Numerical Methods for Simulating Stochastic Reaction-Diffusion Lotka-Volterra Systems

MATHEMATICAL INSTITUTE, UNIVERSITY OF OXFORD

B5.1 Stochastic Modelling of Biological Processes

Candidate number: 1077723

Date: April 22, 2024

Abstract. Many biological reaction-diffusion processes involve a small number of particles. In these cases, continuum-level descriptions are not accurate, and the fluctuating dynamics of the system can only be captured using stochastic simulations. We provide a detailed description and comparison of four numerical methods used to simulate reaction-diffusion phenomena: a deterministic finite difference method and three stochastic algorithms — a particle-based method that uses Brownian dynamics, a compartment-based method that uses the Gillespie algorithm, and a mixed compartment-particle-based method. We analyse their performance in simulating a two-dimensional reaction-diffusion Lotka-Volterra system with random initial conditions and periodic boundary conditions. For all stochastic algorithms, we show that the reaction-controlled regime — i.e. large diffusion coefficients and initial numbers of particles — with medium-sized compartments gives results closest to the deterministic solution. Our analysis demonstrates that, in both diffusion- and reaction-controlled regimes, the mixed method is the least computationally expensive (except for large numbers of particles) because it can apply larger timesteps, and the particle-based method is the best-behaved because it is the most accurate at capturing the system's dynamics.

$\boxed{1}$ Introduction

Reaction-diffusion processes in biological systems are remarkably complex. Often, it is impossible to obtain a full picture of the precise drivers behind their dynamics. Classical models, based on deterministic partial differential equations, provide accurate macroscopic predictions for systems with large numbers of particles. However, when the number of particles is small, they fail to correctly describe the dynamics of systems where the influence of individual particles and the stochastic nature of drivers play a significant role. Then, it is crucial to incorporate stochasticity to account for the discreteness of populations and randomness of events.

A wide spectrum of stochastic methods [1, 2, 3] have been developed for reaction-diffusion systems. On one end, micro-scale approaches are computationally expensive but can precisely keep track of the dynamics of each particle. They are based on Brownian dynamics, and particles react when they are within a certain distance of each other. On the other end, mesoscale approaches are computationally efficient at the expense of representational accuracy by splitting the domain into compartments. They are based on the Gillespie algorithm and are well-suited to describe well-mixed systems. Between these two extremes lies a variety of mixed approaches that combine particle- and compartment-based methods.

In this project, we give a detailed description and comparison of four algorithms that can be used to simulate reaction-diffusion processes: (1) a macroscopic, deterministic, finite differences method (FDM); (2) a stochastic, microscopic, particle-based method (PBM); (3) a stochastic, mesoscopic, compartment-based method (CBM); and (4) a stochastic, mixed compartment-particle-based method developed in [2] (CPBM).

Moreover, we analyse their performance in simulating a two-dimensional reaction-diffusion Lotka-Volterra system. Despite its simplicity, this model manages to capture the essential features of predator-prey interactions and, since it is easily extended, it is the standard starting point for modelling this type of dynamics in the literature.

We hope this project serves as a review of the advantages and disadvantages of some commonly used stochastic numerical methods and an illustration of how these can be applied and what results to expect.

$\boxed{2}$ Model Description

This project uses a two-dimensional reaction-diffusion Lotka-Volterra system with random initial conditions and periodic boundary conditions to test the accuracy and robustness of the numerical methods. In this section, we will introduce such a system by discussing Lotka-Volterra reactions in the absence of diffusion, then incorporating diffusion, and finally addressing the boundary and initial conditions.

2.1. Lotka-Volterra Reactions

2.1.1 Deterministic model

The Lotka-Volterra equations [4, 5] describe predator-prey dynamics operating under the assumptions that: (1) prey species reproduce exponentially in the absence of predators, (2) predators die/emigrate exponentially fast in the absence of prey, and (3) predators reproduce by consuming prey. This behaviour can be written as the reaction scheme of two species of particles A (prey) and B (predator).

$$A \xrightarrow{\sigma_1} 2A, \qquad B \xrightarrow{\sigma_2} \emptyset, \qquad A + B \xrightarrow{\sigma_3} 2B. \tag{1}$$

The deterministic temporal evolution of the concentrations $c_A(t)$ and $c_B(t)$ is

$$\begin{cases} \frac{\partial c_A(t)}{\partial t} = \sigma_1 c_A(t) - \sigma_3 c_A(t) c_B(t), \\ \frac{\partial c_B(t)}{\partial t} = -\sigma_2 c_B(t) + \sigma_3 c_A(t) c_B(t). \end{cases} \tag{2}$$

The dynamics of this system of equations is shown in fig. 1 (a)-(b). There are two fixed points $(c_A, c_B) = (0, 0), (\sigma_2/\sigma_3, \sigma_1/\sigma_3)$ that correspond to the extinction of both species and to co-existence, respectively. In appendix A, a stability analysis is performed to find that such points are a saddle and a centre or elliptic fixed point. Since $(0, 0)$ is unstable, unless the prey population is artificially set to zero, extinction is impossible in this model. Moreover, these equations have an integral of motion

$$G = \sigma_3(c_A(t) + c_B(t)) - (\sigma_1 + \sigma_2) - \sigma_2 \ln\left(\frac{c_A(t)\sigma_3}{\sigma_2}\right) + \sigma_1 \ln\left(\frac{c_B(t)\sigma_3}{\sigma_1}\right). \tag{3}$$

Please refer to appendix A for more details on how to derive it. Hence, all orbits are marginally stable. That is, small perturbations shift the system to an also-stable

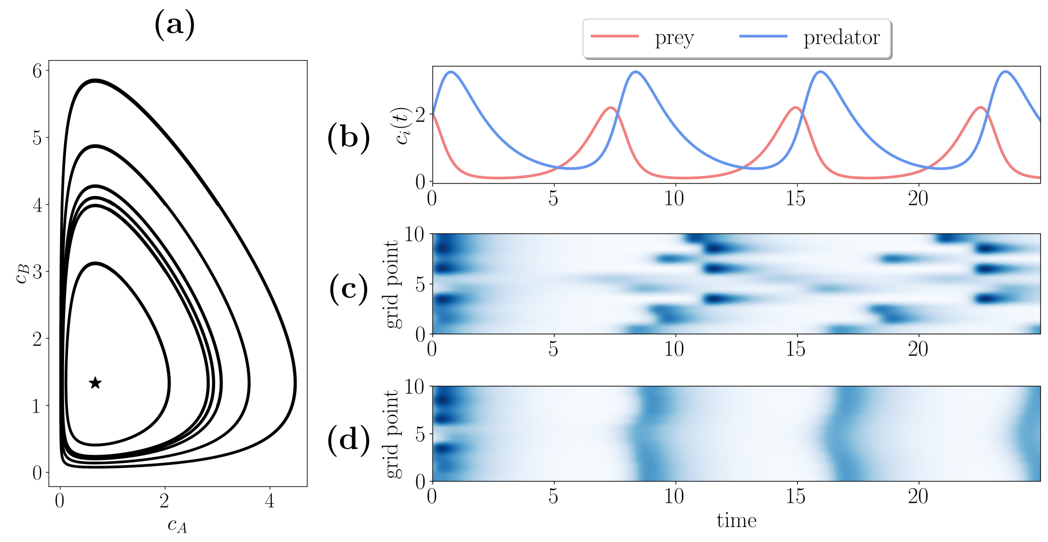

Fig. 1: Deterministic Lotka-Volterra dynamics for $\sigma_1 = 4/3$ s^{-1}, $\sigma_2 = 2/3$ s^{-1}, $\sigma_3 = 1$ m^{2}s^{-1}. **(a)** Phase-plane with $D_A = D_B = 0$ (no diffusion), and several initial conditions. The fixed point at $(\sigma_2/\sigma_3, \sigma_1/\sigma_3)$ is indicated with a star. **(b)** One trajectory of (a) through time. **(c)**-**(d)** Evolution of predator concentrations in a column of a 2D domain $\{(x, y) \in ([0, 10]\text{m}, [0, 10]\text{m})\}$ with $D_A = D_B = 0$ (no diffusion) and $D_A = D_B = 0.1$ m^{2}s^{-1}, respectivelly.

neighbouring orbit, so there is no restoring force nor amplification of the perturbation.

Therefore, overall, for a given initial condition, the populations evolve along a closed orbit (in phase space) around the fixed point $(\sigma_2/\sigma_3, \sigma_1/\sigma_3)$ following a cycle where: (1) predators proliferate when prey is abundant, (2) predators eventually outstrip their food supply and their population declines, and (3) free from predators, prey population increases again.

2.1.2 Stochastic model

Deterministic models fail to take into account the natural stochasticity of birth-death events as well as the discreteness of the populations. If we instead use stochastic simulations, we see a significant behavioural change, giving qualitatively different properties than the deterministic counterpart [4, 5]. Firstly, the system is not restricted to one closed trajectory like in fig. 1 (a) but moves between (stable) neighbouring orbits. Moreover, stochastic Lotka-Volterra interactions not only can, but invariably will result in an extinction event in which the population of either the predator or prey species disappears.

2.2. Lotka-Volterra Reaction-Diffusion

Predator-prey systems in ecology and chemistry usually take place in more than one dimension so that the members of each species are allowed to roam around a domain. In this project, we assume this movement is governed by diffusion so that we get a deterministic spatiotemporal evolution of the concentrations $c_A(t)$ and $c_B(t)$

$$\begin{cases} \frac{\partial c_A(t)}{\partial t} = D_A \nabla^2 c_A + \sigma_1 c_A(t) - \sigma_3 c_A(t) c_B(t), \\ \frac{\partial c_B(t)}{\partial t} = D_B \nabla^2 c_B - \sigma_2 c_B(t) + \sigma_3 c_A(t) c_B(t). \end{cases} \tag{4}$$

However, in reality, other processes may be relevant. For example, prey might be repelled by predators. Figure 1 (c)-(d) shows how, when diffusion is present, instead of independent local processes at each gridpoint, there is a global evolution of the concentration. This new system of equations shows similar periodic behaviour as the system without diffusion, but the stable fixed point is no longer at $(\sigma_2/\sigma_3, \sigma_1/\sigma_3)$. Exploring how extensions of the basic model (eq. (2)) like diffusion (eq. (4)) affect system dynamics is a rich research topic. However, this project focuses on comparing the results of simulating the model using different numerical methods rather than dissecting the effects that certain modelling decisions have on system dynamics.

2.3. Initial and Boundary Conditions

Let us briefly note the boundary and initial conditions we will use throughout this project. Boundary conditions are taken to be periodic (PBCs). This standard practice is employed to model small units within larger systems, aiming to minimize the influence of boundary conditions on computational outcomes as much as possible. Moreover, particles are initially randomly distributed throughout the domain.

3 Numerical Methods

In this section, we give a detailed description of four numerical methods for simulating reaction-diffusion processes. We first outline their general applicability to any reaction system, then demonstrate their implementation for the specific case of a two-dimensional reaction-diffusion Lotka-Volterra system with random initial conditions and PBCs.

3.1. Finite Difference Method (FDM)

Consider K species in a domain

$$\{(x, y, t) \in ([0, L], [0, W], [0, T])\}, \tag{5}$$

where $L, W, T \in \mathbb{R}^+$ describe the size of the 2D spatial domain and temporal domain, respectively. The spatiotemporal evolution of the concentrations $\{c_k(x, y, t) \in \mathbb{R}^+\}_{k=1}^{K}$ of species undergoing reaction-diffusion can be described by a system of equations

$$\frac{\partial c_k}{\partial t} = D_k \nabla^2 c_k + f_k(c_1, ..., c_K), \qquad k = 1, 2, ..., K, \tag{6}$$

where D_k is the diffusion coefficient and f_k is the production/depletion rate of the kth species. These equations can be numerically integrated using a finite difference scheme in a discretised domain

$$\{(x_i, y_j, t_n) := (i\Delta x, j\Delta y, n\Delta t) \in ([0, L], [0, W], [0, T])\}, \tag{7}$$

where $i = 0, 1, ..., I$; $j = 0, 1, ..., J$; $n = 0, 1, ..., N$; $\Delta x = L/I$; $\Delta y = W/J$; and $\Delta t = T/N$. For example, using an explicit Euler scheme, we can evolve the approximation $(C_k)_{i,j}^{n+1}$ of $c_k(x_i, y_j, t_{n+1})$ as

$$\begin{aligned} \frac{(C_k)_{i,j}^{n+1} - (C_k)_{i,j}^{n}}{\Delta t} = D_k \Bigg[& \frac{(C_k)_{i+1,j}^{n} - 2(C_k)_{i,j}^{n} + (C_k)_{i-1,j}^{n}}{(\Delta x)^2} \\ & + \frac{(C_k)_{i,j+1}^{n} - 2(C_k)_{i,j}^{n} + (C_k)_{i,j-1}^{n}}{(\Delta y)^2} \Bigg] + f_k((C_1)_{i,j}^{n}, ..., (C_K)_{i,j}^{n}), \end{aligned} \tag{8}$$

for $i = 1, 2, ..., I - 1$; $j = 1, 2, ..., J - 1$; and $n = 0, 1, ..., N$ from some initial condition $(C_k)_{ij}^{0} = c_k(x_i, y_j, 0)$. To set PBCs we repeat eq. (8) for the boundary points replacing $(i - 1)$ by I for $i = 0$, $(i + 1)$ by 0 for $i = I$, $(j - 1)$ by J for $j = 0$ and $(j + 1)$ by 0 for $j = J$. For Lotka-Volterra reactions (eq. (1)) where the system of equations becomes eq. (4), $f_A(c_A, c_B) = \sigma_1 c_A(t) - \sigma_3 c_A(t) c_B(t)$ and $f_B(c_A, c_B) = -\sigma_2 c_B(t) + \sigma_3 c_A(t) c_B(t)$. Please refer to appendix B.1 for a full Python implementation.

Note, however, that numerical errors in this scheme will result in outward-spiralling solutions [5]. Implicit Euler would equally result in inward-spiralling solutions [5]. Hence, using these simple integration schemes, the populations will either explode or implode into the fixed point, and more sophisticated symplectic schemes for Hamiltonian systems like the Gauss-Legendre methods are required to obtain the expected closed orbits. However, since this project's focus is stochastic numerical methods, explicit Euler was deemed acceptable as a reference if care is taken when selecting the model parameters so that solutions do not become unreasonable.

3.2. Particle-Based Method (PBM)

Continuum representations like eq. (6) are inadequate for systems where the discreteness of populations and randomness of events are significant. In these cases, particle-based methods (PBM) [1, 2, 3] are more appropriate.

Consider K species in the domain defined in eq. (5). The state space consists on the trajectories $\{(x_{k,m}(t), y_{k,m}(t)) \in [0,L] \times [0,W] : m = 1, 2, ..., n_k(t)\}_{k=1}^{K}$ of the $n_k(t)$ particles of each species $k = 1, 2, ..., K$. Diffusion is modelled using Brownian dynamics [1, 2],

$$x_{k,m}(t+\Delta t) = x_{k,m}(t) + \sqrt{2D_k \Delta t}\xi_x, \qquad y_{k,m}(t+\Delta t) = y_{k,m}(t) + \sqrt{2D_k \Delta t}\xi_y, \quad (9)$$

where ξ_x and ξ_y are random variables sampled from the normal distribution with zero mean and unit variance. PBCs are implemented so if a particle leaves the domain — $(x_{k,m}(t+\Delta t), y_{k,m}(t+\Delta t)) \notin [0,L] \times [0,W]$ —, its image enters from the opposite end.

Zeroth and first-order reactions with reaction rates $\tilde{\sigma}_0$ and $\tilde{\sigma}_1$ are easily implemented. From the definition of reaction rate, we know that the probability of a particle reacting within a sufficiently small timestep Δt is $\tilde{\sigma}_0 W L \Delta t \ll 1$ and $\tilde{\sigma}_1 \Delta t \ll 1$. Hence, we generate random numbers υ_0 and υ_1 uniformly distributed in $[0,1]$ and, if $\upsilon_0 < \tilde{\sigma}_0 W L \Delta t$ or $\upsilon_1 < \tilde{\sigma}_1 \Delta t$ the reaction fires (i.e. a particle is created at a random position in the domain or a particle is removed) [1].

Simulating second-order reactions with a reaction rate $\tilde{\sigma}_2$, requires more care. One approach is to consider that two molecules A and B react whenever the distance between them is less than a given radius ρ [1]. A straightforward way to determine ρ is to take that the probability of the particles reacting within a sufficiently large timestep Δt ($\sqrt{2(D_A + D_B)\Delta t} \gg \rho$) is the ratio of areas $(\pi\rho^2)/(WL)$. From the definition of reaction rate, this is also $(\tilde{\sigma}_2 \Delta t)/(WL)$. Hence, $\rho = \sqrt{\tilde{\sigma}_2 \Delta t/\pi}$ and we get the condition $\sqrt{2(D_A + D_B)} \gg \sqrt{\tilde{\sigma}_2/\pi}$. This method is an approximation; it does not account for the fact that not all reactant collisions result in product formation. More advanced methods consider this by assigning a reaction probability when particles are within the reaction radius. For this project, the simpler approach should suffice.

Algorithm 1 shows how this can be implemented for Lotka-Volterra reactions (1). Please refer to appendix B.2 for a full Python implementation.

Algorithm 1 Particle-Based Method (PBM)

```
 1: $L, W, T \in \mathbb{R}^+$                                   ▷ Domain parameters
 2: $\Delta t \in \mathbb{R}^+$                                  ▷ Time discretisation
 3: $D_A, D_B, \sigma_1, \sigma_2, \sigma_3 \in \mathbb{R}^+$    ▷ Reaction-diffusion parameters
 4: $(x_{A,m}(0), y_{A,m}(0)) \in [0, L] \times [0, W] : m = 1, 2, ..., n_A(0)$  ▷ Initial conditions for $A$
 5: $(x_{B,m}(0), y_{B,m}(0)) \in [0, L] \times [0, W] : m = 1, 2, ..., n_B(0)$  ▷ Initial conditions for $B$
 6: $\rho = \sqrt{\sigma_3 \Delta t / \pi}$                      ▷ Set reaction radius
 7: $t \leftarrow 0$
 8: while $t < T$ do                                             ▷ Evolve state
        ▷ Diffusion step
 9:     $\xi_x, \xi_y \leftarrow$ normally distributed random numbers
10:     $\{(x_{A,m}(t+\Delta t), y_{A,m}(t+\Delta t))\}_{m=1}^{n_A(t)} \leftarrow$ eq. (9) + PBCs
11:     $\{(x_{B,m}(t+\Delta t), y_{B,m}(t+\Delta t))\}_{m=1}^{n_B(t)} \leftarrow$ eq. (9) + PBCs
        ▷ Reaction step
12:     for $i \in \{1, 2, ..., n_A(t)\}$ do                      ▷ $A \to 2A$
13:         $\upsilon_1 \leftarrow$ uniformly distributed random number $\in [0, 1]$
14:         if $\upsilon_1 < \sigma_1 \Delta t$ then
15:             Add a new particle of $A$ at the position of particle $i$
16:         end if
17:     end for
18:     for $j \in \{1, 2, ..., n_B(t)\}$ do                      ▷ $B \to \emptyset$
19:         $\upsilon_2 \leftarrow$ uniformly distributed random number $\in [0, 1]$
20:         if $\upsilon_2 < \sigma_2 \Delta t$ then
21:             Remove particle $j$ of type B
22:         end if
23:     end for
24:     for $i \in \{1, 2, ..., n_A(t)\}$ do                      ▷ $A + B \to 2B$
25:         for $j \in \{1, 2, ..., n_B(t)\}$ do
26:             $d_{i,j} \leftarrow \sqrt{(x_{A,i} - x_{B,j})^2 + (y_{A,i} - y_{B,j})^2}$
27:             if $d_{i,j} < \rho$ then
28:                 Remove particle $i$ of type $A$ and add a new particle of type $B$
29:             end if
30:         end for
31:     end for
32:     $t \leftarrow t + \Delta t$
33: end while
```

3.3. Compartment-Based Method (CBM)

PBM can become computationally expensive as the number of particles increases. In systems where the number of particles is considerable but stochasticity remains significant (i.e. continuum representations are still inadequate), compartment-based methods (CBM) [1, 2, 3] provide a computationally efficient alternative by sacrificing some representational accuracy.

Consider K species in the domain defined in eq. (5), which we divide into $C = C_x \times C_y$ compartments. Here, C_x and C_y are the number of compartments into which the horizontal and vertical axes are split, respectively. These compartments have a size of $l \times w = L/C_x \times W/C_y$. Now, the state space is the number of particles $\{(n_k(t))_{ij} \in \mathbb{Z} : i = 0, 1, ..., C_x \text{ and } j = 0, 1, ..., C_y\}_{k=1}^{K}$ of each species in each compartment (i, j). We then approximate Brownian motion as another set of "reactions" in which one molecule can jump into neighbouring compartments

$$\begin{tikzcd}
{\color{gray}(n_k)_{C_yC_x}} \arrow[r, shift left, harpoon, gray] \arrow[from=r, shift left, harpoon, gray] \arrow[d, shift left, harpoon, gray] \arrow[from=d, shift left, harpoon, gray] & {\color{gray}(n_k)_{C_y1}} \arrow[r, shift left, harpoon, gray] \arrow[from=r, shift left, harpoon, gray] \arrow[d, shift left, harpoon, gray] \arrow[from=d, shift left, harpoon, gray] & {\color{gray}(n_k)_{C_y2}} \arrow[r, shift left, harpoon, gray] \arrow[from=r, shift left, harpoon, gray] \arrow[d, shift left, harpoon, gray] \arrow[from=d, shift left, harpoon, gray] & {\color{gray}\ldots} \arrow[r, shift left, harpoon, gray] \arrow[from=r, shift left, harpoon, gray] & {\color{gray}(n_k)_{C_yC_x}} \arrow[r, shift left, harpoon, gray] \arrow[from=r, shift left, harpoon, gray] \arrow[d, shift left, harpoon, gray] \arrow[from=d, shift left, harpoon, gray] & {\color{gray}(n_k)_{C_y1}} \arrow[d, shift left, harpoon, gray] \arrow[from=d, shift left, harpoon, gray] \\
{\color{gray}(n_k)_{1C_x}} \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] \arrow[d, shift left, harpoon, gray] \arrow[from=d, shift left, harpoon, gray] & (n_k)_{11} \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] \arrow[d, shift left, harpoon, blue] \arrow[from=d, shift left, harpoon, blue] & (n_k)_{12} \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] \arrow[d, shift left, harpoon, blue] \arrow[from=d, shift left, harpoon, blue] & \ldots \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] & (n_k)_{1C_x} \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] \arrow[d, shift left, harpoon, blue] \arrow[from=d, shift left, harpoon, blue] & {\color{gray}(n_k)_{11}} \arrow[d, shift left, harpoon, gray] \arrow[from=d, shift left, harpoon, gray] \\
{\color{gray}(n_k)_{2N}} \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] \arrow[d, shift left, harpoon, gray] \arrow[from=d, shift left, harpoon, gray] & (n_k)_{21} \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] \arrow[d, shift left, harpoon, blue] \arrow[from=d, shift left, harpoon, blue] & (n_k)_{22} \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] \arrow[d, shift left, harpoon, blue] \arrow[from=d, shift left, harpoon, blue] & \ldots \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] & (n_k)_{2C_x} \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] \arrow[d, shift left, harpoon, blue] \arrow[from=d, shift left, harpoon, blue] & {\color{gray}(n_k)_{21}} \arrow[d, shift left, harpoon, gray] \arrow[from=d, shift left, harpoon, gray] \\
{\color{gray}\ldots} \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] \arrow[d, shift left, harpoon, gray] \arrow[from=d, shift left, harpoon, gray] & \ldots \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] \arrow[d, shift left, harpoon, blue] \arrow[from=d, shift left, harpoon, blue] & \ldots \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] \arrow[d, shift left, harpoon, blue] \arrow[from=d, shift left, harpoon, blue] & \ldots \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] & \ldots \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] \arrow[d, shift left, harpoon, blue] \arrow[from=d, shift left, harpoon, blue] & {\color{gray}\ldots} \arrow[d, shift left, harpoon, gray] \arrow[from=d, shift left, harpoon, gray] \\
{\color{gray}(n_k)_{C_yC_x}} \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] \arrow[d, shift left, harpoon, gray] \arrow[from=d, shift left, harpoon, gray] & (n_k)_{C_y1} \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] \arrow[d, shift left, harpoon, blue] \arrow[from=d, shift left, harpoon, blue] & (n_k)_{C_y2} \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] \arrow[d, shift left, harpoon, blue] \arrow[from=d, shift left, harpoon, blue] & \ldots \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] & (n_k)_{C_yC_x} \arrow[r, shift left, harpoon, red] \arrow[from=r, shift left, harpoon, red] \arrow[d, shift left, harpoon, blue] \arrow[from=d, shift left, harpoon, blue] & {\color{gray}(n_k)_{C_y1}} \arrow[d, shift left, harpoon, gray] \arrow[from=d, shift left, harpoon, gray] \\
{\color{gray}(n_k)_{1C_x}} \arrow[r, shift left, harpoon, gray] \arrow[from=r, shift left, harpoon, gray] & {\color{gray}(n_k)_{11}} \arrow[r, shift left, harpoon, gray] \arrow[from=r, shift left, harpoon, gray] & {\color{gray}(n_k)_{12}} \arrow[r, shift left, harpoon, gray] \arrow[from=r, shift left, harpoon, gray] & {\color{gray}\ldots} \arrow[r, shift left, harpoon, gray] \arrow[from=r, shift left, harpoon, gray] & {\color{gray}(n_k)_{1C_x}} \arrow[r, shift left, harpoon, gray] \arrow[from=r, shift left, harpoon, gray] & {\color{gray}(n_k)_{11}}
\end{tikzcd} \tag{10}$$

where the grey elements show the PBC, the red arrows represent horizontal diffusion with rate constant $d_{x,k} = D_k/l^2$, and the blue arrows represent vertical diffusion with rate constant $d_{y,k} = D_k/w^2$. Then, our model consists of a system of reactions — namely diffusion "reactions" eq. (10) between compartments and chemical reactions within each compartment — that can be simulated using the Gillespie algorithm [1, 2, 6, 7], as we will now explain.

In "well-mixed" systems, we can ignore spatial dynamics and the reaction intensities only depend on the current state; i.e. the system is a Markov process. If the system is in state $\boldsymbol{X}(t) = \{(n_k(t))_{ij}\}_{k=1}^{K}$ at time t, take $\mathbb{P}[\tau, r|\boldsymbol{X}(t)]\Delta t$ to be the probability that the next reaction is the rth reaction at time $[t+\tau, t+\tau+\Delta t)$ and $\mathbb{P}_0[\tau|\boldsymbol{X}(t)]$ the probability that no interactions take place during the interval $[t, t+\tau)$. The Markov (memoryless) property of the process implies that probabilities scale linearly with time for small enough intervals, so we define the propensity function $\alpha_r(\boldsymbol{X}(t))$ such that the probability of the rth reaction occurring in an interval Δt is $\alpha_r(\boldsymbol{X}(t))\Delta t$. Then,

$$\mathbb{P}[\tau, r|\boldsymbol{X}(t)]\Delta t = \mathbb{P}_0[\tau|\boldsymbol{X}(t)]\alpha_r(\boldsymbol{X}(t+\tau))\Delta t = \mathbb{P}_0[\tau|\boldsymbol{X}(t)]\alpha_r(\boldsymbol{X}(t))\Delta t, \tag{11}$$

where we used that $\boldsymbol{X}(t+\tau) = \boldsymbol{X}(t)$ as the reaction has not occured yet. Defining $\alpha_{\text{sum}}(\boldsymbol{X}(t)) = \sum_{r=1}^{R} \alpha_r(\boldsymbol{X}(t))$ with R being the total number of possible reactions,

$$\mathbb{P}_0(\tau+\Delta t|\boldsymbol{X}(t)) = \mathbb{P}_0(\tau|\boldsymbol{X}(t))[1-\alpha_{\text{sum}}(\boldsymbol{X}(t+\tau))\Delta t] = \mathbb{P}_0(\tau|\boldsymbol{X}(t))[1-\alpha_{\text{sum}}(\boldsymbol{X}(t))\Delta t]. \tag{12}$$

Taking the limit $\Delta t \to 0$ and integrating the ODE gives,

$$\mathbb{P}_0(\tau|\boldsymbol{X}(t)) = e^{-\alpha_{\text{sum}}(\boldsymbol{X}(t))\tau}, \tag{13}$$

and we can re-write eq. (11) as,

$$\mathbb{P}[\tau, r|\boldsymbol{X}(t)] = \frac{\alpha_r(\boldsymbol{X}(t))}{\alpha_{\text{sum}}(\boldsymbol{X}(t))}\alpha_{\text{sum}}(\boldsymbol{X}(t))e^{-\alpha_{\text{sum}}(\boldsymbol{X}(t))\tau}. \tag{14}$$

The fraction on the right-hand side of eq. (14) corresponds to the probability density function of a discrete random variable. To determine which reaction happens next we generate random integers r distributed according to $\alpha_r(\boldsymbol{X}(t))/\alpha_{\text{sum}}(\boldsymbol{X}(t)) \in [0, 1]$. I.e. we take a random number υ_1 distributed uniformly between $[0, 1]$ and take r to be determined by

$$\sum_{\tilde{r}=1}^{r-1} a_{\tilde{r}} \leq \upsilon_1 \alpha_{\text{sum}} < \sum_{\tilde{r}=1}^{r} a_{\tilde{r}}. \tag{15}$$

The rest of the right-hand side of eq. (14) corresponds to the exponential density function of a continuous random variable. To determine when the next reaction occurs, we generate random numbers τ distributed according to $\alpha_{\text{sum}}(\boldsymbol{X}(t))e^{-\alpha_{\text{sum}}\tau}\Delta t$. I.e. we take a random number υ_2 distributed uniformly between $[0,1]$ and set it equal to the auxiliary function $F(\tau) = \int_\tau^\infty \alpha_{\text{sum}}(\boldsymbol{X}(t))e^{-\alpha_{\text{sum}}\tau}\mathrm{d}t = e^{-\alpha_{\text{sum}}\tau}$ that is also uniformly distributed between $[0,1]$, then,

$$\tau = \frac{1}{\alpha_{\text{sum}}} \ln\left(\frac{1}{\upsilon_2}\right) \tag{16}$$

Hence, provided the system is at state $\boldsymbol{X}(t)$ at t, the next state at $t+\tau$ is obtained by updating the number of species according to reaction r.

For the two-dimensional reaction-diffusion Lotka-Volterra system, the system of reactions consists of eq. (10) with propensity functions,

$$(\alpha_{x,k}(t))_{i,j} = d_{x,k}(n_k(t))_{i,j}, \quad (\alpha_{y,k}(t))_{i,j} = d_{y,k}(n_k(t))_{i,j}. \tag{17}$$

Together with eq. (1) on each compartment (i,j)

$$A_{ij} \xrightarrow{\sigma_1} 2A_{ij}, \qquad B_{ij} \xrightarrow{\sigma_2} \emptyset, \qquad A_{ij} + B_{ij} \xrightarrow{\sigma_3} 2B_{ij}, \tag{18}$$

where A_{ij} and B_{ij} refer to A and B particles in compartment (i,j). Their propensity functions are respectively

$$(\alpha_1(t))_{i,j} = \sigma_1(n_A(t))_{i,j}, \quad (\alpha_2(t))_{i,j} = \sigma_2(n_B(t))_{i,j}, \quad (\alpha_3(t))_{i,j} = \frac{\sigma_3}{lw}(n_A(t))_{i,j}(n_B(t))_{i,j}. \tag{19}$$

Algorithm 2 shows how this can be implemented. Please refer to appendix B.3 for a full Python implementation.

Note that CBM is only valid for a range of compartment sizes. They must be small enough ($l \ll L$, $w \ll W$) to correctly capture spatial heterogeneity and diffusion, but big enough ($l, w \gg \sigma_3/(D_A + D_B)$) not to restrict second-order reactions [1].

3.4. Mixed Method (CPBM)

Sometimes PBM is prohibitively computationally expensive, but CBM cannot accurately represent the reactions in the system. For example, for small diffusion coefficients or high second-order reaction rates compartments must be considerably big to correctly represent second-order reactions. However, this impoverishes the accuracy with which diffusion is represented, as particles cannot move diagonally at each timestep. In

Algorithm 2 Compartment-Based Method (CBM)

```
 1: $L, W, T \in \mathbb{R}^+$                                                    ▷ Domain parameters
 2: $C_x, C_y \in \mathbb{Z}^+$                                                   ▷ Domain division into compartments
 3: $D_A, D_B, \sigma_1, \sigma_2, \sigma_3 \in \mathbb{R}^+$                      ▷ Reaction-diffusion parameters
 4: $(n_A(0))_{ij} \in Z : i = 0, 1, ..., C_x$ and $j = 0, 1, ..., C_y$          ▷ Initial conditions for $A$
 5: $(n_B(0))_{ij} \in Z : i = 0, 1, ..., C_x$ and $j = 0, 1, ..., C_y$          ▷ Initial conditions for $B$
 6: $t \leftarrow 0$
 7: while $t < T$ do                                                             ▷ Evolve state
 8:     $\upsilon_1, \upsilon_2 \leftarrow$ uniformly distributed random numbers $\in [0, 1]$
 9:     $(\alpha_{x,k}(t))_{i,j}, (\alpha_{y,k}(t))_{i,j}, (\alpha_1(t))_{i,j}, (\alpha_2(t))_{i,j}, (\alpha_3(t))_{i,j} \leftarrow$ eqs. (17) and (19)
10:     $\alpha_{\text{sum}} = \sum_{i,j,k}(\alpha_{x,k}(t))_{i,j} + (\alpha_{y,k}(t))_{i,j} + (\alpha_1(t))_{i,j} + (\alpha_2(t))_{i,j} + (\alpha_3(t))_{i,j}$
11:     $r, \tau \leftarrow$ eqs. (15) and (16)                    ▷ Calculate next reaction and when it occurs
12:     if $r =$ right/left $(\pm)$ diffusion of $A/B$ in $(i, j)$ compartment then
13:         $(n_{A/B}(t + \tau))_{i,j} \leftarrow (n_{A/B}(t))_{ij} - 1$
14:         $(n_{A/B}(t + \tau))_{i\pm1,j} \leftarrow (n_{A/B}(t))_{i\pm1,j} + 1$  ▷ Or $(0, j)/(I, j)$ if $i = I/0$ (PBC)
15:     else if $r =$ up/down $(\pm)$ diffusion of $A/B$ in $(i, j)$ compartment then
16:         $(n_{A/B}(t + \tau))_{i,j} \leftarrow (n_{A/B}(t))_{ij} - 1$
17:         $(n_{A/B}(t + \tau))_{i,j\pm1} \leftarrow (n_{A/B}(t))_{i,j\pm1} + 1$  ▷ Or $(i, 0)/(i, J)$ if $j = J/0$ (PBC)
18:     else if $r = A \to 2A$ in $(i, j)$ compartment then
19:         $(n_A(t + \tau))_{i,j} \leftarrow (n_A(t))_{ij} + 1$
20:     else if $r = B \to \emptyset$ in $(i, j)$ compartment then
21:         $(n_B(t + \tau))_{i,j} \leftarrow (n_B(t))_{ij} - 1$
22:     else $r = A + B \to 2B$ in $(i, j)$ compartment then
23:         $(n_A(t + \tau))_{i,j} \leftarrow (n_A(t))_{ij} - 1$
24:         $(n_B(t + \tau))_{i,j} \leftarrow (n_B(t))_{ij} + 1$
25:     end if
26:     $t \leftarrow t + \tau$
```

such cases, mixed compartment-particle-based methods (CPBM) [2, 3] are the best option. For example, one might naturally consider using a particle-based method to model diffusion and a compartment-based method to model reactions. This idea was implemented by Choi et al. [2] using the operator-splitting algorithm.

Operator-splitting algorithms approximate eq. (4) with

$$\frac{\partial \tilde{c}_k}{\partial t} = D_k \nabla^2 \tilde{c}_k, \qquad \frac{\partial \tilde{\tilde{c}}_k}{\partial t} = f_k(\tilde{\tilde{c}}_1, \cdots, \tilde{\tilde{c}}_K), \tag{20}$$

so that, from $c_k(t, x, y)$, during the interval $[t, t + \Delta t)$, first particles diffuse from $\tilde{c}_k(t, x, y) = c_k(t, x, y)$ to $\tilde{c}_k(t+\Delta t, x, y)$ and then react from $\tilde{\tilde{c}}_k(t, x, y) = \tilde{c}_k(t+\Delta t, x, y)$ to $\tilde{\tilde{c}}_k(t + \Delta t, x, y)$, which is then used to set $c_k(t + \Delta t, x, y) = \tilde{\tilde{c}}_k(t + \Delta t, x, y)$.

Hence, on each iteration, firstly the diffusion of particles is modelled via Brownian dynamics as described in eq. (9). Secondly, the domain is split into "well-mixed" compartments and reactions are modelled using the Gillespie algorithm for a time of Δt as described in eqs. (14) to (16). Note that in this case, the system of reactions for Gillespie consists only of the chemical reactions in each compartment eq. (18) and not the diffusion "reactions" which are already accounted for through Brownian motion. Moreover, unlike CBM, each timestep may involve several reactions.

The choice of Δt is not trivial, and in [2], they select it depending on whether the system is in a reaction or diffusion-dominated state. A natural timestep for diffusion

$$\tau_D = \min\left\{\frac{l^2}{4D_A}, \frac{l^2}{4D_B}, \frac{w^2}{4D_A}, \frac{w^2}{4D_B}\right\}, \tag{21}$$

is the minimum time during which a particle of any species remains in one compartment on average. Remembering eq. (16), a natural timestep for reaction is

$$\tau_R = T_{R,\min} \ln\left(\frac{1}{\upsilon}\right), \qquad T_{R,\min} = \min\left\{\frac{1}{(\tilde{\alpha}_{\text{sum}})_{i,j}}\right\}_{i,j=1,1}^{C_x,C_y}, \tag{22}$$

where $(\tilde{\alpha}_{\text{sum}})_{i,j}$ is the sum of chemical reaction propensities in each compartment (this is different to α_{sum} which is the sum of chemical and diffusion reaction propensities over all compartments), and $\upsilon \in [0,1]$ is a uniformly distributed random variable. This is the minimum amount of time that you need to wait until a reaction happens [2]. Then,

$$F = \frac{T_{R,\min}}{\tau_D} \tag{23}$$

can be used to classify the system as reaction or diffusion-controlled. This is done by introducing four parameters $\kappa_1, \kappa_1', \kappa_2, \kappa_2'$, so that if $F < \kappa_1$ the system is diffusion-controlled and $\Delta t = \kappa_2 \tau_D$, if $\kappa_1 < F < \kappa_1'$ the system is in an intermediate state and $\Delta t = \kappa_2' \tau_D$, and if $F > \kappa_1'$ the system is reaction-controlled and $\Delta t = 10\tau_D$. Note that $\kappa_1 < \kappa_1'$, $\kappa_2 < \kappa_2' < 10$, and the factor of 10 was chosen by [2]. We set $\kappa_1 = 0.5$ (i.e. diffusion-controlled if $\tau_D > 2T_{R,\min}$), $\kappa_1' = 3$ (i.e. reaction-controlled if $\tau_D < \frac{1}{3}T_{R,\min}$), $\kappa_2 = 2$ (i.e. the probability of a reaction taking place during Δt is 0.86 [2]), and $\kappa_2' = 3$ (by recommendation of [2]).

Algorithm 3 shows how this can be implemented. Please refer to appendix B.4 for a full Python implementation.

Algorithm 3 Mixed Method (CPBM)

```
1: $L, W, T \in \mathbb{R}^+$    ▷ Domain parameters
2: $C_x, C_y \in \mathbb{Z}^+$    ▷ Domain division into compartments
3: $D_A, D_B, \sigma_1, \sigma_2, \sigma_3 \in \mathbb{R}^+$    ▷ Reaction-diffusion parameters
4: $(n_A(0))_{ij} \in Z : i = 0, 1, ..., C_x$ and $j = 0, 1, ..., C_y$    ▷ Initial conditions for $A$
5: $(n_B(0))_{ij} \in Z : i = 0, 1, ..., C_x$ and $j = 0, 1, ..., C_y$    ▷ Initial conditions for $B$
6: $\tau_D \leftarrow$ eq. (21)
7: $t \leftarrow 0$
8: while $t < T$ do    ▷ Evolve state
9:     $T_{R,\min} \leftarrow$ eq. (22)
10:    $F \leftarrow$ eq. (23)
11:    if $F < \kappa_1$ then    ▷ Diffusion-controlled
12:        $\Delta t = \kappa_2 \tau_D$
13:    else if $\kappa_1 < F < \kappa_1'$ then    ▷ Intermediate state
14:        $\Delta t = \kappa_2' \tau_D$
15:    else    ▷ Reaction-controlled
16:        $\Delta t = 10\tau_D$
17:    end if
       ▷ Diffusion step (Particle-based)
18:    $(n_A(t))_{ij} \to \{(x_{A,m}(t), y_{A,m}(t))\}_{m=1}^{n_A(t)}$    ▷ Compartments to particles
19:    $(n_B(t))_{ij} \to \{(x_{A,m}(t), y_{A,m}(t))\}_{m=1}^{n_A(t)}$
20:    $\xi_x, \xi_y \leftarrow$ normally distributed random numbers
21:    $\{(x_{A,m}(t + \Delta t), y_{A,m}(t + \Delta t))\}_{m=1}^{n_A(t)} \leftarrow$ eq. (9) + PBC
22:    $\{(x_{B,m}(t + \Delta t), y_{B,m}(t + \Delta t))\}_{m=1}^{n_B(t)} \leftarrow$ eq. (9) + PBC
       ▷ Reaction step (Compartment-based)
23:    $\{(x_{A,m}(t + \Delta t), y_{A,m}(t + \Delta t))\}_{m=1}^{n_A(t)} \to (n_A(t + \Delta t))_{ij}$ ▷ Back to compartments
24:    $\{(x_{A,m}(t + \Delta t), y_{A,m}(t + \Delta t))\}_{m=1}^{n_A(t)} \to (n_B(t + \Delta t))_{ij}$
25:    while $t < t + \Delta t$ do
26:        $\upsilon_1, \upsilon_2 \leftarrow$ uniformly distributed random numbers $\in [0, 1]$
27:        $(\alpha_1(t))_{i,j}, (\alpha_2(t))_{i,j}, (\alpha_3(t))_{i,j} \leftarrow$ eq. (19)
28:        $\tilde{\alpha}_{\text{sum}} = \sum_{i,j} (\alpha_1(t))_{i,j} + (\alpha_2(t))_{i,j} + (\alpha_3(t))_{i,j}$
29:        $r, \tau \leftarrow$ eqs. (15) and (16)    ▷ Calculate next reaction and when it occurs
30:        if $r = A \to 2A$ in $(i, j)$ compartment then
31:            $(n_A(t + \tau))_{i,j} \leftarrow (n_A(t))_{ij} + 1$
32:        else if $r = B \to \emptyset$ in $(i, j)$ compartment then
33:            $(n_B(t + \tau))_{i,j} \leftarrow (n_B(t))_{ij} - 1$
34:        else $r = A + B \to 2B$ in $(i, j)$ compartment then
35:            $(n_A(t + \tau))_{i,j} \leftarrow (n_A(t))_{ij} - 1$
36:            $(n_B(t + \tau))_{i,j} \leftarrow (n_B(t))_{ij} + 1$
37:        end if
38:        $t \leftarrow t + \tau$
39:    end while
40: end while=0
```

$\boxed{4}$ Performance Analysis

In this section, we conduct a performance analysis for each method described in the previous section when simulating a reaction-diffusion Lotka-Volterra system in a domain $\{(x, y, t) \in ([0, 10]\text{m}, [0, 10]\text{m}, [0, 500]\text{s})\}$ with reaction rates $\sigma_1 = \sigma_2 = 0.05$ s^{-1}, $\sigma_3 = 0.05$ m^2s^{-1}, random initial conditions, and PBCs. We do so in terms of the diffusion constant, the number of particles, the number of compartments, and whether the system is in a diffusion- or reaction-controlled regime. For the FDM and PBM methods, the stepsize is $\Delta t = 0.01$ s. Hence, $\sigma_1 \Delta t = \sigma_2 \Delta t = 0.005 \ll 1$ so PBM's first-order reactions are accurately represented.

The results of the stochastic algorithms for each set of parameters are the average of twenty trials. These runs might not simulate the same total time because some systems might reach extinction. Hence, we measure performance in terms of the average computational time taken per unit of simulated time. Moreover, throughout each simulation, CBM and CPBM may use a variety of different timesteps so we plot the average. Finally, to compare the stochastic trajectories and deterministic orbits, we calculate the average distance from the trajectory/orbit points to the origin of the phase plane. The closer two trajectories/orbits are, the smaller the difference between their average distances should be.

4.1. Diffusion Constant

Figure 2 shows how, for fixed compartment size and initial number of particles, as the diffusion constant increases: (1) the average timestep of CBM and CPBM decreases; (2) the computational time of PBM and (for $D < 0.3$ m^2s^{-1}) CPBM decreases, and the computational time of CBM and (for $D > 0.3$ m^2s^{-1}) CPBM increases; and (3) all stochastic methods tend to the deterministic (FDM) solution. Overall, CPBM is the least computationally expensive, and PBM approaches FDM the fastest.

For $D < 0.3$ m^2s^{-1}, the system is in a diffusion-controlled CPBM regime. For $D > 100$ m^2s^{-1}, the system becomes reaction-controlled. In the next sections we will perform experiments on $D = 0.1$ m^2s^{-1} (diffusion-controlled) and $D = 5$ m^2s^{-1} (almost-reaction-controlled).

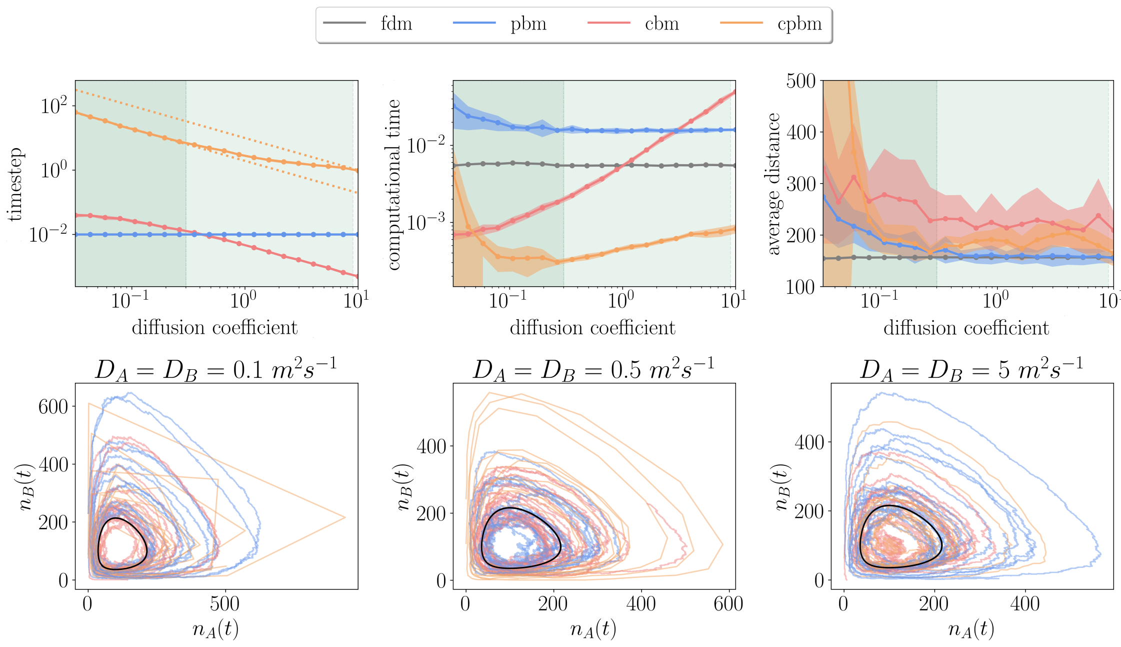

Fig. 2: Effect of the diffusion coefficients on the performance of FDM, PBM, CBM, and CPBM. **(1st row)** Solid lines represent the average of twenty trials per dot, and the shaded regions cover one standard deviation. Dark-to-light-green regions mark the diffusion-controlled, intermediate, and reaction-controlled CPBM regimes. Dotted-yellow lines represent CPBM diffusion-controlled $\kappa_2\tau_D$ (down) and reaction-controlled $10\tau_D$ (up) timesteps. For CBM and CPBM, the domain was split into 25 compartments ($C_x = 5$, $C_y = 5$). The initial number of particles was 50 per species. **(2nd row)** five sample runs of each method for three different diffusion coefficients.

4.2. Number of particles

Figure 3 shows how, for a fixed compartment size and diffusion coefficient, as the initial number of particles increases: (1) the average timestep of CBM and CPBM (if it is not already $\kappa_2\tau_D$) decreases, (2) the computational time of all stochastic methods increases, and (3) all stochastic methods tend to the deterministic (FDM) solution. We plotted the average distance up to 100 initial particles per species because, for larger numbers, extinction is reached faster (as reported by [4]) before full orbits are described, so this is not a representative measure of the algorithms' behaviours. As before, stochastic methods converge closer and faster to the FDM for high diffusion coefficients. Again, CPBM is the least computationally expensive, and PBM approaches FDM the fastest.

For $D = 0.1$ m^{2}s^{-1} and $D = 5$ m^{2}s^{-1}, the system is always in a diffusion-controlled and almost-reaction-controlled CPBM regime, respectively.

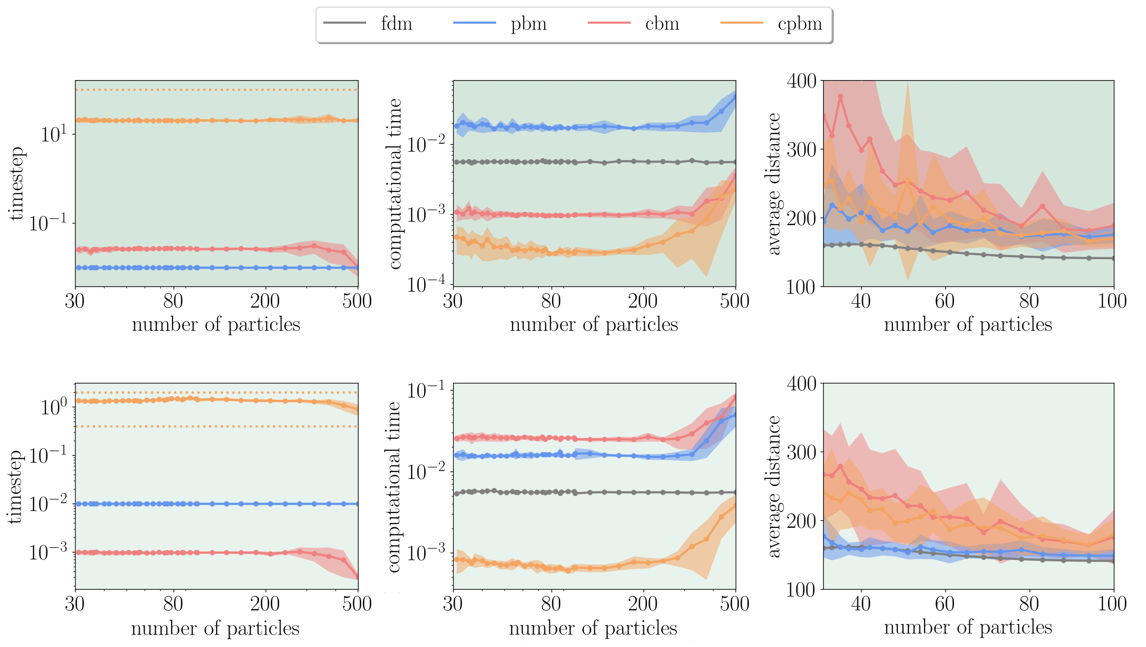

Fig. 3: Effect of the initial number of particles of each species on the performance of FDM, PBM, CBM, and CPBM. Solid lines represent the average of twenty trials per dot, and the shaded regions cover one standard deviation. Dark-to-light-green regions mark the diffusion-controlled, intermediate, and reaction-controlled CPBM regimes. Dotted-yellow lines represent CPBM diffusion-controlled $\kappa_2 \tau_D$ (down) and reaction-controlled $10\tau_D$ (up) timesteps. For CBM and CPBM, the domain was split into 25 compartments ($C_x = 5$, $C_y = 5$). The diffusion coefficients are $D_A = D_B = 0.1$ m^{2}s^{-1} (1st row) and $D_A = D_B = 5$ m^{2}s^{-1} (2nd row).

4.3. Compartment size

Figure 3 shows how, for a fixed diffusion coefficient and initial number of particles, an increasing number of compartments leads to (1) a decrease in timestep for CBM and CPBM, (2) an increase in computational time for CBM and CPBM, and (3) For $D = 0.1$ m^{2}s^{-1}, an initial approach and later deviation of CBM and CPBM to the deterministic solution (FDM). For $D = 5$ m^{2}s^{-1}, CPBM deviates the furthest from FDM in the intermediate regime, and CBM remains more or less at the same distance. PBM and FDM remain unchanged. CPBM is the least computationally expensive.

For $D = 0.1$ m^{2}s^{-1}, the system transitions from a diffusion-controlled to an intermediate regime at $C_x C_y = 36$. For $D = 5$ m^{2}s^{-1}, the system transitions from a diffusion-controlled ($C_x C_y < 4$), to an intermediate ($4 < C_x C_y < 36$), to a reaction-controlled ($C_x C_y > 36$) regime.

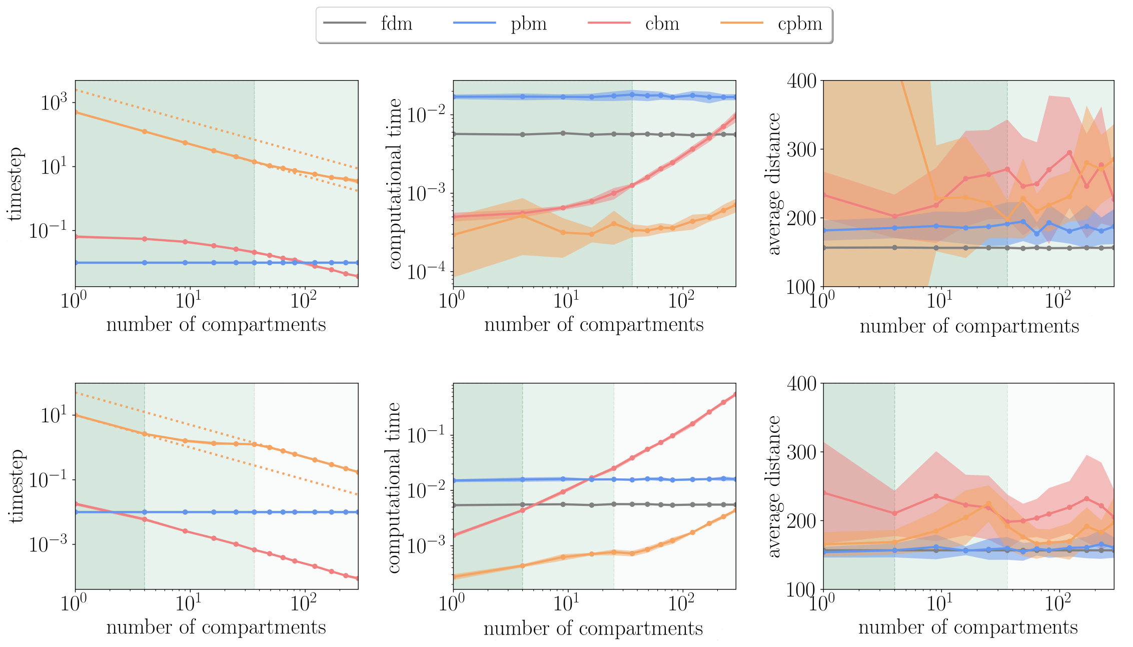

Fig. 4: Effect of the total number of compartments on the performance of CBM and CPBM. Solid lines represent the average of twenty trials per dot, and the shaded regions cover one standard deviation. Dark-to-light-green regions mark the diffusion-controlled, intermediate, and reaction-controlled CPBM regimes. Dotted-yellow lines represent CPBM diffusion-controlled $\kappa_2\tau_D$ (down) and reaction-controlled $10\tau_D$ (up) timesteps. The initial number of particles was 50 per species. The diffusion coefficients are $D_A = D_B = 0.1$ m^{2}s^{-1} (1st row) and $D_A = D_B = 5$ m^{2}s^{-1} (2nd row).

5 Discussion

Throughout all results and regimes, it is clear that CPBM is the least computationally expensive because it can apply larger timesteps, and PBM is the best-behaved because it is the most representationally accurate. Moreover, CPBM follows a closer behaviour to PBM when the system is diffusion-controlled, and a closer behaviour to CBM when it evolves towards reaction-controlled. We note that, overall, results are noisy, and this project would benefit from carrying out more and longer simulations of the system in the future.

5.1. Diffusion

Figure 2 shows the effect of diffusion on the performance of the numerical methods. In agreement with [2], the timesteps of CBM and CPBM decrease because α_{sum} (eq. (16))

increases and F (eq. (23)) decreases respectively. CPBM is the fastest of all methods because it can apply larger time steps. PBM is the slowest except for large diffusion coefficients where CBM exceeds it. This results from the CBM timestep's shrinkage.

As reported by [3], all methods tend to the deterministic solution (FDM) as the diffusion coefficient increases because the system becomes better mixed. This agreement results from (a) an accurate representation of second-order reactions and diffusion by FDM, (b) an accurate representation of second-order reactions by PBM ($\sqrt{2(D_A + D_B)} \gg \sqrt{\sigma_3/\pi} \simeq 0.13 \text{ ms}^{-1/2}$), and (c) CBM's and CPBM's compartmentalisation being a good approximation ($\sigma_3/(D_A + D_B) \ll l, w = 2\text{m}$).

For small diffusion coefficients, the CPBM timestep is so big ($\sim 10^2$) that it only takes a few iterations to complete the simulation, and the distance from these points to the origin is not representative of the algorithm's behaviour (see the second row of fig. 2). That is why we observe such a substantial deviation from CPBM to the rest of the algorithms at these values. In future research, it would be interesting to perform longer simulations so this is not an issue.

5.2. Number of Particles

Figure 3 shows the effect of the number of particles on the performance of the numerical methods. As before, in line with [2]'s results, the timesteps of CBM and CPBM decrease because α_{sum} (eq. (16)) increases and F (eq. (23)) decreases respectively. This timestep shrinkage leads to an increase in computational time, which is also seen in PBM as it needs to keep track of a rising number of particles. Since the computational time of particle-based methods grows faster, there might come a time when CPBM is slower than CBM. Stochastic methods tend to the deterministic solution (FDM) with rising numbers of particles for stochasticity becomes increasingly unimportant.

5.3. Number of Compartments

Figure 4 shows the effect of the number of compartments on the performance of the numerical methods. The interpretation of the timestep and computational time results is the same as for diffusion and again agrees with [2].

For $D = 0.1 \text{ m}^2\text{s}^{-1}$, as expected, there is an initial convergence of CPM and CPBM to FDM as the number of compartments increases for diffusion is better simulated

($l, w \ll L, W = 10$m). Subsequently, they deviate from FDM as the compartments get too small to accurately simulate second-order reactions ($0.25 = \sigma_3/(D_A + D_B) \lll l, w$). Again, the substantial deviation from CPBM to the rest of the algorithms for a small number of compartments is attributed to the timestep ($\sim 10^3$ s) being too big for this distance measure to correctly portray the algorithm's behaviour.

This is not so much the case for $D = 5$ m^{2}s^{-1}. A possible explanation is that diffusion is fast enough for it to be correctly accounted for with big compartments, and the ratio $\sigma_3/(D_A + D_B) = 0.005$ is very small even compared with the smallest compartments ($l, w = 0.59$ m), so second order reactions are always accurately represented. Hence, we do not see the initial decrease and later increase in distance we observe for $D = 0.1$ m^{2}s^{-1}.

6 Conclusion

We provided a detailed description of four algorithms that can simulate reaction-diffusion processes: a finite differences method (FDM), a particle-based method (PBM) that uses Brownian dynamics, a compartment-based method (CBM) that uses the Gillespie algorithm, and a mixed compartment-particle-based method (CPBM).

We gave a detailed comparison of the four algorithms when simulating a two-dimensional reaction-diffusion Lotka-Volterra system with random initial conditions and periodic boundary conditions. For all stochastic algorithms, we showed that the reaction-controlled regime — i.e. large diffusion coefficients and initial numbers of particles — with medium-sized compartments gave results closest to the deterministic solution. Our analysis demonstrated that, in both diffusion- and reaction-controlled regimes, CPBM was the least computationally expensive (except for large numbers of particles) because it could apply larger timesteps, and PBM was the best-behaved because it was the most accurate at capturing the system's dynamics.

In conclusion, this project reviewed the advantages and disadvantages of some commonly used numerical methods and showcased them by simulating a reaction-diffusion Lotka-Volterra system.

References

[1] R. Erban and S. J. Chapman. *Stochastic Modelling of Reaction–Diffusion Processes*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2020.

[2] T. Choi et al. "Stochastic operator-splitting method for reaction-diffusion systems". *The Journal of chemical physics* **137**.18 (2012).

[3] Á. Ruiz-Martínez et al. "Stochastic self-tuning hybrid algorithm for reaction-diffusion systems". *The Journal of Chemical Physics* **151**.24 (2019).

[4] M. Parker and A. Kamenev. "Extinction in the Lotka-Volterra model". *Physical Review E* **80**.2 (2009).

[5] M. Droste. "A functional approach to stochastic Lotka-Volterra equations". MA thesis. Utrecht University, 2021.

[6] D. T. Gillespie. "Exact stochastic simulation of coupled chemical reactions". *The Journal of Physical Chemistry* **81**.25 (1977), pp. 2340–2361.

[7] M. Banaji. *Supplementary Notes Part 2*. Lecture Notes of B5.1 Stochastic Modelling of Biological Processes. Mathematical Institute, University of Oxford, 2024. URL: https://courses.maths.ox.ac.uk/pluginfile.php/103042/mod_resource/content/12/B51NotesPart2.pdf.

$\boxed{\text{Appendix A}}$ Lotka-Volterra Stability Analysis

Throughout this section we will use the notation $\boldsymbol{c} = (c_A, c_B)^T$. The fixed points of the system of eq. (2) are found setting $\partial c_A(t)/\partial t = \partial c_B(t)/\partial t = 0$.

$$\boldsymbol{c}^* = (c_A^*, c_B^*)^T = \begin{cases} (0,0)^T; \\ (\sigma_2/\sigma_3, \sigma_1/\sigma_3)^T. \end{cases}$$

Linearised dynamics To determine their stability the system is linearised around each fixed point using the Jacobian.

$$\frac{\partial \boldsymbol{c}(t)}{\partial t} = J(\boldsymbol{c}^*)[\boldsymbol{c}(t) - \boldsymbol{c}^*] \quad \text{with} \quad J(\boldsymbol{c}^*) = \begin{pmatrix} \sigma_1 - \sigma_3 c_B^* & -\sigma_3 c_A^* \\ \sigma_3 c_B^* & -\sigma_2 + \sigma_3 c_A^* \end{pmatrix}.$$

For the extinction fixed point $\boldsymbol{c}_0^* = (0,0)^T$ the Jacobian is

$$J(\boldsymbol{c}_0^*) = \begin{pmatrix} \sigma_1 & 0 \\ 0 & -\sigma_2 \end{pmatrix},$$

with eigenvalues $\{\lambda_{0,j}^*\}_{j=1}^2 = \{\sigma_1, -\sigma_2\}$. Since these are real and of opposite sign, the fixed point is a saddle point, and hence it is unstable. For the coexistence fixed point $\boldsymbol{c}_1^* = (\sigma_2/\sigma_3, \sigma_1/\sigma_3)^T$ the Jacobian is

$$J(\boldsymbol{c}_1^*) = \begin{pmatrix} 0 & -\sigma_2 \\ \sigma_1 & 0 \end{pmatrix},$$

with eigenvalues $\{\lambda_{0,j}^*\}_{j=1}^2 = \{\pm i\sqrt{\sigma_1\sigma_2}\}$. Since these are imaginary and of opposite sign, the fixed point is either a center or an elliptic fixed point. Such a point is marginally stable, so it is neither attracting nor repelling.

Full dynamics Progress in the description of the dynamics can be made using the full-nonlinear expression. These equations can be combined as follows.

$$\frac{\partial c_A(t)}{\partial c_B(t)} = \frac{c_A(t)}{c_B(t)} \frac{\sigma_1 - \sigma_3 c_B(t)}{-\sigma_2 + \sigma_3 c_A(t)}.$$

Performing separation of variables gives

$$\frac{\sigma_3 c_A(t) - \sigma_2}{c_A(t)} \partial c_A(t) + \frac{\sigma_3 c_B(t) - \sigma_1}{c_B(t)} \partial c_B(t) = 0.$$

Finally, integrating gives the constant

$$G = \sigma_3(c_A(t) + c_B(t)) - (\sigma_1 + \sigma_2) - \sigma_2 \ln\left(\frac{c_A(t)\sigma_3}{\sigma_2}\right) - \sigma_1 \ln\left(\frac{c_B(t)\sigma_3}{\sigma_1}\right).$$

where G was chosen such that $G = 0$ at $\boldsymbol{c}_1^*$ and $G \to \infty$ as cycles approach the axes $(c_A(t), c_B(t) \to 0)$.

Appendix B Python Implementation

B.1. Finite Difference Method (FDM)

```python
import numpy as np
from alive_progress import alive_bar

def discrete_laplacian(M,dx,dy):
    """
    Find the discrete Laplacian of matrix M : finite differences matrix with periodic
    boundary conditions.
    """

    L = - 2*M/(dx**2) - 2*M/(dy**2)
    L += np.roll(M, (0, -1), (0, 1))/(dx**2)  # right neighbour
    L += np.roll(M, (0, +1), (0, 1))/(dx**2)  # left neighbour
    L += np.roll(M, (-1, 0), (0, 1))/(dy**2)  # top neighbour
    L += np.roll(M, (+1, 0), (0, 1))/(dy**2)  # bottom neighbour
    return L


def lotka_volterra_step(A, B, DA, DB, k1, k2, k3, dt, dx, dy):
    """
    Updates a concentration configuration according to a Lotka-Volterra model with
    diffusion coefficients DA and DB, as well as reaction rates k1, k2, k3.
    """

    # Get discrete Laplacians
    LA = discrete_laplacian(A, dx, dy)
    LB = discrete_laplacian(B, dx, dy)

    # Evolve state according to reaction-diffusion equation
    AA = A + (DA * LA + k1 * A - k3 * A * B) * dt
    BB = B + (DB * LB - k2 * B + k3 * A * B) * dt

    return AA, BB


def run_fdm(DA, DB, k1, k2, k3, A0, B0, Niter, dt, dx, dy):
    """
        Runs a finite difference simulation of Lotka-Volterra diffusion-reaction with
        diffusion coefficients DA and DB, as well as reaction rates k1, k2, k3.

        The grid has a spacing of dx and dy.

        The system is updated Niter times with a time interval of dt.

        The initial conditions and the grid size are set by A0 and B0.
    """

    assert A0.shape == B0.shape, "A0 and B0 must have the same shape"
    Ny, Nx = A0.shape

    # Create arrays to store concentrations over time
    A = np.zeros((Ny, Nx, Niter))
    B = np.zeros((Ny, Nx, Niter))

    # Set initial conditions
    A[:, :, 0] = A0
    B[:, :, 0] = B0

    with alive_bar(Niter, force_tty=True) as bar:
        for i in range(1, Niter):

            # Update state
            A[:, :, i], B[:, :, i] = lotka_volterra_step(A[:, :, i-1], B[:, :, i-1],
    DA, DB, k1, k2, k3, dt, dx, dy)
            # Update progression bar
            bar()

    return A, B, dt*np.arange(Niter)
```

```python
import numpy as np
from alive_progress import alive_bar

def lotka_volterra_step(A, B, DA, DB, k1, k2, k3, dt, xmax, ymax):
    """
        Updates a configuration of particles according to a Lotka-Volterra model with
        diffusion coefficients DA and DB, as well as reaction rates k1, k2, k3.
    """

    rowsA, colsA = A.shape
    rowsB, colsB = B.shape

    # compute positions
    A += np.sqrt(2*DA*dt) * np.random.normal(size=(rowsA, colsA))
    B += np.sqrt(2*DB*dt) * np.random.normal(size=(rowsB, colsB))

    # set periodic boundary conditions
    A[:, A[0, :] < 0] = (np.array([xmax, 0]).reshape(-1, 1) + A[:, A[0, :] < 0] - xmax * np.floor(A[0, A[0, :] < 0] / xmax) * np.array([1, 0]).reshape(-1, 1))
    A[:, A[1, :] < 0] = (np.array([0, ymax]).reshape(-1, 1) + A[:, A[1, :] < 0] - ymax * np.floor(A[1, A[1, :] < 0] / ymax) * np.array([0, 1]).reshape(-1, 1))
    B[:, B[0, :] < 0] = (np.array([xmax, 0]).reshape(-1, 1) + B[:, B[0, :] < 0] - xmax * np.floor(B[0, B[0, :] < 0] / xmax) * np.array([1, 0]).reshape(-1, 1))
    B[:, B[1, :] < 0] = (np.array([0, ymax]).reshape(-1, 1) + B[:, B[1, :] < 0] - ymax * np.floor(B[1, B[1, :] < 0] / ymax) * np.array([0, 1]).reshape(-1, 1))

    A[:, A[0, :] > xmax] = (A[:, A[0, :] > xmax]
                            - xmax * np.floor(A[0, A[0, :] > xmax] / xmax) * np.array([1, 0]).reshape(-1, 1))
    A[:, A[1, :] > ymax] = (A[:, A[1, :] > ymax]
                            - ymax * np.floor(A[1, A[1, :] > ymax] / ymax) * np.array([0, 1]).reshape(-1, 1))
    B[:, B[0, :] > xmax] = (B[:, B[0, :] > xmax]
                            - xmax * np.floor(B[0, B[0, :] > xmax] / xmax) * np.array([1, 0]).reshape(-1, 1))
    B[:, B[1, :] > ymax] = (B[:, B[1, :] > ymax]
                            - ymax * np.floor(B[1, B[1, :] > ymax] / ymax) * np.array([0, 1]).reshape(-1, 1))

    # A -> 2A
    r = np.random.rand(colsA)
    if any(r < k1*dt):
        new_cols = np.hstack([A[:, i].reshape(-1, 1) for i in range(colsA) if r[i] < k1*dt])
        A = np.hstack([A, new_cols])

    # B -> 0
    r = np.random.rand(colsB)
    B = np.delete(B, [i for i in range(colsB) if r[i] < k2*dt], 1)

    # Re-evaluate rows and columns because they have changed
    rowsA, colsA = A.shape
    rowsB, colsB = B.shape

    # A + B -> 2B
    rho = np.sqrt(k3*dt/np.pi)  # Reaction radius
    euclidian_distances = np.sqrt((A*A).sum(axis=0).reshape((colsA, 1))*np.ones(shape=(1, colsB)) + (B*B).sum(axis=0) * np.ones(shape=(colsA, 1)) - 2 * A.T.dot(B))  # (a-b)^2=a*a-2*a*b+b*b

    idx_possible_reactions = np.argwhere(euclidian_distances < rho)
    np.random.shuffle(idx_possible_reactions)

    A_reacted = []
    B_reacted = []
    for idx in idx_possible_reactions:
        if (idx[0] not in [A_idx for A_idx in A_reacted]
                and idx[1] not in [B_idx for B_idx in B_reacted]):
            A_reacted.append(idx[0])
            B_reacted.append(idx[1])
    if len(A_reacted) != 0 and len(B_reacted) != 0:
        A = np.delete(A, [i for i in A_reacted], 1)
        new_cols = np.hstack([B[:, i].reshape(-1, 1) for i in B_reacted])
        B = np.hstack([B, new_cols])

    return A, B


def run_pbm(DA, DB, k1, k2, k3, A0, B0, Niter, dt, xmax, ymax):
    """
        Runs a particle-based stochastic simulation of Lotka-Volterra diffusion
        -reaction with diffusion coefficients DA and DB, as well as reaction rates
        k1, k2, k3.

        The domain is [0, xmax] x [0, ymax].

        The system is updated Niter times with a time interval of dt.

        The initial conditions are A0 and B0.
    """

    assert A0.shape[0] == 2, "A0 and B0 must be two-dimensional"

    # Create variables to store extinction information
    extinction = False
    extinction_time = None

    # Create arrays to store concentrations over time
    A = []
    B = []

    # Set initial conditions
    Anow = A0
    Bnow = B0
    A.append(Anow.copy())
    B.append(Bnow.copy())

    with alive_bar(Niter-1, force_tty=True) as bar:

        for i in range(1, Niter):

            # Update state
            Anow, Bnow = lotka_volterra_step(A[-1], B[-1], DA, DB, k1, k2, k3, dt,
    xmax, ymax)
            A.append(Anow.copy())
            B.append(Bnow.copy())

            # Update progression bar
            bar()

            # Break if extinction
            if np.sum(Anow) == 0 or np.sum(Bnow) == 0:
                extinction = True
                extinction_time = dt*i
                break

    t = dt * np.arange(len(A))

    return A, B, t, extinction, extinction_time
```

```python
import numpy as np
from alive_progress import alive_bar

def lotka_volterra_step(A, B, DA, DB, k1, k2, k3, dx, dy):
    """
        Updates a configuration of particles according to a Lotka-Volterra model with
        diffusion coefficients DA and DB, as well as reaction rates k1, k2, k3.
    """

    ny, nx = A.shape  # size array

    # Reaction rates

    kx_A = DA/(dx**2)  # A horizontal diffusion
    ky_A = DA/(dy**2)  # A vertical diffusion

    kx_B = DB/(dx**2)  # B horizontal diffusion
    ky_B = DB/(dy**2)  # B vertical diffusion

    # Generate two random numbers r1, r2, uniformly distributed in [0,1]

    r1, r2 = np.random.rand(2)

    # Compute the propensity function of each reaction

    chi_A_ij = A * kx_A  # A: horizontal diffusion
    ups_A_ij = A * ky_A  # A: vertical diffusion
    chi_B_ij = B * kx_B  # B: horizontal diffusion
    ups_B_ij = B * ky_B  # B: vertical diffusion

    alpha_ij = A * k1  # A -> 2A
    beta_ij = B * k2  # B -> 0
    gamma_ij = A * B * k3 / (dx*dy)  # A + B -> 2B

    # Compute array of propensities

    a = np.concatenate(
        [chi_A_ij.flatten(), chi_A_ij.flatten(), ups_A_ij.flatten(), ups_A_ij.flatten(),  # diffusion A
         chi_B_ij.flatten(), chi_B_ij.flatten(), ups_B_ij.flatten(), ups_B_ij.flatten(),  # diffusion B
         alpha_ij.flatten(), beta_ij.flatten(), gamma_ij.flatten()])  # Lotka-Volterra reactions

    # Compute total propensity
    a_sum = np.sum(a)

    if a_sum == 0:
        return A, B, 0

    # Compute the time at which the chemical reaction takes place tau

    tau = (1/a_sum)*np.log(1/r1)

    # Compute which reaction takes place

    cumsum_a = np.cumsum(a / np.sum(a))
    r_index = np.min(np.where(r2 < cumsum_a))

    # Update number of particles

    if r_index < nx*ny:  # A x-dir right
        ix = r_index % ny
        iy = int(r_index / ny)

        A[iy, ix] -= 1
        if ix+1 == nx: A[iy, 0] += 1
        else: A[iy, ix+1] += 1

    elif r_index < 2*nx*ny:  # A x-dir left
        ix = (r_index-nx*ny) % ny
        iy = int((r_index-nx*ny) / ny)

        A[iy, ix] -= 1
        A[iy, ix-1] += 1

    elif r_index < 3*nx*ny:  # A y-dir up
        ix = (r_index - 2*nx*ny) % ny
        iy = int((r_index - 2*nx*ny) / ny)

        A[iy, ix] -= 1
        if iy+1 == ny: A[0, ix] += 1
        else: A[iy+1, ix] += 1

    elif r_index < 4*nx*ny:  # A y-dir down
        ix = (r_index - 3*nx*ny) % ny
        iy = int((r_index - 3*nx*ny) / ny)

        A[iy, ix] -= 1
        A[iy-1, ix] += 1

    elif r_index < 5*nx*ny:  # B x-dir right
        ix = (r_index - 4*nx*ny) % ny
        iy = int((r_index - 4*nx*ny) / ny)

        B[iy, ix] -= 1
        if ix+1 == nx: B[iy, 0] += 1
        else: B[iy, ix+1] += 1

    elif r_index < 6*nx*ny:  # B x-dir left
        ix = (r_index - 5*nx*ny) % ny
        iy = int((r_index - 5*nx*ny) / ny)

        B[iy, ix] -= 1
        B[iy, ix-1] += 1

```

```python
    elif r_index < 7*nx*ny:  # B y-dir up
        ix = (r_index - 6*nx*ny) % ny
        iy = int((r_index - 6*nx*ny) / ny)

        B[iy, ix] -= 1
        if iy+1 == ny: B[0, ix] += 1
        else: B[iy+1, ix] += 1

    elif r_index < 8*nx*ny:  # B y-dir down
        ix = (r_index - 7*nx*ny) % ny
        iy = int((r_index - 7*nx*ny) / ny)

        B[iy, ix] -= 1
        B[iy-1, ix] += 1

    elif r_index < 9*nx*ny:  # A -> 2A
        ix = (r_index - 8*nx*ny) % ny
        iy = int((r_index - 8*nx*ny) / ny)

        A[iy, ix] += 1

    elif r_index < 10*nx*ny:  # B -> 0
        ix = (r_index - 9*nx*ny) % ny
        iy = int((r_index - 9*nx*ny) / ny)

        B[iy, ix] -= 1

    elif r_index < 11*nx*ny:  # A + B -> 2B
        ix = (r_index - 10*nx*ny) % ny
        iy = int((r_index - 10*nx*ny) / ny)

        A[iy, ix] -= 1
        B[iy, ix] += 1

    else:
        raise "Error: reaction index out of range"

    return A, B, tau

def run_cbm(DA, DB, k1, k2, k3, A0, B0, ttot, dx, dy):
    """
        Runs a compartment-based stochastic simulation of Lotka-Volterra diffusion
        -reaction with diffusion coefficients DA and DB, as well as reaction rates
        k1, k2, k3.

        The grid has a spacing of dx and dy.

        The system is updated until time ttot is reached.

        The initial conditions and the grid size are set by A0 and B0.
    """

    assert A0.shape == B0.shape, "A0 and B0 must have the same shape"

    # Create variables to store extinction information
    extinction = False
    extinction_time = None

    # Create arrays to store concentrations over time
    A = []
    B = []

    # Set initial conditions
    Anow = A0
    Bnow = B0
    A.append(Anow.copy())
    B.append(Bnow.copy())
    t = [0]

    with alive_bar(manual=True, force_tty=True) as bar:

        while t[-1] < ttot:

            # Update state
            Anow, Bnow, tau = lotka_volterra_step(A[-1], B[-1], DA, DB, k1, k2, k3,
dx, dy)
            A.append(Anow.copy())
            B.append(Bnow.copy())
            t.append(t[-1] + tau)

            # Update progression bar
            bar(t[-1]/ttot)

            # Break if extinction
            if np.sum(Anow) == 0 or np.sum(Bnow) == 0:
                extinction = True
                extinction_time = t[-1]
                break

    A = np.dstack(A)
    B = np.dstack(B)
    t = np.array(t).flatten()

    return A, B, t, extinction, extinction_time
```

```python
import numpy as np
from alive_progress import alive_bar
import time

def find_F(A, B, k1, k2, k3, dx, dy, tau_D):
    """
        Finds the time fraction F=T_R,min/tau_D for a Lotka-Volterra system with
        reaction rates k1, k2, k3 divided in compartments of size dx x dy.
    """

    # Compute the propensity function of each reaction

    alpha_ij = A * k1  # A -> 2A
    beta_ij = B * k2  # B -> 0
    gamma_ij = A * B * k3 / (dx * dy)  # A + B -> 2B

    # Compute array of propensities

    a = np.dstack([alpha_ij, beta_ij, gamma_ij])

    # Compute total propensity
    a_sum = np.sum(a, 2)

    # Compute constants
    T_min = np.min(1/np.max(a_sum))
    F = T_min/tau_D

    return F

def diffusion_step(A_compartments, B_compartments, DA, DB, dt, dx, dy):
    """
        Carries out a diffusion step for particles A and B with diffusion constants
        DA and DB distributed in compartments of size dx x dy.
    """

    Ky, Kx = A_compartments.shape
    xmax, ymax = Kx*dx, Ky*dy

    # from compartments to particles
    A_particles = []
    B_particles = []
    for i in range(Ky):
        for j in range(Kx):
            for m in range(int(A_compartments[i,j])):
                rand = np.random.rand(2)*np.array([dx, dy])+np.array([dx*j, dy*i])
                A_particles.append(rand.reshape(-1, 1))
            for n in range(int(B_compartments[i,j])):
                rand = np.random.rand(2) * np.array([dx, dy]) + np.array([dx*j, dy*i
    ])
                B_particles.append(rand.reshape(-1, 1))
    A_particles = np.hstack(A_particles)
    B_particles = np.hstack(B_particles)

    rowsA, colsA = A_particles.shape
    rowsB, colsB = B_particles.shape

    # compute positions
    A_particles += np.sqrt(2 * DA * dt) * np.random.normal(size=(rowsA, colsA))
    B_particles += np.sqrt(2 * DB * dt) * np.random.normal(size=(rowsB, colsB))

    # set periodic boundary conditions
    A_particles[:, A_particles[0, :] < 0] = (np.array([xmax, 0]).reshape(-1, 1) +
    A_particles[:, A_particles[0, :] < 0] - xmax * np.floor(A_particles[0,
    A_particles[0, :] < 0] / xmax) * np.array([1, 0]).reshape(-1, 1))
    A_particles[:, A_particles[1, :] < 0] = (np.array([0, ymax]).reshape(-1, 1) +
    A_particles[:, A_particles[1, :] < 0] - ymax * np.floor(A_particles[1,
    A_particles[1, :] < 0] / ymax) * np.array([0, 1]).reshape(-1, 1))
    B_particles[:, B_particles[0, :] < 0] = (np.array([xmax, 0]).reshape(-1, 1) +
    B_particles[:, B_particles[0, :] < 0] - xmax * np.floor(B_particles[0,
    B_particles[0, :] < 0] / xmax) * np.array([1, 0]).reshape(-1, 1))
    B_particles[:, B_particles[1, :] < 0] = (np.array([0, ymax]).reshape(-1, 1) +
    B_particles[:, B_particles[1, :] < 0] - ymax * np.floor(B_particles[1,
    B_particles[1, :] < 0] / ymax) * np.array([0, 1]).reshape(-1, 1))

    A_particles[:, A_particles[0, :] > xmax] = (A_particles[:, A_particles[0, :] >
    xmax] - xmax * np.floor(A_particles[0, A_particles[0, :] > xmax] / xmax) * np.
    array([1,0]).reshape(-1,1))
    A_particles[:, A_particles[1, :] > ymax] = (A_particles[:, A_particles[1, :] >
    ymax] - ymax * np.floor(A_particles[1, A_particles[1, :] > ymax] / ymax) * np.
    array([0,1]).reshape(-1,1))
    B_particles[:, B_particles[0, :] > xmax] = (B_particles[:, B_particles[0, :] >
    xmax] - xmax * np.floor(B_particles[0, B_particles[0, :] > xmax] / xmax) * np.
    array([1,0]).reshape(-1,1))
    B_particles[:, B_particles[1, :] > ymax] = (B_particles[:, B_particles[1, :] >
    ymax] - ymax * np.floor(B_particles[1, B_particles[1, :] > ymax] / ymax) * np.
    array([0,1]).reshape(-1,1))

    # from particles to compartments
    A_compartments = np.zeros_like(A_compartments)
    B_compartments = np.zeros_like(A_compartments)
    for coord_A in A_particles.T:
        j = int(coord_A[0]/dx)
        i = int(coord_A[1]/dy)
        A_compartments[i, j] += 1
    for coord_B in B_particles.T:
        j = int(coord_B[0]/dx)
        i = int(coord_B[1]/dy)
        B_compartments[i, j] += 1

    return A_compartments, B_compartments

def reaction_step(A, B, k1, k2, k3, dx, dy):
    """
        Carries out a reactions during a time interval dt at each compartment
        of size dx x dy of a Lotka-Volterra system with reaction rates k1, k2
        and k3.
    """

    ny, nx = A.shape  # size array

    # Generate two random numbers r1, r2, uniformly distributed in (0,1)

    r1, r2 = np.random.rand(2)

    # Compute the propensity function of each reaction

    alpha_ij = A * k1  # A -> 2A
    beta_ij = B * k2  # B -> 0
    gamma_ij = A * B * k3 / (dx * dy)  # A + B -> 2B

    # Compute array of propensities

    a = np.concatenate(
        [alpha_ij.flatten(), beta_ij.flatten(), gamma_ij.flatten()])

    # Compute total propensity
    a_sum = np.sum(a)

    if a_sum == 0:
        return A, B, 0

    # Compute the time at which the chemical reaction takes place tau

    tau = (1 / a_sum) * np.log(1 / r1)

    # Compute which reaction takes place

    cumsum_a = np.cumsum(a / np.sum(a))
    r_index = np.min(np.where(r2 < cumsum_a))

    # Update number of particles

    if r_index < nx * ny:  # reaction 1: A -> 2A
        ix = r_index % ny
        iy = int(r_index / ny)

        A[iy, ix] += 1

    elif r_index < 2 * nx * ny:  # reaction 2: B -> 0
        ix = (r_index - nx * ny) % ny
        iy = int((r_index - nx * ny) / ny)

        B[iy, ix] -= 1

    elif r_index < 3 * nx * ny:  # reaction 3: A + B -> 2B
        ix = (r_index - 2 * nx * ny) % ny
        iy = int((r_index - 2 * nx * ny) / ny)

        A[iy, ix] -= 1
        B[iy, ix] += 1

    else:
        raise "Error: reaction index out of range"

    return A, B, tau

def run_cpbm(DA, DB, k1, k2, k3, A0, B0, ttot, dx, dy, alpha_1 = 0.5, alpha_11 = 3,
    alpha_2 = 2, alpha_22 = 3):

    """
        Runs a mixed compartment-particle-based stochastic simulation of Lotka-
        Volterra diffusion-reaction with diffusion coefficients DA and DB, as
        well as reaction rates k1, k2, k3.

        The grid has a spacing of dx and dy.

        The system is updated until time ttot is reached.

        The initial conditions and the grid size are set by A0 and B0.
    """

    start = time.time()
    assert A0.shape == B0.shape, "A0 and B0 must have the same shape"

    # Create variables to store extinction information
    extinction = False
    extinction_time = None

    # Create arrays to store concentrations over time
    A = []
    B = []

    # Set initial conditions
    Anow = A0
    Bnow = B0
    A.append(Anow.copy())
    B.append(Bnow.copy())
    t = [0]

    # Set diffusion time
    d = 2  # system dimensionality
    tau_D = min(dx**2/(2*max(DA, DB)*d), dy**2/(2*max(DA, DB)))

    with alive_bar(manual=True, force_tty=True) as bar:

        while t[-1] < ttot:

            # set timestep
            F = find_F(A[-1], B[-1], k1, k2, k3, dx, dy, tau_D)
            tau = None
            if F < alpha_1:  # diffusion controlled
                tau = alpha_2*tau_D
            elif alpha_1 < F < alpha_11:  # mixed zone
                tau = alpha_22*tau_D
            else:  # reaction controlled
                tau = 10*tau_D

            # diffusion process
            Anow, Bnow = diffusion_step(A[-1], B[-1], DA, DB, tau, dx, dy)

            # reaction process
            t_reac = 0
            while t_reac < tau:
                Anow, Bnow, tnow = reaction_step(Anow, Bnow, k1, k2, k3, dx, dy)
                t_reac += tnow

            # update state
            A.append(Anow.copy())
            B.append(Bnow.copy())
            t.append(t[-1] + tau)

            # Update progression bar
            bar(t[-1]/ttot)

            # Break if extinction
            if np.sum(Anow) == 0 or np.sum(Bnow) == 0:
                extinction = True
                extinction_time = t[-1]
                break

            # Break if taking too long
            now = time.time()
            if now-start > 30*60:
                break

    A = np.dstack(A)
    B = np.dstack(B)
    t = np.array(t).flatten()

    return A, B, t, extinction, extinction_time
```