, 4]$
Computational domain	$[-8, 8] \times [-8, 8]$
Domain points	$2^9 \times 2^9$
AMR threshold	$1 \cdot 10^{-5}$
RBF radius	$1 \cdot 10^{-5}$

(b) List of the input parameters of the algorithm.

4.2. Van der Pol oscillator

The Van der Pol system holds importance in various fields due to its ability to simulate real-world behaviors like oscillations and irregular patterns. When varying the system parameter, μ, the Van der Pol oscillator shifts from orderly periodic oscillations to chaotic dynamics via a period-doubling route. This intricate behavior stems from the interaction between nonlinearity and damping, giving rise to complex patterns with implications in diverse domains like electronics, physics, and biology.

We consider the following form for the randomized Van der Pol oscillator:

$$\begin{aligned} &\ddot{X}(t,\omega) - \mu(\omega)\left(1 - X^2(t,\omega)\right)\dot{X}(t) + 5X(t,\omega) = 0, \quad t \geq 0, \\ &X(0,\omega) = X_0(\omega) \in \mathcal{L}_2 \cap \mathcal{L}_\infty(\Omega, \mathbb{P}), \\ &\dot{X}(0,\omega) = \dot{X}_0(\omega) \in \mathcal{L}_2 \cap \mathcal{L}_\infty(\Omega, \mathbb{P}). \end{aligned} \tag{4.6}$$

To launch the simulations, we set the simulation parameters specified in Table 2a. Using these we have seen (numerically) that the PDF total mass is always concentrated inside $\mathcal{D} = [-5.5, 5.5] \times [-5.5, 5.5]$. To compute the range of possible timesteps, we compute the Jacobian matrix for the Van der Pol oscillator Eq. (4.6):

$$J_t(x_1, x_2) = \begin{bmatrix} 0 & 1 \\ -5 - 2\mu x_1 x_2 & \mu(1 - x_1^2) \end{bmatrix} \quad \Rightarrow \quad \sup_{\mathcal{D} \times \text{supp}\{\mu(\omega)\}} \|J_t\|_2 \simeq 197.11040. \tag{4.7}$$

The comparison shown in the set of figures shown below (Figs. 8 and 9) were computed with an RBF radius of $7.49h$, where h denotes the discretization length of the domain. Using the results of Remark 2.2, we obtain the maximal reinitialization timestep of $\Delta t_{\text{Reinit}} = 0.0102$, which corresponds to ~ 5 integration steps using Δt as defined in Table 2b.

Fig. 8 shows the joint PDF obtained via the Liouville equation with the proposed solver and Montecarlo simulations with spatial binning. It can be seen that the Liouville solver provides a smooth evolution of the joint PDF and, contrarily to the MC simulations, it maintains a proper form without the severe particle distortion that affects the joint PDF's structure in phase space (compare Figs. 8e and 8f). This may result from MC being very sensitive to nonlinear phenomena when choosing a low-to-moderate number of particles. Particles are scattered easily throughout the phase space, where the Liouville equation solver can preserve the location of the most *relevant* particles due to the particle reinitialization at the underlying mesh and AMR. We may conclude that more particles must be chosen for a better, comparable, MC-based simulation.

Also, Fig. 9 shows the computed marginals from the Liouville equation and the MC simulations. We can see that the Liouville-based marginal PDFs achieve higher values than their MC counterparts, at least in the position component (compare Figs. 9a and 9b or Figs. 9c and 9d). This is the same as discussed in the previous paragraph: MC is very sensitive to nonlinear phenomena when choosing a low-to-moderate number of particles. However, there is a smaller difference than with the joint PDFs since we are accumulating particles in the marginal components; therefore, the sensitivity to particles is slightly dampened.

4.3. Mathieu equation

Let us consider the Mathieu equation, in a sense similar to the Van der Pol model:

$$\ddot{X}(t,\omega) + (a(\omega) - 2q\cos(2t))\,X(t,\omega) = 0. \tag{4.8}$$

The Mathieu equation is characterized by periodic coefficients and it describes parametric resonance, where perturbations can lead to large oscillations. This model and similar ones find applications to periodically variable systems and their control [42,47,34] in areas such as mechanical vibrations, electromagnetic waves, and quantum mechanics, as it helps in understanding the stability and behavior of systems with periodic parameters. An interesting question is how does the Mathieu equation behaves if some of its parameters are considered random variables. If one strives to define the regions of stability numerically, it is beneficial to use geometric integrators

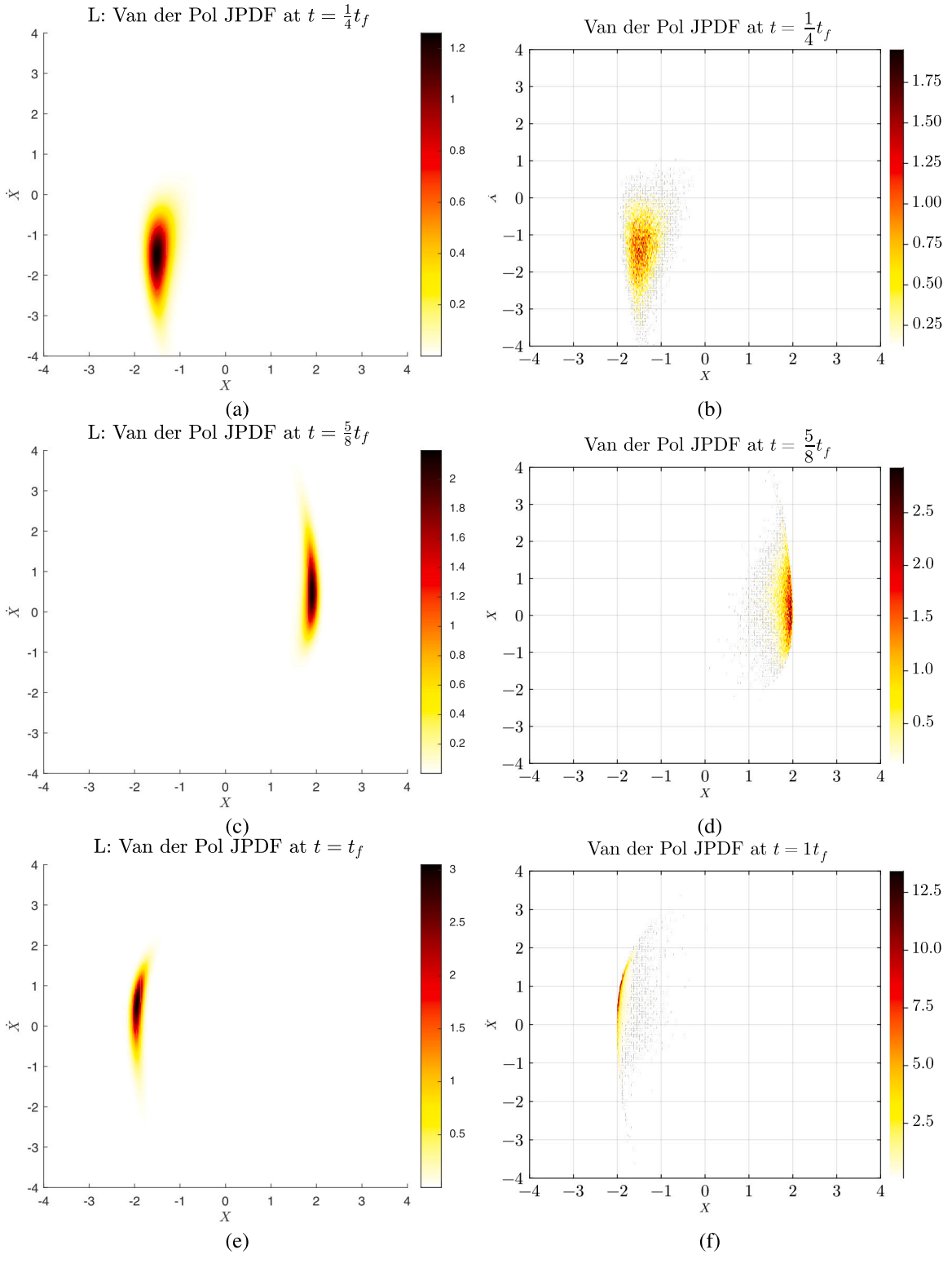


Fig. 8. Comparison of the joint probability density functions (JPDFs) of the solution to the Van der Pol oscillator by the Liouville method (left) and the Monte Carlo simulation with $m = 2^{14}$ samples (right). Here, $t_f = 4$ as defined in Table 2b.

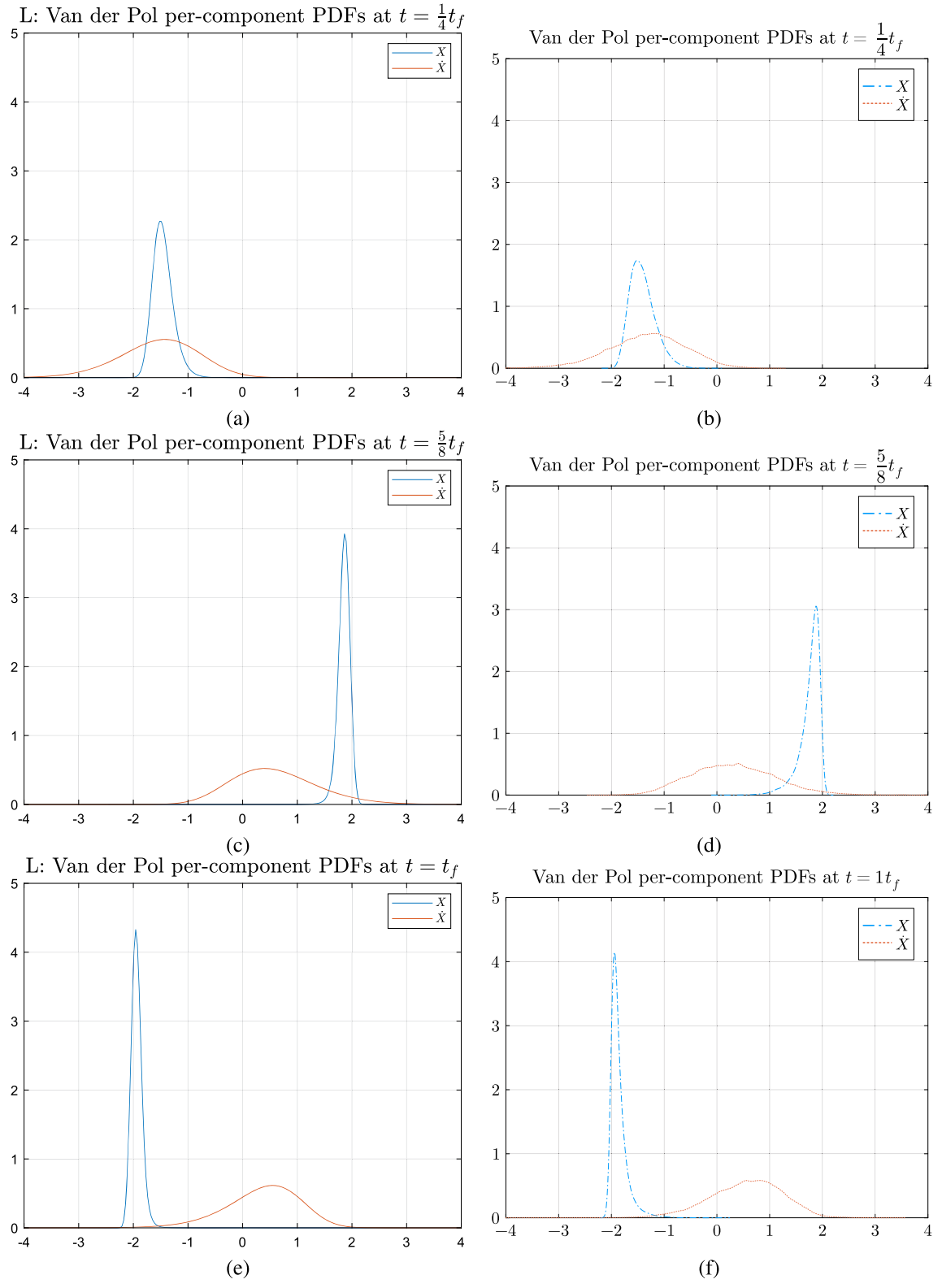


Fig. 9. Comparison of the PDFs of the components of the solution to the random Van der Pol oscillator by the Liouville method (left) and the Monte Carlo simulation with $m = 2^{14}$ samples (right). Here, $t_f = 4$ as defined in Table 2b.

Table 3
Parameters of the random Mathieu equation.

Par.	Distr.	Mean	Variance	Samples
X_0	Normal	0.0	0.09	—
$\dot{X}_0$	Normal	-1.0	0.09	—
a	Normal	3	0.09	64
q_{stable}	Delta	-1	—	1
q_{resonant}	Delta	-8	—	1

(a) Random parameters' statistical information. All parameters are pairwise independent. Initial conditions are truncated in the corresponding problem domain (see right table).

Input	Description
Δt	$\pi/500 (\sim 0.0063)$
Δt_{Reinit}	$\pi/250$
Time span	$[0, 2\pi]$
Computational domain	$[-8, 8] \times [-8, 8]$
Domain points	$2^9 \times 2^9$
AMR threshold	$1 \cdot 10^{-6}$

(b) List of the input parameters of the algorithm.

[14,3]. Table 3b shows the statistical information for the system's and simulation's parameters. For this problem, we use the geometric Runge–Kutta–Munthe-Kaas Euler method [39] instead of the classical RK method $X_{n+1} = \exp\left(hA(t_n, X_n)\right) X_n$.[3]

Fig. 10 shows similar features to the Van der Pol case. The Liouville equation solver can smoothly track the PDF evolution, while the MC samples are scattered throughout the phase space. However, the situation is better than in the Van der Pol case. This is also seen in the marginal PDFs in Fig. 11; the corresponding marginals are always very alike.

The situation is completely different in the unstable case, which shows one of the limitations of the current implementation of the Liouville equation solver. As the system evolution domain is set beforehand, the chosen window in the phase space cannot correctly track the system's resonant exploding growth (case $q_{\text{resonant}} = -8$) with the variance reaching the order of 10^3.

However, our solver forces the total density preservation inside the window, while in the resonant case, most of the density's mass should eventually escape the window. Nonetheless, this is not a fundamental limitation of the method and can be alleviated by moving (and expanding, if necessary) the window alongside the system trajectories, that is one of the points of the future research.

4.4. Susceptible-infected-recovered epidemiological model

The SIR model plays a crucial role in epidemiology, providing a framework to understand and predict the spread of infectious diseases within a population. Its simplicity allows for exploring various scenarios related to disease transmission, recovery rates, and vaccination strategies. The model helps estimate the potential impact of interventions and public health measures by categorizing individuals into susceptible, infectious, or recovered groups. Its application extends beyond infectious diseases, serving as a foundational tool for analyzing diverse phenomena in network dynamics, behavioral sciences, and risk assessment, offering valuable insights to guide health policies and interventions.

We consider a randomized SIR model with vital dynamics (birth and death) for the three-dimensional case. The SIR model is a well-known compartmental model in epidemiology. It is a nonlinear model that admits a closed solution only under certain circumstances. Particularly, we consider the following model:

$$S'(t,\omega) = \Delta(\omega) - \mu(\omega)S(t,\omega) - \beta(\omega)S(t,\omega)I(t,\omega), \qquad S(t_0,\omega) = S_0(\omega), \tag{4.9}$$

$$I'(t,\omega) = \beta(\omega)S(t,\omega)I(t,\omega) - \left(\gamma(\mu) + \mu(\omega)\right)I(t,\omega), \qquad I(t_0,\omega) = I_0(\omega), \tag{4.10}$$

$$R'(t,\omega) = \gamma(\omega)I(t,\omega) - \mu(\omega)R(t,\omega), \qquad R(t_0,\omega) = R_0(\omega), \tag{4.11}$$

where S, I and R denote the susceptible, infected, and recovered populations, respectively, Δ, μ denote the birth and death rates; and β, γ denote the susceptible–infected and infected–recovered transfer rates, respectively. We assume all parameters and the initial condition vector (S_0, I_0, R_0) are independent random variables.

With the SIR model definition Eqs. (4.9) and (4.11) and the distribution information from Table 4a, we may compute the maximum time step via the relations in Eqs. (2.25) and (2.26):

$$\sup\{\|J_t\|_2 : t \geq t_0, (S, I, \Lambda, \mu, \beta, \gamma) \in \tilde{D}\} \simeq 0.77703,$$

where $\tilde{D}$ denotes the joint positively invariant domain for all the random variables in the SIR model. Considering a RBF support radius of $6.49h$, we get the following time step bounds:

$$0 \leq \Delta t_{\max} \leq \frac{1}{0.77703}\log(6) \Rightarrow \Delta t_{\max} \simeq 2.31, \tag{4.12}$$

$$0 \leq \Delta t_{\min} \leq -\frac{1}{0.77703}\log\left(\frac{1}{6}\right) \Rightarrow \Delta t_{\min} \simeq 2.31, \tag{4.13}$$

which means that $\Delta t_{\text{Reinit}} \leq 2.31$, which is considerably larger than the previous examples.

[3] For simplicity of implementation in CUDA and due to the low impact of the integrator's order in the resulting simulation, we have used a Taylor series approximation of the matrix exponential. However, it is favorable to use diagonal Padé approximants.

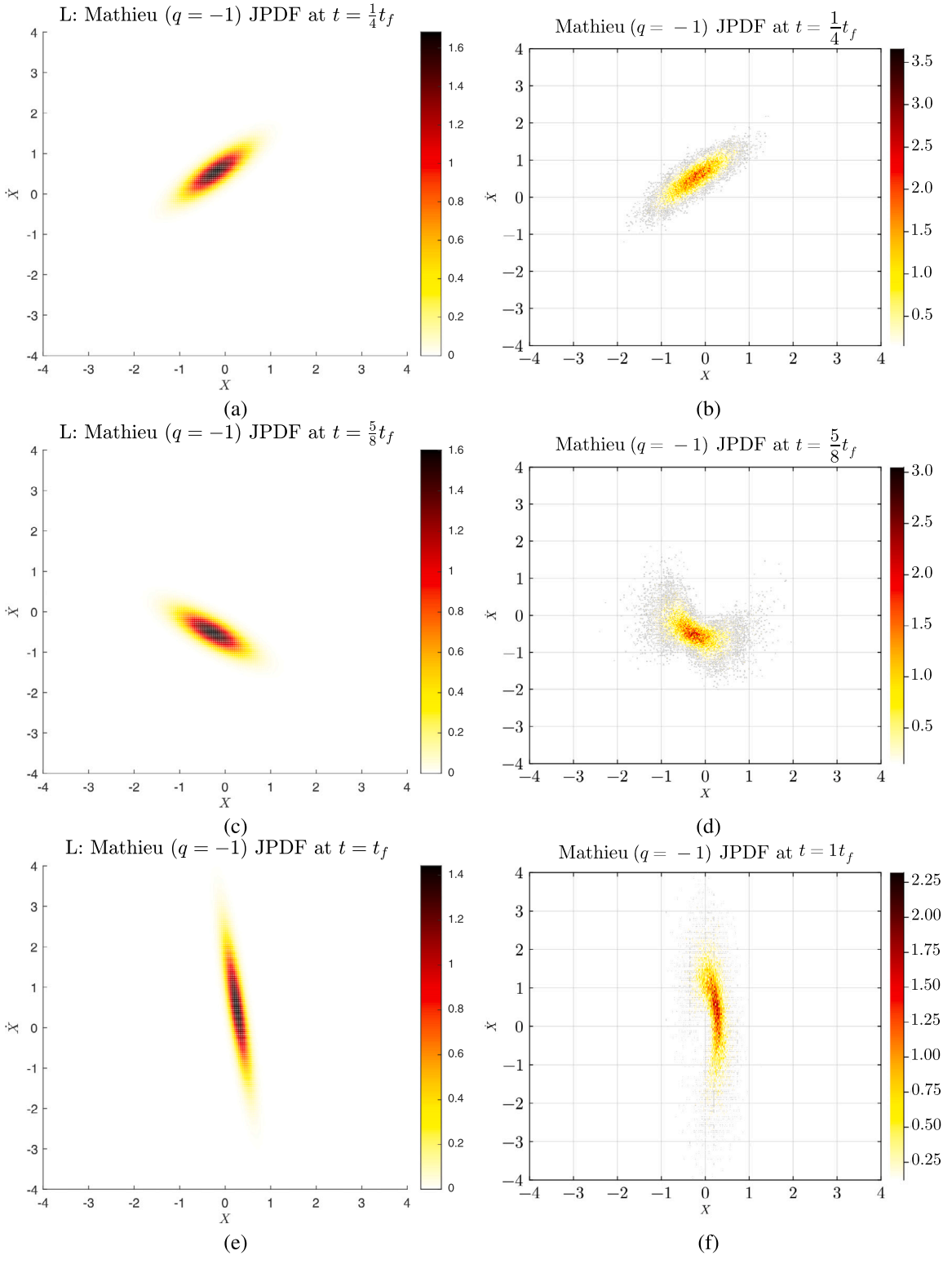


Fig. 10. Comparison of the joint probability density functions (JPDFs) of the solution to the Mathieu equation (stable case with $q=-1$) by the Liouville method (left) and the Monte Carlo simulation with $m=2^{14}$ samples (right). Here, $t_f=2\pi$ as defined in Table 3b.

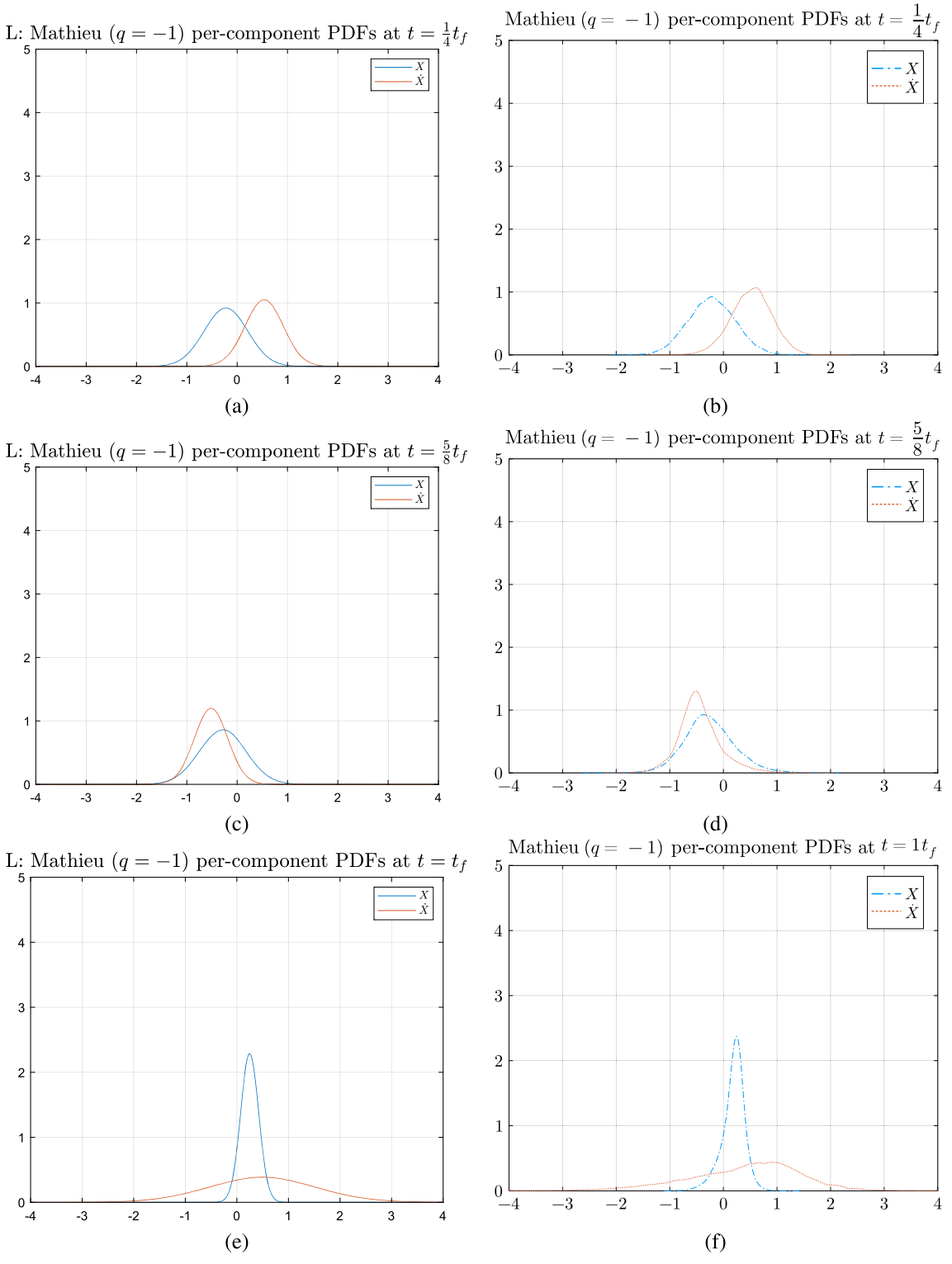


Fig. 11. Comparison of the PDFs of the components of the solution to the Mathieu equation (stable case $q = -1$) by the Liouville method (left) and the Monte Carlo simulation with $m = 2^{14}$ samples (right). Here, $t_f = 2\pi$ as defined in Table 3b.

Table 4
List of parameters for the random SIR model.

Par.	Distr.	Mean	Variance	Samples
S_0	Normal	0.75	0.0001	—
I_0	Normal	0.15	0.0001	—
R_0	Normal	0.1	0.0001	—
Δ	Delta	0.025	—	1
μ	Delta	0.025	—	1
β	Normal	0.3	0.0001	8
γ	Gamma	0.2	0.0001	8

(a) Random parameters' statistical information for the SIR model. The initial condition components do not have a prescribed numberof random samples as the AMR procedure defines them. All parameters are pairwise independent.

Input	Description
Δt	0.5
Δt_{Reinit}	1.0
Time span	$[0, 30]$
Domain	$[0,1]\times[0,1]\times[0,1]$
Domain points	$2^8 \times 2^8 \times 2^8$
AMR threshold	$1 \cdot 10^{-3}$

(b) List of the input parameters of the algorithm.

Figs. 12 show the difference between the marginal PDFs computed via MC and the Liouville equation solver. Once again, we can see that the MC simulation tends to separate particles more giving higher-variance PDFs, although the expected values and general structure of the PDF are very similar. We expect that a much higher number of samples would be needed to lead similar results.

Also, Figs. 13 show the 95% prediction surface of the PDF given by the Liouville equation solver and the MC version of the prediction surface for several time instants.[4] Once again, the Liouville equation solver improves the MC simulations giving a smooth surface, as expected by the nature of the random SIR model and showing the superiority of the Liouville equation in this kind of problems. The entire simulation in the [0, 30] timespan has been computed in ~ 4 minutes.

5. Conclusion

This contribution introduces and analyzes a novel numerical approach for efficiently solving the Liouville equation in the context of random ordinary differential equations (RODEs) using General-Purpose Graphics Processing Units (GPGPUs). Our methodology integrates wavelet compression-based adaptive mesh refinement, Lagrangian particle methods, and radial basis function approximation to develop a versatile, accurate and computationally efficient numerical algorithm.

We validated the performance of our approach through several mathematical models, including the van der Pol oscillator, Mathieu equation, and SIR model. These examples demonstrate the method's applicability to various problems and its compatibility with various numerical integrators for the underlying systems. Not to be overlooked are the illustration of the limitation of the method in its current implementation. Future research directions include:

- Investigating the interconnection of the method's parameters to enable their automatic adjustment for specific problems. Specifically, researching the quality and stability of the RBF interpolation.
- Developing a moving and adaptive window for domain integration representation.
- Implementing an inverse refining algorithm that begins with coarse approximations and progressively refines them.
- Finding an appropriate approach for higher dimensional systems ($d \geq 5$).

Our findings suggest that the numerical method described in the present contribution holds significant potential for quantifying forward uncertainty in random differential equations via probability density evolution, greatly improving on the information obtained by simply performing Montecarlo simulations.

CRediT authorship contribution statement

V.J. Bevia: Writing – review & editing, Writing – original draft, Visualization, Software, Investigation, Formal analysis, Conceptualization. **S. Blanes:** Writing – review & editing, Writing – original draft, Investigation, Formal analysis, Conceptualization. **J.C. Cortés:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Investigation, Funding acquisition, Formal analysis, Conceptualization. **N. Kopylov:** Writing – review & editing, Writing – original draft, Visualization, Investigation, Formal analysis, Conceptualization. **R.J. Villanueva:** Writing – review & editing, Writing – original draft, Supervision, Project administration, Investigation, Funding acquisition, Formal analysis, Conceptualization.

Acknowledgements

This work has been supported by the grant PID2020-115270GB–I00 granted by MCIN/AEI/10.13039/501100011033.

[4] Further details about the prediction region finder can be found in [13].

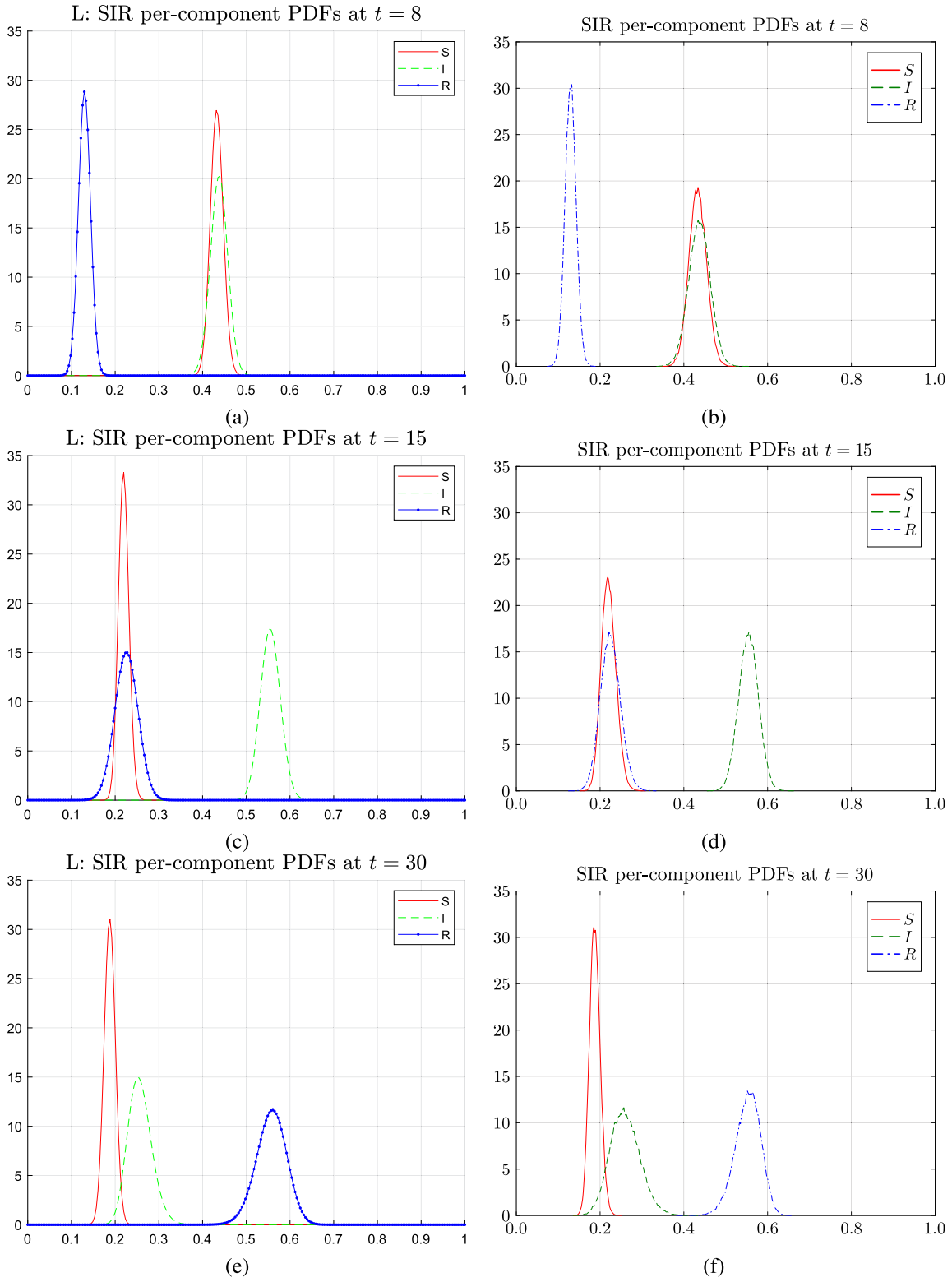


Fig. 12. Comparison of the marginal PDFs of the components of the solution to the SIR system by the Liouville method (left) and the Monte Carlo simulation with $m = 2^{16}$ samples (right). Here, $t_f = 30$ as defined in Table 4b.

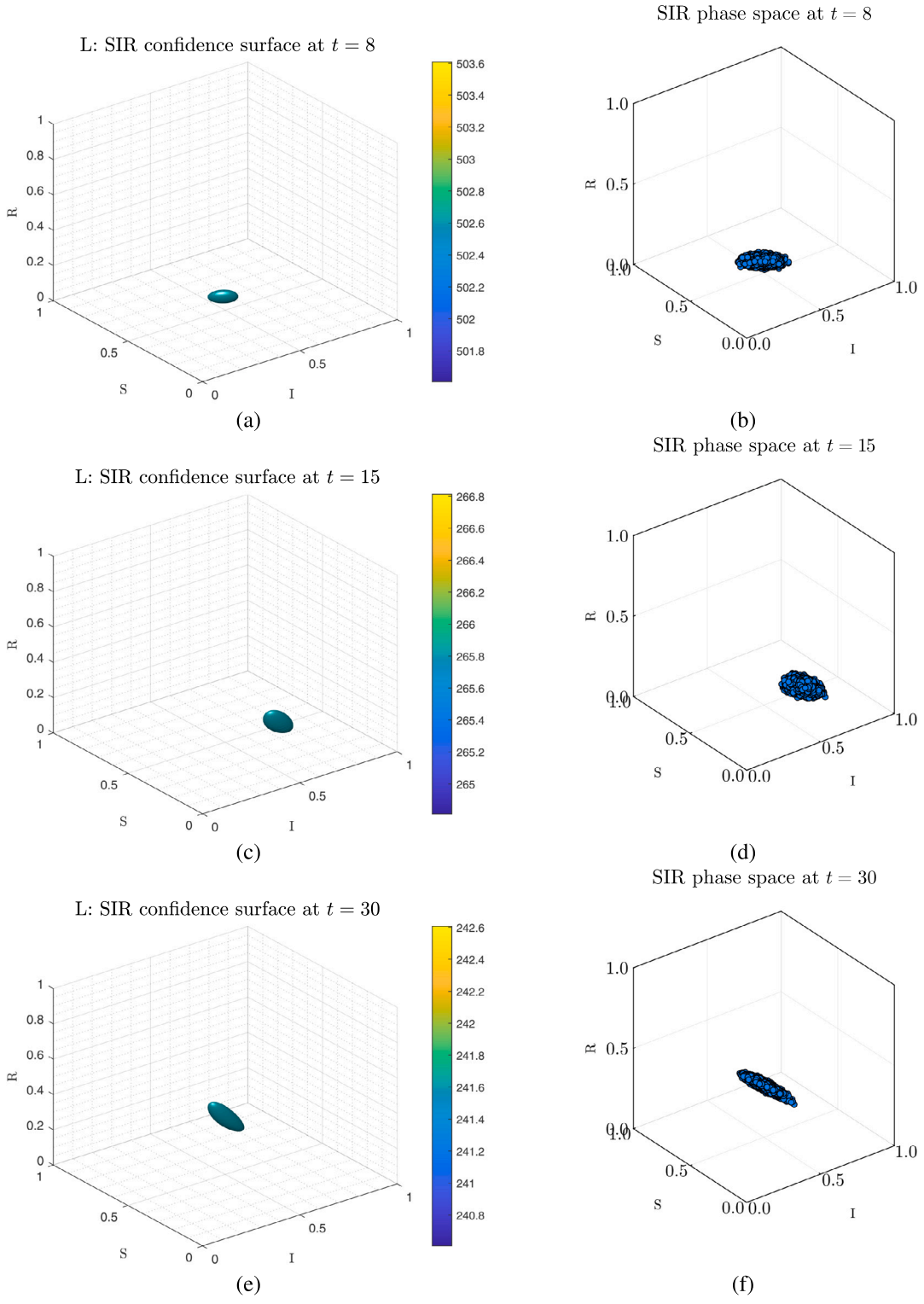


Fig. 13. Prediction ellipsoid for the SIR system solution by the Liouville method (left) and the phase space by the Monte Carlo simulation with $m = 2^{16}$ samples (right). Here, $t_f = 4$ as defined in Table 4b.

References

[1] A. C.-H., A. Jensen, Ripples in Mathematics: The Discrete Wavelet Transform, Springer, Berlin, Heidelberg, 2001.
[2] H.S.A.V. Skorokhod, F.C. Hoppensteadt, Random Perturbation Methods with Applications in Science and Engineering, Springer, 2002.
[3] P. Bader, S. Blanes, E. Ponsoda, M. Seydaoğlu, Symplectic integrators for the matrix hill equation, J. Comput. Appl. Math. 316 (May 2017) 47–59.
[4] G. Batchelor, An Introduction to Fluid Dynamics, Cambridge University Press, 1973.
[5] N. Bell, M. Garland, Efficient sparse matrix-vector multiplication on CUDA, Report NVR-2008-005, 2008.
[6] M. Bergdorf, P. Koumoutsakos, A Lagrangian particle-wavelet method, Multiscale Model. Simul. 5 (3) (2006) 980–995, 1.
[7] V.J. Bevia, N-Dimensional Liouville solver [computer software], https://doi.org/10.5281/zenodo.7673678, 2023. (Accessed February 2023).
[8] V.-J. Bevia, C. Andreu-Vilarroig, J.-C. Cortés, R.-J. Villanueva, Probability density function computation in evolutionary model calibration with uncertainty, in: J.E. Fieldsend, M. Wagner (Eds.), Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO'22, New York, NY, USA, 2022, pp. 1902–1908, 7, Association for Computing Machinery.
[9] V.-J. Bevia, C. Burgos, J.-C. Cortés, A. Navarro-Quiles, R.-J. Villanueva, Analysing differential equations with uncertainties via the Liouville-Gibbs theorem: theory and applications, in: D. Zeidan, S. Padhi, A. Burqan, P. Ueberholz (Eds.), Computational Mathematics and Applications, Springer, Singapore, 2020, pp. 1–23, Singapore.
[10] V.-J. Bevia, C. Burgos, J.-C. Cortés, R.-J. Villanueva, Uncertainty quantification of random microbial growth in a competitive environment via probability density functions, Fractal Fract. 5 (2) (2021) 26, 3.
[11] V.-J. Bevia, J. Calatayud, J.-C. Cortés, M. Jornet, On the generalized logistic random differential equation: theoretical analysis and numerical simulations with real-world data, Commun. Nonlinear Sci. Numer. Simul. 116 (2023) 106832, 1.
[12] V.-J. Bevia, J.-C. Cortés, M. Jornet, R.-J. Villanueva, Probabilistic analysis of a general class of nonlinear random differential equations with state-dependent impulsive terms via probability density functions, Commun. Nonlinear Sci. Numer. Simul. 119 (2023) 107097, 5.
[13] V.-J. Bevia, J.-C. Cortés, R.-J. Villanueva, Forward uncertainty quantification in random differential equation systems with delta-impulsive terms: theoretical study and applications, Math. Methods Appl. Sci. (2023) 1–21, 3.
[14] S. Blanes, F. Casas, A Concise Introduction to Geometric Numerical Integration, Chapman & Hall/CRC Monographs and Research Notes in Mathematics, Apple Academic Press, Oakville, MO, 2016, p. 5.
[15] P.A. Bosler, J. Kent, R. Krasny, C. Jablonowski, A Lagrangian particle method with remeshing for tracer transport on the sphere, J. Comput. Phys. 340 (2017) 639.
[16] C. Burgos, J.-C. Cortés, L. Villafuerte, R.-J. Villanueva, A mean square convergent numerical solutions of random fractional differential equations: approximations of moments and density, J. Comput. Appl. Math. 378 (2020) 112925, 11.
[17] J. Butcher, Runge–Kutta Methods, vol. 7, John Wiley & Sons, Ltd, 2016.
[18] S. Cook, CUDA Programming: A Developer's Guide to Parallel Computing with GPUs, 1st edition, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2012.
[19] G. Cottet, P. Koumoutsakos, High order semi-Lagrangian particle methods, in: M.L. Bittencourt, N.A. Dumont, J.S. Hesthaven (Eds.), Spectral and High Order Methods for Partial Differential Equations, ICOSAHOM 2016, vol. 119, Springer International Publishing, 2017, pp. 103–117, Cham.
[20] G. Dimarco, R. Loubère, J. Narski, T. Rey, An efficient numerical method for solving the Boltzmann equation in multidimensions, J. Comput. Phys. 353 (2018) 46.
[21] F. Dorini, M. Cecconello, L. Dorini, On the logistic equation subject to uncertainties in the environmental carrying capacity and initial population density, Commun. Nonlinear Sci. Numer. Simul. 33 (2016) 160.
[22] H. Evans, P. Kloeden, Random Ordinary Differential Equations and Their Numerical Solution. Probability Theory and Stochastic Modelling, Springer, Singapore, 2017.
[23] L.C. Evans, Partial Differential Equations, vol. 3, American Mathematical Society, Providence, R.I., 2010.
[24] G. Evtushenko, Sparse matrix-vector multiplcation with cuda, 2019.
[25] R.B.G. Casella, Statistical Inference, 4th edition, Cengage Learning, 2008, p. 3.
[26] C. Gasquet, P. Witomski, Fourier Analysis and Applications. Filtering, Numerical Computation, Wavelets, Springer, 1998.
[27] T.S.G.J. Lord, C. Powell, An Introduction to Computational Stochastic PDEs, Cambridge University Press, 2014.
[28] E. Hairer, C. Lubich, G. Wanner, Geometric Numerical Integration, 2 edition, Springer Series in Computational Mathematics, vol. 3, Springer, Berlin, Germany, 2006, p. 12.
[29] M. Hestenes, E. Stiefel, Methods of conjugate gradients for solving linear systems, J. Res. Natl. Bur. Stand. 49 (6) (1952).
[30] Y.C. Hon, R. Schaback, M. Zhong, The meshless kernel-based method of lines for parabolic equations, Comput. Math. Appl. 68 (2014) 2057–2067 (12, Part A).
[31] A. Hussein, H. Slama, M. Selim, A full probabilistic solution of a stochastic red blood cells model using RVT technique, Eur. Phys. J. Plus 136 (4) (2021) 381, 4.
[32] A. Iske, V.I. Arnold, Multiresolution Methods in Scattered Data Modelling, Springer Verlag, 2004.
[33] M. Jornet, Liouville's equations for random systems, Stoch. Anal. Appl. 40 (6) (2022) 1–22, 10.
[34] I. Kovacic, R. Rand, S. Mohamed Sah, Mathieu's equation and its generalizations: overview of stability charts and their features, Appl. Mech. Rev. 70 (2) (Feb. 2018).
[35] F.S.L. Debnath, Wavelet Transforms and Their Applications, 2nd edition, Springer Science + Business Media, New York, 2015, 2014.
[36] R.I. McLachlan, G.R.W. Quispel, Geometric integrators for odes, J. Phys. A, Math. Gen. 39 (19) (2006) 5251, 4.
[37] R.C. Mittal, S. Pandit, A numerical algorithm to capture spin patterns of fractional Bloch nuclear magnetic resonance flow models, J. Comput. Nonlinear Dyn. 14 (8) (May 2019).
[38] M. Mohammadi, R. Mokhtari, R. Schaback, A meshless method for solving the 2D Brusselator reaction-diffusion system, Comput. Model. Eng. Sci. 101 (2) (2014) 113–138.
[39] H. Munthe-Kaas, Runge-Kutta methods on Lie groups 38 (1) (1998) 92–111.
[40] M. Neumann, Lecture 13: radial basis function networks, https://www.cse.wustl.edu/~m.neumann/sp2016/cse517/lecturenotes/lecturenote13.html. (Accessed 31 July 2023).
[41] NVIDIA, P. Vingelmann, F. Fitzek, CUDA documentation, 2023, release: 12.2.
[42] W. Paul, Electromagnetic traps for charged and neutral particles, Rev. Mod. Phys. 62 (3) (July 1990) 531–540.
[43] C. Rackauckas, Q. Nie, Differentialequations.jl – a performant and feature-rich ecosystem for solving differential equations in julia, J. Open Res. Softw. 5 (1) (2017), Exported from https://app.dimensions.ai on 2019/05/05.
[44] P.S.R.G. Ghanem, Stochastic Finite Elements: A Spectral Approach, Springer, 1991.
[45] S.U.P.S. Papaulis, Probability, Random Variables and Stochastic Processes, Education, 4th edition, McGraw-Hill, 2015.
[46] F. Santambrogio, Optimal Transport for Applied Mathematicians. Calculus of Variations, PDEs and Modeling, Birkhäuser, Cham, 2015.
[47] J.M. Sanz-Serna, Stabilizing with a hammer, Stoch. Dyn. 08 (01) (Mar. 2008) 47–57.
[48] R. Schaback, H. Wendland, Using Compactly Supported Radial Basis Functions to Solve Partial Differential Equations, Southampton, Boston, 1999.
[49] R.C. Smith, Uncertainty Quantification: Theory, Implementation and Applications. Computational Science and Engineering, SIAM, New York, 2014.
[50] T.T. Soong, Random Differential Equations in Science and Engineering, Academic Press, New York, 1973.

[51] F.R.T. Neckel, Random Differential Equations in Scientific Computing, Versita, 2013.
[52] V.W.T. Plewa, T. Linde, Adaptive Mesh Refinement - Theory and Applications, Springer, Berlin, Heidelberg, 2005.
[53] T.T. Warner, Numerical Weather and Climate Prediction, Cambridge University Press, Cambridge, UK, 2012.
[54] H. Wendland, Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree, Adv. Comput. Math. 4 (1) (1995) 389–396, 12.
[55] H. Wendland, Error estimates for interpolation by compactly supported radial basis functions of minimal degree, J. Approx. Theory 93 (2) (1998) 258–272, 5.
[56] D. Xiu, Numerical Methods for Stochastic Computations: A Spectral Method Approach. Computational Science and Engineering, Princeton University Press, New Jersey, 2010, p. 7.
[57] H.R. Zadeh, M. Mohammadi, E. Babolian, Solving a class of PDEs by a local reproducing kernel method with an adaptive residual subsampling technique, Comput. Model. Eng. Sci. 108 (6) (2015) 375–396.