Journal of Computational Physics 488 (2023) 112157

Contents lists available at ScienceDirect

Journal of Computational Physics

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

The asymmetric particle population density method for simulation of coupled noisy oscillators

Ningyuan Wang [a], Daniel B. Forger [a,b,*]

[a] Department of Mathematics, University of Michigan, 530 Church St, Ann Arbor, 48109, MI, United States
[b] Department of Computational Medicine and Bioinformatics, 100 Washtenaw Ave Rm 2017, Ann Arbor, 48109, MI, United States

ARTICLE INFO

Article history:
Received 14 June 2021
Received in revised form 13 February 2023
Accepted 14 April 2023
Available online 5 May 2023

Keywords:
Convection-diffusion equation
Particle method
Hodgkin-Huxley model
Neuronal population
Population density approach

ABSTRACT

Many biological phenomena can be modeled by the collective activity of a population of individual units. A common strategy for simulating such a system, the population density approach, is to take the macroscopic limit and track a population density function. Here, we develop the asymmetric particle population density (APPD) method that efficiently and accurately simulates populations with complex behaviors that are infeasible for previous population density-based methods. The APPD method is well-suited for a parallel implementation. Our method can accurately reproduce complex macroscopic behaviors such as inhibitory coupling-induced clustering and noise-induced firing while being faster than the direct simulation. We compare the method's performance against direct Monte-Carlo simulation and verify its accuracy by applying it to the well-studied Hodgkin-Huxley model with a range of challenging scenarios.

1. Introduction

Many biological behaviors can be modeled by a large population of coupled oscillating elements, with examples including the rhythmic flashing of fireflies [1], the electrical activity of neurons [2], and circadian rhythm [3]. To simulate macroscopic behavior, one can take the number of elements to infinity and simulate the evolution as a continuous population density function. The mean field method is a classical and widely used approach to solve these problems [4–8]. However, a mean-field approach has severe constraints on the oscillator model, such as requiring the oscillator to follow a limit cycle closely or that the coupling is weak. A more direct approach is to simulate the population density directly. This approach is first used for a one-dimensional model as higher dimensions are considered prohibitively expensive [9]. In a more recent development, Stinchcombe and Forger [10] noted many of the models, while having a high dimensional state space, are dissipative with a coupling that concentrates the distribution; consequently, only a fraction of the possible system states have a non-negligible number of oscillators at any given time. As such, a population can be efficiently discretized by a particle method that only covers the distribution but not the entire state space. Here, we improve the method in two ways: First, we use an asymmetric kernel that covers the distribution more efficiently. Second, we simulate the diffusion term by deforming the particle independently instead of as an interaction between particles (as defined in [11]).

Using an asymmetric deforming kernel to solve a convection-diffusion equation or a Fokker-Planck equation is not new. We took our inspiration from a very different context of Kalman-Bucy filtering [12–16] which works when the system's dynamics can be approximated as locally linear. In the context of convection-diffusion equations, this has also been discussed

* Corresponding author.
E-mail address: forger@umich.edu (D.B. Forger).

https://doi.org/10.1016/j.jcp.2023.112157
0021-9991/

in [17,18]. Unlike Kalman-Bucy filtering, however, the population density cannot be satisfactorily represented using a single multivariate normal distribution.

An asymmetric particle method must split and combine particles as they deform and the local linear approximation is no longer satisfactory. Such particle methods are first investigated in [19], with later works [20,21]. In a very recent work by Berchet [22], they investigated the mass transport problem with a similar update-split-combine scheme that we are introducing here.

However, the previous works are still insufficient for the problem of interest: previous particle methods are restricted to 2-dimensional scenarios, whereas the Hodgkin-Huxley model [2] has a 4-dimensional state space. The higher dimension poses two main challenges: numerical stability with finite-precision linear algebra and an exponentially growing space requirement. We designed the method with these constraints in mind. First, our method naturally tracks a *square-root* of the covariance, which gives improved numerical stability, as known in the Kalman filter community [12,13] but has not been introduced in this context. Second, we avoid the need for Ito-Taylor expansion with an *a priori* timestep to represent diffusion, which enables an adaptive timestep method to be directly applied. Third, for the splitting particle criteria, we avoided using Laplacian as in [22] and used a single-direction criterion to avoid numerical instability, as the covariance matrix can be close to a singular (see numerical example in Fig. 8). Fourth, splitting a single particle into three particles with optimized weight and distance introduces less error than a 2-particle split as in [22]. Fifth, for combining particles, we introduced hashed cubic blocks and restricted particle combinations within each block only to speed up the computation without exhausting computer memory.

The remainder of the paper is structured as follows: Sec. 2 introduces the method, with an overview in Sec. 2.1 followed by a detailed discussion in the following subsections. In Sec. 3, we look at several numerical examples that illustrate our method's working and show its advantages.

2. Method

2.1. Overview of the method

A large population of coupled noisy oscillators is commonly simulated by Monte-Carlo methods. Here, we use the Monte Carlo method as described in [23] for comparison. In the direct Monte-Carlo method, each oscillator is updated individually based on its dynamics, noise, and an averaged coupling. The intuition for our method is simple: a particle not only represents oscillators exactly at one state but also nearby oscillators with some locally linear approximation. We now discuss how this is implemented: how the system's dynamics and noise are updated for a single particle in a deterministic algorithm, how the local linearity is preserved by splitting the particle, and other technicalities that arise.

In the asymmetrical particle population density (APPD) method, a single particle is a weighted multivariate Gaussian. Simulating the probability distribution function (PDF) of a similar stochastic process using a single multivariate Gaussian is a well-studied topic in the context of Kalman-Bucy filtering [24] and its generalizations [12–15]. These methods utilize the fact that a Gaussian function is preserved if the system's dynamics are linear in space. Therefore, when the PDF is concentrated in a small region where the ordinary differential equation (ODE) can be considered approximately locally linear, and the Gaussian is preserved. However, these existing methods for Kalman-Bucy filtering [12–15] are based on the Itô-Taylor expansion of the underlying equation and require a pre-determined fixed timestep. In [16], we proposed the Level Set Kalman Filter (LSKF), which takes arbitrary adaptive timestep and has superior accuracy than the Continuous-Discrete Cubature Kalman Filter. Therefore we base the particle in our proposed method on the *time-update* of the LSKF.

While the Fokker-Planck equation for the LSKF and the convection-diffusion equation for the population takes the same form, some adjustments are needed. First, oscillators can have a limit cycle with strong contraction, resulting in the PDF being very thin in certain directions, corresponding to a close-to-singular covariance matrix in a multivariate-Gaussian approximation. Therefore the method has to be robust with respect to singular covariance. The LSKF is suitable as it works with a square root of the covariance matrix and is demonstrated to be robust for semi-positive definite covariances in [16]. The next challenge is that the distribution is more complicated: in Kalman-Bucy filtering, continuous or periodic measurements keep the distribution to be closely approximated by a multivariate distribution, and the distribution is concentrated in a small volume where the dynamics can be approximated by a local linear approximation centered at mean of the distribution. In the absence of such measurements, the distribution can no longer be approximated by a single normal distribution; instead, a linear combination of many particles, each having their own local linear approximation is used to approximate the distribution. This also introduced the need for a scheme that splits and combines particles.

2.2. Problem formulation

We simulate a population consisting of identical oscillators subject to dynamics described by a velocity field $\mathbf{v}$, and noise that can be described by a Wiener process. We assume that an oscillator can be described by a d-dimensional state variable. Suppose that the oscillators are also subject to noise; then this probability density u is evolved according to the following convection-diffusion equation:

$$\frac{\partial u}{\partial t} = \nabla \cdot \mathbf{K} \nabla u - \nabla \cdot (\mathbf{v} u) \tag{1}$$

in which $\mathbf{K}$ is a constant **diffusion matrix**, and $\mathbf{v} = \mathbf{v}(\mathbf{x}, u)$ is the **convection velocity** that corresponds to the ordinary differential equation of each oscillator, and the coupling between oscillators. In this work, we assume $\mathbf{K}$ and $\mathbf{v}$ are constant in time or changing slowly so that their time derivatives can be omitted.

We further assume that the population is infinitely large such that individual fluctuations alone would not affect the population. Hence the probability distribution (1) is the population distribution function of the entire population. This assumption is essential as the coupling would be more complex otherwise.

To accommodate coupling, we assume the velocity field takes the following form:

$$\mathbf{v} = \mathbf{v}_d(\mathbf{x}) + \mathbf{v}_c(u, \mathbf{x}) \tag{2}$$

in which $\mathbf{v}_d$ is the **dynamic velocity** corresponding to the dynamics that are determined by the state variable $\mathbf{x}$ only. $\mathbf{v}_c$ is the **coupling velocity** corresponding to the coupling term that depends on the state variable $\mathbf{x}$, and the population distribution u at the same time. Moreover, for each particle, we take a local linear approximation and assume the coupling velocity $\mathbf{v}_c$ is of the following form:

$$\mathbf{v}_c(u, \mathbf{x}) = \mathbf{w}_c(\mathbf{x}) \cdot \mathbf{L}[u] \tag{3}$$

in which $\mathbf{w}_c(\cdot)$ is a vector-valued function, $L[\cdot]$ is a linear functional on the PDF u.

In principle, the dependence of coupling velocity on the density makes (1) a nonlinear differential equation, which gives rise to complex behaviors. However, since the coupling velocity $\mathbf{v}_c$ as defined in (3) depends on global quantities of u over the domain and can be assumed to change relatively slowly for our applications. Therefore, we approximate $\mathbf{v}_c = \mathbf{v}_c(\mathbf{x})$ to be fixed except when the velocity function $\mathbf{v}_c$ is updated at some fixed timesteps. This approximation keeps (1) linear in the following derivations.

A discretization is required to simulate the population density numerically. In a **particle method**, the population density is discretized by approximating the population density as a linear combination of **particles**:

$$u(\mathbf{x}) \approx \sum_{i \in I} w_i K_i(\mathbf{x} - \mathbf{x_i}) \tag{4}$$

where for each particle with index i, $K_i(\mathbf{x})$ is the **kernel function** of the particle, w_i is the **weight** of the particle, and $\mathbf{x_i}$ is the **center location** of the particle. It should be noted that the linear approximation $L[\cdot]$ defined in (3) is different for each particle, as they have different center locations.

In our method, each particle is a weighted multivariate Gaussian. In particular, each Gaussian particle is represented by its **weight** w_i, **center location** $\mathbf{x}_i$ and **covariance matrix** $\mathbf{\Sigma}_i$, and its **kernel function** K_i is given by:

$$K_i(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^d \det(\mathbf{\Sigma}_i)}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mathbf{x}_i)^T \mathbf{\Sigma}_i^{-1} (\mathbf{x} - \mathbf{x}_i)\right). \tag{5}$$

To improve the numerical stability and derive a simpler update algorithm, we track and store a **square root** of the covariance matrix $\mathbf{\Sigma} = \mathbf{M}\mathbf{M}^T$ instead and only evaluate the covariance matrix when needed.

2.3. Single particle update

This section describes the particle update algorithm, which is a special case of the Level Set Kalman Filter method as derived in [16]. Only the description is included here. Readers interested in the derivation and proof should refer to [16].

Recall (1):

$$\frac{\partial u}{\partial t} = \nabla \cdot \mathbf{K} \nabla u - \nabla \cdot (\mathbf{v} u). \tag{6}$$

For a single particle centered at $\mathbf{x}_0$, the probability density function is given by:

$$u(\mathbf{x}, 0) = \frac{1}{\sqrt{(2\pi)^d \det(\mathbf{\Sigma})}} \exp\left(-\frac{(\mathbf{x} - \mathbf{x}_0)^T \mathbf{\Sigma}^{-1} (\mathbf{x} - \mathbf{x}_0)}{2}\right). \tag{7}$$

Consider the **level set** of the function F:

$$\left\{\mathbf{x} | F(\mathbf{x}, t) := \frac{u(\mathbf{x}, t)}{u(\mathbf{0}, t)} = c\right\} \quad (0 < c < 1). \tag{8}$$

For u being a Gaussian particle, all level sets are ellipsoids. Tracking the movement of the Gaussian particle is equivalent to tracking one of its ellipsoid level sets as defined in (8). A factorization of the covariance matrix $\mathbf{\Sigma} = \mathbf{M}\mathbf{M}^T$ (where $\mathbf{M}$ is called a **square root** of $\mathbf{\Sigma}$) represents a level set in the sense that the column vectors lie on the same ellipsoid. We represent the columns of the matrix $\mathbf{M}$ as $\mathbf{M}_i$, and advance them in time. The ordinary differential equation is given by:

$$\frac{\partial \mathbf{M}_i}{\partial t} = \frac{1}{2}\left(\mathbf{v}(\mathbf{x}_0 + \mathbf{M}_i) - \mathbf{v}(\mathbf{x}_0 - \mathbf{M}_i)\right) + \mathbf{K}(\mathbf{M}^T)^{-1}\mathbf{e}_i \tag{9}$$

where $\mathbf{x}_0$ is the **center location** of the Gaussian particle, and the matrix inverse is understood as solving the equation with a backward-stable solver.

A matrix short-hand form is as follows (where the matrix-vector additions are defined entrywise):

$$\frac{\partial \mathbf{M}}{\partial t} = \frac{1}{2}\left(\mathbf{v}(\mathbf{x}_0 + \mathbf{M}) - \mathbf{v}(\mathbf{x}_0 - \mathbf{M})\right) + \mathbf{K}(\mathbf{M}^T)^{-1} \tag{10}$$

and the velocity for the center is given by:

$$\frac{\partial \mathbf{x}_0}{\partial t} = \frac{1}{2d}\sum_{i=1}^{d} \mathbf{v}(\mathbf{x}_0 + \mathbf{M}_i) + \mathbf{v}(\mathbf{x}_0 - \mathbf{M}_i). \tag{11}$$

Notice that to evaluate the velocity for one point $\mathbf{x}_i$ on the level set, the center location, and all other points $\mathbf{x}_j$ for this Gaussian kernel are needed, hence the points on a level set cannot be updated independently, and the dimension for the ODE solver is $d \times (d+1)$. However, the updates of different Gaussian particles are independent.

As a summary, a particle is updated as follows (Table 1):

Table 1
The particle update algorithm.

Algorithm 1 Particle Update.
Require: center $\mathbf{x}$, and a square root $\mathbf{M}$ of the covariance matrix at the previous time step. Pass $\mathbf{M}$ and $\mathbf{x}$ as the state variable to an ODE solver with derivative defined by (10).
return center $\mathbf{x}'$, and a square root $\mathbf{M}'$ of the covariance matrix.

2.4. Splitting a particle

Since updating the center location and the square root of the covariance matrix for a single particle uses a linear approximation of the convection velocity, we need a method to control the size of the Gaussian particle that would otherwise expand due to diffusion. Though the Gaussian particles have infinite support, most of the density is within 2 standard deviations in each eigenvector direction, and we consider this region to be its effective support. In [22], the Laplacian is used as the criterion for the quality of the local linear approximation. However, we would like to avoid the use of an explicit Laplacian, as it is one of the advantages of LSKF [16]. Numerical differentiation to find the Laplacian turns out to be not numerically stable with the close-to-singular covariance matrices. (This is investigated in Sec. 3.2.) With the numerical stability issue in mind, we check the soundness of the linear approximation in the direction of the off-center points used in the particle update. We decide if the particle needs to be split if this particular relative error for any of the off-center points is larger than a threshold chosen by the user:

$$\epsilon = \frac{\|(\mathbf{v}(\mathbf{x} + 2\Delta\mathbf{x}_i) - \mathbf{v}(\mathbf{x})) - 2(\mathbf{v}(\mathbf{x} + \Delta\mathbf{x}_i - \mathbf{v}(\mathbf{x})))\|}{2\|\mathbf{v}(\mathbf{x})\|} \tag{12}$$

where $\mathbf{x}$ is the center of the particle, and $\Delta\mathbf{x}_i$ is the offset of the off-center point in direction i. The choice of ϵ has a significant effect on computational speed and is related to the combination of particles discussed in Sec. 2.5. In Sec. 3.2, we will further investigate the relation between the tolerance ϵ and the computation speed. For the numerical examples that follow, unless otherwise noticed, the value is chosen as 0.05.

Once the velocity field $\mathbf{v}$ in the effective support deviates from the linear approximation larger than this tolerance, we need a method to reduce the covariance, thereby reducing the effective support of the particle. We achieve this by splitting the particle into three thinner particles in the corresponding direction with half variance in that direction.

Up to some rotation, the kernel function (5) can be reformulated as:

$$K_i(\mathbf{x}) = \frac{1}{(2\pi)^{d/2}\sqrt{\sigma_1\sigma_2\ldots\sigma_d}} \exp\left(-\frac{1}{2}\left(\frac{x_1^2}{\sigma_1} + \frac{x_2^2}{\sigma_2} + \cdots + \frac{x_d^2}{\sigma_d}\right)\right) \tag{13}$$

in which $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_d$ are the eigenvalues of $\mathbf{\Sigma}$. We here consider a method to reduce the width of the population in x_i direction by representing the original population as a summation of three particles with variance in x_i direction reduced by half. The particle in the center would weigh $1 - 2\omega$, and have its location the same as what it replaced, while the two off-center particles would both weigh ω, and have their center location shift by a and $-a$ in the direction of x_i respectively. Note the weight of the three children particles add up to 1, and the total weight of the particle is conserved after the operation.

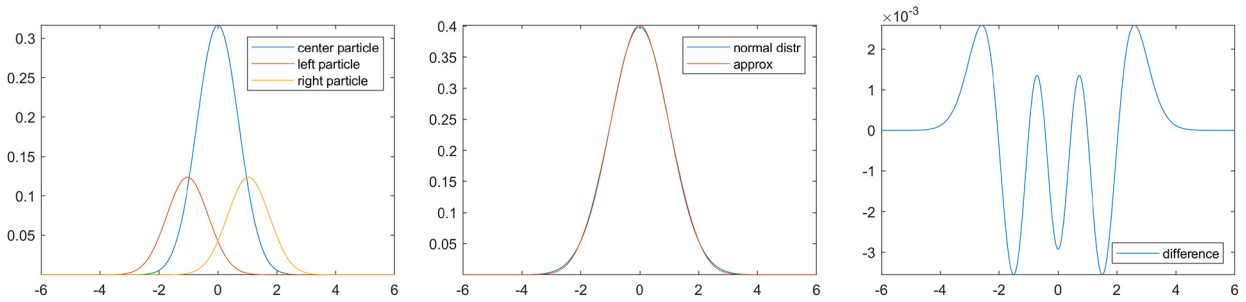

Fig. 1. Optimal one-dimension particle split with standard deviation 1 to three particles with standard deviation $\frac{1}{\sqrt{2}}$. The left panel shows the three components of the split particle with $a = 1.0332$ and $\omega = 0.21921$ as defined in (14). The central panel shows the normal distribution and its approximation using the three particles defined in the left panel. The right figure shows the difference between the normal distribution and its approximation. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

Table 2
The Eigen-decomposition particle split algorithm.

Algorithm 2 Particle Split (Eigen-decomposition).
Require: Particle with weight 1 centered at origin with a Diagonal covariance matrix $\mathbf{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_d)$
Define the new Diagonal covariance matrix $\mathbf{\Sigma}' = \text{diag}(\sigma_1/2, \sigma_2, \ldots, \sigma_d)$
Find center locations of the off-center particles given by $\mathbf{x}_c = (1.03332\sqrt{\sigma_1}, 0, \ldots, 0)$
Construct center particle centered at the origin, with weight $1 - 2\omega$ and covariance $\mathbf{\Sigma}'$
Construct left and right particle centered at $\pm\mathbf{x}_c$, with weight ω and covariance $\mathbf{\Sigma}'$
Apply this method to the new particles until all variances are sufficiently small

In order to optimally find the value of the parameter of ω and a that is independent of the dimension of the state space, we first consider the following optimization problem in a single variable x:

$$\min_{a,\omega} \max_{x} \left| K(x|0,\sigma^2) - ((1-2\omega)K(x|0,\frac{\sigma^2}{2}) + \omega K(x|-a,\frac{\sigma^2}{2}) + \omega K(x|a,\frac{\sigma^2}{2})) \right| \tag{14}$$

Where $K(x|\mu,\sigma^2)$ is the probability density function of the normal distribution with expectation μ and standard deviation σ. Using numerical optimization, we find $a = 1.03332\sigma, \omega = 0.21921$.

Then, for a multivariate normal distribution, note that in an eigendecomposition, the probability distribution formed by the three particles is still a normal distribution when projected into the orthogonal eigenvectors. Additionally, (14) is defined without dimension-dependent terms. Consequently, this process can be repeated in other directions with the a and ω still optimal until variance in all directions are sufficiently small such that (12) is satisfied. This process introduces a maximum relative error of 0.73% for each iteration. This is enough to bias a careful analysis of convergence as the size of particles is reduced (and thus, we do not provide convergence results). However, it does offer sufficient accuracy in our simulations since the error introduced has little effect on the linear coupling velocity term. Therefore, the error introduced does not significantly affect the global behavior of the population, as will be shown in the examples below. If higher accuracy is required, one needs to reduce the difference between the variance of the particles. The method and a demonstration of the split is in Fig. 1 (Table 2).

The above derivation uses the eigendecomposition of the covariance matrix, which could be expensive and undesirable in some applications. However, as the error introduced is so small, we would like to apply this method with the identical a and ω for an arbitrary matrix decomposition as well. While this choice of parameters may no longer be optimal (as the claim *projection of split distribution are normal* no longer holds), the error bound 0.73% is still valid.

Specifically, given a particle centered at the origin with weight 1, and a covariance matrix is given by its square root decomposition $\mathbf{\Sigma} = \mathbf{M}\mathbf{M}^T$, suppose split is needed along direction $\mathbf{M}_1$ (the first column vector of $\mathbf{M}$), then the children particles are computed as follows:

- Center particle: centered at the origin, weight $1-2\omega$, covariance matrix defined by the square root decomposition $\mathbf{N}\mathbf{N}^T$;
- Left particle: centered at $1.03332\mathbf{M}_1$, weight ω, covariance matrix defined by the square root decomposition $\mathbf{N}\mathbf{N}^T$;
- Right particle: centered at $-1.03332\mathbf{M}_1$, weight ω, covariance matrix defined by the square root decomposition $\mathbf{N}\mathbf{N}^T$.

w is equal to 0.21921, and the matrix $\mathbf{N}$ defined column-wise by:

$$\mathbf{N}_i = \mathbf{M}_i - \left(1 - \frac{1}{\sqrt{2}}\right) \frac{<\mathbf{M}_1, \mathbf{M}_i>}{<\mathbf{M}_1, \mathbf{M}_1>} \mathbf{M}_1 \tag{15}$$

in which $<\cdot,\cdot>$ denotes the inner product.

Table 3
The square root particle split algorithm.

Algorithm 3 Particle Split (square root).
Require: A particle with weight 1 centered at the origin and a square root decomposition $\mathbf{M}\mathbf{M}^T$ of the Covariance matrix Find new square root decomposition $\mathbf{N}$ define in (15) Find center locations of the off-center particles $\mathbf{x}_c = 1.03332\mathbf{M}_1$ Construct center particle centered at the origin, with weight $1-2\omega$ and decomposition $\mathbf{N}$ Construct left and right particle centered at $\pm\mathbf{x}_c$, with weight ω and decomposition $\mathbf{N}$ Apply this method to the new particles until all variances are sufficiently small

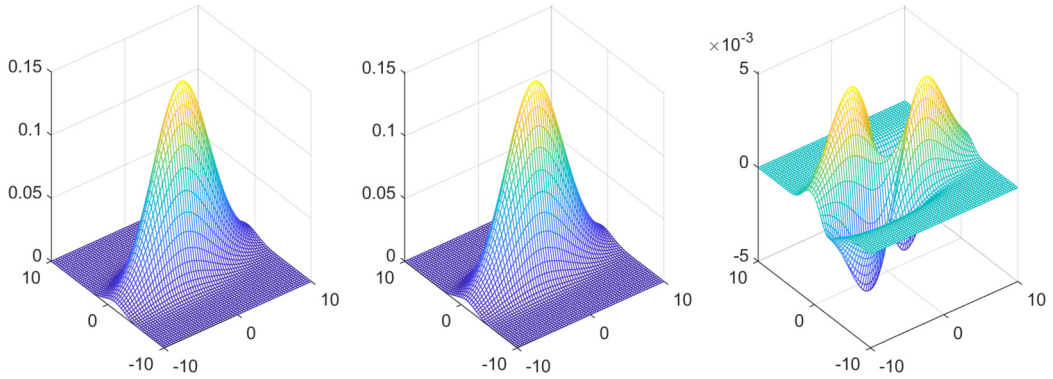

Fig. 2. Particle split along an arbitrary direction. The left figure shows the PDF of a particle with covariance $\left[\begin{smallmatrix}16 & 2\\ 2 & 3\end{smallmatrix}\right]$. The middle figure and right figure show the three-particle approximation and absolute error introduced when the particle is split along $(0,1)$ direction, which is not an eigenvector.

It can be seen in Fig. 2 that the error introduced has more complicated shapes, and the projection of the split distribution into the other direction is no longer a normal distribution. A summary of the split algorithm is given by Table 3.

2.5. Combining particles

The previous method introduces new particles to the system. Therefore, a method to remove particles is required: otherwise, the number of particles can grow geometrically, rendering this method impossible to use.

The description of the method for combining the particles can be divided into two parts: first, given a set of particles, a method to combine these particles into a single one optimally; second, a strategy to determine which particles should be merged into a single one.

First, consider the method for combining particles. With the requirement that the total weight adds to 1, we find a multivariate normal distribution that best approximates the PDF of the old population. Therefore we choose the new particle such that it has the same mean (center location) and covariance matrix of the distribution, and the weight is the sum of all the particles to be combined, conserving the total weight.

Consider a probability distribution consist of N particles with each particle approximated as:

$$p(\mathbf{x}) = \sum_{n=1}^{N} \frac{w_n}{\sqrt{(2\pi)^d \det(\boldsymbol{\Sigma}_n)}} \exp\left(-\frac{(\mathbf{x}-\boldsymbol{\mu}_n)^T \boldsymbol{\Sigma}_n^{-1}(\mathbf{x}-\boldsymbol{\mu}_n)}{2}\right) \tag{16}$$

where ω_n, $\boldsymbol{\mu}_n$, and $\boldsymbol{\Sigma}_n$ are the weight, center location, and covariance matrix for the nth component particle.

Then the expectation of the new distribution $\boldsymbol{\mu}$ is the weighted average of $\boldsymbol{\mu}_n$:

$$\boldsymbol{\mu} = \frac{1}{\Omega}\sum_{n=1}^{N} w_n \boldsymbol{\mu}_n \tag{17}$$

where $\Omega = \sum_{n=1}^{N} \omega_n$ is the sum of particle weights.

$(\boldsymbol{\Sigma})_{ij}$, the ijth entry of the covariance matrix $\boldsymbol{\Sigma}$ for the combined particle, is given by the definition

$$(\boldsymbol{\Sigma})_{ij} := E[(X_i-\mu_i)(X_j-\mu_j)] = \frac{1}{\Omega}\sum_{n=1}^{N} \omega_1 E\left[(Y_{ni}-\mu_i)(Y_{nj}-\mu_j)\right] \tag{18}$$

where X_i denotes the ith component of the random variable $\mathbf{X}$ (note X_i is not bold to indicate that it is a *scalar*), μ_i is the ith component of the mean $\boldsymbol{\mu}$, $\mathbf{Y}_n$ is the normal random variable corresponding to the nth particle with expectation $\boldsymbol{\mu}_n$ and covariance matrix $\boldsymbol{\Sigma}_n$, and Y_{ni} denotes the ith component of the nth particle.

A single summand in the right hand side of (18) can be evaluated as:

$$\begin{aligned} E[(Y_{ni}-\mu_i)(Y_{nj}-\mu_j)] &= E\left[(Y_{ni}-\mu_{ni}+(\mu_{ni}-\mu_i))(Y_{nj}-\mu_{nj}+(\mu_{nj}-\mu_j))\right] \\ &= E\left[(Y_{ni}-\mu_{ni})(Y_{nj}-\mu_{nj})\right]+(\mu_{ni}-\mu_i)E(Y_{nj}-\mu_{nj}) \\ &+(\mu_{nj}-\mu_j)E(Y_{ni}-\mu_{ni})+(\mu_{ni}-\mu_i)(\mu_{nj}-\mu_j) \\ &+(\mu_{nj}-\mu_j)E(Y_{ni}-\mu_{ni})+(\mu_{ni}-\mu_i)(\mu_{nj}-\mu_j) \\ &+(\mu_{nj}-\mu_j)E(Y_{ni}-\mu_{ni})+(\mu_{ni}-\mu_i)(\mu_{nj}-\mu_j) \\ &= (\boldsymbol{\Sigma}_n)_{ij}+(\mu_{ni}-\mu_i)(\mu_{nj}-\mu_j) \end{aligned} \tag{19}$$

in which μ_{nj} is the jth component of $\boldsymbol{\mu}_n$, and $(\boldsymbol{\Sigma}_n)_{ij}$ is the ijth entry of $\boldsymbol{\Sigma}_n$. Therefore, we find that the covariance matrix M is given by:

$$(\boldsymbol{\Sigma})_{ij}=\frac{1}{\Omega}\sum_n w_n\left((\boldsymbol{\Sigma}_n)_{ij}+(\mu_{ni}-\mu_i)(\mu_{nj}-\mu_j)\right). \tag{20}$$

In matrix form:

$$\boldsymbol{\Sigma}=\frac{1}{\Omega}\sum_n w_n\left(\boldsymbol{\Sigma}_n+(\mu_n-\mu)(\mu_n-\mu)^T\right). \tag{21}$$

Then we describe the method to determine which particles should be combined. We choose criteria for combining particles based on their distance being sufficiently close, and the combined particle does not need to be immediately split. Since we are more concerned about the computation time for finding combinations rather than having the minimum count of new particles, here we describe a method that scales linearly in the count of particles.

To try combining a particle with others, one can first find all neighbors within a fixed radius, then combine with these neighbors, and iterate through all particles. While finding all particles within a radius threshold appears to need to compute pairwise distance, it turns out that this fixed-radius near neighbors problem achieves linear scaling in the number of particles [25]. Unfortunately, there is a lack of implementation for these algorithms for dimensions higher than 3. Here we describe a much-simplified version that only finds a subset of all neighbors within a fixed radius since we are only interested in reducing particles quickly, not optimally.

The method is based on a subdivision of the state-space into cubic-shaped buckets and using a hash function to utilize the sparsity, as illustrated in Fig. 4. For the sake of simplicity, let us assume the combining criteria is that the distance between particles is less than radius r. Since the particles are not structured, a pairwise computation of the distance between particles would scale quadratically with the number of particles. However, we divide the state space into cubic-shaped buckets with sidelength larger than r, and only lookup for pairs of particles within each bucket, then the computational should scale by the number of particles multiplied by the average number of particles in a bucket. Table 4 gives an outline of the method.

The challenge is that this would use a prohibitively large amount of memory, as discretization is required in every dimension. However, as the population is sparsely distributed in the whole space, we can use a hash function to store the occupied buckets: First, we pick the sidelength of the spatial grid. Since the state space is bounded, each cubic bucket can be indexed by its order in each direction. Then, we iterate through particles to find the buckets they are in and store the nonempty buckets in a hash table. After all the points are checked, we iterate through each nonempty bucket and combine particles inside into a single particle.

The advantage of using a hash table is that it is asymptotically efficient in both space and time: While the length of the keys grows linearly with the spatial dimension d, the time taken to evaluate the hash function can be considered a constant. Under this assumption, the average case writes and access time for a hash table of the occupied bins are practically constant. Hence we achieve practically linear scaling in the number of particles and constant scaling in the number of dimensions if the number of particles is fixed. A hash table also takes space that asymptotically grows linearly with the number of occupied buckets. Hash tables are also suitable for a parallel algorithm. In our implementation, we used the `concurrent_hash_map` included in the open-source `intel oneAPI` library.

The choice of grid sidelength is controlled by the parameter τ. The choice of optimal τ is affected by both the characteristic of the problem and the tolerance defined in (12). Larger grids miss fewer possible combinations at the expense of more pairwise computations, as illustrated in Fig. 3. The effect of τ in computation time is further discussed in Sec. 3.2 with actual timing in Fig. 7. In addition to combining particles, we remove particles with very small (less than 10^{-8} of the population) weights due to successive splitting without combination and redistribute the removed weight evenly to all particles. This is needed since one can otherwise run into floating point underflow, which could then cause a division by zero error.

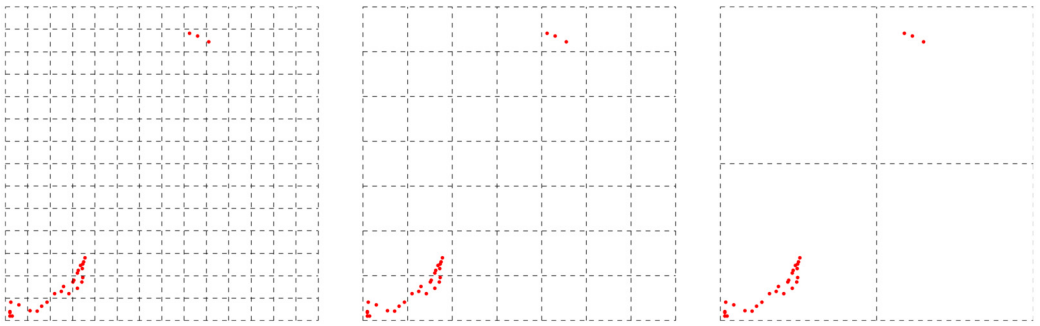

Fig. 3. Tradeoff for grid sidelength: larger grids miss fewer possible combinations between cubic blocks, at the expense of more pairwise comparisons.

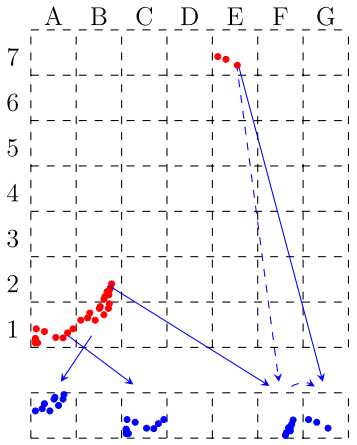

Fig. 4. Particle combination with hashed buckets: Instead of storing all the buckets A1 to G7, only the hash table is stored in memory. A hash function (mapping indicated by arrows) maps occupied buckets to the hash table. In case of clashing hash values (indicated by a dashed arrow), the hash table automatically maps to an unoccupied place in the hash table. The combine particle algorithm only checks within each bucket.

Table 4
The particle combine algorithm.

Algorithm 4 Particle Combine.

Divide the state space into sufficiently small grids, and find which grid block each particle has its center located (using a Hash function)
for each grid block containing more than one particle **do**
combine particles with center location defined as (17), and covariance matrix defined as (21)
end for

3. Simulation results and discussion

3.1. Motivating example: Van der Pol oscillators

First, consider a motivating example of a population of Van der Pol oscillators as an illustration of the method. Recall (1):

$$\frac{\partial u}{\partial t} = \nabla \cdot \mathbf{K} \nabla u - \nabla \cdot (\mathbf{v} u) \tag{22}$$

where, $\mathbf{v} = \mathbf{v}_d + \mathbf{v}_c$. In the case of the Van der Pol model, the oscillator dynamic velocity field $\mathbf{v}_d = \partial \mathbf{x}/\partial t$ is defined by:

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} \mu(x_1 - \frac{1}{3}x_1^3 - x_2) \\ \frac{1}{\mu}x_1 \end{bmatrix}. \tag{23}$$

The coupling velocity $\mathbf{v}_c = \mathbf{v}_c(u)$ is defined by:

$$\mathbf{v}_c(u) = \alpha \int_{\mathbb{R}^2} y_1 u(\mathbf{y}) d\mathbf{y} \tag{24}$$

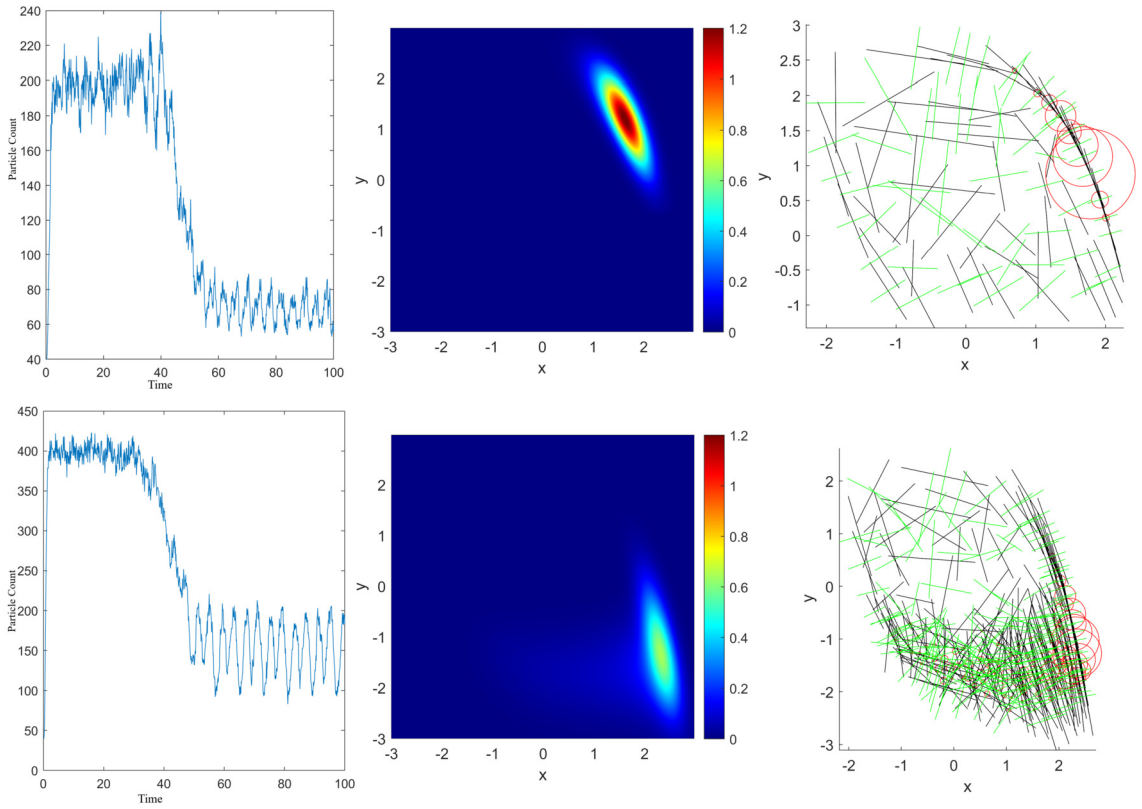

Fig. 5. The population Van der Pol oscillator with $\mu = 1.5, \alpha = 0.5$ and diffusion coefficient $k = 0.05$ (top panels) and $k = 0.1$ (bottom panels). The left panels show the number of particles versus time. In the middle panels, the heat map of the distribution at $t = 100$. In the right panels: the location and geometry of particles at $t = 100$: the red circles indicate the weight of individual particles, while the black and green lines are the eigenvectors of the covariance matrix scaled by their corresponding eigenvalues. Movies are provided in the supplement.

in which α is a coupling coefficient. This is commonly referred to as a *mean-field* coupling. We set up the initial population along the limit cycle of this oscillator without coupling. The results from a sample simulation are shown in Fig. 5, which serves as an illustrative example of how the APPD method represents the population and noise over the process of simulation.

We choose the Van der Pol oscillator to serve as an illustrative example for the usage of our algorithm, as the state space is 2-dimension and can be plotted without projection. In the example used in Fig. 5, we select an uncoupled initial condition along the limit cycle and observe the effects of coupling. It can be seen that both populations are coupled; however, there is a slight phase difference due to the noise level difference. It started at a high value since the population is set to start in an uncoupled state. As the coupling synchronizes the population, the population density is concentrated in a smaller region, and the number of particles drops accordingly. Since the computational cost is proportional to particle count, our method is more efficient if there is stronger coupling and weaker diffusion.

Additionally, we note that synchronization occurs unevenly and that once enough particles start to synchronize, many of the other particles quickly follow suit. This occurs in a similar way to the phase transitions seen in coupled oscillator theory [26]. As the population synchronizes, the number of particles shrinks.

3.2. Hodgkin-Huxley model with threshold coupling

The Hodgkin-Huxley model is a model that describes the electrical activity of a neuron based on ionic currents. Coupled neurons form the basis of computational neuroscience. These models are highly nonlinear and have multiple attractors, which creates challenges for simulation. Additionally, the model exhibits complex behaviors that are highly dependent on the noise level, such as noise-induced synchronization [27,28], and a noise-induced coexistence of firing and resting neurons [29]. These interesting macroscopic phenomena require an accurate simulation of noise to reproduce. We use the Hodgkin-Huxley model in two ways. First, we show that the APPD method accurately reproduces the mentioned phenomena, which can be further separated into two cases. Then, we compare it against direct Monte Carlo simulation to show that the method is accurate and fast.

The Hodgkin-Huxley model used here is described as follows: We take $\mathbf{x} = (0.01V, m, n, h)$, where V is the membrane potential in millivolts, m, h are proportion (for each cell) of activating and inactivating subunits of the sodium channel, and

n corresponds to that of potassium channel subunits. Here V is scaled by 0.01 such that all the values have the same order of magnitude since the particle combination method described in Sec. 2.5 uses cubic-shaped buckets.

The model equations are as follows:

$$C\frac{\partial}{\partial t}V = -G_{\mathrm{Na}}m^3h(V-V_{\mathrm{Na}}) - G_{\mathrm{K}}n^4(V-V_{\mathrm{K}}) - G_L(V-V_L) - G_{\mathrm{coupling}}(V-V_{\mathrm{coupling}}) - I_{\mathrm{app}}, \tag{25}$$

$$\frac{\partial m}{\partial t} = \alpha_m(V)(1-m) - \beta_m(V)m, \tag{26}$$

$$\frac{\partial h}{\partial t} = \alpha_h(V)(1-h) - \beta_h(V)h, \tag{27}$$

$$\frac{\partial n}{\partial t} = \alpha_n(V)(1-n) - \beta_n(V)n \tag{28}$$

with the following equations for the subunit dynamics:

$$\alpha_m(V) = 0.1\frac{V-25}{1-\exp(-\frac{V-25}{10})}, \tag{29}$$

$$\beta_m(V) = 4\exp(-\frac{V}{18}), \tag{30}$$

$$\alpha_h(V) = 0.07\exp(-\frac{V}{20}), \tag{31}$$

$$\beta_h(V) = \frac{1}{1+\exp(-\frac{V-30}{10})}, \tag{32}$$

$$\alpha_n(V) = 0.01\frac{V-10}{1-\exp(-\frac{V-10}{10})}, \tag{33}$$

$$\beta_n(V) = 0.125\exp(-\frac{V}{80}) \tag{34}$$

in which G_{coupling} is conductance due to coupling, as explained below.

In a neuron population, it is usually assumed that individual neurons are independent, except when a neuron firing occurs and post-synaptic neurons are coupled with the firing neuron. In a population density model, the neurons are indistinguishable except for their state; therefore, an all-to-all coupling (or probabilistic coupling) between neurons is assumed. However, contrary to a neural mass model where coupling strength is determined from the average membrane potential [7,30], the more realistic threshold coupling can be implemented. Specifically, G_{coupling} is proportional to the flow rate across the hyperplane $V = V_{\mathrm{threshold}}$ from $V < V_{\mathrm{threshold}}$ to $V > V_{\mathrm{threshold}}$ side. Since the threshold value is chosen where the voltage changes quickly, the contribution of diffusion to coupling can be ignored, and the flow rate is given by the velocity of particles multiplied by the marginal density along $V_{\mathrm{threshold}}$. In the actual implementation, we used the averaged flow rate over the previous timestep as the flow rate for the current step, which, considering that actual coupling is not instantaneous, is still a reasonable assumption. The threshold firing potential is chosen at $V_{\mathrm{threshold}} = 45$ mV.

Since the subunit variables are proportions of some quantity, their domain is restricted in $[0,1]$. The dynamics have inward-pointing velocity on the boundary. Therefore the center of particles will not leave the domain without splitting. However, as the particles have volume, the support of centers can be outside this domain due to the effects of Gaussian noise. [31] and [32] discuss more detailed handling of the noise near boundary conditions. Nevertheless, the following alternative method can still produce sufficiently accurate results.

The noise can result in off-domain points in two ways: When a split occurs and when checking an off-center point of a particle in the LSKF method. If a split result in the center of a particle outside the domain, the center is shifted to the closest location inside the domain. If an off-center point is outside the domain while the center is inside, we choose the offset of the point from the particle's center to be its negative, which will be inside the domain and on the level set.

The value of the constants are listed as follows: $C = 1\ \mu\mathrm{F\ cm^{-2}}$, $G_{\mathrm{Na}} = 120\ \mu\mathrm{A\ mV^{-1}\ cm^2}$, $E_{\mathrm{Na}} = 115$ mV, $G_{\mathrm{K}} = 36\ \mu\mathrm{A\ mV^{-1}\ cm^2}$, $E_{\mathrm{K}} = -12$ mV, $G_L = 0.3\ \mu\mathrm{A\ mV^{-1}\ cm^2}$, $E_L = 10.613$ mV, $I_{\mathrm{app}} = 10\ \mu$A. The coupling potential $V_{\mathrm{coupling}} = -35$ mV for the inhibitory case, and $V_{\mathrm{coupling}} = 50$ mV for the excitatory case, keeping in mind that in this original Hodgkin-Huxley model, the neuron rests near 0 mV. We choose the diffusion to be homogeneous after V is scaled, with a diffusion matrix $\mathbf{K} = k\mathbf{I}_d$.

The coupling is defined analogously to neuron firing: That is, when a neuron reaches a threshold membrane potential from below, the neuron sends a signal to all post-synaptic neurons. Consequently, G_{coupling} in (25) is defined by $G_{\mathrm{coupling}} =$

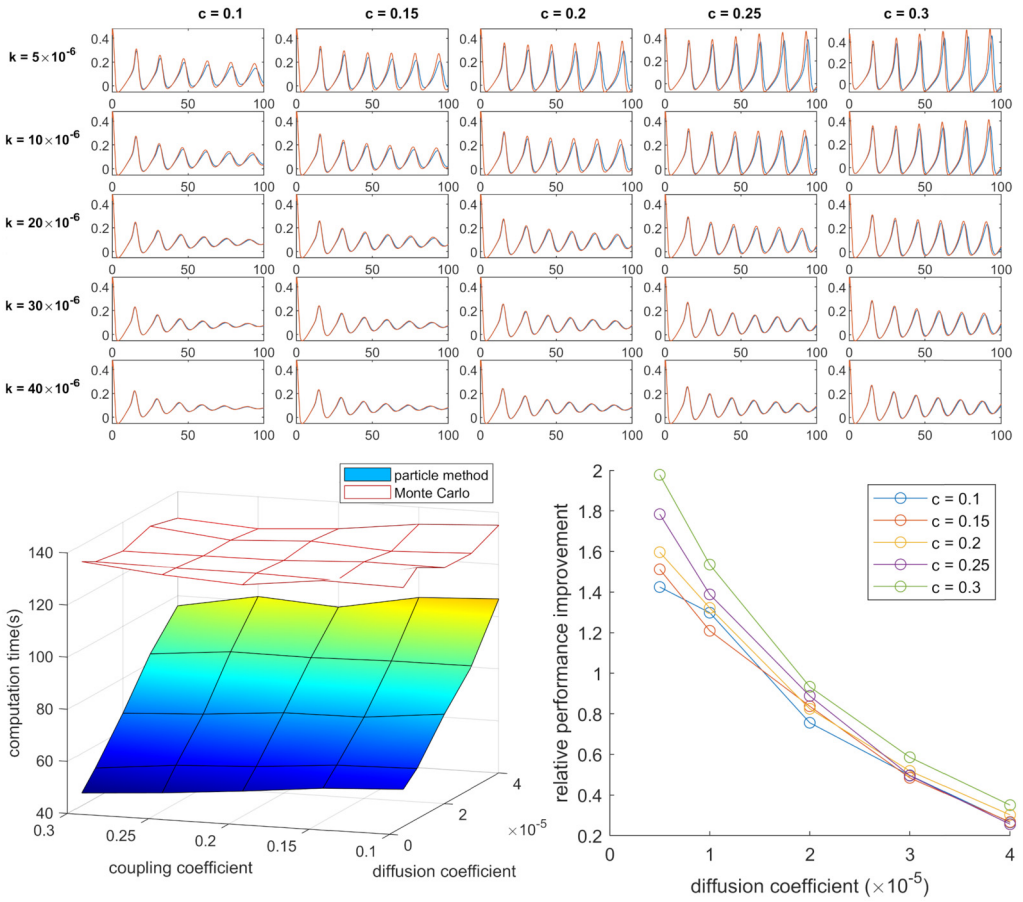

Fig. 6. The Hodgkin-Huxley model: Comparing APPD and Monte Carlo. Top panel: the Average membrane potential V versus time in milliseconds. From left to right, the coupling coefficient c increases from 0.1 to 0.3 with a stepsize of 0.05. From top to bottom, the diffusion coefficient increases, with $k = 0.5, 1, 2, 3, 4 \times 10^{-5}$. The APPD curve is in blue; the Monte Carlo curve is in orange. Lower panels: the computation time with the same parameters: the left panel shows the computation time, whereas the right panel shows the relative performance improvement over the Monte Carlo method. Note for the APPD method, the computation is faster when the population is more synchronized, indicated by drastically changing average membrane potential in the top panel.

$20Qc$, where constant c is the coupling coefficient, Q is the flow rate of neurons as a proportion of total population per millisecond. Therefore, Q has the unit ms^{-1}.

We compare our simulations to direct Monte Carlo simulations. Note we choose the number of elements $N = 41080$ for the Monte Carlo simulation, and the improvement in performance would be more significant if a larger number of elements is used. For the range of parameters chosen and our implementation of the Monte Carlo method, the APPD method is faster than the direct Monte Carlo method, as shown in Fig. 6. It is important to note that while the Monte Carlo method takes about a constant amount of time for all simulation cases, there are large variations for the APPD. At a high noise, low coupling scenario ($k = 4 \times 10^{-5}, c = 0.1$) where the population is asynchronous at the end of the simulation, the APPD method is faster than the Monte Carlo simulation by 20%. Whereas for a low noise strong coupling scenario ($k = 0.5 \times 10^{-5}, c = 0.3$), the population remains synchronized, and the APPD method is 200% faster, as shown in the lower-right panel of Fig. 6. This again shows that the APPD method is most suitable to simulate neurons with strong coupling at a lower noise level and that the APPD method can adapt to create computational efficiencies.

We now explore the choice of tolerance ϵ in (12) and its effect on computation time. It should be noted that the tolerance ϵ also affects the optimal choice of grid sidelength τ. The effect of ϵ, τ and computation time is investigated in Fig. 7.

We asserted that the distribution is compressed in some directions, and asymmetric particles are introduced to fully take advantage of this fact. With the Hodgkin-Huxley example, we investigate the extent of the asymmetry by plotting how fast the singular value of the covariance matrix decays in Fig. 8. Notice that the fourth singular value can be less than 10^{-4} of the largest singular value. The number of dimensions combined with the difference of magnitude makes evaluating Laplacian using numerical differentiation infeasible, and the particle split criteria (12) in Sec. 2.4 is defined in a single direction as a result.

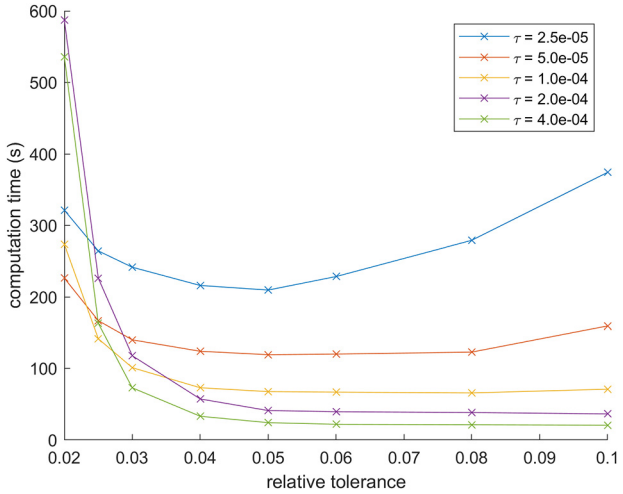

Fig. 7. Relation between computational time, tolerance ϵ, and grid sidelength factor τ.

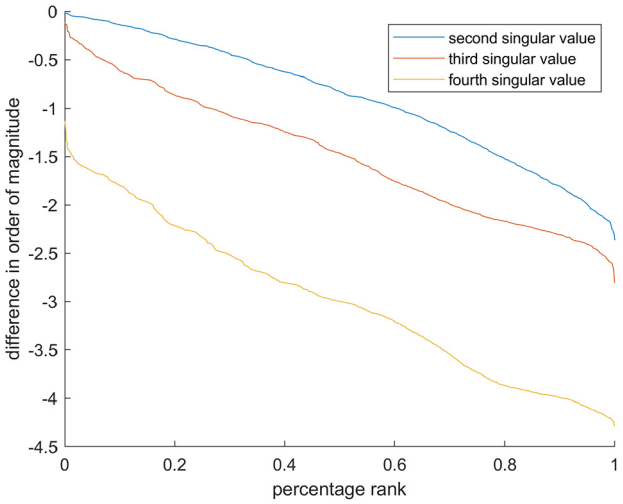

Fig. 8. Decay of singular value: The plot shows the distribution of the ratio of the second, third, and fourth singular value versus the first singular value of the covariance matrices of particles. The covariance matrix is taken from a snapshot of the distribution from the Hodgkin-Huxley simulation with 1732 particles. The ratio is ranked in descending order, with the percentage ratio indicated on the horizontal axis. Since the values are orders of magnitude different, the ratio is plotted in the common log of base 10, as indicated by the vertical axis.

4. Conclusions

The APPD method provides a fast and accurate method to study population-level behaviors while accounting for noise and coupling in biologically realistic models. We compared our method against direct Monte Carlo simulation and tested its ability to reproduce complex macroscopic phenomena across various parameters.

While the APPD method is conceptually inspired by the particle method by Stinchcombe and Forger [10], the method presented here has two significant improvements: 1) particles are asymmetric, allowing it to better track the population; 2) computation of the noise term is computed by modifying a single particle instead of interactions between nearby particles, which reduces computation, and allows almost perfect parallel implementation. When compared against existing asymmetric particle methods [17,18,20,22], our method is the first to be applied to a 4-dimensional problem instead of the typically considered 2-dimensional problems. Such an extension is nontrivial as it depends on the improved numerical stability of our method by tracking a *square-root* of the covariance instead of the covariance.

Looking at the examples, we checked whether we achieved the goal of developing a middle ground between the direct simulation and existing population density-based methods. From the form of (10), it is evident that the dynamics of the oscillator are retained, and no manual dimension reduction is required. The choice of the kernel (5) enabled an accurate representation of the effect of noise. Through the examples, we implemented both all-to-all-coupled oscillators and threshold-coupled oscillators; both resemble the direct simulation.

The advantages of the APPD method are the versatility through different models and a range of parameters without any model-specific simplifications and the robustness that the macroscopic behavior matches the direct simulation (Fig. 6). These observations lead us to conclude that the APPD method can be applied to analyze the complex macroscopic behaviors of a population of coupled noisy oscillators, which is in contrast to many population density approaches that require a model-specific understanding of its behavior to be implemented.

APPD can be extended by future work in the following ways: First, the derivation in Sec. 2.3 assumed the diffusion matrix and velocity field are constant in time or change slowly, but it could be extended to a more general situation. Second, the criteria for combining particles can be improved; for example, particles with small weights can be combined more aggressively.

CRediT authorship contribution statement

Ningyuan Wang: Methodology, Software, Validation, Writing Original Draft.
Daniel B. Forger: Securing funding, Conceptualization, Methodology, Writing Review and Editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgement

We would like to thank Adam Stinchcombe for his input through the development of the method and shaping of the manuscript. We appreciate the depth the reviewers went into the manuscript, with many insights that make the manuscript more comprehensive and readable. This project is partially supported by the NSF DMS2052499 grant, HFSP program grant RGP 0019-2018, and DoD MURI grant W911NF-22-1-0223.

Appendix A. Supplementary material

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.jcp.2023.112157.

References

[1] J. Buck, Synchronous rhythmic flashing of fireflies. II, Q. Rev. Biol. 63 (3) (1988) 265–289.
[2] A.L. Hodgkin, A.F. Huxley, B. Katz, Measurement of current-voltage relations in the membrane of the giant axon of loligo, J. Physiol. 116 (4) (1952) 424.
[3] T.-L. To, M.A. Henson, E.D. Herzog, F.J. Doyle III, A molecular model for intercellular synchronization in the mammalian circadian clock, Biophys. J. 92 (11) (2007) 3792–3803.
[4] A.T. Winfree, Biological rhythms and the behavior of populations of coupled oscillators, J. Theor. Biol. 16 (1) (1967) 15–42.
[5] B.H. Jansen, V.G. Rit, Electroencephalogram and visual evoked potential generation in a mathematical model of coupled cortical columns, Biol. Cybern. 73 (4) (1995) 357–366.
[6] T.D. Sanger, Probability density estimation for the interpretation of neural population codes, J. Neurophysiol. 76 (4) (1996) 2790–2793.
[7] D.Q. Nykamp, D. Tranchina, A population density approach that facilitates large-scale modeling of neural networks: analysis and an application to orientation tuning, J. Comput. Neurosci. 8 (1) (2000) 19–50.
[8] Y. Kuramoto, Chemical Oscillations, Waves, and Turbulence, Courier Corporation, 2003.
[9] E. Haskell, D.Q. Nykamp, D. Tranchina, Population density methods for large-scale modelling of neuronal networks with realistic synaptic kinetics: cutting the dimension down to size, Netw. Comput. Neural Syst. 12 (2) (2001) 141.
[10] A.R. Stinchcombe, D.B. Forger, An efficient method for simulation of noisy coupled multi-dimensional oscillators, J. Comput. Phys. 321 (2016) 932–946, https://doi.org/10.1016/j.jcp.2016.05.025, http://www.sciencedirect.com/science/article/pii/S0021999116301620.
[11] S. Mas-Gallic, The diffusion velocity method: a deterministic way of moving the nodes for solving diffusion equations, Transp. Theory Stat. Phys. 31 (4–6) (2002) 595–605.
[12] S. Sarkka, On unscented Kalman filtering for state estimation of continuous-time nonlinear systems, IEEE Trans. Autom. Control 52 (9) (2007) 1631–1641.
[13] I. Arasaratnam, S. Haykin, T.R. Hurd, Cubature Kalman filtering for continuous-discrete systems: theory and simulations, IEEE Trans. Signal Process. 58 (10) (2010) 4977–4993.
[14] F. Gustafsson, G. Hendeby, Some relations between extended and unscented Kalman filters, IEEE Trans. Signal Process. 60 (2) (2011) 545–555.
[15] G.Y. Kulikov, M.V. Kulikova, Accurate continuous–discrete unscented Kalman filtering for estimation of nonlinear continuous-time stochastic models in radar tracking, Signal Process. 139 (2017) 25–35.
[16] N. Wang, D.B. Forger, The level set Kalman filter for state estimation of continuous-discrete systems, IEEE Trans. Signal Process. 70 (2021) 631–642.
[17] L.F. Rossi, Resurrecting core spreading vortex methods: a new scheme that is both deterministic and convergent, SIAM J. Sci. Comput. 17 (2) (1996) 370–397.
[18] L.F. Rossi, Achieving high-order convergence rates with deforming basis functions, SIAM J. Sci. Comput. 26 (3) (2005) 885–906.
[19] A. Beaudoin, S. Huberson, E. Rivoalen, Méthode particulaire anisotrope pour des écoulements de fluide visqueux, C. R., Méc. 332 (7) (2004) 499–504.

[20] Z. Xie, D. Pavlidis, J.R. Percival, J.L. Gomes, C.C. Pain, O.K. Matar, Adaptive unstructured mesh modelling of multiphase flows, Int. J. Multiph. Flow 67 (2014) 104–110.
[21] R. Abgrall, H. Beaugendre, C. Dobrzynski, An immersed boundary method using unstructured anisotropic mesh adaptation combined with level-sets and penalization techniques, J. Comput. Phys. 257 (2014) 83–101.
[22] A.B. Berchet, A. Beaudoin, S.H. Huberson, Adaptive particle method based on moments for simulating the mass transport in natural flows, Comput. Part. Mech. 8 (3) (2021) 525–534.
[23] S. Pope, A Monte Carlo method for the pdf equations of turbulent reactive flow, Combust. Sci. Technol. 25 (5–6) (1981) 159–174, https://doi.org/10.1080/00102208108547500.
[24] R.E. Kalman, R.S. Bucy, New Results in Linear Filtering and Prediction Theory, 1961.
[25] J.L. Bentley, D.F. Stanat, E. Williams, The complexity of finding fixed-radius near neighbors, Inf. Process. Lett. 6 (6) (1977) 209–212, https://doi.org/10.1016/0020-0190(77)90070-9, http://www.sciencedirect.com/science/article/pii/0020019077900709.
[26] D.B. Forger, Biological Clocks, Rhythms, and Oscillations: The Theory of Biological Timekeeping, MIT Press, 2017.
[27] Y. Wang, D.T. Chik, Z. Wang, Coherence resonance and noise-induced synchronization in globally coupled Hodgkin-Huxley neurons, Phys. Rev. E 61 (1) (2000) 740.
[28] N. Brunel, D. Hansel, How noise affects the synchronization properties of recurrent networks of inhibitory neurons, Neural Comput. 18 (2006) 1066–1110, https://doi.org/10.1162/089976606776241048.
[29] I. Bashkirtseva, A.B. Neiman, L. Ryashko, Stochastic sensitivity analysis of noise-induced suppression of firing and giant variability of spiking in a Hodgkin-Huxley neuron model, Phys. Rev. E 91 (5) (2015) 052920.
[30] O. David, K.J. Friston, A neural mass model for meg/eeg:: coupling and neuronal dynamics, NeuroImage 20 (3) (2003) 1743–1755.
[31] E. Alòs, S. Bonaccorsi, Stochastic partial differential equations with Dirichlet white-noise boundary conditions, Ann. Inst. Henri Poincaré Probab. Stat. 38 (2002) 125–154.
[32] N. Krepysheva, L. Di Pietro, M.-C. Néel, Space-fractional advection-diffusion and reflective boundary condition, Phys. Rev. E 73 (2) (2006) 021104.