Efficient methods for simulating stochastic reaction-diffusion models on evolving domains



Bartosz Jan Bartmanski
St Hilda's College
University of Oxford

A thesis submitted for the degree of
Doctor of Philosophy

Trinity Term 2020

To mum.

Acknowledgements

First, I would like to thank my supervisor, Ruth Baker, for all the help and guidance she has provided over the last couple of years. Thanks to my examiners, Joe Pitt Francis and Thomas Woolley, for very useful feedback. I would not have been able to finish this thesis without the help of Victoria Ponce. Her love and support got me through it. I would also like to thank Gianni Klesse and Martin Robinson for helpful hints related to programming and Andrew Krause for interesting discussions about various topics. Lastly, I would like to thank my mum for supporting me no matter what.

Abstract

Mathematical modelling of complex biological phenomena allows us to understand the contributions of different processes to observed behaviours. Many of these phenomena involve the reaction and diffusion of biomolecules and so we use so-called reaction-diffusion models to describe them mathematically. By discretising space into compartments and letting system dynamics be governed by the reaction-diffusion master equation, it is possible to derive and simulate a stochastic model of reaction and diffusion on an arbitrary domain. However, there are many implementation choices involved in this process, such as the choice of discretisation and method of derivation of the diffusive jump rates, and it is not clear *a priori* how these affect model predictions.

To shed light on how model implementation choices affect simulation results, in this thesis we explore how a variety of discretisations and methods for derivation of the diffusive jump rates affect the outputs of stochastic simulations of reaction-diffusion models, in particular using Turing's model of pattern formation as a key example. While only minor differences are observed for simple reaction-diffusion systems, there can be vast differences in model predictions for systems that include complicated reaction kinetics, such as Turing's model of pattern formation.

To date, very few works have considered domain growth for stochastic models. Hence, we consider both static and uniformly growing domains and demonstrate that domain growth can impact how the different diffusive jump rates affect simulation results. Furthermore, we consider discretisations of the domain using complex meshes. Using complex meshes allows models to capture geometric characteristics of a system. We investigate the effects of using complex meshes and the corresponding choices of diffusive jump rates on simulation results. This thesis highlights that care must be taken in using the reaction-diffusion master equation to make predictions as to the dynamics of stochastic reaction-diffusion systems.

Contents

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Stochastic modelling and simulation . . . 5

1.2.1 Stochastic modelling frameworks . . . 5

1.2.2 Stochastic simulations . . . 9

1.3 Pattern analysis . . . 10

1.3.1 Linear stability analysis . . . 12

1.3.2 Weak noise expansion . . . 16

1.4 Thesis structure . . . 22

2 Simulation of stochastic reaction-diffusion models on static domains **24**

2.1 The jump rates . . . 25

2.1.1 Derivations . . . 26

2.1.1.1 Diffusion of a single molecule . . . 26

2.1.1.2 The finite difference method . . . 29

2.1.1.3 The finite element method . . . 31

2.1.1.4 The finite volume method . . . 33

2.1.1.5 The first exit time method . . . 35

2.1.2 Comparison between methods . . . 40

2.2 Case studies . . . 42

2.2.1 Diffusion . . . 42

2.2.2 Zeroth- and first-order reactions . . . 47

2.2.3 Second-order reactions . . . 48

2.2.4 Schnakenberg kinetics . . . 50

2.3 Discussion . . . 58

3 Growing domains and Cartesian meshes **60**

3.1 Mathematical techniques . . . 61

3.1.1 Uniform domain growth . . . 61

3.1.2 Derivations of jump rates on a growing domain . . . 63

3.2 Case studies . . . 68

3.2.1 Diffusion . . . 70
3.2.2 Schnakenberg kinetics . . . 72
3.2.3 Gray-Scott kinetics . . . 78
3.3 Discussion . . . 84

4 The RDME framework on unstructured meshes with domain growth 86
4.1 Mathematical techniques . . . 87
4.1.1 Meshes and dual meshes . . . 87
4.1.2 Jump rates on unstructured meshes . . . 88
4.1.2.1 Finite element method . . . 90
4.1.2.2 Finite volume method . . . 92
4.2 Case studies . . . 95
4.2.1 Diffusion only . . . 96
4.2.1.1 Including domain growth . . . 99
4.2.2 Schnakenberg kinetics . . . 101
4.2.2.1 Square domain . . . 102
4.2.2.2 Other domain shapes . . . 107
4.3 Discussion . . . 109

5 An efficient computational implementation of the RDME framework 111
5.1 StoSpa2 . . . 112
5.1.1 CME example . . . 115
5.1.2 RDME example . . . 118
5.1.3 Growing domain example . . . 121
5.1.4 Schnakenberg example . . . 124
5.1.5 Unit-testing . . . 129
5.2 DualMesh . . . 134

6 Discussion 140
6.1 Chapter 2 . . . 140
6.2 Chapter 3 . . . 142
6.3 Chapter 4 . . . 143
6.4 Chapter 5 . . . 144
6.5 Future work . . . 145
6.6 Concluding remarks . . . 146

A Simulation of stochastic reaction-diffusion models on static domains 147
A.1 Finite element method derivation 147
A.2 First exit time method derivation 153
A.3 Analytic solution 155
A.4 Zeroth- and first-order reactions 157

B Growing domains and Cartesian meshes 158
B.1 Analytic solution 158

Bibliography 160

List of Figures

2.1 Schematic of jump rates 28
2.2 Schematic diagrams of stencils 29
2.3 Schematic for FVM derivation of the jump rates 35
2.4 Schematic for FET derivation of the jump rates 36
2.5 Effects of varying implementation parameters on the FDM and FET jump rates 40
2.6 Comparison of FDM, FEM, FVM and FET methods of derivation of the jump rates 41
2.7 Comparison of the FET-derived jump rates when the parameter β is varied 41
2.8 Example output from simulations of diffusion-only system 44
2.9 Errors as a function of parameters α and κ for simulations that use FDM-derived jump rates 45
2.10 Errors as a function of parameters β and κ for simulations that use FET-derived jump rates 45
2.11 Errors as a function of parameter κ 46
2.12 Effect of compartment size on stationary distribution 49
2.13 Comparison of stationary distributions for different methods of derivation of jump rates 50
2.14 Effects of varying implementation parameter α in FDM-derived jump rates in the Schnakenberg system 53
2.15 Effects of varying implementation parameter β in FET-derived jump rates in the Schnakenberg system when $\kappa = 1.0$ 55
2.16 Effects of varying implementation parameter β in FET-derived jump rates in the Schnakenberg system when $\kappa = 1.4$ 56
2.17 Comparison of using FDM-, FEM-, FVM- and FET-derived jump rates in simulations of the Schnakenberg system 57

3.1 Domain growth functions considered for diffusion-only system 70
3.2 Errors against time for simulations with domain growth 71
3.3 Errors against κ for simulations with domain growth 72
3.4 Simulation output of Schnakenberg system with exponentially growing domain 73

3.5 Domain growth functions considered for Schnakenberg system 74
3.6 Comparison of simulations output of Schnakenberg system for different domain growth functions 75
3.7 Comparison of power spectra of Schnakenberg system for different domain growth functions 76
3.8 Comparison of dominant wavemodes of Schnakenberg system for different domain growth functions 78
3.9 Domain growth functions considered for Gray-Scott system 79
3.10 Initial conditions for Gray-Scott model 80
3.11 Comparison of simulations output of Gray-Scott system for different domain growth functions 82
3.12 Comparison of power spectra of Gray-Scott system for different domain growth functions 83

4.1 Examples of Delauney triangulation and Voronoi tessellation 88
4.2 Schematic diagram of critical angles in FEM derivation on unstructured meshes. 90
4.3 Schematic for FVM derivation 94
4.4 Meshes comparison 96
4.5 Histogram of compartment sizes in an unstructured mesh 97
4.6 Example of simulation output of diffusion-only system on an unstructured mesh 98
4.7 Comparison of errors between simulation on structured and unstructured meshes 99
4.8 Domain growth functions considered for diffusion-only system 100
4.9 Comparison of errors between simulations of diffusion-only system on structured and unstructured meshes 101
4.10 Domain growth functions considered for Schnakenberg system 102
4.11 Comparison of simulations output of Schnakenberg system for different domain growth functions 104
4.12 Comparison of power spectra of Schnakenberg system for different domain growth functions 105
4.13 Comparison of power spectra of Schnakenberg system for different meshes . 107
4.14 Comparison of simulation output for Schnakenberg system on annulus domain 108

5.1 Examples of structured and unstructured meshes 114
5.2 Example of simulation output for system with decay of molecules 118
5.3 Example of simulation output for diffusion-only system 121
5.4 Comparison of simulations on static and growing domains. 124

5.5 The state of the simulation at $T = 1800$ of the Schnakenberg system with the following reaction rates: $D_u = 1 \times 10^{-5}\, m^2 s^{-1}$, $D_v = 1 \times 10^{-3}\, m^2 s^{-1}$, $k_1 = 1\, s^{-1}$, $k_2 = 0.02\, s^{-1}$, $k_3 = 1 \times 10^{-6}\, s^{-1}$ and $k_4 = 3\, s^{-1}$. 129
5.6 Examples of Delauney triangulation and the corresponding Voronoi tessellation 135

A.1 Errors against κ for simulations of systems with zeroth- and first-order reactions 157

Chapter 1

Introduction

1.1 Motivation

Through the scientific method we aim to understand the world around us. This method hinges on making hypotheses and gathering evidence either for or against them. The field of systems biology aims to develop a quantitative understanding of many biological phenomena. It does so through the development of mathematical models that allow us to form hypotheses, which when validated using experimental data, give us an insight into biological mechanisms. The applications of systems biology range from the study of tumour development [1, 5, 8, 9], epidemiology [7, 34, 50, 62] and pattern formation [51, 52], to name but a few.

Huge swathes of fundamental biological phenomena are controlled by chemical reaction networks, and hence the mathematical theory of chemical reaction networks has been extensively developed. One of the first mathematical models of chemical kinetics was developed in 1864 by Waage and Gulberg [28] to study the interaction of alcohol molecules with organic acids, amongst a few other examples of chemical reactions. In this work, the

authors showed that the rate of a chemical reaction depends on the concentration of each reactant. Since then, investigations in this field have branched in many different directions: network classifications based on their mathematical properties; physical analysis based on thermodynamic assumptions; and timescale analysis to determine different long-time behaviours. Many biological systems can be represented as networks of chemical reactions, hence the tools used to model chemical reactions can and have been used for the study of other phenomena, such as gene expression [58] or pattern formation [69].

An important aspect to consider in studying chemical reaction networks, and in many biological systems, is the noise within the system. Noise plays an important role when only small numbers of molecules are present, and the individual behaviours can drastically change the system dynamics. Noise can also be introduced from external sources that influence the system, such as temperature or pH variations. There exist many biological phenomena in which noise plays an important role: defining cell fate decisions, contributing to cancer mutations, amongst others. A classical example of such a system where stochastic effects play an important role is gene expression [6, 33, 38, 55]. Given the many situations where noise plays an important role, it is important that we understand when and how to take it into account.

One common class of models for chemical reaction networks is deterministic models, which assume that there is a negligible amount of noise within the system. Usually in a deterministic model, the state of the system is described by a system of differential equations that, given the initial state of the system, the necessary model parameters and, if necessary, any boundary conditions, the state of the system can be predicted at any future point in time. An example of a deterministic model used to describe the behaviour of a chemical reaction network is a system of ordinary differential equations (ODEs). This

system models the evolution of the concentrations of all the species in the system over time. However, ODEs do not take into account any spatial information about the system. Partial differential equations (PDEs), on the other hand, describe the concentrations of the reactants at every position within the domain, thus taking into account spatial information and heterogeneities. Typically, deterministic models are preferred due to their low computational cost and the extensive mathematical theory behind them. However, deterministic models are not generally appropriate when the number of molecules is low or when the noise drives the observed behaviour, as these models ignore the noise within the system.

Stochastic models aim to take into account noise within the system under consideration. There are a multitude of approaches that we can take to include stochastic effects – from the chemical master equation (CME) framework to stochastic differential equations (SDEs) [56, 65]. The CME is a very commonly used stochastic model. It describes the evolution of the probability of the system to have a particular number of molecules of each type of chemical species present. This model relies on the assumption that the timescale of the diffusion of molecules is much smaller compared to the timescale of all the reactions within the system, *i.e.* that the system is well-mixed. For only very few systems does the CME model have an analytic solution, and so usually stochastic simulations algorithms (SSAs) are used to generate sample paths of the system. For systems in which the simulations of the CME are too computationally expensive [58], the chemical Langevin equation can be a viable alternative.

The stochastic models we mentioned thus far do not take into account the positions of the molecules. An extension to the CME that takes into account spatial effects is called the reaction-diffusion master equations (RDME). The RDME framework, much like the

CME, rarely has an analytic solution and so simulations are usually used to generate sample paths. In the RDME framework, the domain of the system is discretised into non-overlapping compartments amongst which the molecules are distributed, and diffusion is modelled as a series of jumps between adjacent compartments. Hence, the positions of the molecules are taken into account through specifying which of the compartments contain each of the molecules. Alternative approaches to spatial stochastic modelling include the Doi model [13, 14] and the Smoluchowski diffusion limited reaction model [22, 32]. Both the Doi and Smoluchowski approaches model the precise positions of all the molecules, which react if they are less than a certain distance from each other, called a reaction distance [22]. When within the reaction distance, for the Doi model molecules react at a certain rate, while for the Smoluchowski model the react instantaneously. In this work we will focus on the RDME framework, because of its popularity in the field of stochastic modelling, and as it offers an appropriate level of modelling detail for many systems [60], without being prohibitively computationally costly.

Many aspects of modelling using the RDME framework are areas of active research. As already mentioned, diffusion is modelled as jumps between adjacent compartments within the RDME framework. For two- and three-dimensional systems, it has been shown that there exists a choice in defining the rates of molecules jumping between compartments [30, 47, 48]. However, there has to-date been no exploration of how this choice affects predictions made using RDME models. A more general question, is how to calculate the jump rates given any shape of a compartment. In most cases, models are simulated on simple geometries using Cartesian meshes. However, in order to use the RDME framework in complex domains, it is necessary to allow unrestricted compartment shapes. Deriving methods to calculate the appropriate jump rates remains an open question. Furthermore,

some RD systems take place on growing domains and so, it is necessary for models of such systems to include domain growth. How best to do this within the RDME framework is still an open question. Our aim in this thesis is to address these questions to determine how model implementation choices affect simulation results within the RDME framework. To understand the contributions of this thesis, in Sections 1.2 and 1.3 we first give a brief introduction to the mathematical theory behind the RDME and pattern analysis, respectively. We outline the structure of the rest of this thesis in Section 1.4.

1.2 Stochastic modelling and simulation

In this section we introduce basic concepts behind the theory of stochastic modelling. First, in Section 1.2.1, we introduce the mathematical background and then in Section 1.2.2 we describe the Gillespie algorithm, which allows to directly simulate the underlying process. Standard references on this topic are [27, 45, 56, 65].

1.2.1 Stochastic modelling frameworks

Let us start with defining mathematically a chemical reaction network. A chemical reaction network is a mathematical representation of a set of real-world chemical reactions. It comprises of L chemical species, U^ℓ, henceforth referred to as simply species, and the interactions between those species are referred to as reactions. Given a set of species $\mathrm{U}^1, \ldots, \mathrm{U}^L$, a reaction, r, can be written as

$$\sum_{\ell=1}^{L} s_r^\ell \mathrm{U}^\ell \xrightarrow{k_r} \sum_{\ell=1}^{L} \tilde{s}_r^\ell \mathrm{U}^\ell \,, \tag{1.1}$$

where k_r is the rate at which the reaction happens and s_r^ℓ and $\tilde{s}_r^\ell$ represent the number of molecules of each species before and after a reaction happens, respectively. For example, two species, U^1 and U^2, interact to make a third species, U^3, at rate k_1 can be written as

$$\mathrm{U}^1 + \mathrm{U}^2 \xrightarrow{k_1} \mathrm{U}^3 \,. \tag{1.2}$$

Historically, the evolution of such chemical reaction networks has been described by using ODEs, where the rate of change of each species is described by an algebraic expression. The following ODEs describe the evolution of the concentrations of species U^ℓ

$$\frac{\mathrm{d}\phi^\ell}{\mathrm{d}t} = \sum_{r=1}^{R} \nu_r^\ell f(\boldsymbol{\phi}) \quad \text{for} \quad \ell = 1, \ldots, L \,, \tag{1.3}$$

where $\nu_r^\ell = \tilde{s}_r^\ell - s_r^\ell$ is an element of the stoichiometry vector $\boldsymbol{\nu}_r$, which represents the net change of all species given a reaction, r, and ϕ^ℓ is the concentration of species U^ℓ. The function $f(\boldsymbol{\phi})$ represents the rate at which the reaction r occurs. However, there are two assumptions underlying this model of chemical reaction network:

1. the concentrations of all species are large enough such that fluctuations can neglected and stochastic effects more generally are not important;

2. the system is assumed to be well-mixed, *i.e.* the diffusion of all species is much faster than other reactions.

When the first assumption breaks and the system involves small number of molecules, then their dynamics cannot be accurately modelled using concentrations and other frameworks are better suited. The CME framework allows us to model a chemical reaction network by describing the evolution of the probability of the system to be in a given state [27, 45, 56, 65]. We define $\boldsymbol{U}(t) = (U^1(t), \ldots, U^L(t))$ to be the state vector of the system,

i.e. a vector that describes how many molecules of each species there are within the system at time t, where U^ℓ represents the number of molecules of species U^ℓ, for $\ell = 1, \ldots, L$ and $\mathbb{P}(\boldsymbol{u}, t) = \mathbb{P}(\boldsymbol{U}(t) = \boldsymbol{u} | \boldsymbol{U}(0) = \boldsymbol{u}_0)$ to be the probability of the system to be in a given state. We also need to define

$$\begin{aligned} a^r(\boldsymbol{u}(t))\mathrm{d}t \;\; \coloneqq \;\; & \mathbb{P}\left(\text{reaction } r \text{ fires during time interval } [t, t+\mathrm{d}t)\right. \\ & \left. \text{given the system is in state } \boldsymbol{u} \text{ at time } t\right), \end{aligned} \tag{1.4}$$

where $\mathrm{d}t$ is an infinitesimal time increment and $a_i^r(\boldsymbol{u}(t))$ is the propensity function of reaction r. For more details of the specific forms of the propensity functions see [58].

The CME, which describes how the probabilities of being in each state evolve in time, can be written as

$$\frac{\partial}{\partial t}\mathbb{P}\left(\boldsymbol{u}, t\right) = \sum_{r=1}^{R}\left[a^r\left(\boldsymbol{u} - \nu_r\right)\mathbb{P}\left(\boldsymbol{u} - \boldsymbol{\nu}_r, t\right) - a^r\left(\boldsymbol{u}\right)\mathbb{P}\left(\boldsymbol{u}, t\right)\right]. \tag{1.5}$$

We note at this point that the evolution of the mean values of each species matches the ODE description, shown in Equation (1.3), if the system contains only zeroth- and first-order reactions [31]. The main assumption underlying the CME is that the system is well-mixed, *i.e.* the concentrations of all species being considered are homogeneous within their domain [23].

We consider a reaction-diffusion (RD) system consisting of L species, where $U_i^\ell(t)$ is now the number of molecules of species ℓ in compartment i at time t, for $\ell = 1, \ldots, L$ and $i = 1, \ldots, I$. We define $\boldsymbol{U}^\ell(t) = (U_1^\ell(t), \ldots, U_I^\ell(t))$ to be the spatial vector of species ℓ at time t, $\boldsymbol{U}(t) = (\boldsymbol{U}^1(t), \ldots, \boldsymbol{U}^L(t))$ to be the state matrix of the system and $\mathbb{P}(\boldsymbol{u}, t) = \mathbb{P}(\boldsymbol{U}(t) = \boldsymbol{u} | \boldsymbol{U}(0) = \boldsymbol{u}_0)$. Changes to the state matrix occur through either reactions

within compartments or diffusive jumps between compartments. We consider a system of $r = 1, \ldots, R$ chemical reactions such that reaction r has propensity function a_i^r in compartment i and is defined such that

$$\begin{aligned} a_i^r(\boldsymbol{u}(t))\mathrm{d}t \;\; &\coloneqq \;\; \mathbb{P}\left(\text{reaction } r \text{ fires in compartment } i \text{ during } [t, t+\mathrm{d}t)\right. \\ &\qquad \left.\text{given the system is in state } \boldsymbol{u} \text{ at time } t\right), \end{aligned} \tag{1.6}$$

where $\mathrm{d}t$ is an infinitesimal time interval. Similarly, diffusive jumps have propensity function $d_{i,j}^{\ell}$ where

$$d_{i,j}^{\ell}(\boldsymbol{u}(t))\mathrm{d}t = \lambda_{i,j}^{\ell} u_i^{\ell}(t)\mathrm{d}t, \tag{1.7}$$

is the probability that a molecule of species ℓ jumps from compartment i to compartment j in the infinitesimal time interval $[t, t+\mathrm{d}t)$, given the system is in state $\boldsymbol{u}$ at time t. Hence, the RDME can be written as

$$\begin{aligned} \frac{\partial}{\partial t}\mathbb{P}\left(\boldsymbol{u}, t\right) \;\; &= \;\; \sum_{i=1}^{I}\sum_{r=1}^{R}\left[a_i^r\left(\mathbf{u}-\boldsymbol{\nu}_{i,r}\right)\mathbb{P}\left(\boldsymbol{u}-\boldsymbol{\nu}_{i,r}, t\right) - a_i^r\left(\mathbf{u}\right)\mathbb{P}\left(\boldsymbol{u}, t\right)\right] \\ &\quad + \sum_{i=1}^{I}\sum_{\substack{j=1 \\ j\neq i}}^{I}\sum_{\ell=1}^{L}\left[\lambda_{i,j}^{\ell}\left(u_i^{\ell}(t)+1\right)\mathbb{P}\left(\boldsymbol{u}-\boldsymbol{e}_{i,j}^{\ell}, t\right) - \lambda_{i,j}^{\ell}u_i^{\ell}(t)\mathbb{P}\left(\boldsymbol{u}, t\right)\right], \end{aligned} \tag{1.8}$$

where $\boldsymbol{\nu}_{i,r}$ is the stochiometric matrix that describes the change in state upon the firing of reaction r in compartment i, and $\boldsymbol{e}_{i,j}^{\ell}$ is the stochiometric matrix that describes the change in state upon a diffusive jump of a molecule of species ℓ from compartment i to compartment j [31]. The RDME framework is an extension of the CME where the spatial domain is discretised into compartments, each of which is assumed well-mixed, and diffusion is modelled using a set of first-order reactions that describe the jumps of molecules between neighbouring compartments [58].

1.2.2 Stochastic simulations

In general, the CME is intractable and, as a result, Gillespie [27] developed a SSA to generate sample paths of the system under consideration. Again, the RDME is rarely tractable, and so stochastic simulation algorithms have been developed to simulate RD systems modelled using the RDME framework [17, 29].

To generate sample paths for a RD system within the RDME framework described in Equation (1.8), we use an SSA called the next sub-volume method [17, 29]. Let t_i be the time until the next reaction within compartment i for $i = 1, \ldots, I$, and let the total propensity for compartment i be

$$a_i^0(\boldsymbol{u}(t)) = \sum_{k=1}^{K} a_i^k(\boldsymbol{u}(t)) + \sum_{\substack{j=1 \\ j \neq i}}^{I} \sum_{\ell=1}^{L} d_{i,j}^{\ell}(\boldsymbol{u}(t)). \tag{1.9}$$

Then, given that T is the end time of a simulation, we can simulate sample paths consistent with the RDME in Equation (1.8) using the SSA given in Algorithm 1.

Algorithm 1 Next sub-volume method SSA

Set $t = 0$.
Initialise $\boldsymbol{u}$.
for i in $\{1, 2, \ldots, I\}$ **do**
    Initialise the time until the next reaction for compartment i: $t_i \sim \exp(a_i^0(\boldsymbol{u}(t)))$.
end for
while $t < T$ **do**
    Find argmin of the set $\{t_1, t_2, \ldots, t_I\}$ and denote it m.
    Set $t = t_m$.
    Generate a random number $r \sim U(0, 1)$.
    Choose reaction / diffusion q to fire, that is the smallest integer satisfying $\sum_{k=1}^{q} a_i^k(\boldsymbol{u}(t)) > r a_i^0(\boldsymbol{u}(t))$. Note that reaction k fires with probability $a_i^k(\boldsymbol{u}(t))/a_i^0(\boldsymbol{u}(t))$ and diffusion of a molecule of species ℓ from box i to box j occurs with probability $d_{i,j}^{\ell}(\boldsymbol{u}(t))/a_i^0(\boldsymbol{u}(t))$.
    Update molecule numbers: $\boldsymbol{u} := \boldsymbol{u} + \boldsymbol{\nu}_q$, where $\boldsymbol{\nu}_q$ is the stoichiometric matrix of reaction / diffusion q.
    Let Γ be the set of indices of non-zero elements of $\boldsymbol{\nu}_q$.
    for γ in Γ **do**
        Update $a_i^k(\boldsymbol{u}(t))$, $d_{i,j}^{\ell}(\boldsymbol{u}(t))$ and $a_i^0(\boldsymbol{u})$ accordingly.
        Generate a random number $\Delta t \sim \exp(a_\gamma^0(\boldsymbol{u}))$.
        Update the time until the next reaction: $t_\gamma = t + \Delta t$.
    end for
end while

Although the RDME as outlined in Equation (1.8), together with the SSA of Algorithm 1, holds in full generality, for the rest of this work, we assume for simplicity that all diffusion coefficients and reaction rates are spatially homogeneous.

1.3 Pattern analysis

In this thesis we are interested in the application of stochastic modelling frameworks of chemical reactions to investigate how model choices affect simulation results. One type of chemical reaction networks we particularly focus on are those that model pattern formation via a diffusion-driven instability [26, 51, 53, 57, 64, 69]. The most obvious examples of patterns in the natural world are animal coats where we have a multitude of examples. The emergence of patterns on animal coats is typically related to increasing an organism's

chances of survival, either by making a prey more difficult to spot for a predator [51] or vice versa *i.e.* camouflage. Less well-known examples of where patterning is important lie in developmental biology. Key examples in this area include branching in kidney and lung formation [12] and digit formation in the vertebrate limb [63]. Mathematical models of pattern formation provide us with a framework where we can formulate and test hypotheses of the underlying governing mechanisms. The mathematical study of pattern formation started with Alan Turing, who heavily influenced the field of mathematical biology. Turing [64] developed a model of pattern formation to suggest a mechanism by which genes of a zygote influence its development. He was the first person to suggest a diffusion-driven instability model of pattern formation as an explanation for morphogenesis. Since then, many extensions have been done to this work [11, 26, 43, 44, 57, 69], including generalisations to chemotaxis systems and high-dimensional signal transduction systems, to name a few. Currently, a big focus is on the interaction between biochemical patterning and cell mechanics and tissue growth, as many biological systems in development and cancer rely on a complicated interplay between biochemical reactions and mechanical interactions between cells.

The main type of RD system we focus on in this work are those that display diffusion-driven instabilities. These systems are such that the reactions on their own cannot lead to patterns, but with the addition of diffusion to the system patterns become viable. In this section, we introduce a number of tools that help with analysing the models and results of simulations of these RD systems. We first introduce linear stability analysis (LSA) in Section 1.3.1 and the weak-noise expansion (WNE) in Section 1.3.2.

1.3.1 Linear stability analysis

As mentioned in the previous section, the mean behaviour of a stochastic model (CME, RDME) can be similar to the corresponding PDE description when the number of molecules is high enough. Therefore, we first use analysis of the PDE model as a guide to what to expect from a RDME model. By using this fact we can compare properties of the stochastic system with the corresponding mean-field counterpart.

One important tool in determining the spatial distribution of species of molecules in a deterministic RD pattern formation model is LSA. We outline the derivation of the conditions for spatial patterns to form for general RD kinetics, closely following the derivation outlined by Murray [52]. Consider a PDE description of a two-species RD system,

$$\frac{\partial u}{\partial t}(\mathbf{x},t) = D_u \nabla^2 u(\mathbf{x},t) + f(u,v), \tag{1.10}$$

$$\frac{\partial v}{\partial t}(\mathbf{x},t) = D_v \nabla^2 v(\mathbf{x},t) + g(u,v), \tag{1.11}$$

where u is the concentration of species U and v is the concentration of species V on the domain Ω. We consider only two-species systems as they are both simple yet representative of the dynamics of more complicated RD systems that model pattern formation. The initial conditions for this system are

$$u(\mathbf{x},0) = u_0(\mathbf{x}) \quad \text{and} \quad v(\mathbf{x},0) = v_0(\mathbf{x}), \tag{1.12}$$

with Neumann boundary conditions

$$\begin{aligned} (\mathbf{n}\cdot\nabla)\, u = 0 \quad \text{for} \quad \mathbf{x}\in\partial\Omega, \\ (\mathbf{n}\cdot\nabla)\, v = 0 \quad \text{for} \quad \mathbf{x}\in\partial\Omega. \end{aligned} \tag{1.13}$$

Let $\bar{u}$ and $\bar{v}$ be the homogeneous steady states of this system, *i.e.* the values of u and v that satisfy the following equalities: $f(\bar{u},\bar{v}) = 0$ and $g(\bar{u},\bar{v}) = 0$.

We linearise about the spatially uniform steady state by letting $u = \bar{u} + \delta u$ and $v = \bar{v} + \delta v$, where $|\delta u|, |\delta v| \ll 1$. This leads to the following system of equations

$$\frac{\partial \mathbf{w}}{\partial t} = D\nabla^2\mathbf{w} + A\mathbf{w}, \tag{1.14}$$

where $\mathbf{w} = (\delta u, \delta v)^T$. Let us denote

$$\left.\frac{\partial f}{\partial u}\right|_{(\bar{u},\bar{v})} = f_u, \quad \left.\frac{\partial f}{\partial v}\right|_{(\bar{u},\bar{v})} = f_v, \quad \left.\frac{\partial g}{\partial u}\right|_{(\bar{u},\bar{v})} = g_u, \quad \left.\frac{\partial g}{\partial v}\right|_{(\bar{u},\bar{v})} = g_v, \tag{1.15}$$

so that we can write the matrices in Equation (1.14) as

$$A = \begin{bmatrix} f_u & f_v \\ g_u & g_v \end{bmatrix} \quad \text{and} \quad D = \begin{bmatrix} D_u & 0 \\ 0 & D_v \end{bmatrix}. \tag{1.16}$$

Now, let us consider the system without diffusion, *i.e.*

$$\frac{\partial \mathbf{w}}{\partial t} = A\mathbf{w}, \tag{1.17}$$

which is a linear system of ODEs. As a result, the solution is a linear superposition of

solutions of the following type

$$\mathbf{w} = \mathbf{c}e^{\lambda t}. \tag{1.18}$$

The vector $\mathbf{c}$ is a constant vector and λ is a constant to be determined. If we now substitute the above solution into Equation (1.17), then

$$\lambda \mathbf{w} = A\mathbf{w}, \tag{1.19}$$

which has a non-trivial solution if and only if $|A - \lambda I| = 0$. Therefore, we can find the value of λ to be

$$\lambda_{\pm} = \frac{1}{2}\left[f_u + g_v \pm \sqrt{(f_u + g_v)^2 - 4(f_u g_v - f_v g_u)}\right]. \tag{1.20}$$

As stated in [52], we seek patterns driven by diffusion and, as a result, we require that the homogeneous solution be linearly stable, and hence $\mathrm{Re}(\lambda_{\pm}) < 0$. Therefore, the first two conditions necessary for a diffusion-driven instability are

$$f_u + g_v < 0 \quad \text{and} \quad f_u g_v - f_v g_u > 0. \tag{1.21}$$

We now consider the system with diffusion present and assume that the solution is of the form

$$\mathbf{w} = \sum_k \mathbf{c}_k e^{\lambda t}\mathbf{W}_k(\mathbf{x}), \tag{1.22}$$

where $\mathbf{W}_k$ satisfies the eigenvalue problem

$$\nabla^2 W_k + k^2 W_k = 0 \quad \text{with} \quad (\mathbf{n}\cdot\nabla)\mathbf{W}_k = 0 \quad \text{for} \quad \mathbf{x} \in \partial\Omega. \tag{1.23}$$

Here, $\mathbf{n}$ is the vector normal to the domain boundary, $\partial\Omega$. If we now substitute Equation (1.22) into Equation (1.14), we have

$$0 = (A - k^2 D - \lambda I)\mathbf{W}_k, \tag{1.24}$$

which has non-trivial solutions if and only if

$$\left|A - k^2 D - \lambda I\right| = 0. \tag{1.25}$$

The above can be simplified to

$$\lambda^2 + ((D_u + D_v)k^2 - (f_u + g_v))\lambda + h(k^2) = 0, \tag{1.26}$$

where $h(k^2) = D_u D_v k^4 - (D_u g_v + D_v f_u)k^2 + f_u g_v - f_v g_u$. Since we are seeking conditions under which spatial patterns can form, we need linear instability, *i.e.* $\mathrm{Re}(\lambda) > 0$ for $k^2 > 0$. This means that the system is linearly unstable for the values of k at which $h(k^2) < 0$ or when the coefficient of λ is negative. Due to Equation (1.21), we have

$$(D_u + D_v)k^2 - (f_u + g_v) > 0\,, \tag{1.27}$$

since $k^2 \geq 0$. As a result, the only way we can have the system be linearly unstable is for $h(k^2) < 0$. So let us look at the function $h(k^2)$ in more detail.

To determine when the system is linearly unstable, we need to determine for what values of k^2, if any, $h(k^2)$ becomes negative. For $h(k^2)$ to be negative, the coefficient of

k^2 needs to be positive and so we have

$$(D_u g_v + D_v f_u) > 0\,. \tag{1.28}$$

We obtain the last condition by determining the minimum value of $h(k^2)$, which gives us

$$f_u g_v - f_v g_u < \frac{(D_u g_v + D_v f_u)^2}{4 D_u D_v}\,. \tag{1.29}$$

Given details of Ω we can find which wavenumbers, (k_x, k_y), where $k^2 = k_x^2 + k_y^2$, that can be excited in any given system [52].

1.3.2 Weak noise expansion

While for LSA we consider the solutions of the RD systems in terms of the eigenvectors of the Laplacian, a method based on the weak-noise expansion (WNE) [65] allows us to separate the scales and consider the effects of weak noise within the system. Using this approach we analyse the conditions for diffusion-driven instability when using the RDME framework. While this approach is more complex, it is more accurate compared to the LSA, as it takes into account stochasticity, which the LSA does not. We outline the WNE approach, as developed by Woolley *et al.* [68], for which we use the two-dimensional spatial, discrete Fourier cosine transform

$$\hat{f}(k_x, k_y) = \Delta_x \Delta_y \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} \cos\left(k_x \Delta_x (i-1)\right) \cos\left(k_y \Delta_y (j-1)\right) f(x, y). \tag{1.30}$$

We aim to determine which wavenumbers, $\mathbf{k} = (k_x, k_y)$, can be eventually excited in the course of a RDME simulation. Given a system with R reactions and with two species,

U and V, spread over I compartments, we have the RDME as follows

$$\frac{\partial}{\partial t}P\left(\mathbf{W},t|\mathbf{W}_0,t_0\right)=\sum_{i=1}^{2I}\sum_{j=1}^{R}\left[P\left(\mathbf{W}-\boldsymbol{\nu}_j,t|\mathbf{W}_0,t_0\right)a_{i,j}\left(\mathbf{W}-\boldsymbol{\nu_j}\right)\right.$$
$$\left.-\,P\left(\mathbf{W},t|\mathbf{W}_0,t_0\right)a_{i,j}\left(\mathbf{W}\right)\right]. \tag{1.31}$$

The vector $\mathbf{W}=(\mathbf{U},\mathbf{V})$ is composed of $\mathbf{U}=(U_1,U_2,\ldots,U_I)^T$ and $\mathbf{V}=(V_1,V_2,\ldots,V_I)^T$, which are vectors of the number of molecules in each compartment of species U and V, respectively. The function $a_{i,j}$ is the propensity function for reaction j in compartment i, and ν_{ij} is the stoichiometric matrix. Now, we assume that the number of molecules in compartment χ_i can be expressed as

$$U_i=\phi_i\Theta+\eta_{ui}\sqrt{\Theta}\quad\text{and}\quad V_i=\psi_i\Theta+\eta_{vi}\sqrt{\Theta}, \tag{1.32}$$

where ϕ_i and ψ_i are the expected ratios of the number of molecules to the system parameter, Θ, which is the magnitude of the smallest homogeneous steady state, and η_{ui} and η_{vi} are random variables that describe the noise in species U and V, respectively. As we change variables, we redefine the probability P to

$$P(\mathbf{U},\mathbf{V},t)=\Pi(\boldsymbol{\eta}_u,\boldsymbol{\eta}_v,t)\,, \tag{1.33}$$

which we can differentiate with respect to time, keeping the number of molecules constant,

to arrive at

$$\frac{\partial P}{\partial t} = \frac{\partial \Pi}{\partial t} + \sum_{i=1}^{I} \left(\frac{\mathrm{d}\eta_{ui}}{\mathrm{d}t} \frac{\partial \Pi}{\partial \eta_{ui}} + \frac{\mathrm{d}\eta_{vi}}{\mathrm{d}t} \frac{\partial \Pi}{\partial \eta_{vi}} \right) \tag{1.34}$$

$$= \frac{\partial \Pi}{\partial t} - \sqrt{\Theta} \sum_{i=1}^{I} \left(\frac{\partial \phi_i}{\partial t} \frac{\partial \Pi}{\partial \eta_{ui}} + \frac{\partial \psi_i}{\partial t} \frac{\partial \Pi}{\partial \eta_{vi}} \right). \tag{1.35}$$

To simplify further calculations, let us define new variables

$$\boldsymbol{\zeta} = (\boldsymbol{\eta}_u, \boldsymbol{\eta}_v) \quad \text{and} \quad \boldsymbol{\varphi} = (\boldsymbol{\phi}, \boldsymbol{\psi}). \tag{1.36}$$

Therefore, Equation (1.35) becomes

$$\frac{\partial P}{\partial t} = \frac{\partial \Pi}{\partial t} - \sqrt{\Theta} \sum_{i=1}^{2I} \frac{\partial \varphi_i}{\partial t} \frac{\partial \Pi}{\partial \zeta_i}. \tag{1.37}$$

Before we proceed further, we need to make a distinction between the macroscopic propensity function, $\mathrm{a}_{i,j}$, and the microscopic propensity function, $a_{i,j}$. The macroscopic propensity function depends on the expected ratios of both species, $\mathrm{a}_{i,j}(\boldsymbol{\phi}, \boldsymbol{\psi})$, while the microscopic propensity function depends on the number of molecules of both species, $a_{i,j}(\mathbf{U}, \mathbf{V})$. This is important to note, as when we substitute Equation (1.32) into the microscopic propensity function, we have

$$a_{i,j}(\mathbf{W} - \nu_j) = a_{i,j}(\boldsymbol{\phi}\Theta + \boldsymbol{\eta}\sqrt{\Theta} - \boldsymbol{\nu}_j) \tag{1.38}$$

$$= \Theta \mathrm{a}_{i,j} \left(\boldsymbol{\phi} + \frac{\boldsymbol{\eta}}{\sqrt{\Theta}} - \frac{\boldsymbol{\nu}_j}{\Theta} \right) + o\left(\frac{1}{\Theta}\right) \tag{1.39}$$

$$= \Theta \left(\mathrm{a}_{i,j}(\boldsymbol{\phi}) + \frac{1}{\sqrt{\Theta}} \sum_{i=1}^{I} \frac{\partial \mathrm{a}_{i,j}}{\partial \phi_i} \eta_{ui} + o\left(\frac{1}{\Theta}\right) \right). \tag{1.40}$$

If we substitute Equation (1.32) into the RDME, Equation (1.31), we have

$$\frac{\partial \Pi}{\partial t} - \sqrt{\Theta}\sum_{i=1}^{2I}\frac{\partial \varphi_i}{\partial t}\frac{\partial \Pi}{\partial \zeta_i} = \sum_{j=1}^{J}\left(-\sqrt{\Theta}\sum_{i=1}^{2I}\nu_{ij}\frac{\partial}{\partial \zeta_i} + \frac{1}{2}\sum_{i=1}^{2I}\sum_{l=1}^{2I}\nu_{ij}\nu_{lj}\frac{\partial^2}{\partial \zeta_i \zeta_l}\right) \times \left(\mathrm{a}_{i,j}(\boldsymbol{\varphi}) + \frac{1}{\sqrt{\Theta}}\sum_{i=1}^{2I}\frac{\partial \mathrm{a}_{i,j}}{\partial \varphi_i}\right)\Pi(\boldsymbol{\zeta}). \tag{1.41}$$

Next, we expand the above in powers of Θ. The leading term, of order $\sqrt{\Theta}$, gives

$$\sum_{i=1}^{2I}\frac{\partial \varphi_i}{\partial t}\frac{\partial \Pi}{\partial \zeta_i} = \sum_{j=1}^{R}\sum_{i=1}^{2I}\nu_{ij}\frac{\partial \Pi}{\partial \zeta_i}\mathrm{a}_{i,j}, \tag{1.42}$$

which is satisfied by

$$\frac{\partial \varphi_i}{\partial t} = \sum_{j=1}^{R}\nu_{ij}\mathrm{a}_{i,j}(\boldsymbol{\varphi}), \tag{1.43}$$

i.e. the mean-field equations. The next terms in the expansion, terms of order 1, give

$$\frac{\partial \Pi}{\partial t} = -\sum_{j=1}^{R}\left[\sum_{i=1}^{2I}\sum_{l=1}^{2I}\nu_{ij}\frac{\partial \mathrm{a}_{i,j}}{\partial \varphi_i}\frac{\partial(\zeta_l \Pi)}{\partial \zeta_i} + \frac{1}{2}\sum_{i=1}^{2I}\sum_{l=1}^{2I}\mathrm{a}_{i,j}\nu_{ij}\nu_{lj}\frac{\partial^2 \Pi}{\partial \zeta_i \partial \zeta_l}\right] \tag{1.44}$$

$$= -\sum_{i=1}^{2I}\sum_{l=1}^{2I}A_{il}\frac{\partial(\zeta_l \Pi)}{\partial \zeta_i} + \frac{1}{2}\sum_{i=1}^{2I}\sum_{l=1}^{2I}B_{il}\frac{\partial^2 \Pi}{\partial \zeta_i \partial \zeta_l}, \tag{1.45}$$

which is the Fokker-Planck equation, as stated in [68].

We wish to use this method to determine which wavemodes can be excited in a model of pattern formation. As such, we need to find the discrete cosine transform of the Laplacian, as was done in [69]. The authors of [69] have developed this method for one-dimensional systems, and so here we extend the method to two-dimensional systems. We consider covariances

$$\langle \eta_{ul}\eta_{um}\rangle = \int \Pi \eta_{ul}\eta_{um}\,\mathrm{d}\eta_{u1}\ldots\mathrm{d}\eta_{uI}\,. \tag{1.46}$$

With the above definition we can write the equations for the evolution of the Fourier-

transformed covariances as follows

$$\frac{\mathrm{d}}{\mathrm{d}t}\left\langle \widehat{\eta}_{uk_xk_y}\widehat{\eta}_{uk_xk_y}\right\rangle = 2d_1\left\langle \widehat{\eta}_{uk_xk_y}\widehat{\eta}_{uk_xk_y}\right\rangle + 2b_0\left\langle \widehat{\eta}_{uk_xk_y}\widehat{\eta}_{vk_xk_y}\right\rangle + v_1\,, \tag{1.47}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left\langle \widehat{\eta}_{uk_xk_y}\widehat{\eta}_{vk_xk_y}\right\rangle = c_0\left\langle \widehat{\eta}_{uk_xk_y}\widehat{\eta}_{uk_xk_y}\right\rangle + (d_1+d_2)\left\langle \widehat{\eta}_{uk_xk_y}\widehat{\eta}_{vk_xk_y}\right\rangle + b_0\left\langle \widehat{\eta}_{vk_xk_y}\widehat{\eta}_{vk_xk_y}\right\rangle + v_2\,, \tag{1.48}$$

$$\frac{\mathrm{d}}{\mathrm{d}t}\left\langle \widehat{\eta}_{uk_xk_y}\widehat{\eta}_{uk_xk_y}\right\rangle = 2d_2\left\langle \widehat{\eta}_{vk_xk_y}\widehat{\eta}_{vk_xk_y}\right\rangle + 2c_0\left\langle \widehat{\eta}_{uk_xk_y}\widehat{\eta}_{vk_xk_y}\right\rangle + v_3\,, \tag{1.49}$$

where the coefficients are

$$d_1 = 2\lambda_1^u(\cos(k_x\kappa h)-1) + 2\lambda_2^u(\cos(k_y h)-1) + 4\lambda_3^u(\cos(k_x\kappa h)\cos(k_y h)-1) + f_\phi, \tag{1.50}$$

$$d_2 = 2\lambda_1^v(\cos(k_x\kappa h)-1) + 2\lambda_2^v(\cos(k_y h)-1) + 4\lambda_3^v(\cos(k_x\kappa h)\cos(k_y h)-1) + g_\psi, \tag{1.51}$$

$$b_0 = g_\phi, \tag{1.52}$$

$$c_0 = f_\psi, \tag{1.53}$$

$$v_1 = \frac{IA^2}{2}\left(\lambda_0^u\bar{u} - 2\lambda_1^u\bar{u}\cos(k_x\kappa h) - 2\lambda_3^u\bar{u}\cos(k_y h) - 4\lambda_2^u\bar{u}\cos(k_x\kappa h)\cos(k_y h) + f(\boldsymbol{\phi},\boldsymbol{\psi})\right), \tag{1.54}$$

$$v_2 = \frac{IA^2}{2}\tilde{h}(\boldsymbol{\phi},\boldsymbol{\psi}), \tag{1.55}$$

$$v_3 = \frac{IA^2}{2}\left(\lambda_0^v\bar{v} - 2\lambda_1^v\bar{v}\cos(k_x\kappa h) - 2\lambda_3^v\bar{v}\cos(k_y h) - 4\lambda_2^v\bar{v}\cos(k_x\kappa h)\cos(k_y h) + g(\boldsymbol{\phi},\boldsymbol{\psi})\right). \tag{1.56}$$

The functions f and g in this section have the same form as in Equations (1.10) and (1.11), while the function $\tilde{h}(\boldsymbol{\phi},\boldsymbol{\psi})$ is composed of terms present in both functions f and g, *i.e.* the terms that represent reactions involving both species. We also assume the domain, Ω, is a

square that is discretised by rectangles. As such, due to symmetries of this discretisation we can denote the rates of diffusive jumps in the x-direction by λ_1^u, diagonal jumps by λ_2^u, and jumps in the y-direction by λ_3^u for species U and likewise for species V.

Equation (1.47), Equation (1.48) and Equation (1.49) form a system of linear ODEs and, as a result, can be written

$$\dot{\mathbf{X}} = M\mathbf{X} + V, \tag{1.57}$$

where $\mathbf{X} = \left(\left\langle \widehat{\eta}_{uk_xk_y}\widehat{\eta}_{uk_xk_y}\right\rangle, \left\langle \widehat{\eta}_{uk_xk_y}\widehat{\eta}_{vk_xk_y}\right\rangle, \left\langle \widehat{\eta}_{vk_xk_y}\widehat{\eta}_{vk_xk_y}\right\rangle\right)^T$, $V = (v_1, v_2, v_3)^T$ and

$$M = \begin{bmatrix} 2d_1 & 2b_0 & 0 \\ c_0 & (d_1+d_2) & b_0 \\ 0 & c_0 & 2d_2 \end{bmatrix}. \tag{1.58}$$

Therefore, Equation (1.57) can be solved in the same manner as the system of ODEs in Equation (1.17) and we have the following eigenvalues

$$e_0 = d_1 + d_2, \tag{1.59}$$

$$e_\pm = e_0 \pm \sqrt{4b_0c_0 + (d_1 - d_2)^2}. \tag{1.60}$$

For spatial patterns to form, we require at least one of the eigenvalues to have a positive real part. However, since $\cos(\cdot) - 1 \leq 0$ and the condition for a Turing instability is $f_\phi + g_\psi < 0$, as stated in Section 1.3.1, then $d_1 + d_2 \leq 0$. As a result, $\mathrm{Re}(e_0) \leq 0$ and $\mathrm{Re}(e_-) \leq 0$. Therefore, the only way for the patterns to form is if $\mathrm{Re}(e_+) \geq 0$, and so the condition for patterns to form is

$$d_1d_2 - b_0c_0 < 0\,. \tag{1.61}$$

1.4 Thesis structure

Having covered the basic details of the RDME and pattern analysis, we delve into the intricacies of stochastic modelling using the RDME framework. As we already mentioned, there exists a choice in defining jumping rates between compartments and understanding the impact of this choice on the simulation results has not to-date been explored. In this thesis we aim to understand the impact of this modelling choice on characteristic models of RD systems.

In Chapter 2, we focus first on the different methods to derive the diffusion jump rates and their effects on model predictions on a static domain discretised by a Cartesian mesh. We show that using different descriptions of diffusion can have significant effects. We analyse how the different implementations of diffusion affect simulation results in simple systems and systems describing pattern formation. In the case of pattern formation, we show that the model can predict very different patterns given different implementations of diffusion.

Next, in Chapter 3, we evaluate the effects of incorporating domain growth in our modelling framework when the domain is discretised by a Cartesian mesh. Some RD systems require domain growth to be incorporated, for example, to allow for modelling of tissue growth. In RD systems that model pattern formation, it is important that growth takes place at the same time as patterning. Hence, we incorporate domain growth into the model presented in Chapter 2 and we assess how different descriptions of diffusion affect model predictions on growing domains. We also look at how different types of growth affect model predictions.

In Chapter 4 we explore the effects of using complex meshes to discretise the domain have on model predictions. Using complex meshes to discretise the domain allows us to

model RD systems where complex geometry needs to be considered, such as patterning in ligaments of arcoid bivalves [43]. To consider complex meshes, we introduce Delauney triangulations and Voronoi tessellations and we generalise the derivations of the jump rates to this type of unstructured meshes. After we show the derivations of the jump rates, we investigate the effects of using a different discretisation of the domain compared to Chapters 2 and 3.

Following on from that, we outline the software packages we developed in Chapter 5. First, we describe the software package *StoSpa2*, which is used for simulations within RDME framework. Second, we outline the Python software package *Dualmesh*, which we developed for generating a specific kind of mesh in Chapter 4.

Lastly, we discuss all of the results and contributions to the field in Chapter 6. We also discuss the limitations of our approaches and possible next steps that can be taken.

Chapter 2

Simulation of stochastic reaction-diffusion models on static domains

Stochastic models capture the indisputable uncertainty of natural systems. As discussed in the previous chapter, the RDME is one of the most popular mesoscopic frameworks and is one that we focus on in this thesis. To model diffusion, the RDME framework requires computation of the rate per unit time of a molecule to jump from one compartment to a neighbouring one. Previous research in this direction includes the work by Engblom *et al.* [19], where the finite element method is used to calculate the diffusive jump rates, and that of Hellander and Petzold [30] who developed a description of diffusion for more general compartment shapes. In addition, Meinecke and Lötstedt explored the impact of different methods of diffusive jump rate derivation upon simulation of diffusive systems [39, 47, 48].

However, there has been no exploration, to-date, of how the choice of derivation of diffusive jump rates, nor the implementation parameters within these derivations affects

the predictions of models when reactions are also present. As such, our aim in this chapter is to investigate the effects of different methods of derivation of diffusive jump rates on stochastic RD systems modelled using the RDME. To do so, we replicate, extend and apply the results of Meinecke and Lötstedt [48] to a number of RD systems, that include a range of reaction types, on a static domain. We first outline the derivations of the diffusive jump rates in Section 2.1. Next, in Section 2.2, we study a number of RD systems to determine the effects of different jump rates on simulation results.

2.1 The jump rates

In this chapter, for simplicity we consider the evolution of a RD system in the square domain $\Omega = [0, L\,m]^2$, and partition Ω using a Cartesian mesh into non-overlapping rectangular sub-domains C_i, called compartments, such that $\Omega = \bigcup_{i=1}^{I} C_i$ (Figure 2.1) [48]. Each compartment is of size $\kappa h \times h$, where κ is the aspect ratio of a compartment, and the total number of compartments, I, can be expressed as $I = n_x \times n_y$, where n_x and n_y are the numbers of compartments in the x-direction and y-direction, respectively. We number the compartments starting at the bottom-left corner of the domain and then from left to right, so that, for site i, the adjacent compartments have indices $i \pm 1$ in the x-direction and indices $i \pm n_x$ in the y-direction.

When using the CME and following law of mass action, we only have a single choice of rate for the reactions. However, in the RDME we have multiple choices for the rates of molecules jumping between adjacent compartments, *i.e.* jump rates, and it is important to see how they compare in terms of model predictions. In this section we use different methods to derive the jump rates, following the work by Meinecke and Lötstedt [48]. We use the finite difference method (FDM), finite element method (FEM), and finite

volume method (FVM) numerical schemes, as well as a method developed by Meinecke and Lötstedt called the *first exit time method* (FET). After we outline the derivations, we compare the impact of different choices of the jump rates for a range of RD models.

2.1.1 Derivations

We first consider a diffusion-only system with a single molecular species, U. As already outlined in Section 1.2, diffusion is modelled as a series of jumps between neighbouring compartments where the propensity function for a jump from compartment i to compartment j is $d_{i,j}(\boldsymbol{u}(t)) = \lambda_{i,j} u_i(t)$. Note that we have dropped the superscript ℓ for clarity of notation since we are currently considering only a single species. In this work we will assume that $\lambda_{i,j} = 0$, $i = 1, \ldots, I$, unless j is one of the eight nearest neighbour compartments of i (indices $i \pm 1$, $i \pm n_x$ and $i \pm (n_x \pm 1)$), as depicted in Figure 2.1. In this section we outline four methods for derivation of the $\lambda_{i,j}$.

2.1.1.1 Diffusion of a single molecule

Consider a single molecule diffusing within the domain $\Omega = [0, L\,m]^2$ with diffusion coefficient D. Then evolution of the position, $\mathbf{x}(t) = (x(t), y(t))$, of this molecule can be described using the following SDEs [20]:

$$x(t + \mathrm{d}t) = x(t) + \sqrt{2D}\mathrm{d}W_x, \tag{2.1}$$

$$y(t + \mathrm{d}t) = y(t) + \sqrt{2D}\mathrm{d}W_y, \tag{2.2}$$

where W_x and W_y are the usual Wiener processes. The associated Fokker-Planck equation (FPE) is

$$\frac{\partial p(\mathbf{x}, t)}{\partial t} = D\nabla^2 p(\mathbf{x}, t) \quad \text{for} \quad \mathbf{x} \in \Omega, \tag{2.3}$$

so that $p(\mathbf{x},t)\mathrm{d}\mathbf{x}$ is the probability of finding the molecule in the region $\mathrm{d}\mathbf{x} = [x, x+\mathrm{d}x) \times [y, y+\mathrm{d}y)$ at time t, where the scalar quantity $\mathrm{d}\mathbf{x}$ denotes the area of an infinitesimal region in the domain Ω.

For simplicity, throughout this work we will use Neumann boundary conditions

$$(\mathbf{n}\cdot\nabla)\,p = 0 \quad \text{for} \quad \mathbf{x}\in\partial\Omega, \tag{2.4}$$

and we take the initial distribution for the molecule to be

$$p(\mathbf{x},0) = p_0(\mathbf{x}). \tag{2.5}$$

Using the Cartesian mesh shown in Figure 2.1, we discretise the Laplacian as follows:

$$D\nabla^2 p(\mathbf{x}_i,t) \approx \sum_{j=1}^{I} \lambda_{i,j}\,\tilde{p}_j(t) - \sum_{j=1}^{I} \lambda_{j,i}\,\tilde{p}_i(t), \tag{2.6}$$

where $\tilde{p}_i(t)$ is the approximate solution to Equation (2.3) at point $\mathbf{x}_i$, *i.e.* at the centre of compartment i at time t. The compartments are all of the same size, $\kappa h \times h$, with area $A = \kappa h^2$ (Figure 2.1). Hence, exploiting translational symmetry and noting that the diffusion coefficient, D, is spatially homogeneous, we refer to the jump rates, $\lambda_{i,j}$, from now on as λ_n for $n \in \{1,2,\ldots,8\}$ (Figure 2.1), with

$$\lambda_0 = \sum_{n=1}^{8} \lambda_n. \tag{2.7}$$

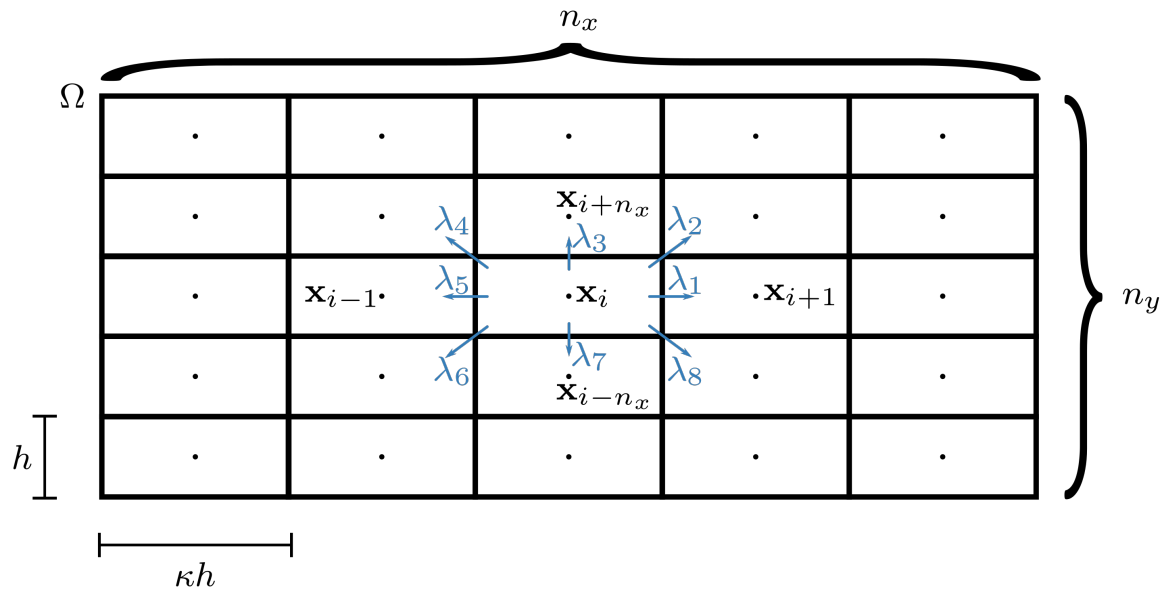


Figure 2.1: Schematic diagram of the discretisation of the two-dimensional domain Ω, with the dimensions of the compartments and the indexing of the jump rates, λ_n, for $n \in \{1, \ldots, 8\}$ for a given compartment i shown.

We now make a few remarks. First, note that this description allows for jumps between diagonally neighbouring compartments. Second, the indexing of the jump rates starts at one for a jump to the right and continues in an anti-clockwise fashion, as shown in Figure 2.1. Third, due to the reflective symmetries of a Cartesian mesh and since D is spatially homogeneous, we have the following identities

$$\lambda_1 = \lambda_5\,, \quad \lambda_3 = \lambda_7\,, \quad \lambda_2 = \lambda_4 = \lambda_6 = \lambda_8. \tag{2.8}$$

Finally, note that in what follows we present derivations of the jump rates for a two-dimensional domain but that all derivations naturally extend to the three-dimensional case.

2.1.1.2 The finite difference method

The first method we consider to derive the diffusive jump rates is the FDM. Meinecke and Lötsedt [48] chose a 9-point stencil which allows one to include a parameter α to study the effects of diagonal jumps, while maintaining a discretisation of the Laplacian that is still second order accurate [61]. The standard 5-point stencil, that is typically used to discretise the Laplacian with the FDM, can be recovered by setting $\alpha = 0$, and hence it neglects diagonal jumps. An illustration of both the 5-point stencil and the 9-point stencil can be seen in Figure 2.2.

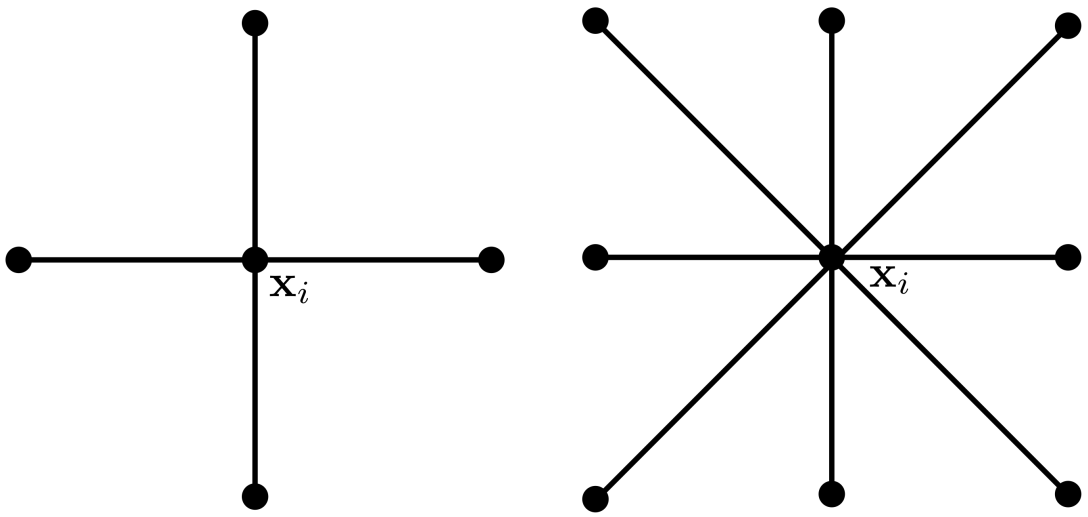


Figure 2.2: On the left we have the standard 5-point stencil and on the right we have the 9-point stencil used in the derivation of the FDM jump rates.

We have

$$\nabla^2 p\left(\mathbf{x}_i, t\right) \approx \delta_x^2 \tilde{p}_i + \delta_y^2 \tilde{p}_i + \frac{1}{2}\alpha\kappa h^2 \delta_x^2 \delta_y^2 \tilde{p}_i, \tag{2.9}$$

where α is a free parameter controlling the frequency of diagonal jumps and κ is the

aspect ratio of a compartment, and the difference operators are defined as follows:

$$\begin{aligned}\delta_x^2 \tilde{p}_i &= \frac{1}{\kappa^2 h^2}\left(\tilde{p}_{i+1} + \tilde{p}_{i-1} - 2\tilde{p}_i\right); \\ \delta_y^2 \tilde{p}_i &= \frac{1}{h^2}\left(\tilde{p}_{i+n_x} + \tilde{p}_{i-n_x} - 2\tilde{p}_i\right). \end{aligned} \tag{2.10}$$

As such, the right-hand side of Equation (2.3) can be approximated as

$$\begin{aligned} D\nabla^2 p \approx\ & \frac{D - D\alpha\kappa}{\kappa^2 h^2}\left(\tilde{p}_{i+1} + \tilde{p}_{i-1}\right) + \frac{D\kappa - D\alpha}{\kappa h^2}\left(\tilde{p}_{i+n_x} + \tilde{p}_{i-n_x}\right) \\ & + \frac{D\alpha}{2\kappa h^2}\left(\tilde{p}_{i+n_x+1} + \tilde{p}_{i-n_x+1} + \tilde{p}_{i+n_x-1} + \tilde{p}_{i-n_x-1}\right) \\ & - 2D\frac{\kappa^2 - \alpha\kappa + 1}{\kappa^2 h^2}\tilde{p}_i \,. \end{aligned} \tag{2.11}$$

Comparing the general expression for the discretised Laplacian in Equation (2.6) with the discretised Laplacian in Equation (2.11) we have

$$\lambda_1 = \frac{D - D\alpha\kappa}{\kappa^2 h^2}, \tag{2.12}$$

$$\lambda_2 = \frac{D\alpha}{2\kappa h^2}, \tag{2.13}$$

$$\lambda_3 = \frac{D\kappa - D\alpha}{\kappa h^2}, \tag{2.14}$$

$$\lambda_0 = \frac{2D(\kappa^2 - \alpha\kappa + 1)}{\kappa^2 h^2}. \tag{2.15}$$

We note that to have non-negative jump rates, we have the following bound on κ for a given a value of α, as stated in [48]:

$$0 \leq \alpha \leq \min\left(\kappa, \frac{1}{\kappa}\right) \leq 1\,. \tag{2.16}$$

2.1.1.3 The finite element method

The second numerical method we will use to derive the jump coefficients is the FEM. We multiply Equation (2.3) by a test function $w(\mathbf{x})$, which is a function in Sobolev space [18] (*i.e.* is bounded and has bounded first partial derivatives), and integrate over the whole domain Ω:

$$\begin{aligned} \int_\Omega \frac{\partial p}{\partial t} v \,\mathrm{d}x\,\mathrm{d}y &= D\int_\Omega w\nabla^2 p \,\mathrm{d}x\,\mathrm{d}y \\ &= D\int_\Omega \left[\nabla\cdot(w\nabla p) - \nabla p\cdot\nabla w\right] \,\mathrm{d}x\,\mathrm{d}y. \end{aligned} \tag{2.17}$$

Now we use the divergence theorem to give

$$\int_\Omega \frac{\partial p}{\partial t} w \,\mathrm{d}x\,\mathrm{d}y = D\int_{\partial\Omega} \mathbf{n}\cdot[w\nabla p]\,\mathrm{d}s - D\int_\Omega \nabla p\cdot\nabla w \,\mathrm{d}x\,\mathrm{d}y, \tag{2.18}$$

where $\mathrm{d}s$ is the line element of the domain boundary, $\partial\Omega$, and $\mathbf{n}$ is the normal vector to the domain boundary at the point $\mathbf{x}$. Due to the Neumann boundary conditions, the first term in Equation (2.18) vanishes and we are left with

$$\int_\Omega \frac{\partial p}{\partial t} w \,\mathrm{d}x\,\mathrm{d}y = -D\int_\Omega \nabla p\cdot\nabla w \,\mathrm{d}x\,\mathrm{d}y. \tag{2.19}$$

We express the functions p, $\partial p/\partial t$ and v in terms of the basis functions φ_j,

$$w(x,y) = \varphi_j(x,y), \tag{2.20}$$

$$p(x,y,t) \approx \sum_{i=1}^{I} \tilde{p}_i(t)\varphi_i(x,y), \tag{2.21}$$

$$\frac{\partial p}{\partial t} \approx \sum_{i=1}^{I} \frac{\mathrm{d}\tilde{p}_i}{\mathrm{d}t}(t)\varphi_i(x,y), \tag{2.22}$$

where the φ_i are defined as

$$\varphi_i(x,y) = \begin{cases} \left(1 - \dfrac{x - x_i}{\kappa h}\right)\left(1 - \dfrac{y - y_i}{h}\right) & \text{for} \quad \mathbf{x} \in [x_i, x_i + \kappa h] \times [y_i, y_i + h], \\ \left(1 + \dfrac{x - x_i}{\kappa h}\right)\left(1 - \dfrac{y - y_i}{h}\right) & \text{for} \quad \mathbf{x} \in [x_i - \kappa h, x_i] \times [y_i, y_i + h], \\ \left(1 + \dfrac{x - x_i}{\kappa h}\right)\left(1 + \dfrac{y - y_i}{h}\right) & \text{for} \quad \mathbf{x} \in [x_i - \kappa h, x_i] \times [y_i - h, y_i], \\ \left(1 - \dfrac{x - x_i}{\kappa h}\right)\left(1 + \dfrac{y - y_i}{h}\right) & \text{for} \quad \mathbf{x} \in [x_i, x_i + \kappa h] \times [y_i - h, y_i], \\ 0 & \text{otherwise,} \end{cases} \tag{2.23}$$

for $i \in \{1, \ldots, I\}$, and $\varphi_j(\mathbf{x}) = 0$ for $\mathbf{x} \notin \Omega$. We choose linear basis functions for simplicity, however, any form will suffice, as long as the following condition is met:

$$\varphi_j(x_i, y_i) = \begin{cases} 1 & \text{if} \quad i = j; \\ 0 & \text{if} \quad i \neq j. \end{cases} \tag{2.24}$$

Substituting Equations (2.20), (2.21) and (2.22) into Equation (2.19) gives, for each j,

$$\sum_{i=1}^{I} \left(\int_\Omega \varphi_i \varphi_j \, \mathrm{d}x \, \mathrm{d}y \right) \frac{\mathrm{d}\tilde{p}_i}{\mathrm{d}t} = -\sum_{i=1}^{I} \left(\int_\Omega D \nabla \varphi_i \cdot \nabla \varphi_j \, \mathrm{d}x \, \mathrm{d}y \right) \tilde{p}_i, \tag{2.25}$$

which can be expressed as a matrix equation

$$M \frac{\mathrm{d}\tilde{\mathbf{p}}}{\mathrm{d}t} = -N\tilde{\mathbf{p}}\,, \tag{2.26}$$

where

$$\tilde{\mathbf{p}} = (\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_I), \tag{2.27}$$

$$M_{ij} = \int_\Omega \varphi_i \, \varphi_j \, \mathrm{d}x \, \mathrm{d}y, \tag{2.28}$$

$$N_{ij} = \int_\Omega D \nabla \varphi_i \cdot \nabla \varphi_j \, \mathrm{d}x \, \mathrm{d}y. \tag{2.29}$$

Evaluating the elements of the matrices M and N, and using the lumped mass matrix approach (see Appendix A.1), we finally arrive at the following expressions for the jump coefficients:

$$\lambda_1 = \frac{D(2-\kappa^2)}{3\kappa^2 h^2}; \tag{2.30}$$

$$\lambda_2 = \frac{D(\kappa^2+1)}{6\kappa^2 h^2}; \tag{2.31}$$

$$\lambda_3 = \frac{D(2\kappa^2-1)}{3\kappa^2 h^2}; \tag{2.32}$$

$$\lambda_0 = \frac{4D(\kappa^2+1)}{3\kappa^2 h^2}. \tag{2.33}$$

Also, we note that for the FEM-derived jump rates to be non-negative, we have the following bound on κ, as stated in [48]:

$$\frac{1}{\sqrt{2}} \leq \kappa \leq \sqrt{2}\,. \tag{2.34}$$

2.1.1.4 The finite volume method

Finally, we consider derivation of the jump rates using the FVM. We start by averaging the Laplacian at $\mathbf{x}_i$ by integrating over compartment i:

$$D\nabla^2 p(\mathbf{x}_i) \approx \frac{D}{A} \int_{C_i} \nabla^2 p \, \mathrm{d}\Omega, \tag{2.35}$$

where A is the area of the compartment i. Using the divergence theorem, we rewrite Equation (2.35) as

$$D\nabla^2 p(\mathbf{x}_i) \approx \frac{D}{A} \int_{\partial C_i} (\mathbf{n} \cdot \nabla)\, p \, \mathrm{d}s, \tag{2.36}$$

where $\mathrm{d}s$ is a line element of the compartment boundary, ∂C_i, and $\mathbf{n}$ is the normal vector to the compartment boundary at the point $\mathbf{x}$.

Next, we split the integral in Equation (2.36) into integrals along segments of the compartment boundary ∂C_i, as shown in Figure 2.3:

$$D\nabla^2 p(\mathbf{x}_i) \approx \frac{D}{A} \sum_{k=1}^{4} \int_{\partial C_i^k} (\mathbf{n} \cdot \nabla)\, p \, \mathrm{d}s, \tag{2.37}$$

and we approximate the gradient of p across the different parts of the boundary of the compartment using finite differences:

$$\mathbf{n} \cdot \nabla p \Big|_{\partial C_i^1} \approx \frac{\tilde{p}_{i+1} - \tilde{p}_i}{\kappa h}; \tag{2.38}$$

$$\mathbf{n} \cdot \nabla p \Big|_{\partial C_i^2} \approx \frac{\tilde{p}_{i+n_x} - \tilde{p}_i}{h}; \tag{2.39}$$

$$\mathbf{n} \cdot \nabla p \Big|_{\partial C_i^3} \approx \frac{\tilde{p}_{i-1} - \tilde{p}_i}{\kappa h}; \tag{2.40}$$

$$\mathbf{n} \cdot \nabla p \Big|_{\partial C_i^4} \approx \frac{\tilde{p}_{i-n_x} - \tilde{p}_i}{h}. \tag{2.41}$$

Substituting Equations (2.38)–(2.41) into Equation (2.37) gives

$$\lambda_1 = \frac{D}{\kappa^2 h^2}, \tag{2.42}$$

$$\lambda_2 = 0, \tag{2.43}$$

$$\lambda_3 = \frac{D}{h^2}, \tag{2.44}$$

$$\lambda_0 = \frac{2D(\kappa^2 + 1)}{\kappa^2 h^2}. \tag{2.45}$$

Jump rates	Finite Difference Method (FDM)	Finite Element Method (FEM)	Finite Volume Method (FVM)
λ_1	$\frac{D(1-\alpha\kappa)}{\kappa^2 h^2}$	$\frac{D(2-\kappa^2)}{3\kappa^2 h^2}$	$\frac{D}{\kappa^2 h^2}$
λ_2	$\frac{D\alpha}{2\kappa h^2}$	$\frac{D(\kappa^2+1)}{6\kappa^2 h^2}$	0
λ_3	$\frac{D(\kappa-\alpha)}{\kappa h^2}$	$\frac{D(2\kappa^2-1)}{3\kappa^2 h^2}$	$\frac{D}{h^2}$

Table 2.1: Comparison of jump rates arising from different numerical derivations.

Table 2.1 shows the expressions for all the jump rates we have derived thus far.

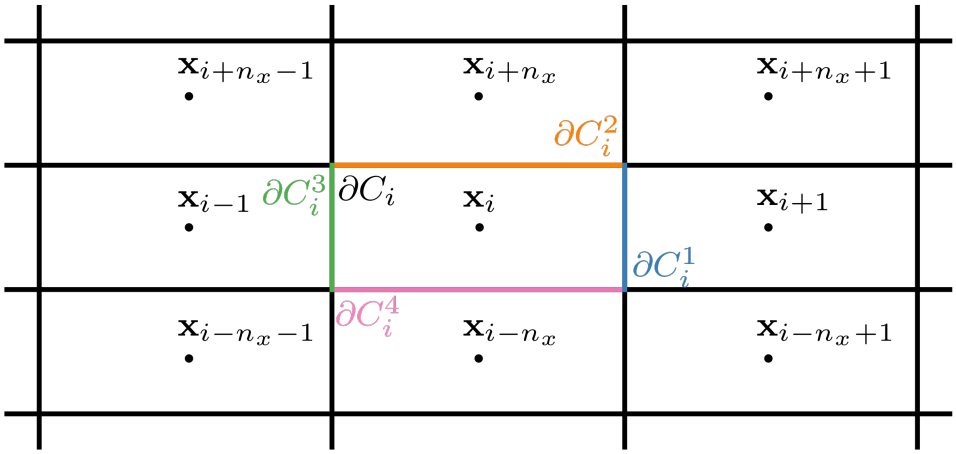


Figure 2.3: Schematic diagram of how compartment i boundary, ∂C_i, is split into four segments ∂C_i^k for $k = 1, 2, 3, 4$.

2.1.1.5 The first exit time method

Meinecke and Löstedt proposed an additional method of calculating the jump rates [39, 48]. This method involves considering the FET distribution from a domain ω_i (centered

on C_i) of a single molecule undergoing Brownian motion that is placed initially at the centre of C_i. Note that to model diffusion in a RDME framework using FET-derived jump rates we approximate the probability distribution function (PDF) for the FET by an exponential distribution with the same mean. We follow the choice of domain ω_i from [48], where ω_i is defined by its boundary, $\partial\omega_i$, and taken to be a rectangle that intersects the centres of neighbouring compartments of compartment i, as shown in Figure 2.4, so that ω_i completely contains the compartment i. The choice of ω_i has been briefly discussed in [39], where different shapes are considered, such as a circle or a rhombus. The conclusion of [39] was that the choice of ω_i used in this work is the most accurate.

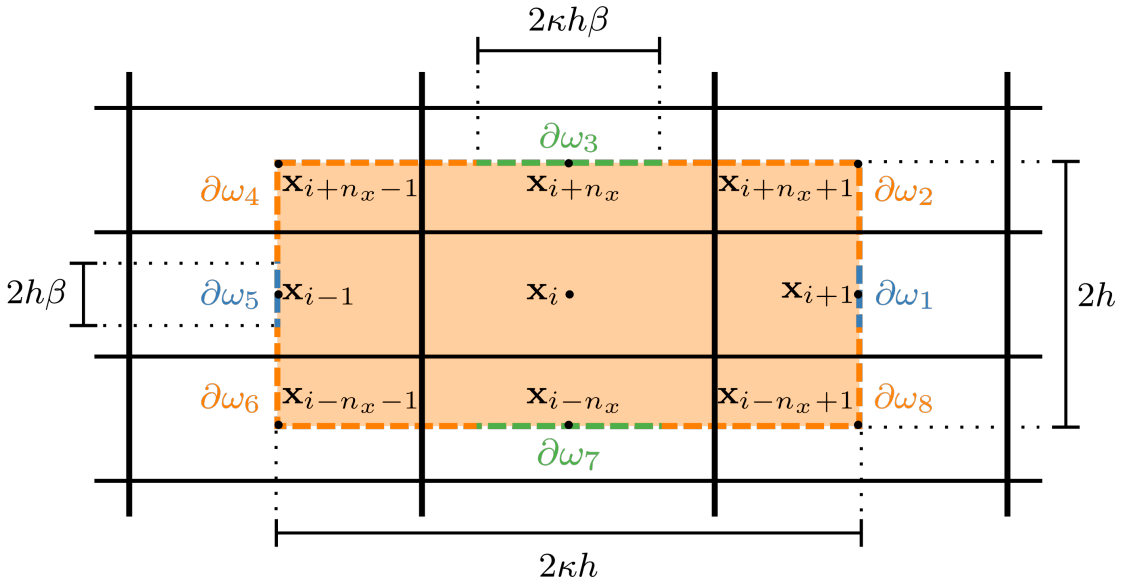


Figure 2.4: Diagram of the domain ω_i, which has been shaded in orange. The boundary of the domain ω_i, indicated by the dashed line, is split into eight sections coloured as follows: $\partial\omega_1$ is the rightmost blue; $\partial\omega_2$ is the upper-right orange; $\partial\omega_3$ is the uppermost green; $\partial\omega_4$ is the upper-left orange; $\partial\omega_5$ is the leftmost blue; $\partial\omega_6$ is the lower-left orange blue; $\partial\omega_7$ is the lower green; and $\partial\omega_8$ is the lower-right orange.

We follow Meinecke and Löstedt [48] in choosing jump rates of the form

$$\lambda_n = \lambda_0\,\theta_n, \tag{2.46}$$

where θ_n is the probability that a molecule exits ω_i into the nth neighbouring compartment (see Figure 2.1). We define a molecule to exit ω_i into the nth neighbouring compartment if it exits through the segment of the domain boundary denoted by $\partial\omega_i^n$ in Figure 2.4. There is flexibility in the choice of discretisation of the boundary $\partial\omega_i$ into segments $\partial\omega_i^n$, $n = 1, \ldots, 8$, and we define the discretisation using parameter β as follows:

$$\begin{aligned} |\partial\omega_1| &= |\partial\omega_5| = 2\beta h; \\ |\partial\omega_3| &= |\partial\omega_1| = 2\beta\kappa h; \\ |\partial\omega_2| &= |\partial\omega_4| = |\partial\omega_6| = |\partial\omega_8| = (1-\beta)(1+\kappa)h. \end{aligned} \tag{2.47}$$

We will explore the effect of changing the parameter β on the jump rates in Section 2.1.2.

We proceed by considering two random variables associated with a molecule governed by Brownian motion exiting from the domain ω_i: τ and η. The random variable τ denotes the FET out of ω_i, whereas η denotes the exit segment. We use τ to calculate λ_0 by, as explained earlier, equating the mean of the PDF of τ with the rate parameter of the exponential distribution used in Algorithm 1. The random variable η, on the other hand, is used to calculate θ_n.

We calculate the PDF of τ using the FPE (as outlined in Section 2.1.1) that describes the probability for the molecule to be at position $\mathbf{x}$ at time t:

$$\frac{\partial p}{\partial t} = D\nabla^2 p \quad \text{for} \quad \mathbf{x} \in \omega_i, \tag{2.48}$$

where D is the diffusion coefficient. To calculate the PDF of the FET, we use Dirichlet (absorbing) boundary conditions

$$p(\mathbf{x}, t) = 0 \quad \text{for} \quad \mathbf{x} \in \partial\omega_i, \tag{2.49}$$

and assume that initially the molecule at the centre of ω_i so that

$$p(\mathbf{x}, 0) = \delta(\mathbf{x} - \mathbf{x}_i). \tag{2.50}$$

Using $p(\mathbf{x}, t)$ we calculate the survival probability, $S(t)$, which is the probability that a molecule remains in the domain ω_i at least until time t, as [56]

$$S(t) = \mathbb{P}(\tau \geqslant t) = \int_{\omega_i} p(\mathbf{x}, t)\, \mathrm{d}\omega, \tag{2.51}$$

where $\mathrm{d}\omega$ is a domain element of the domain ω_i. Now, given $S(t)$, we derive the probability that a molecule exits ω_i at time t, *i.e.* the PDF of the random variable τ, denoted by $\mathbb{P}(\tau = t)$, as

$$\mathbb{P}(\tau = t) = -\frac{\partial S}{\partial t}\,. \tag{2.52}$$

Substituting Equation (2.51) into Equation (2.52) results in

$$\begin{aligned} \mathbb{P}(\tau = t) &= -\int_{\omega_i} \frac{\partial p}{\partial t}\, \mathrm{d}\omega \\ &= -D \int_{\omega_i} \nabla^2 p\, \mathrm{d}\omega \\ &= -D \int_{\partial\omega_i} \mathbf{n} \cdot \nabla p\, \mathrm{d}s, \end{aligned} \tag{2.53}$$

where $\mathrm{d}s$ is a line element of $\partial\omega_i$. Finally, we calculate the mean of the PDF for τ as

$$E_\tau = \mathbb{E}\left[\tau\right] = \int_0^\infty t\, \mathbb{P}(\tau = t)\, \mathrm{d}t = \int_0^\infty S(t)\, \mathrm{d}t, \tag{2.54}$$

and set

$$\lambda_0 = \frac{1}{E_\tau}. \tag{2.55}$$

To calculate the PDF for η, we use the law of total probability

$$\mathbb{P}(\eta = n) = \int_0^\infty \mathbb{P}(\eta = n|\tau = t)\mathbb{P}(\tau = t)\,\mathrm{d}t. \tag{2.56}$$

Following [48], we have the probability of a molecule exiting through segment $\partial\omega_i^n$ given that a molecule exits at time t, *i.e.* the probability $\mathbb{P}(\eta = n|\tau = t)$, as follows

$$\mathbb{P}(\eta = n|\tau = t) = \frac{\int_{\partial\omega_i^n} \mathbf{n}\cdot\nabla p\,\mathrm{d}s}{\int_{\partial\omega_i} \mathbf{n}\cdot\nabla p\,\mathrm{d}s}. \tag{2.57}$$

Substituting Equation (2.53) and Equation (2.57) into Equation (2.56), we have

$$\begin{aligned}
\theta_n &= \mathbb{P}(\eta = n) \\
&= \int_0^\infty \frac{\int_{\partial\omega_i^n} \mathbf{n}\cdot\nabla p\,\mathrm{d}s}{\int_{\partial\omega_i} \mathbf{n}\cdot\nabla p\mathrm{d}s}\left(-D\int_{\partial\omega_i} \mathbf{n}\cdot\nabla p\,\mathrm{d}s\right)\mathrm{d}t \\
&= -D\int_0^\infty\int_{\partial\omega_i^n} \mathbf{n}\cdot\nabla p\,\mathrm{d}s\,\mathrm{d}t.
\end{aligned} \tag{2.58}$$

Therefore, given solutions to Equation (2.46) and Equation (2.48), we can calculate the jump rates λ_n for $n = 1, \ldots, 8$. To see the full expressions for the jump coefficients derived using the FET method see Appendix A.2.

2.1.2 Comparison between methods

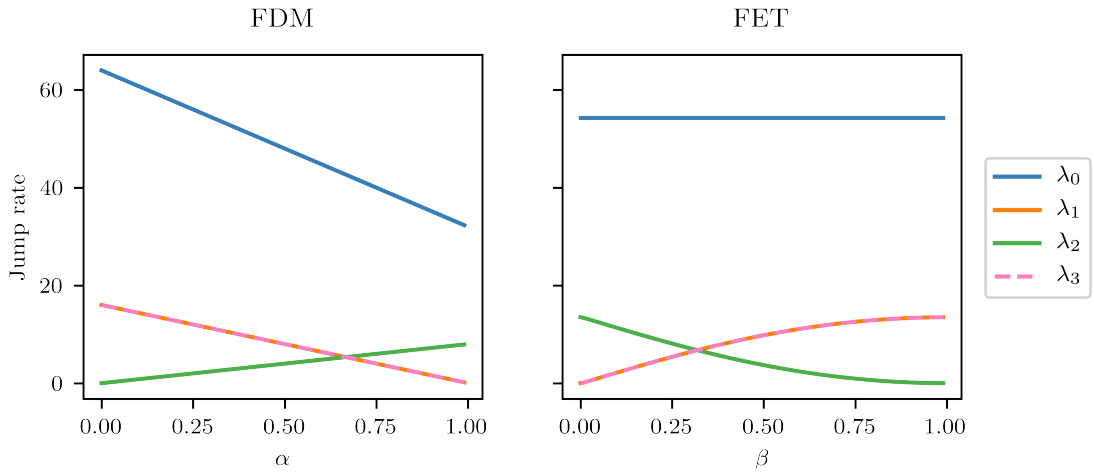


Figure 2.5: Plot of the jump rates for the FDM and FET methods of derivation as a function of the implementation parameters α and β, respectively, for $h = 0.25\,m$, $\kappa = 1.0$ and $D = 1.0\,m^2 s^{-1}$.

Figure 2.5 shows how the FDM and FET methods of derivation compare to one another as the parameters α and β, respectively, are varied. Both α and β control the frequency of diagonal jumps. For the FDM-derived jump rates, increasing the parameter α increases λ_2, the rate of diagonal jumps. On the other hand, increasing β has the opposite effect on λ_2 in the FET method case.

To compare all the methods of derivation of the jump rates, we plot them in Figure 2.6. When $\alpha = 1.0$ for the FDM, the total jump rate, λ_0, is the lowest of all the methods and only diagonal jumps can happen, as evidenced by only λ_2 being non-zero. We also see in Figure 2.6 that the total jump rate derived using the FVM is the highest of all the methods and only has non-diagonal jumps. The jump rates derived using the FEM differ from the other methods, by having jump rates in all directions equal to each other. Lastly, the FET method, when $\beta = 0.5$, has a high total jump rate, much like the FVM, but unlike the FVM it has non-zero diagonal jump rate. We choose the value of $\alpha = 1.0$ as this returns the regime the furthest away from the FVM, and we choose $\beta = 0.5$ as the

resulting jump rates are similar to the ones derived using both the FEM and FVM.

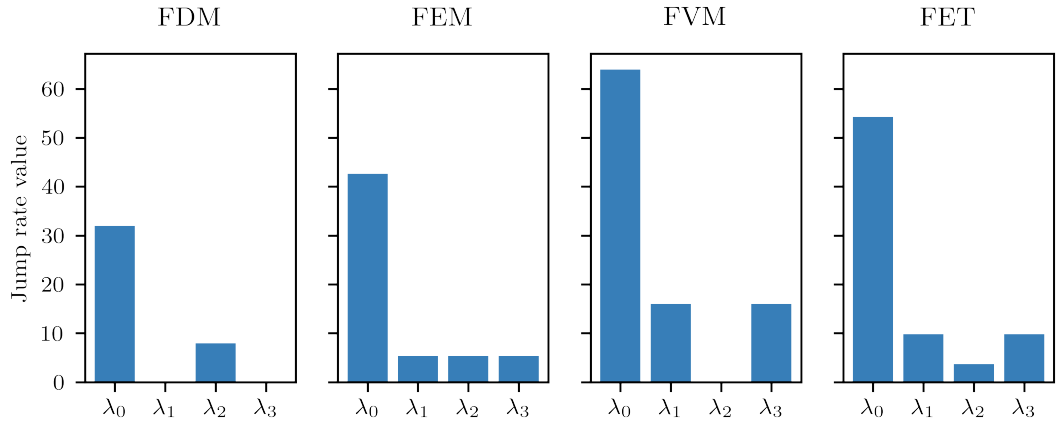


Figure 2.6: Comparison of the jump rates for all four methods of derivation for $h = 0.25\,m$, $\kappa = 1.0$ and $D = 1.0\,m^2s^{-1}$. For the FDM-derived jump rates (left-most plot) we used $\alpha = 1.0$, while for the FET-derived jump rates we used $\beta = 0.5$.

Furthermore, Figure 2.7 shows that by varying the value of the derivation parameter β, we can obtain similar distributions of jump rates to other methods of derivation. When $\beta = 0.0$ only λ_2 is non-zero, which means that there are only diagonal jumps, similarly to the case of FDM-derived jump rates when $\alpha = 1.0$ in Figure 2.6. In the same manner, when $\beta = 0.319$ and $\beta = 1.0$, the FET-derived jump rates are similar to the FEM- and FVM-derived jump rates used in Figure 2.6, respectively. Hence, the variation in the parameter β allows the FET paradigm to emulate other methods.

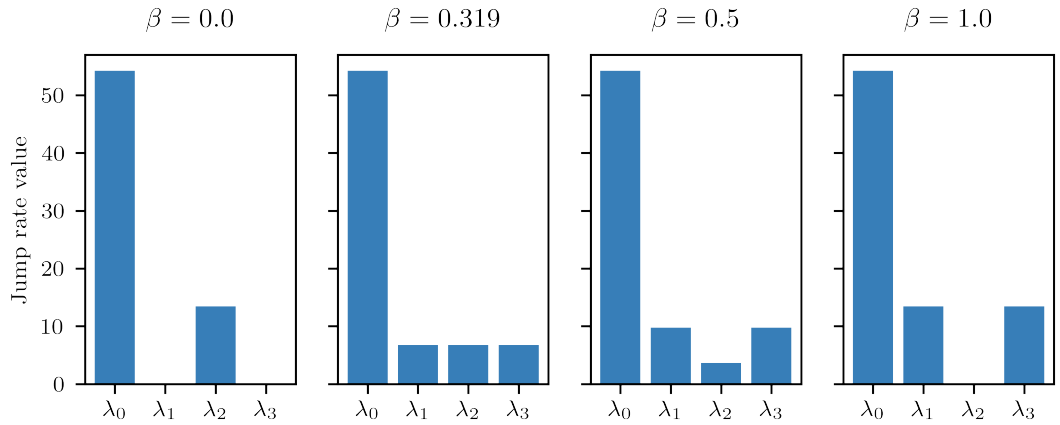


Figure 2.7: Comparison of the FET-derived jump rates for $h = 0.25\,m$, $\kappa = 1.0$ and $D = 1.0\,m^2s^{-1}$, as we vary the derivation parameter β.

2.2 Case studies

The SSA outlined in Algorithm 1 of Section 1.2.2 allows us to simulate a range of RD systems and test the impact of the different diffusive jump derivations on model predictions.

For simple systems, *e.g.* those that contain only zeroth and first order reactions, it can be shown that the mean behaviour of the RDME model matches the analytical solution of the corresponding macroscopic PDE model (see Appendix A.3 for more details of the models and their analytic solution). Hence, analysing the discrepancy between the two helps to determine the differences in accuracy of different sets of jump rates. The PDE represents the evolution of mean molecule concentration in time and space, so to compare with the results of stochastic simulation, we use the following error measure:

$$e(t) = \sqrt{\sum_{i=1}^{I} \frac{1}{M_0} \left| U_i(t) - Au(\mathbf{x}_i, t) \right|^2}, \tag{2.59}$$

where A is the compartment area, M_0 is the initial number of molecules and $u(\mathbf{x}_i, t)$ is the analytical solution of the corresponding macroscale PDE at $\mathbf{x}_i$ at time t. We consider a high number of molecules in simulations to be able to easily compare results to the analytical solutions [35].

2.2.1 Diffusion

We first investigate the effects of varying the compartment aspect ratio, κ. The macroscale PDE that describes the evolution of molecule concentration is

$$\frac{\partial u(\mathbf{x}, t)}{\partial t} = D\nabla^2 u(\mathbf{x}, t) \quad \text{for} \quad \mathbf{x} \in \Omega, \tag{2.60}$$

with Neumann boundary conditions

$$(\mathbf{n} \cdot \nabla)\, u = 0 \quad \text{for} \quad \mathbf{x} \in \partial\Omega, \tag{2.61}$$

where $\mathbf{n}$ is the normal vector to the domain boundary $\partial\Omega$. We choose the initial condition to be

$$u(\mathbf{x}, 0) = \begin{cases} 5 \times 10^6/A & \text{for } 0 < x < \kappa h, 0 < y < h, \\ 0 & \text{otherwise,} \end{cases} \tag{2.62}$$

which corresponds to 5×10^6 molecules placed in the lower-left compartment. We use the analytic solution of this PDE (see Appendix A.3) to calculate the error, $e(t)$, using Equation (2.59). We can see side-by-side comparison of simulation output and the analytic solution to Equation (2.60) in Figure 2.8, where the two look very similar, hence, we use the error defined by Equation (2.59). Figures 2.9 and 2.10 show how the error varies with respect to the implementation parameters α and β, respectively, and with respect to the compartment aspect ratio, κ.

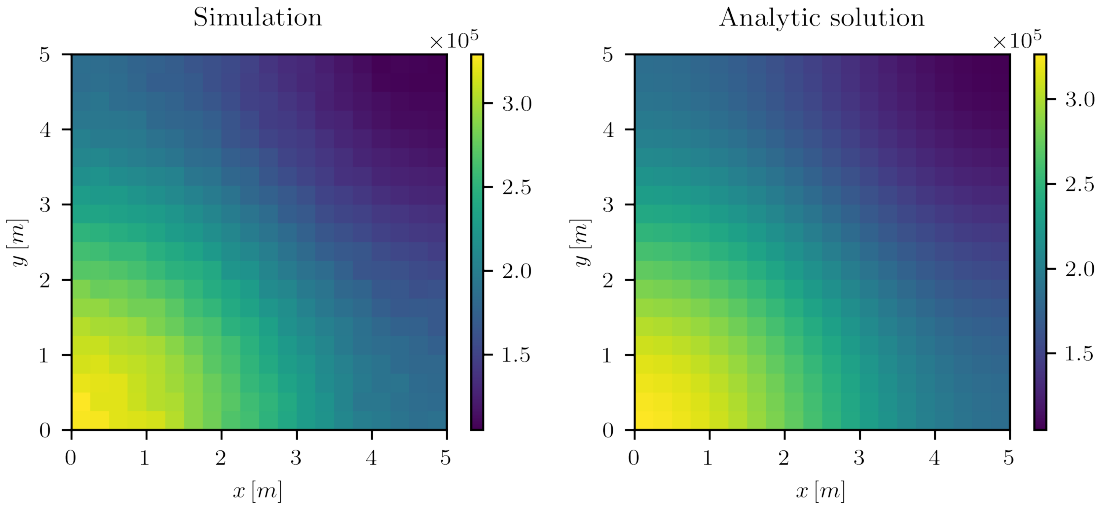


Figure 2.8: Heat maps of simulation output and analytic solution to Equation (2.60) at $T = 5.0$. The domain, $\Omega = [0, 5\,m]^2$ is discretised into 20×20 compartments and 5×10^6 molecules are placed in the compartment next to the origin. FVM-derived jump rates are used for the stochastic simulation.

In Figure 2.9 we see how increasing value of α increases the error. Increasing κ, on the other hand, decreases the error in simulations with FDM-derived jump rates. The effect that κ has on the error is due to the fact that increasing κ makes the discretisation of the domain finer, as we discretised the domain $\Omega = [0, 5\,m]^2$ into $20 \times 20\kappa$ compartments.

Next, we see in Figure 2.10 how the error is affected by the FET implementation parameter, β, and it tells a very different story to Figure 2.9. In this context, changing κ seems to have small effect on the error, while β has a non-linear relationship with the error at a minimum at around $\beta = 0.5$.

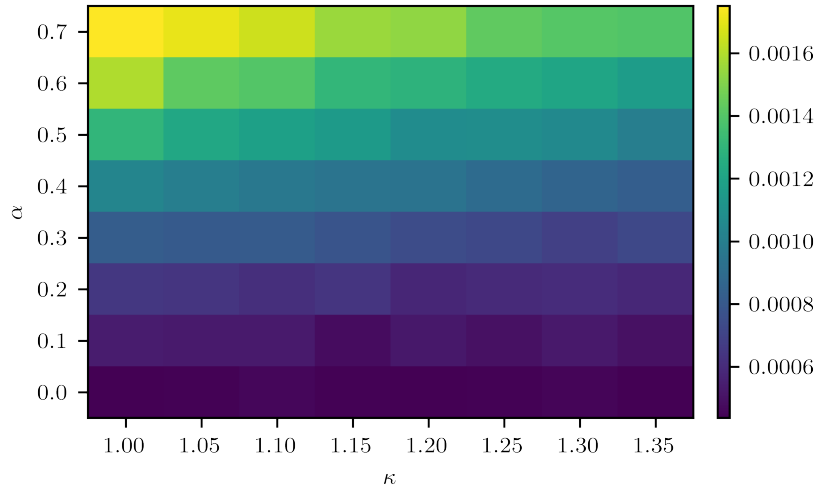


Figure 2.9: Heat map of error at time $T = 5.0\,s$ as defined in Equation (2.59), as a function of parameters α and κ for simulations that use FDM-derived jump rates. The domain used in the simulations is $\Omega = [0, 5\,m]^2$, which has been discretised into $20 \times 20\kappa$ compartments and 5×10^6 molecules are placed in the compartment next to the origin.

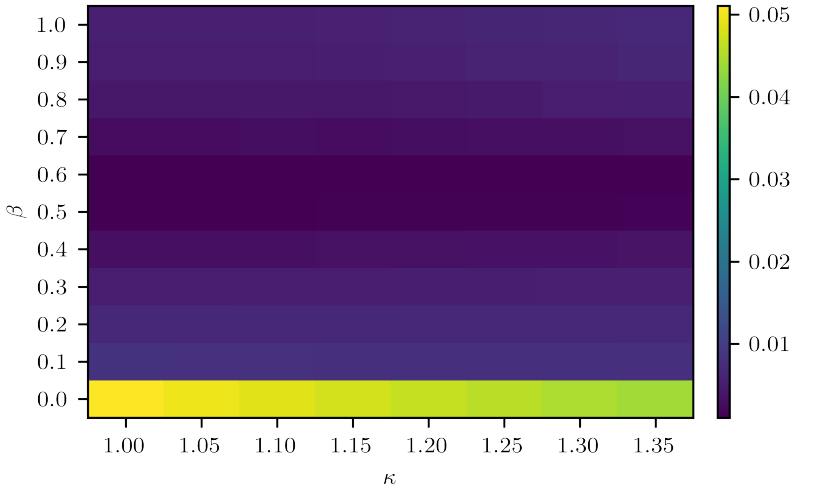


Figure 2.10: Heat map of error at time $T = 5.0\,s$ as defined in Equation (2.59), as a function of parameters β and κ for simulations that use FET-derived jump rates. The domain used in the simulations is $\Omega = [0, 5\,m]^2$, which has been discretised into $20 \times 20\kappa$ compartments and 5×10^6 molecules are placed in the compartment next to the origin.

To compare all the methods of derivation of the jump rates, we restrict the values of the implementation parameters α and β for the FDM and FET derivations, respectively, in

Figure 2.11. We see that the error is relatively unaffected by variation in the compartment aspect ratio, κ, for simulations with FVM-derived jump rates and these give rise to the smallest error. For the value of α used, the errors in simulations with FDM-derived jump rates are very similar to the errors seen in simulations with FEM-derived jump rates. Note that for $\alpha = 0$ the diffusive jump rates for the FDM and the FVM are identical and, as shown in Figure 2.9, as we increase the value of α, we increase the error. The errors for simulations with FDM- and FEM-derived jump rates decrease as we increase κ, which is as expected, as increasing κ leads to a finer discretization. This is in direct contrast to the errors for simulations with FET-derived jump rates, where the error increases as κ is increased. Figure 2.10 shows how much the choice of β affects the value of error in the simulations.

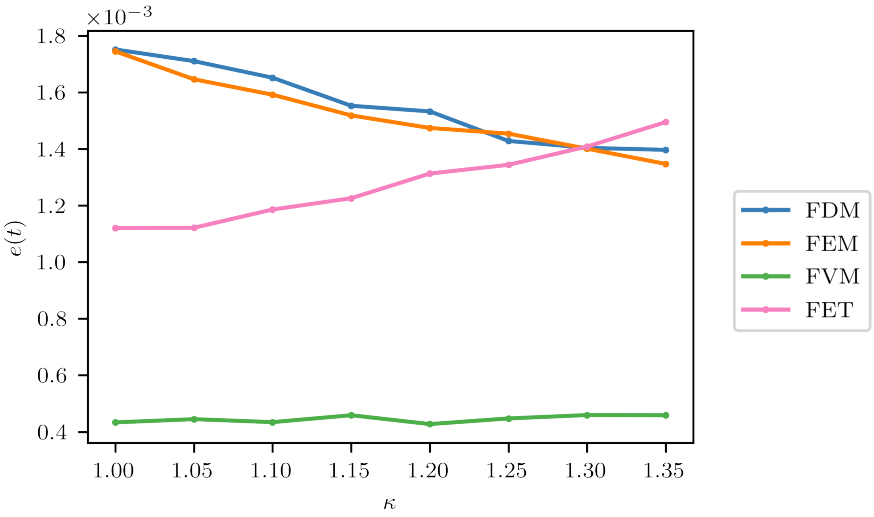


Figure 2.11: Plot of the error at time $T = 5\,s$ as a function of κ, the compartment aspect ratio, for simulations that used FDM-, FEM-, FVM- and FET-derived jump rates. The domain $\Omega = [0, 5\,m]^2$ is discretised into $20 \times 20\kappa$ compartments and 5×10^6 molecules are initially placed in the compartment adjacent to the origin. Parameters are: $D = 1\,m^2 s^{-1}$, $\alpha = 0.7$, $\beta = 0.5$.

2.2.2 Zeroth- and first-order reactions

We now consider RD systems involving the decay of molecules, given by the chemical equation

$$\mathrm{U} \xrightarrow{k_{-1}} \emptyset, \tag{2.63}$$

and the production of molecules, given by the chemical equation

$$\emptyset \xrightarrow{k_1} \mathrm{U}, \tag{2.64}$$

alongside diffusion. The corresponding propensity functions take the form

$$a_i^{\mathrm{decay}}(\mathbf{u}(t)) = k_{-1}u_i(t), \tag{2.65}$$

$$a_i^{\mathrm{prod}}(\mathbf{u}(t)) = k_1 A(t), \tag{2.66}$$

for the decay reaction and the production reaction in compartment i, respectively. As before, $A(t)$ is the area of a compartment at time t, and $u_i(t)$ is the number of molecules in compartment i at time t. We consider production-only, decay-only and production-and-decay systems, and the macroscale PDEs that describe the evolution of molecule concentration are, respectively,

$$\frac{\partial u(\mathbf{x},t)}{\partial t} = D\nabla^2 u(\mathbf{x},t) + k_1, \tag{2.67}$$

$$\frac{\partial u(\mathbf{x},t)}{\partial t} = D\nabla^2 u(\mathbf{x},t) - k_{-1}u, \tag{2.68}$$

$$\frac{\partial u(\mathbf{x},t)}{\partial t} = D\nabla^2 u(\mathbf{x},t) + k_1 - k_{-1}u. \tag{2.69}$$

Again, we use Neumann boundary conditions

$$(\mathbf{n} \cdot \nabla)\, u = 0 \quad \text{for} \quad \mathbf{x} \in \partial\Omega, \tag{2.70}$$

where $\mathbf{n}$ is the normal vector to the domain boundary $\partial\Omega$, and initially 5×10^6 molecules are placed in the central compartment.

The plots of error for these systems are essentially identical to those in Figure 2.11, hence we omit them here (see Appendix A.4). Again, the FVM out-performs the other three methods (FDM, FEM and FET) for the chosen values of α and β.

2.2.3 Second-order reactions

To investigate the effects of different derivations of the jump rates in systems involving second-order reactions, we now consider a system composed of two species, U and V, which undergo diffusion (with rates D_u and D_v, respectively) and the following reactions

$$\mathrm{U} + \mathrm{V} \xrightarrow{k_1} \mathrm{V}, \quad \emptyset \xrightarrow{k_2} \mathrm{U}, \tag{2.71}$$

i.e. U is consumed upon contact with V at rate k_1, and U is produced at rate k_2 on the domain $\Omega = [0, 1\,m]^2$. We use the same boundary conditions, *i.e.* Neumann boundary conditions, as in the previous section. The corresponding propensity functions take the form

$$a_i^1(\mathbf{u}(t)) = \frac{k_1}{A(t)} u_i(t) v_i(t), \tag{2.72}$$

$$a_i^2(\mathbf{u}(t)) = k_2 A(t). \tag{2.73}$$

We cannot write down equivalent closed-form macroscale PDEs for the evolution of mean U and V concentrations over time because the system contains second order reactions. As such, we investigate how the stationary distribution, estimated from stochastic simulations, is affected by the choice of compartment size, for the four different methods of deriving the diffusive jump rates. In Figure 2.12 we see that as the number of compartments increases the stationary distribution shifts to the right, towards higher molecule numbers. This is because the effective rate of bimolecular reactions decreases as the number of compartments increases and hence compartment size decreases [32]. Figure 2.13 shows how the stationary distribution changes with the method of derivation of the diffusive jump rates. All methods give slightly different results; in particular, using $\beta = 0.1$ or $\beta = 0.9$ in the FET results in stationary distributions that are shifted noticeably to the left and have a higher peak.

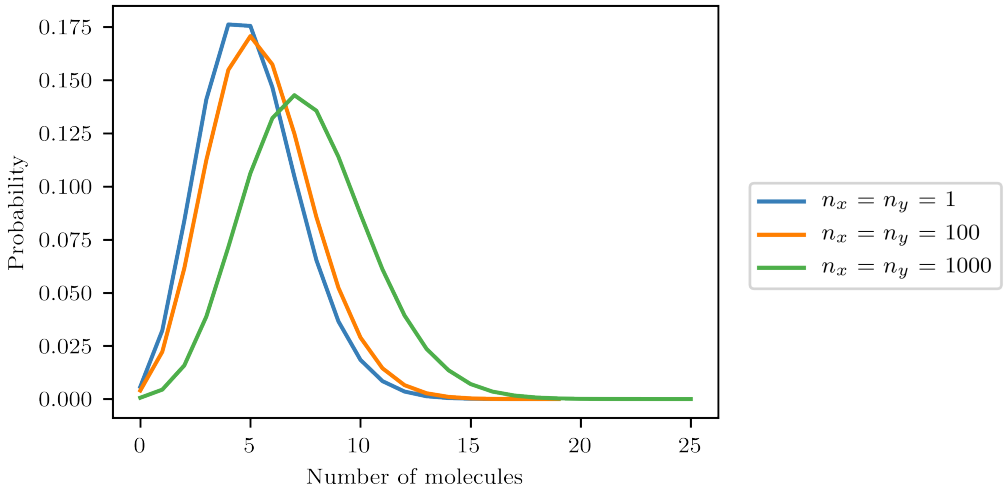


Figure 2.12: Comparison of the stationary distributions as the compartment number is varied, where we used FVM-derived jump rates. Stationary distributions as calculated from simulations run until $T = 1 \times 10^6 s$, on the domain $\Omega = [0, 1\,m]^2$, with the following reaction rates: $k_1 = 0.2\,m^2 s^{-1}$ and $k_2 = 1.0\,m^{-2} s^{-1}$. We initialised the simulations with five molecules of U and one molecule of V distributed at random across the compartments. To calculate the stationary distributions, the number of molecules was recorded every $0.5\,s$.

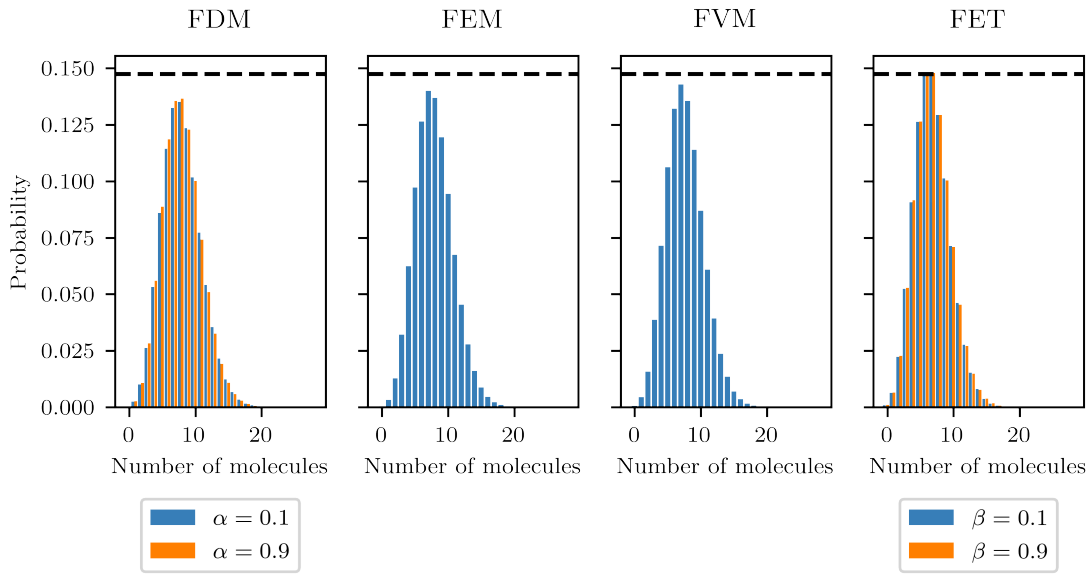


Figure 2.13: Comparison of the stationary distributions for different methods of derivation of the diffusive jump rates. The stationary distributions are calculated from simulations run until $T = 1 \times 10^6 s$, on the domain $\Omega = [0, 1\,m]^2$ discretised into 1000×1000, with the following reaction rates: $k_1 = 0.2\,m^2 s^{-1}$ and $k_2 = 1.0\,m^{-2} s^{-1}$. We initialised the simulations with five molecules of U and one molecule of V distributed at random across the compartments.

2.2.4 Schnakenberg kinetics

Finally, we investigate the effects of different derivations of the diffusive jump rates on Turing's RD model of pattern formation [64]. We focus on Schnakenberg kinetics [26, 57] as they are simple to simulate. The system is composed of two species, U and V, which undergo diffusion (with rates D_u and D_v, respectively) on the domain $\Omega = [0, 1\,m]^2$ and the following reactions

$$\emptyset \xrightarrow{k_1} \mathrm{U}, \quad \mathrm{U} \xrightarrow{k_2} \emptyset, \quad 2\mathrm{U} + \mathrm{V} \xrightarrow{k_3} 3\mathrm{U}, \quad \emptyset \xrightarrow{k_4} \mathrm{V}. \tag{2.74}$$

The corresponding propensity functions take the form

$$a_i^1(\mathbf{u}(t), \mathbf{v}(t)) = k_1 A(t), \tag{2.75}$$

$$a_i^2(\mathbf{u}(t), \mathbf{v}(t)) = k_2 u_i(t), \tag{2.76}$$

$$a_i^3(\mathbf{u}(t), \mathbf{v}(t)) = \frac{k_3}{A^2(t)} u_i(t)(u_i(t) - 1)v_i(t), \tag{2.77}$$

$$a_i^4(\mathbf{u}(t), \mathbf{v}(t)) = k_4 A(t). \tag{2.78}$$

We also assume Neumann boundary conditions as in the previous sections.

We quantify the patterns formed using the discrete Fourier cosine transform. Let $f(x, y) : \mathbb{R}^2 \to \mathbb{R}$, then the discrete Fourier cosine transform is defined as

$$\hat{f}(k_x, k_y) = \Delta_x \Delta_y \sum_{i=1}^{n_x} \sum_{j=1}^{n_y} \cos\left(k_x \Delta_x (i-1)\right) \cos\left(k_y \Delta_y (j-1)\right) f(x, y), \tag{2.79}$$

where Δ_x is the spacing between points $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$ and Δ_y is the spacing between points $\mathbf{x}_i$ and $\mathbf{x}_{i+n_x}$ (here, $\Delta_x = \kappa h$ and $\Delta_y = h$). The wavenumbers, k_x and k_y, give the spatial frequency of a pattern, with wavemodes m_x and m_y such that $k_x = m_x \pi / L$ and $k_y = m_y \pi / L$, where L is the side length of the square domain Ω [69]. To showcase results, we plot the power spectrum, defined as

$$P_s = \left| \widehat{f}(k_x, k_y) \right|^2, \tag{2.80}$$

where $\widehat{f}(\cdot, \cdot)$ is the discrete Fourier cosine transform defined in Equation (2.79).

We can predict which patterns are possible in equivalent PDE models of pattern formation using LSA [52], outlined in Section 1.3.1. In addition, Woolley *et al.* [68] developed a method using a weak noise expansion (WNE) of the RDME [65] to predict the

wavemodes that can evolve in stochastic simulations, which we outlined in Section 1.3.2. The advantage of the WNE approach over LSA in determining whether a particular wavemode is possible for a given parameter set is that the WNE takes into account the details of the diffusive jump rates. We compare the simulation results to the predictions of the WNE and LSA to see whether the patterns formed fall within the ranges of wavemodes predicted by those methods.

All the simulations are run until $T = 1800\,s$ on the domain $\Omega = [0, 1\,m]^2$, which is discretised into $40 \times \lfloor 40\kappa \rfloor$ compartments. We use the following reaction rates: $D_u = 1 \times 10^{-5}\,m^2 s^{-1}$, $D_v = 1 \times 10^{-3}\,m^2 s^{-1}$, $k_1 = 1\,s^{-1}$, $k_2 = 0.02\,s^{-1}$, $k_3 = 1 \times 10^{-6}\,s^{-1}$ and $k_4 = 3\,s^{-1}$. We initialise the system at the spatially uniform steady state, placing 200 molecules of species U and 75 molecules of species V in each compartment.

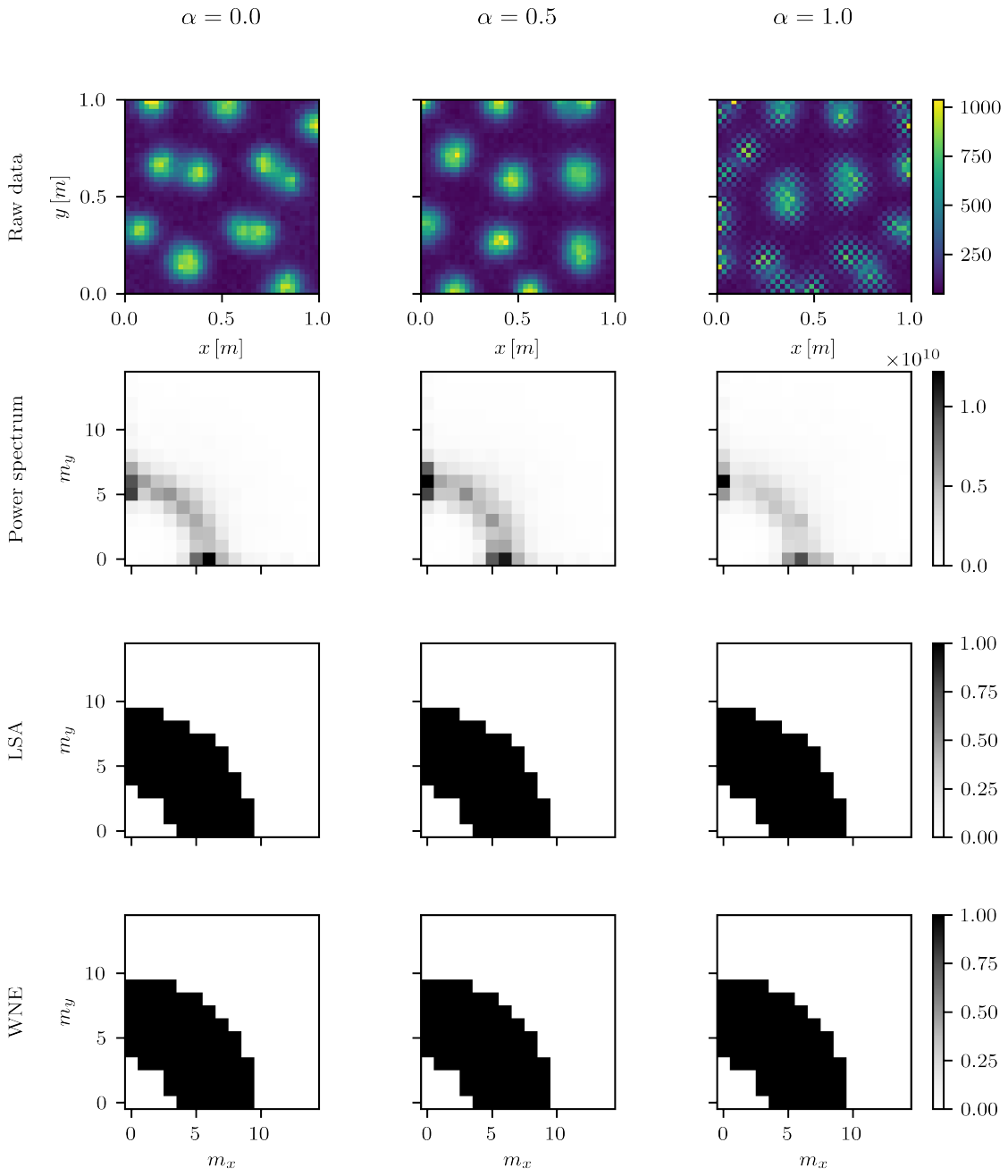


Figure 2.14: Results from simulations with FDM-derived diffusive jump rates with $\kappa = 1.0$ for the Schnakenberg system. Raw data row shows an example pattern. Power spectrum row shows power spectra averaged over 100 simulations. LSA row shows wavemodes predicted using LSA and the bottom row, WNE row, shows wavemodes predicted using the WNE.

The top row of Figure 2.14 shows examples of patterns formed with FDM-derived diffusive jump rates, while the second-from-top row of Figure 2.14 shows corresponding power spectra averaged over 100 simulations. There is good agreement with the possible wavemodes predicted using the WNE (bottom row of Figure 2.14) and LSA (second-from-bottom row of Figure 2.14). Our results indicate that varying the parameter α, which controls the rate of diagonal jumping, has no discernible effect on pattern formation, except when $\alpha = 1.0$ (only diagonal jumps are possible) where checkerboard patterns arise.

Next, we investigate the effects of using FET-derived diffusive jump rates on pattern formation. Based on the averaged power spectrum shown in the second-from-top row of Figure 2.15, we see that changing the value of β shifts the range of excited wavemodes towards larger values. This observation is consistent with the predictions made using the WNE (bottom row of Figure 2.15). In this respect, the predictions of the LSA fall down as LSA does not take into account details of the diffusive jump rates (second-from-bottom row of Figure 2.15).

To explore whether changing the compartment aspect ratio has any bearing on the patterns formed, we also simulated the system with $\kappa = 1.4$. As in the case of $\kappa = 1.0$, larger wavemodes appear as the value of β increases (second-from-top row of Figure 2.16). However, the range of wavemodes observed in the x-direction is more significantly affected than the range in the y-direction. This observation is, again, consistent with the predictions made using the WNE, but cannot be predicted using LSA.

In both cases, $\kappa = 1.0$ and $\kappa = 1.4$, changing the parameter β, which controls the rate of molecules jumping diagonally in the FET derivation, has a significant effect on the type of patterns formed, as shown in Figure 2.15 and Figure 2.16.

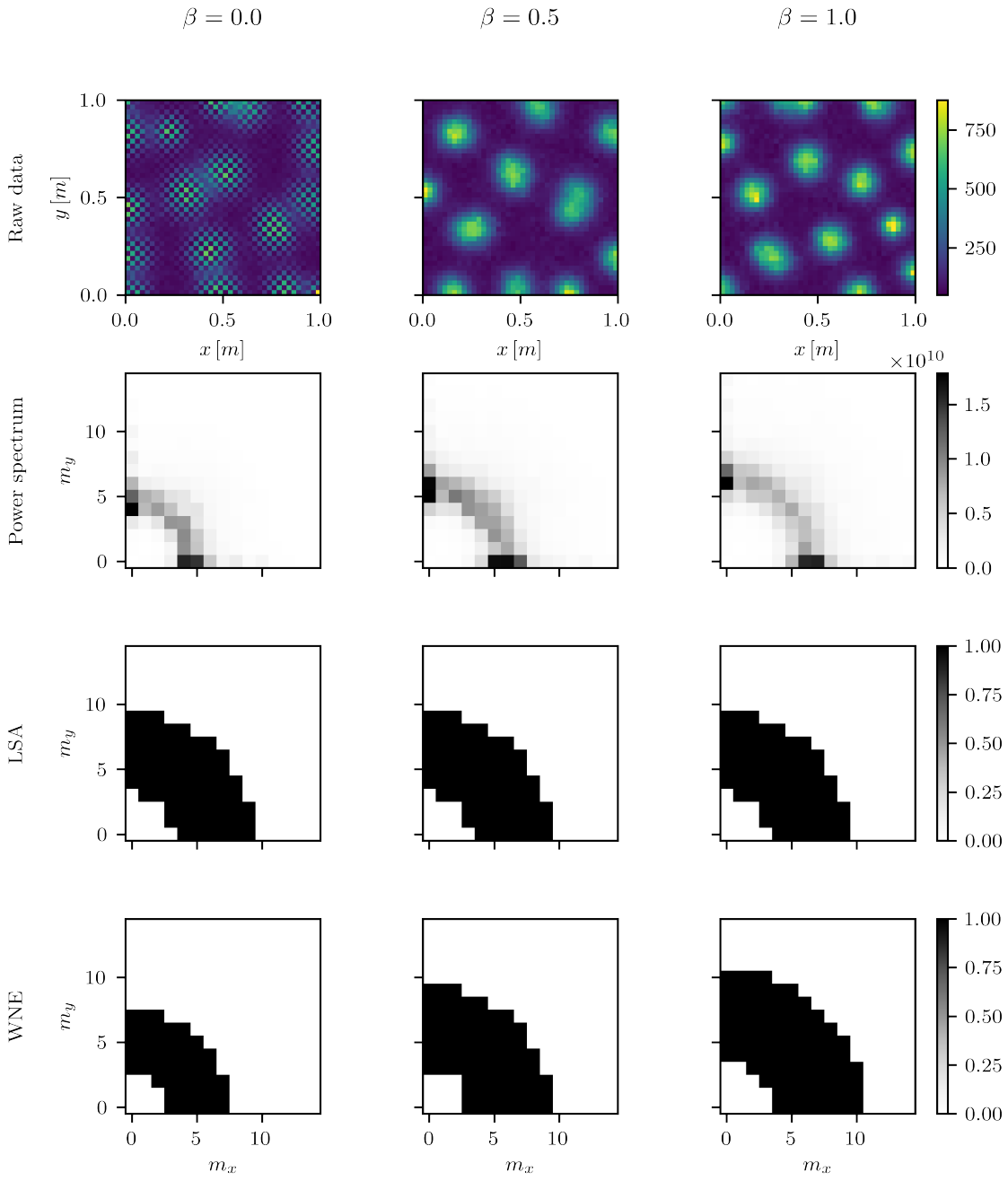


Figure 2.15: Results from simulations with FET-derived diffusive jump rates with $\kappa = 1.0$ for the Schnakenberg system. Raw data row shows an example pattern. Power spectrum row shows power spectra averaged over 100 simulations. LSA row shows wavemodes predicted using LSA and the bottom row, WNE row, shows wavemodes predicted using the WNE.

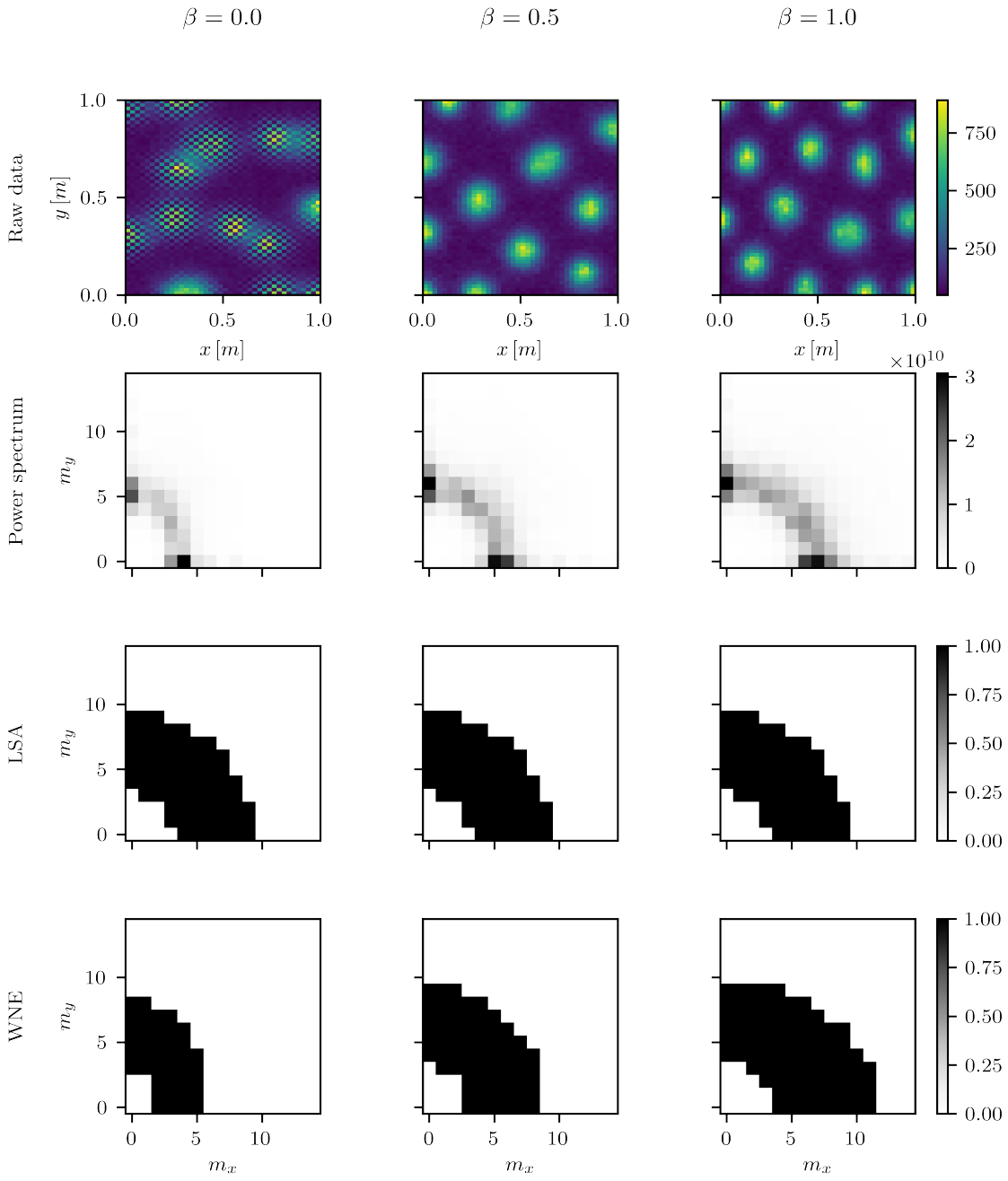


Figure 2.16: Results from simulations with FET-derived diffusive jump rates with $\kappa = 1.4$ for the Schnakenberg system. Raw data row shows an example pattern. Power spectrum row shows power spectra averaged over 100 simulations. LSA row shows wavemodes predicted using LSA and the bottom row, WNE row, shows wavemodes predicted using the WNE.

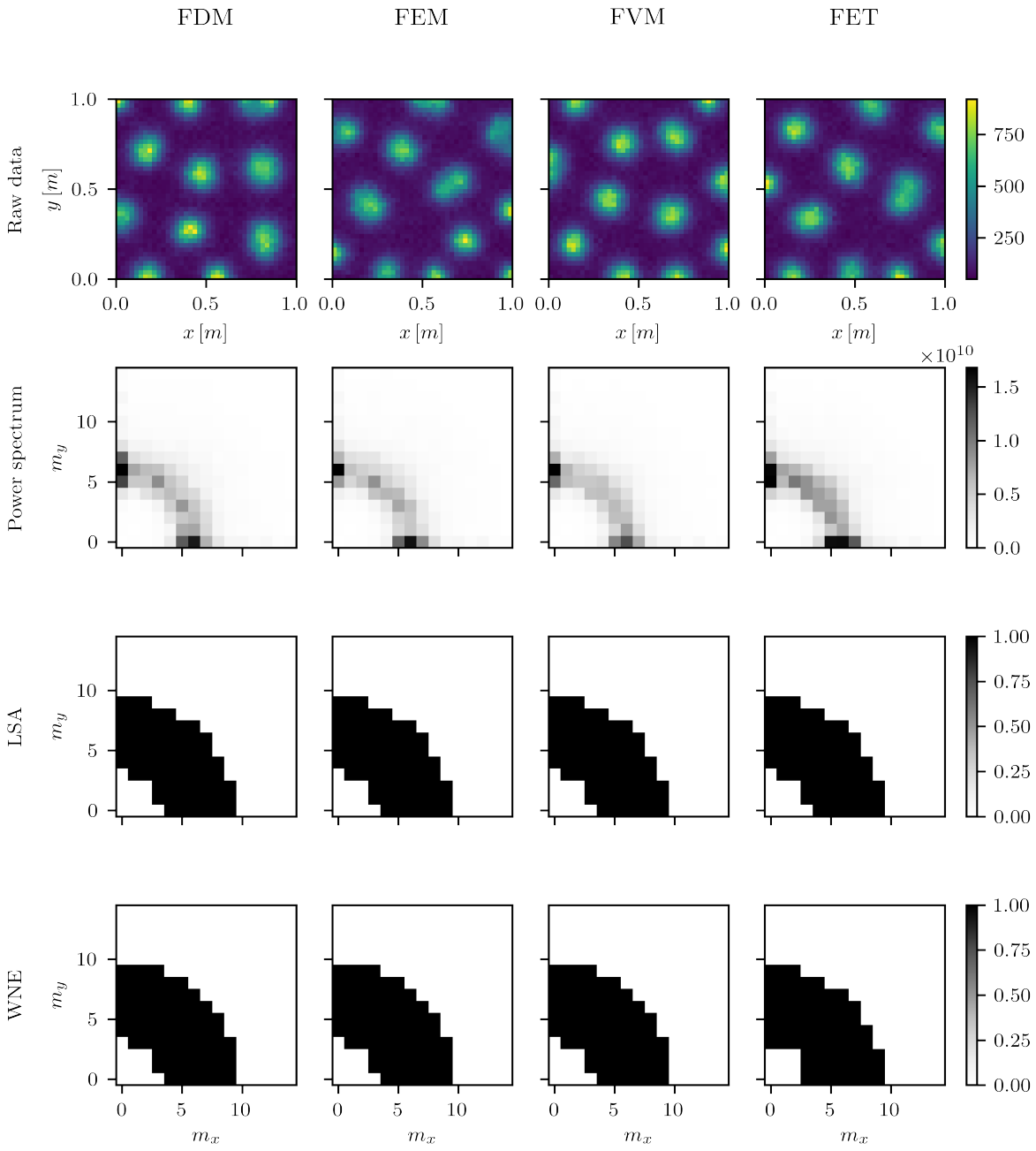


Figure 2.17: Comparison of the results from simulations with $\kappa = 1.0$ for the Schnakenberg system. Raw data row shows an example pattern. Power spectrum row shows power spectra averaged over 100 simulations. LSA row shows wavemodes predicted using LSA and the bottom row, WNE row, shows wavemodes predicted using the WNE. Simulations that used FDM-derived jump rates have $\alpha = 0.5$, while simulations that used FET-derived jump rates have $\beta = 0.5$.

Finally, we compare the results of simulations of the Schnakenberg system when using different methods of derivation of the jump rates. Figure 2.17 shows that the results from simulations with FEM- and FVM-derived jump rates are consistent with the predictions from LSA and WNE and are very similar to each other. When we compare results from simulations that used the FEM and the FVM to the simulations that used the FDM and the FET (with $\alpha = 0.5$ and $\beta = 0.5$) in Figure 2.17, we see that the simulation output is very similar.

2.3 Discussion

In this chapter, we followed the derivations of Meinecke and Lötsted [48] to arrive at different derivations of rates of molecules jumping between adjacent compartments, *i.e.* jump rates, on a domain discretised by a Cartesian mesh. We then presented different examples of RD systems in order to determine the differences in simulation results depending on method of derivation of the jump rates.

For the production-decay examples, which consisted solely of zeroth and first order reactions, the FVM generally gives the most reliable results in terms of the error between stochastic simulations and solution of the corresponding macroscale PDE (as does the FDM for $\alpha = 0$ where there are no diagonal jumps and we have the same diffusive jump rates as for the FVM) (Figure 2.11). When bimolecular reactions are included, we showed that the spatial discretisation affects results, for example shifting the stationary distribution (Figure 2.13).

Subsequently, using a RD system in Section 2.2.3, we showed how the stationary state of a RD system can be affected by the choice of jump rates. The FET-derived jump rates caused the stationary distribution for a system involving second-order reactions to

be noticeably different to the stationary distribution produced by simulations that used the other derivations (FDM, FEM, FVM).

We then moved to study the effects of different methods of derivation of diffusive jump rates on pattern formation, using the Turing RD model with Schnakenberg kinetics. We compared the averaged power spectra from simulations with the wavemodes predicted using LSA, which analyses the corresponding PDE model, and the WNE, which takes into account the details of the diffusive jump rates (Figures 2.14–2.15). Our results show that the WNE approach provides good predictions of the range of possible wavemodes, as expected from the work of Woolley *et al.* [68]. In particular, we showed that the aspect ratio of the compartments can significantly affect the patterns formed (see Figure 2.16).

Taken together, the results presented in this chapter demonstrate that care should be taken when choosing how to model diffusion in the context of the RDME models of stochastic RD systems, with a preference for FVM-derived jump rates for stochastic simulations using RDME framework.

This chapter assumed that the domain is static, however in many systems the size and shape of the domain evolves over time. As such, growth is a natural phenomenon to incorporate into RD models. Hence, the focus of the next chapter is to determine how the different methods of derivation of the jump rates extend to a growing domain, Ω_t.

Chapter 3

Growing domains and Cartesian meshes

Growth is a fundamental part of life. From *E. Coli* to humans, most living organisms grow. As such, if we want to harness mathematical models of biological systems, we need to develop approaches to take growth into account. However, stochastic models that incorporate domain growth are few and far between [67, 68, 69]. Woolley *et al.* [67] have investigated the choice choice of diffusive jump rates derivations on growing domains, but only on one-dimensional domains. We, however, investigate these choices on two-dimensional domains.

The aim of this chapter is to extend the RDME framework we introduced in Chapter 2 to model domain growth for RD systems on a Cartesian mesh and investigate the effects of domain growth and the different derivations of diffusive jump rates on simulation results. First, in Section 3.1, we outline the mathematical description of domain growth we use in this chapter. Then, in Section 3.2, we show simulation results comparing different methods of derivation of the jump rates alongside different functions that describe domain growth.

3.1 Mathematical techniques

In this section we will describe the tools that we use to extend the mathematical techniques established in the previous chapter. First, we show the mathematical description of tissue growth in Section 3.1.1. We then present a derivation of the jump rates for a RD system on a growing domain in Section 3.1.2.

3.1.1 Uniform domain growth

In this section, we derive a mathematical description of reaction and diffusion on a growing domain, which is then used in the derivation of jump rates on a growing domain. This description is independent of the modelling formalism; it can be used with PDEs [10, 11, 40, 41, 42] and in a RDME framework (as is the case in this work). Our objective is first to determine the velocity field, $\mathbf{v}(\mathbf{x}, t)$, that defines domain growth.

Let us consider a domain evolving in time, Ω_t, for a RD system, such that $\Omega_t = [0, L(t)\, m]^2$. Much like in the previous chapter, we discretise the domain, Ω_t, into I rectangular compartments, n_x and n_y compartments in x- and y-directions, respectively. Hence, one side of the domain, say in the x-direction, can be written as the sum of widths of compartments along that direction

$$L_x(t) = \sum_{j=1}^{n_x} \left(\tilde{x}_j(t) - \tilde{x}_{j-1}(t)\right) , \tag{3.1}$$

where i denotes the compartment (j, k) by defining $i = j + n_x k$. The x-coordinates of the boundaries parallel to the y-axis, $\tilde{x}_j$ and $\tilde{x}_{j-1}$ for compartment i, can be expressed as $\tilde{x}_j = jL_x(t)/n_x$.

Now, let us consider the domain at a later time, $t + \delta t$, where δt is a small positive

increment in time. We have

$$L_x(t+\delta t) = \sum_{j=1}^{n_x} \left(\tilde{x}_j(t+\delta t) - \tilde{x}_{j-1}(t+\delta t)\right) . \tag{3.2}$$

We model domain growth by assuming that the movement of the points $\tilde{x}_j(t)$ due to growth evolves according to

$$\tilde{x}_j(t+\delta t) = \tilde{x}_j(t) + v_x(\tilde{x}_j(t), t)\delta t \,, \tag{3.3}$$

where $v_x(x,t)$ is a velocity field. Next, we use Equation (3.3) in Equation (3.2) to get the following:

$$L_x(t+\delta t) = \sum_{j=1}^{N} \left(\tilde{x}_j(t) + v_x(\tilde{x}_j(t), t)\delta t - \tilde{x}_{j-1}(t) - v_x(\tilde{x}_{j-1}(t), t)\delta t\right) , \tag{3.4}$$

which can be rearranged to give

$$\frac{L_x(t+\delta t) - L_x(t)}{\delta t} = \sum_{j=1}^{N} \left(v_x(\tilde{x}_j(t), t) - v_x(\tilde{x}_{j-1}(t), t)\right) . \tag{3.5}$$

We observe that

$$\lim_{\delta t \to 0} \left(\frac{L_x(t+\delta t) - L_x(t)}{\delta t} \right) = \frac{\mathrm{d}L_x}{\mathrm{d}t} \tag{3.6}$$

and

$$\lim_{\delta x \to 0} \left(\sum_{j=1}^{N} \left(v_x(\tilde{x}_j(t), t) - v_x(\tilde{x}_{j-1}(t), t)\right) \right) = \int_0^{L_x(t)} \frac{\partial v_x}{\partial x} \, \mathrm{d}x \,, \tag{3.7}$$

where $\delta x = \tilde{x}_j(t) - \tilde{x}_{j-1}(t)$. Therefore, we have

$$\frac{\mathrm{d}L_x}{\mathrm{d}t} = \int_0^{L_x(t)} \frac{\partial v_x}{\partial x} \, \mathrm{d}x \,. \tag{3.8}$$

If we assume, for simplicity, that the domain is growing uniformly, such that $\partial v_x/\partial x$ is independent of x, then we can rearrange Equation (3.8) to give

$$\frac{\partial v_x}{\partial x} = \frac{1}{L_x(t)} \frac{\mathrm{d}L_x}{\mathrm{d}t}\,, \tag{3.9}$$

and so the velocity field $v_x(x,t)$ can be expressed as

$$v_x(x,t) = \frac{x}{L_x(t)} \frac{\mathrm{d}L_x}{\mathrm{d}t}\,. \tag{3.10}$$

To derive the equivalent relationship between $L_y(t)$ and $v_y(y,t)$, we follow the above steps to arrive at

$$v_y(y,t) = \frac{y}{L_y(t)} \frac{\mathrm{d}L_y}{\mathrm{d}t}\,. \tag{3.11}$$

We assume that the growth of the domain is isotropic by specifying that the length of the domain in both x- and y-direction follows the same evolution, and so $L(t) = L_x(t) = L_y(t)$. Hence, the domain growth velocity field, $\mathbf{v}(\mathbf{x},t)$, can expressed as

$$\mathbf{v}(\mathbf{x},t) = \left(\frac{x}{L(t)} \frac{\mathrm{d}L}{\mathrm{d}t}, \frac{y}{L(t)} \frac{\mathrm{d}L}{\mathrm{d}t} \right)\,. \tag{3.12}$$

Therefore, once we choose a function that describes how the domain is growing, $L(t)$, we can calculate the velocity field function, $\mathbf{v}(\mathbf{x},t)$. This will be important in the next section in deriving jump rates on a growing domain.

3.1.2 Derivations of jump rates on a growing domain

To derive the diffusive jump rates, much like in the previous chapter, we start by considering the Brownian motion of a molecule at point $\mathbf{x}(t) = (x(t), y(t))$ which evolves

according to the following SDEs

$$\begin{aligned} x(t+\mathrm{d}t) &= x(t) + v_x(x,t)\mathrm{d}t + \sqrt{2D}\mathrm{d}W_x, \\ y(t+\mathrm{d}t) &= y(t) + v_y(y,t)\mathrm{d}t + \sqrt{2D}\mathrm{d}W_y, \end{aligned} \tag{3.13}$$

where W_x and W_y are the usual Wiener processes. Note that Equation (3.13) has additional terms, $v_x(x,t)\Delta t$ and $v_y(y,t)\Delta t$, that model domain growth by way of advection of a molecule in the x- and y- directions, respectively, as introduced in Section 3.1.1. A standard result in SDE theory is that the above equations satisfy the Fokker-Planck equation [58]

$$\frac{\partial p}{\partial t} = \frac{1}{2}\nabla^2\left(g(\mathbf{x},t)^2 p\right) - \nabla\cdot(\mathbf{f}(\mathbf{x},t)p), \tag{3.14}$$

where $p(\mathbf{x},t)$ is the probability density of a molecule to be at position $\mathbf{x} = (x,y)$ at time t. In Equation (3.14), $g(\mathbf{x},t) = \sqrt{2D}$ and $\mathbf{f}(\mathbf{x},t) = \mathbf{v}(\mathbf{x},t) = (v_x(x,t), v_y(y,t))^T$ and so that

$$\frac{\partial p}{\partial t} = D\nabla^2 p - \nabla\cdot(\mathbf{v}p) \quad \text{for} \quad \mathbf{x}\in[0,L(t)]^2 = \Omega_t, \tag{3.15}$$

with Neumann boundary conditions

$$\nabla p|_{\partial\Omega_t} = 0, \tag{3.16}$$

and initial condition

$$p(\mathbf{x},0) = p_0(\mathbf{x}). \tag{3.17}$$

Again, we want to calculate the jump rates between adjacent compartments, and we use the evolution of the probability of a molecule to be in compartment i, C_i. We discretise a two-dimensional domain Ω_t into K rectangular compartments, such that there

are n_x compartments in the x-direction and n_y compartments in the y-direction. Since, the domain grows in both directions, the boundaries of the compartments move over time. Hence, C_i is the set of points

$$C_i = [\tilde{x}_{j-1}(t), \tilde{x}_j(t)] \times [\tilde{y}_{k-1}(t), \tilde{y}_k(t)], \tag{3.18}$$

where $i = j + n_x k$ for $j = 1, \ldots, n_x$ and $k = 1, \ldots, n_y$, such that $x_j = jL(t)/n_x$ and $y_k = kL(t)/n_y$. Similarly to the previous chapter, we calculate the probability of a molecule to be in C_i at time t, $\mathbb{P}(C_i, t)$, by integrating the probability density function $p(\mathbf{x}, t)$ over the area of C_i, as such

$$\mathbb{P}(C_i, t) = \int_{\tilde{x}_{j-1}(t)}^{\tilde{x}_j(t)} \int_{\tilde{y}_{k-1}(t)}^{\tilde{y}_k(t)} p(\mathbf{x}, t)\, \mathrm{d}x\, \mathrm{d}y. \tag{3.19}$$

To proceed with the calculations we define the new coordinates as follows

$$(x, y, t) \to (\xi, \eta, \tau) = \left(\frac{x}{L(t)}, \frac{y}{L(t)}, t \right). \tag{3.20}$$

Hence, the probability $\mathbb{P}(C_i, t)$ is

$$\mathbb{P}(C_i, t) = \int_{\tilde{\xi}_{j-1}}^{\tilde{\xi}_j} \int_{\tilde{\eta}_{k-1}}^{\tilde{\eta}_k} p(\boldsymbol{\xi}, \tau) L^2(\tau)\, \mathrm{d}\xi \mathrm{d}\eta, \tag{3.21}$$

and the expression for $\mathbf{v}(\mathbf{x}, t)$, *i.e.* Equation (3.12), becomes

$$\mathbf{v}(\boldsymbol{\xi}, \tau) = \boldsymbol{\xi} \frac{\mathrm{d}L}{\mathrm{d}\tau}, \tag{3.22}$$

and Equation (3.15) becomes

$$\begin{aligned}
&\frac{\partial p}{\partial \tau} - \frac{1}{L(\tau)}\frac{\mathrm{d}L}{\mathrm{d}\tau}\boldsymbol{\xi}\cdot\nabla_\xi p = \frac{D}{L^2(\tau)}\nabla_\xi^2 p - \frac{1}{L(\tau)}\nabla_\xi\cdot\left(\boldsymbol{\xi}\frac{\mathrm{d}L}{\mathrm{d}\tau}p\right),\\
\Rightarrow&\frac{\partial p}{\partial \tau} = \frac{D}{L^2(\tau)}\nabla_\xi^2 p - \frac{1}{L(\tau)}\frac{\mathrm{d}L}{\mathrm{d}\tau}\left(\nabla_\xi\cdot(\boldsymbol{\xi}p) - \boldsymbol{\xi}\cdot\nabla_\xi p\right),\\
\Rightarrow&\frac{\partial p}{\partial \tau} = \frac{D}{L^2(\tau)}\nabla_\xi^2 p - \frac{1}{L(\tau)}\frac{\mathrm{d}L}{\mathrm{d}\tau}\left(p\nabla_\xi\cdot\boldsymbol{\xi}\right),\\
\Rightarrow&\frac{\partial p}{\partial \tau} = \frac{D}{L^2(\tau)}\nabla_\xi^2 p - \frac{2}{L(\tau)}\frac{\mathrm{d}L}{\mathrm{d}\tau}p,
\end{aligned} \tag{3.23}$$

where $\boldsymbol{\xi} = (\xi, \eta)$ and $\nabla_\xi = (\partial/\partial\xi, \partial/\partial\eta)$. Therefore, when we consider the evolution of the probability for a molecule to be in C_i at time t in the new coordinate system, we have

$$\frac{\mathrm{d}}{\mathrm{d}\tau}\mathbb{P}(C_i,\tau) = \int_{\tilde{\xi}_{j-1}}^{\tilde{\xi}_j}\int_{\tilde{\eta}_{k-1}}^{\tilde{\eta}_k}\left[\frac{\mathrm{d}p(\boldsymbol{\xi},\tau)}{\mathrm{d}\tau}L^2(\tau) + 2p(\boldsymbol{\xi},\tau)L(\tau)\frac{\mathrm{d}L}{\mathrm{d}\tau}\right]\mathrm{d}\xi\mathrm{d}\eta, \tag{3.24}$$

which can also be obtained by using the chain rule on Equation (3.21). If we substitute Equation (3.23) into Equation (3.24) we have

$$\begin{aligned}
\frac{\mathrm{d}}{\mathrm{d}\tau}\mathbb{P}(C_i,\tau) = \int_{\tilde{\xi}_{j-1}}^{\tilde{\xi}_j}\int_{\tilde{\eta}_{k-1}}^{\tilde{\eta}_k}\Bigg\{&\left[\frac{D}{L^2(\tau)}\nabla_\xi^2 p(\boldsymbol{\xi},\tau) - \frac{2}{L(\tau)}\frac{\mathrm{d}L}{\mathrm{d}\tau}p(\boldsymbol{\xi},\tau)\right]L^2(\tau)\\
&+2L(\tau)\frac{\mathrm{d}L}{\mathrm{d}\tau}p(\boldsymbol{\xi},\tau)\Bigg\}\,\mathrm{d}\xi\mathrm{d}\eta,
\end{aligned} \tag{3.25}$$

which leads to

$$\frac{\mathrm{d}}{\mathrm{d}\tau}\mathbb{P}(C_i,\tau) = \frac{D}{L^2(\tau)}\nabla_\xi^2\int_{\tilde{\xi}_{j-1}}^{\tilde{\xi}_j}\int_{\tilde{\eta}_{k-1}}^{\tilde{\eta}_k} p(\boldsymbol{\xi},\tau)L^2(\tau)\,\mathrm{d}\xi\mathrm{d}\eta. \tag{3.26}$$

Therefore, $\mathbb{P}(C_i,\tau)$ satisfies the diffusion equation with diffusion coefficient $D/L^2(\tau)$, and so we can use any method described in Chapter 2 to discretise $\mathbb{P}(C_i,\tau)$ in order to derive the diffusive jump rates.

As an example, if we follow the steps in Section 2.1.1.4 of the previous chapter to use the finite volume method (FVM) to discretise Equation (3.26), we transform back to coordinates (x, t) and let $P_i(t) = \mathbb{P}(C_i, t)$, to arrive at

$$\frac{\mathrm{d}}{\mathrm{d}t}(P_i(t)) = \frac{D}{\kappa^2 h^2(t)}\left(P_{i+1}(t) + P_{i-1}(t)\right) + \frac{D}{h^2(t)}\left(P_{i+n_x}(t) + P_{i-n_x}(t)\right) - \left(\frac{D}{\kappa^2 h^2(t)} + \frac{D}{h^2(t)}\right) P_i(t). \tag{3.27}$$

Therefore, looking at the coefficients in Equation (3.27), the jump rates using the FVM are as follows

$$\lambda_1(t) = \frac{D}{\kappa^2 h^2(t)}, \tag{3.28}$$

$$\lambda_2(t) = 0, \tag{3.29}$$

$$\lambda_3(t) = \frac{D}{h^2(t)}, \tag{3.30}$$

with $\lambda_0(t) = \sum_{i=1}^{8} \lambda_i(t)$.

Derivations for the finite difference method (FDM) and finite element method (FEM) follow analogously (see Table 3.1). Note that we no longer consider the first exit time (FET) approach to deriving the diffusive jump rates. Simpson *et al.* [59] have previously shown that on a growing domain the survival probability does not approach zero in the limit $t \to \infty$. As a result, the FET distribution has infinite mean so we cannot use it to parameterise the exponential distribution, as required in Equation (2.55).

A method used by other authors to incorporate domain growth uses Dynkin's formula [68, 69]. Woolley *et al.* in [68, 69] consider domain growth as a consequence of compartment splitting and, by using Dynkin's formula, they derived a finite value for the expected

first exit time, that was then used to define the jump rates in their framework. Dynkin's formula cannot be used in the derivations of the jump rates within our description of domain growth, since the expected first exit time does not converge, as shown in [59].

Jump rates	Finite Difference Method (FDM)	Finite Element Method (FEM)	Finite Volume Method (FVM)
$\lambda_1(t)$	$\frac{D(1-\alpha\kappa)}{\kappa^2 h^2(t)}$	$\frac{D(2-\kappa^2)}{3\kappa^2 h^2(t)}$	$\frac{D}{\kappa^2 h^2(t)}$
$\lambda_2(t)$	$\frac{D\alpha}{2\kappa h^2(t)}$	$\frac{D(\kappa^2+1)}{6\kappa^2 h^2(t)}$	0
$\lambda_3(t)$	$\frac{D(\kappa-\alpha)}{\kappa h^2(t)}$	$\frac{D(2\kappa^2-1)}{3\kappa^2 h^2(t)}$	$\frac{D}{h^2(t)}$

Table 3.1: Comparison of jump rates from different derivations on a growing domain.

3.2 Case studies

In this section, we explore the effects of using the different diffusive jump rate derivations on a growing domain. As for the static domain case, we first consider cases where we can directly compare the mean behaviour with the analytic solution of the corresponding macroscale PDE, and we then consider models for pattern formation. We assume domain growth to be isotropic and such that $\Omega_t = [0, L(t)\,m]^2$. We consider various domain growth functions, $L(t)$, such as exponential, linear and logistic, and compare how these growth functions affect simulation results. Table 3.2 shows the domain growth functions we consider in this chapter and how the parameters of those functions are varied depending on the case study.

growth type	$L(t)$
exponential	$L_0 e^{rt}$
linear	$L_0(1+rt)$
log	$L_0(1+\log(1+rt))$
logistic	$2L_0/(1+e^{-rt})$

Table 3.2: Table of growth functions we consider in this chapter.

As we now have time-dependent propensity functions, we cannot use the simulation algorithm stated in Section 1.2.2. Instead we adapt Algorithm 1 using the Extrande method [66]. Each of the propensity functions decreases with time, and so we use the propensity function calculated at time t as the upper bound in the time interval $[t, t+\delta t]$ in the Extrande method. Hence, the modified algorithm is as stated in Algorithm 2.

Algorithm 2 Modified next sub-volume method SSA

Set $t = 0$.
Initialise $\boldsymbol{u}$.
for i in $\{1, 2, \ldots, I\}$ **do**
  Calculate the total propensity in compartment i, $a_i^0(\boldsymbol{u}(t), t)$.
  Set the upper bound, $\tilde{a}_i(\boldsymbol{u}(t), t) = a_i^0(\boldsymbol{u}(t), t)$
  Initialise the time until the next reaction for compartment i: $t_i \sim \exp(\tilde{a}_i(\boldsymbol{u}(t), t))$.
end for
while $t < T$ **do**
  Find argmin of the set $\{t_1, t_2, \ldots, t_I\}$ and denote it m.
  Set $t = t_m$.
  Generate a random number $r \sim U(0, 1)$.
  Choose reaction / diffusion q to fire from possible reactions in compartment m. Note that reaction q fires with probability $a_i^q(\boldsymbol{u}(t), t)/\tilde{a}_i(\boldsymbol{u}(t), t)$, diffusion of a molecule of species ℓ from box i to box j occurs with probability $d_{i,j}^{\ell}(\boldsymbol{u}(t), t)/\tilde{a}_i(\boldsymbol{u}(t), t)$ and no reaction happens with probability $(\tilde{a}_i(\boldsymbol{u}(t), t) - a_i^0(\boldsymbol{u}(t), t))/\tilde{a}_i(\boldsymbol{u}(t), t)$.
  Update molecule numbers: $\boldsymbol{u} := \boldsymbol{u} + \boldsymbol{\nu}_q$, where $\boldsymbol{\nu}_q$ is the stoichiometric matrix of reaction / diffusion q.
  Let Γ be the set of indices of non-zero elements of $\boldsymbol{\nu}_q$.
  for γ in Γ **do**
    Update $a_\gamma^k(\boldsymbol{u}(t), t)$, $d_{\gamma,j}^{\ell}(\boldsymbol{u}(t), t)$, $a_\gamma^0(\boldsymbol{u}(t), t)$ and $\tilde{a}_i(\boldsymbol{u}(t), t)$ accordingly.
    Generate a random number $\Delta t \sim \exp(\tilde{a}_\gamma(\boldsymbol{u}(t), t))$.
    Update the time until the next reaction: $t_\gamma = t + \Delta t$.
  end for
end while

3.2.1 Diffusion

As in the static case, there is an analytic solution to the macroscale PDE (see Appendix B.1). We explore how the error, as defined in Section 2.2, evolves in time. Figure 3.1 shows how the width of the domain, $L(t)$, evolves with time for various domain growth functions and parameters of those functions. We chose the growth rate, r, such that by the end of the simulations on a growing domain, the domain increases from one static case ($L_0 = 5\,m$) to the other ($L_0 = 10\,m$) over the course of the simulation.

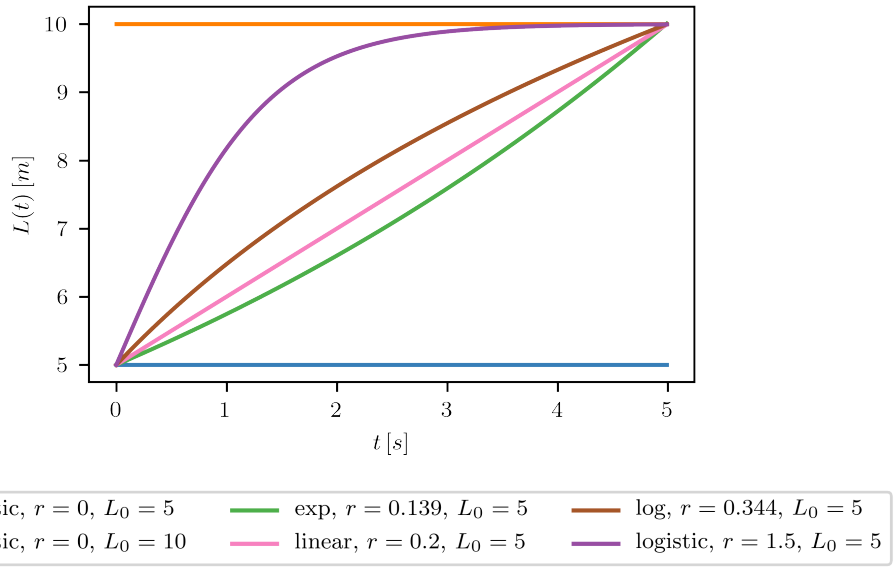


Figure 3.1: Plot of the domain growth functions we consider in Section 3.2.1. We consider two static cases: $L_0 = 5\,m$ and $L_0 = 10\,m$. We chose the growth rate, r, such that for simulations where the domain is growing, the domain size starts off at one static case under consideration ($L_0 = 5\,m$) and by the end of the simulation is the same size as the other static case ($L_0 = 10\,m$).

Figure 3.2 shows the errors of simulations with FDM-, FEM- and FVM-derived jump rates, as well as how they differ when the domain growth function changes. We can see that for all static and growing domain cases shown, the simulations with FVM-derived diffusive jump rates have the lowest error, similar to results presented in the previous chapter. Furthermore, the errors are much higher for simulations with FDM-dervied jump

rates, which is very much expected as we used FDM-derived jump rates with parameter $\alpha = 1.0$, which means that molecules can only jump to diagonally adjacent compartments. Varying α between 0 and 1 led to the same behaviour as in Figure 2.9 (resuts not shown). The errors were also higher for the static domain case of size $L = 10\,m$, compared to other cases, but this again is as expected because the number of compartments used was the same in all simulations, and as a result the discretisation of the domain was coarser.

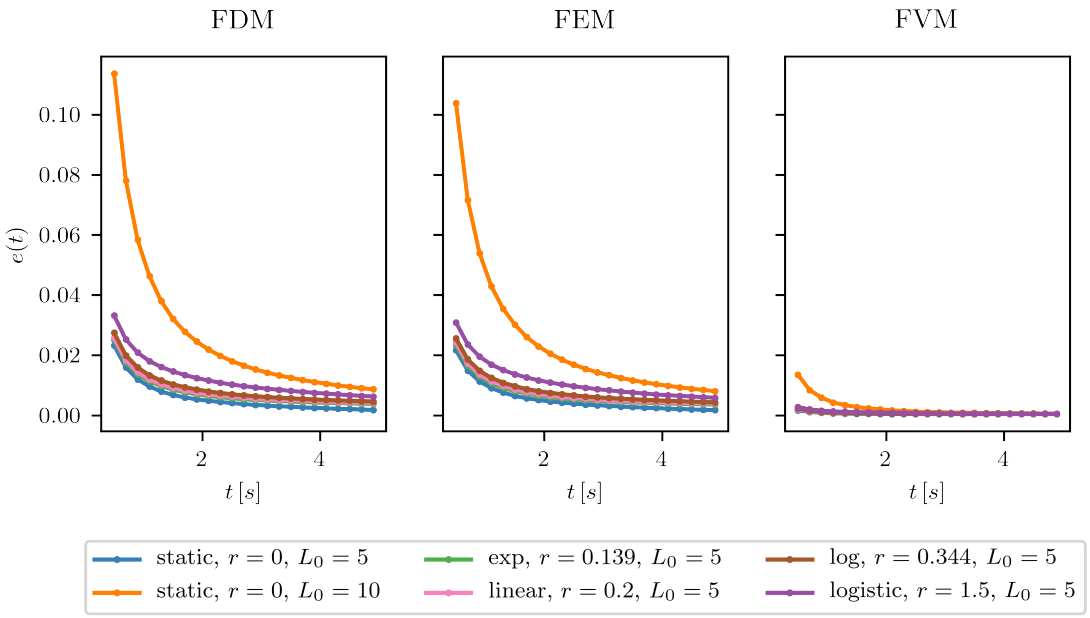


Figure 3.2: Plots of the error, $e(t)$, as a function of time for simulations with diffusion coefficient $D = 1.0\,m^2s^{-1}$. For simulations with FDM-derived jump rates $\alpha = 0.7$ was used. The domain has been discretised into 20×20 compartments and all 5×10^6 molecules were initially placed into the bottom-right compartment.

Next, we explore the effects of changing the compartment aspect ratio, κ, on the error, presenting results in Figure 3.3. The FVM-derived diffusive jump rates provide the smallest error across the range of compartment aspect ratios considered. For both the FEM- and FDM-derived diffusive jump rates, there is a slight decrease in the error as the compartment aspect ratio is increased, similar to results in Section 2.2.1. However, the choice of the growth rate function appears to impact the value of the error for simulations with FDM- and FEM-derived jump rates. The simulations which used FDM or FVM

derivations on a static domain of size $L = 5\,m$ had the smallest error, while the simulations on a static domain of size $L = 10\,m$ had the biggest error, which is as expected due to the coarser discretisation of the domain. However, the simulations on growing domains had errors between the two static cases at the end of the simulations, even though the size of the domain was the same as the static $L = 10\,m$ case.

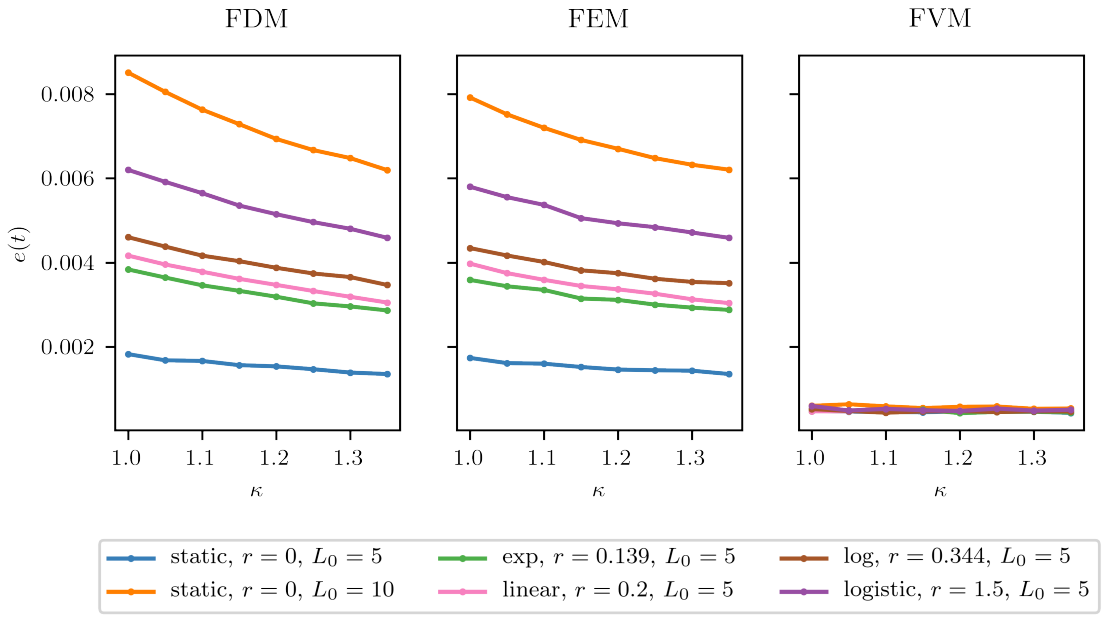


Figure 3.3: Plots of the error, $e(t)$, as a function of compartment aspect ratio, κ, for simulations with diffusion coefficient $D = 1.0\,m^2 s^{-1}$. For simulations with FDM-derived jump rates $\alpha = 0.7$ was used. We compare different derivations of the diffusive jump rates as we vary κ. The domain, of initial size $[0, 5\,m]^2$, is discretised into $20 \times 20\kappa$ compartments and all 5×10^6 molecules were initially placed into the bottom-right compartment.

3.2.2 Schnakenberg kinetics

We now consider pattern formation on a growing domain to determine to what extent the different derivations of diffusive jump rates affect the patterns formed. We use the Schnakenberg system, as in Section 2.2.4. Figure 3.4 showcases the range of patterns that form as the domain growth rate is varied, where we can see that if the growth rate, r, is too small there is no discernable difference from simulations on a static domain. On the other hand, if r is too large, as is the case in the bottom row of Figure 3.4, the

compartments become too large for us to discern any patterns. Hence, we pick the growth rates for each domain growth function such that the domain size, $L(t)$, doubles over the course of a simulation, as shown in Figure 3.5.

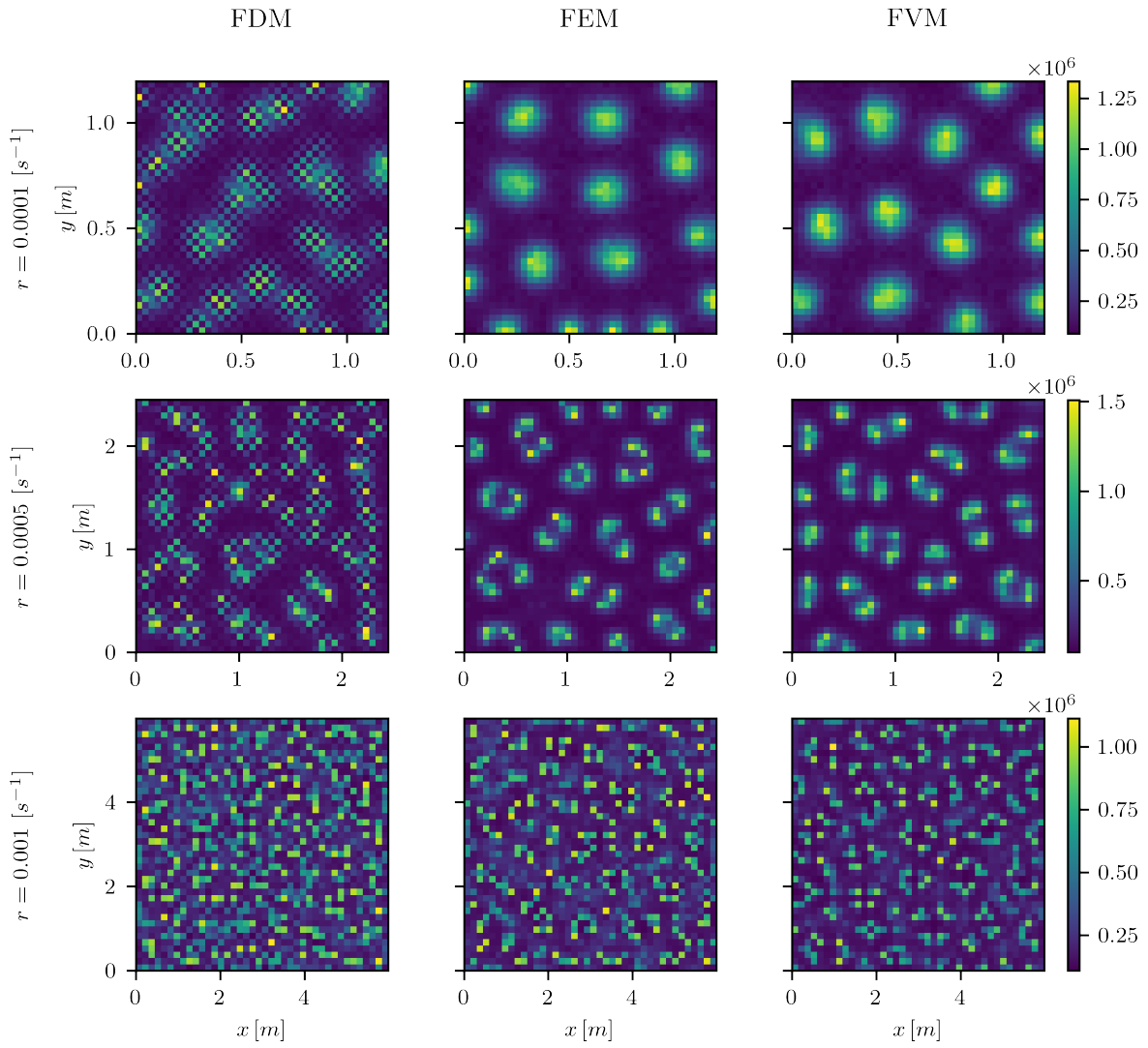


Figure 3.4: Comparison of different growth rates for simulations of the Schnakenberg system with the following parameters: $D_u = 1 \times 10^{-5}\, m^2 s^{-1}$, $D_v = 0.001\, m^2 s^{-1}$, $k_1 = 1.0\, m^{-2} s^{-1}$, $k_2 = 0.02\, s^{-1}$, $k_3 = 1 \times 10^{-6}\, m^4 s^{-1}$, $k_4 = 3.0\, m^{-2} s^{-1}$ run until $T = 1800\, s$. The domain has been discretised into 40×40 compartments and the domain is growing according to exponential growth, $L(t) = L_0 e^{rt}$.

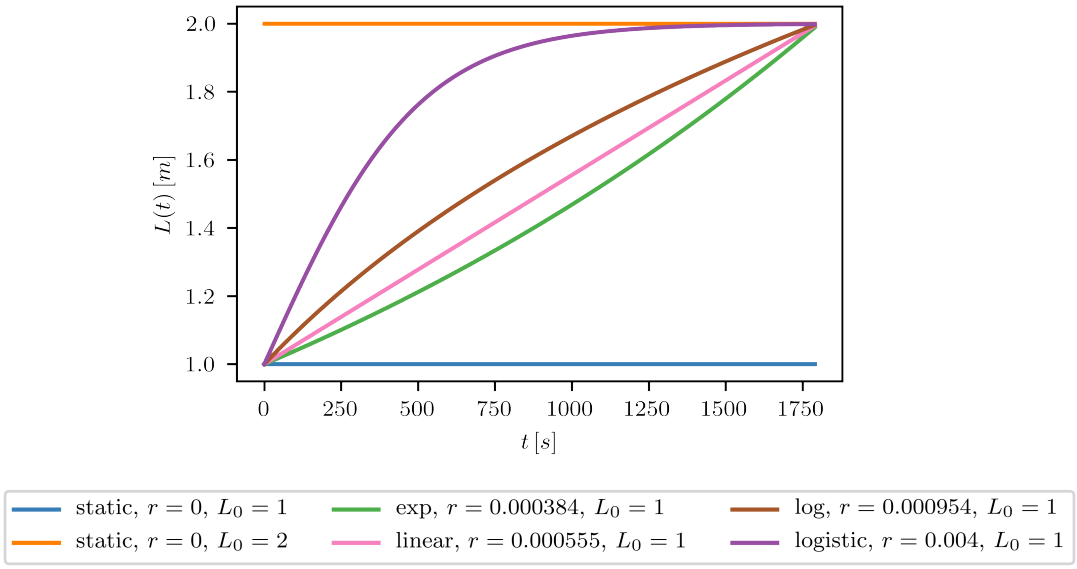


Figure 3.5: Plot of the domain growth functions we consider in Section 3.2.2. We consider two static cases: $L_0 = 1\,m$ and $L_0 = 2\,m$. We chose the growth rate, r, such that for simulations where the domain is growing, the domain size starts off at one static case under consideration ($L_0 = 1\,m$) and by the end of the simulation is the same size as the other static case ($L_0 = 2\,m$).

We see in Figure 3.4 that the patterns formed on an exponentially growing domain are different compared to the static case. However, using different derivations for the diffusive jump rates does not seem to markedly affect the patterns produced. The reason we see different patterns as the domain grows is because higher wavemodes become available, and the inherent stochasticity in the system causes clusters of molecules (peaks) to split and these higher wavemodes are then realised. This change in the wavemode is highlighted in the middle row in Figure 3.4.

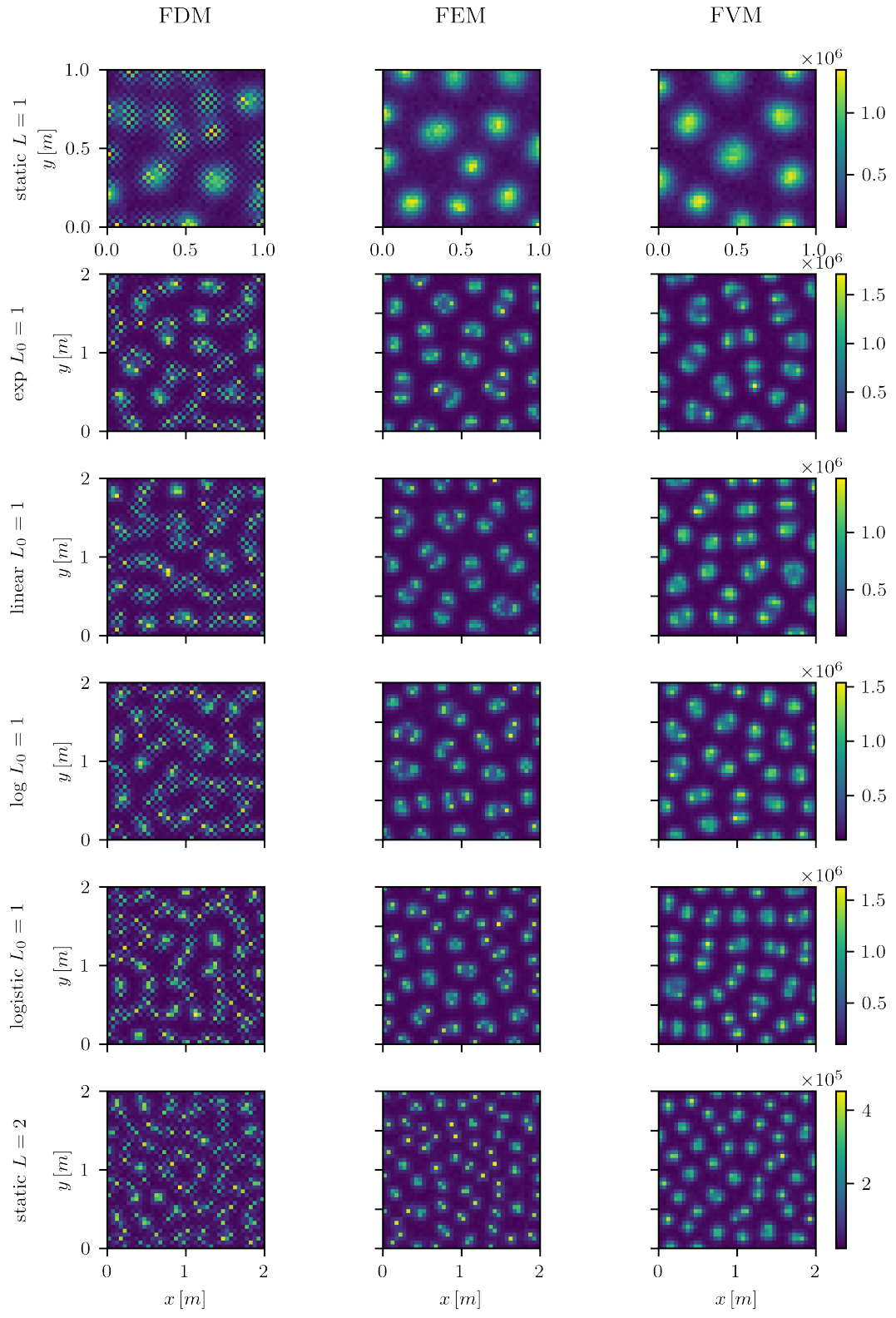


Figure 3.6: Results from simulations of the Schnakenberg system with the following parameters: $D_u = 1 \times 10^{-5}\, m^2 s^{-1}$, $D_v = 0.001\, m^2 s^{-1}$, $k_1 = 1.0\, m^{-2} s^{-1}$, $k_2 = 0.02\, s^{-1}$, $k_3 = 1 \times 10^{-6}\, m^4 s^{-1}$, $k_4 = 3.0\, m^{-2} s^{-1}$, and $\alpha = 1.0$ run until $T = 1800\, s$.

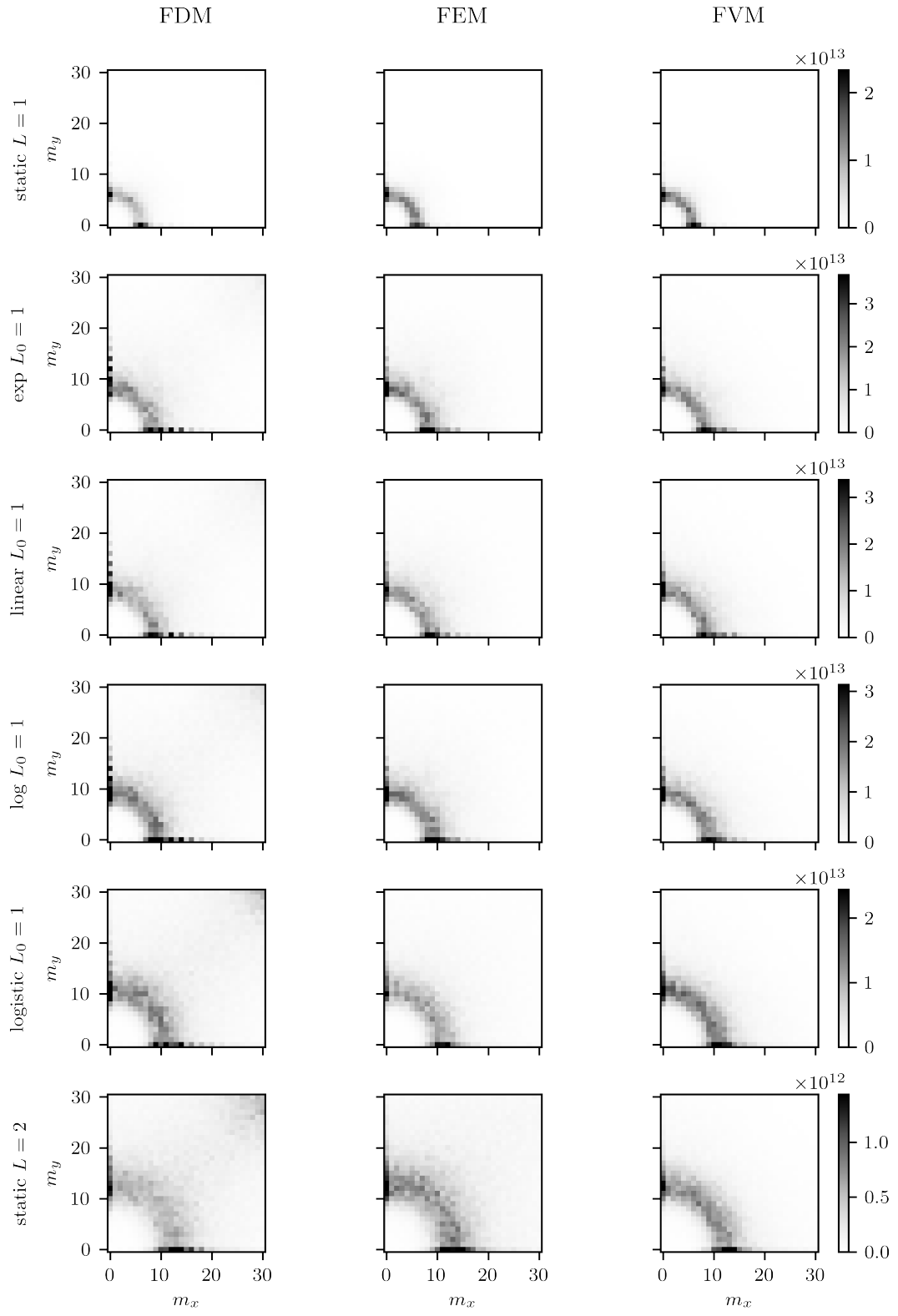


Figure 3.7: Power spectra of simulations of the Schnakenberg system at $T = 1800\,s$ averaged over 100 simulations with the following parameters: $D_u = 1 \times 10^{-5}\,m^2s^{-1}$, $D_v = 0.001\,m^2s^{-1}$, $k_1 = 1.0\,m^{-2}s^{-1}$, $k_2 = 0.02\,s^{-1}$, $k_3 = 1 \times 10^{-6}\,m^4s^{-1}$, $k_4 = 3.0\,m^{-2}s^{-1}$, and $\alpha = 1.0$.

Figure 3.6 shows the comparison of domain growth functions and methods of derivation of the jump rates for the Schnakenberg system. As expected, in this figure we see many more spots when the domain is growing compared to the static domain of size $L = 1$. However, for exponential, logarithmic and linear growth, the spots are not as small as for the static domain of size $L = 2$ (bottom row in Figure 3.6). Furthermore, there does not appear to be any discernable difference between methods of derivation of the jump rates as the domain growth function is changed. The only difference being the checker-board pattern for simulations with FDM-derived jump rates where parameter $\alpha = 1.0$ was used which means that molecules can only jump diagonally.

The comparison between the methods of derivation in Figure 3.6 is difficult just based on the state of the system at the end of a simulation. This is why we also analyse the results using the Fourier cosine transform, as defined by Equation (2.79), and calculate the power spectrum using Equation (2.80). For an easier comparison between domain growth functions, we take the radial component of the power spectra and plot the wavemode with the highest intensity, which we will refer to as the dominant wavemode. We see in Figure 3.8 that, in the case of a static domain of size $L = 1$, the dominant is either 5 or 6, and when the domain is static and of size $L = 2$, the dominant wavemode is between 10 and 13. When the domain is growing with an exponential, linear or logarithmic growth function, the dominant wavemode increases from 5 to 8 or 9, indicating that the size of the spots decreases and spots split as the domain is growing. However, only in the case of logistic domain growth does the dominant wavemode get close to the value of the dominant wavemode seen for the static domain of size $L = 2$, unlike the other choices of domain growth function. Lastly, the choice of method of derivation of the jump rates does not seem to have a significant effect on the results of simulations of the Schnakenberg

system based on Figure 3.8.

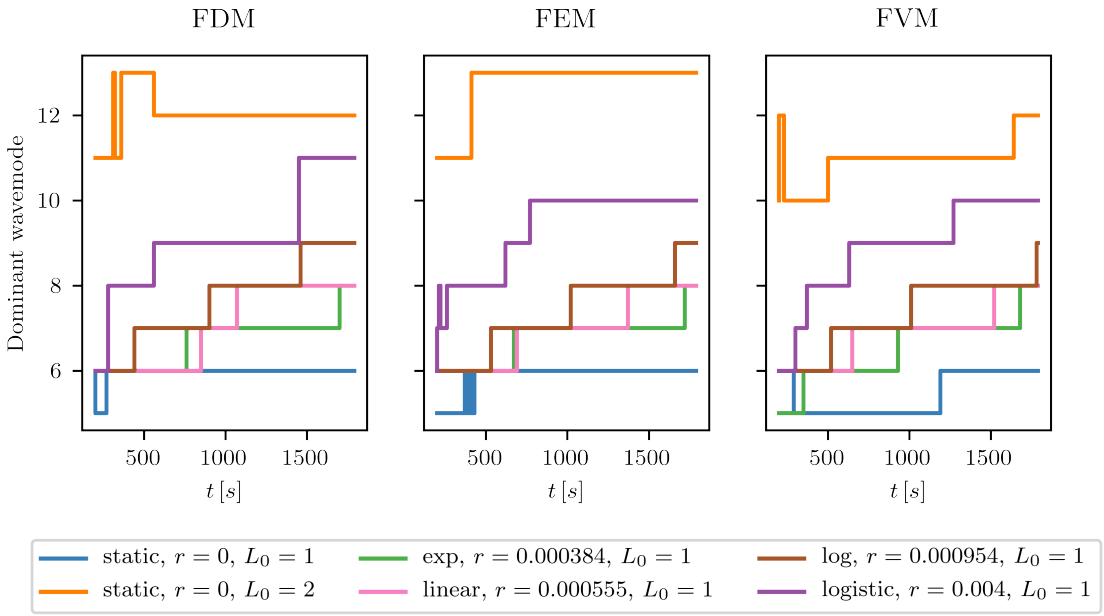


Figure 3.8: Comparison of dominant wavemode for simulations of the Schnakenberg system when varying the domain growth function. For each choice of domain growth function, the dominant wavemode is calculated based on the averaged power spectrum of 100 simulations.

3.2.3 Gray-Scott kinetics

As the strongest effects of modelling choices on simulation results were seen for RD systems that model pattern formation, we also investigate the behaviour of the Gray-Scott system. This system has a different mechanism of pattern formation and is described by the following reactions:

$$\mathrm{U} + 2\mathrm{V} \xrightarrow{k_1} 3\mathrm{V}, \quad \mathrm{U} \xrightarrow{k_2} \emptyset, \quad \mathrm{V} \xrightarrow{k_3} \emptyset, \quad \emptyset \xrightarrow{k_4} \mathrm{U}\,, \tag{3.31}$$

where U and V are the two species in this system, which also diffuse with rates D_u and D_v, respectively. The corresponding propensity functions are as follows:

$$a_i^1(\mathbf{u}(t), \mathbf{v}(t)) = \frac{k_1}{A(t)^2} u_i(t) v_i(t)(v_i(t) - 1); \tag{3.32}$$

$$a_i^2(\mathbf{u}(t), \mathbf{v}(t)) = k_2 u_i(t); \tag{3.33}$$

$$a_i^3(\mathbf{u}(t), \mathbf{v}(t)) = k_3 v_i(t); \tag{3.34}$$

$$a_i^4(\mathbf{u}(t), \mathbf{v}(t)) = k_4 A(t). \tag{3.35}$$

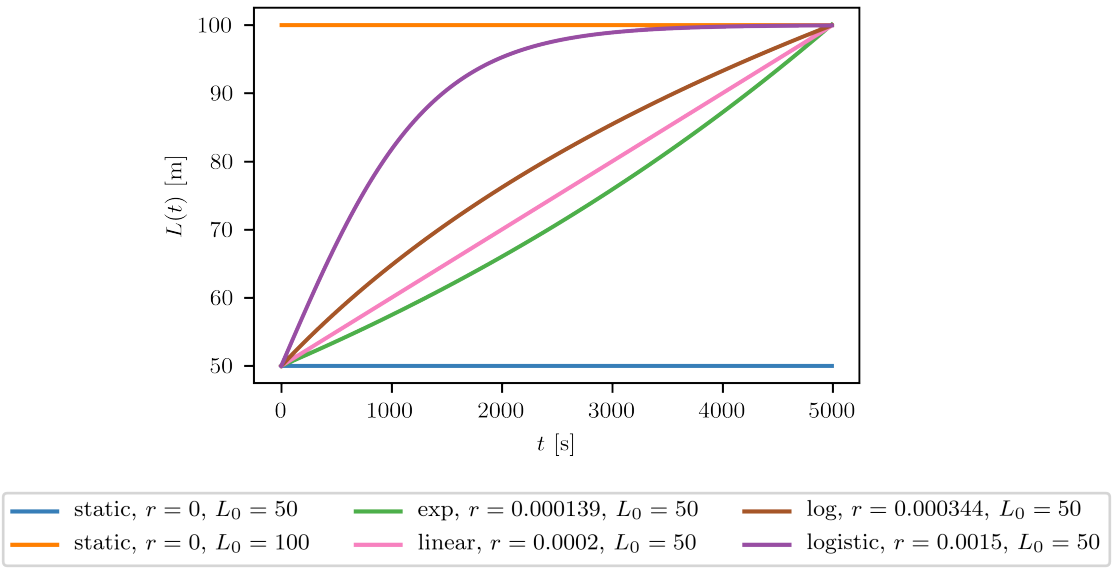


Figure 3.9: Plot of domain growth functions we consider in Section 3.2.3. We consider two static cases: $L_0 = 50$ and $L_0 = 100$. We chose the growth rate, r, such that in simulations where the domain is growing, the domain size starts off at one static case under consideration ($L_0 = 50$) and by the end of the simulation is the same size as the other static case ($L_0 = 100$).

We simulate the Gray-Scott model until $T = 5000\,s$ on the domain $\Omega = [0, L(t)\,m]^2$, which is discretised into 50×50 compartments. We use the following reaction rates: $D_u = 0.14\,m^2 s^{-1}$, $D_v = 0.06\,m^2 s^{-1}$, $k_1 = 2 \times 10^{-5}\,s^{-1}$, $k_2 = 7.0\,s^{-1}$, $k_3 = 0.035\,s^{-1}$ and

$k_4 = 0.1\,s^{-1}$. We initialise the system with the following initial condition:

$$u(\mathbf{x}, 0) = \begin{cases} 100 & \text{for } \mathbf{x} \in \left[\frac{3L_0}{10}, \frac{7L_0}{10}\right]^2, \\ 200 & \text{otherwise}, \end{cases} \tag{3.36}$$

$$v(\mathbf{x}, 0) = \begin{cases} 50 & \text{for } \mathbf{x} \in \left[\frac{3L_0}{10}, \frac{7L_0}{10}\right]^2, \\ 0 & \text{otherwise}, \end{cases} \tag{3.37}$$

which we can see in Figure 3.10.

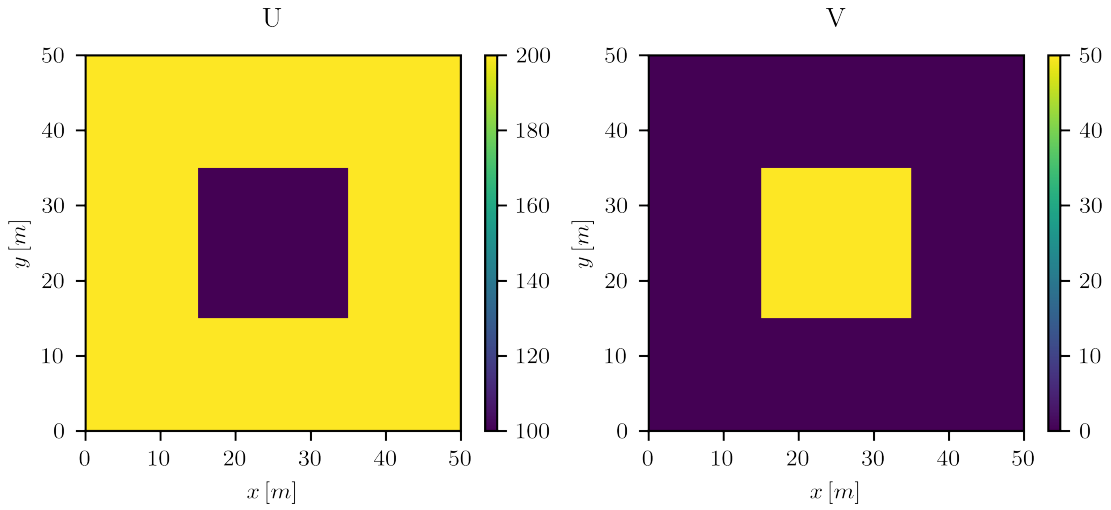


Figure 3.10: Initial conditions for Gray-Scott model.

In Figure 3.11 we see that domain growth, unlike in the Schnakenberg case, has a strong effect on pattern formation in the Gray-Scott RD system. We see that when the domain is growing slowly enough, spots form by the end of simulations irrespective of the jump rate derivation method. In the cases where the domain is growing (exponentially, linearly, logistically or logarithmically) in Figure 3.11, the domain size is $[0, 50\,m]^2$ initially and grows to be $[0, 100\,m]^2$ by the end of the simulations. For comparison we show the simulations where the domain was static for both of these sizes alongside simulations

where the domain was growing. We see that, even though patterns do not completely form when FDM- and FEM-derived jump rates are used in the case of a static domain with $L = 100\,m$, patterns do form when the domain grows slowly enough, *i.e.* when the domain is growing according to exponential, linear and logarithmic growth functions. That is unlike for the case of logistic domain growth, where we see in Figure 3.11 that patterns do not appear to form in simulations with FDM- and FEM-derived jump rates.

The striking results of Figure 3.11, compared to the analogous results of the Schnakenberg system in the previous section, are potentially a result of the pattern formation mechanism being more complex for the Gray-Scott system. The spots in the simulations of the Gray-Scott system form in a different way compared to the Schnakenberg system and from very different initial conditions. Starting from homogenous initial conditions, all the spots appear at the same time in the simulations of the Schnakenberg system. On the other hand, the number of molecules of species V initially in the Gray-Scott system is higher at the centre of the domain compared to the rest (and the opposite is true for species U), as seen in Figure 3.10, forming the initial spot; as the simulation proceeds, the initial spot proliferates, leading to spots covering the entire domain by the end of the simulation. This difference in mechanism by which the patterns form potentially explains why patterns do not form if the domain grows too quickly.

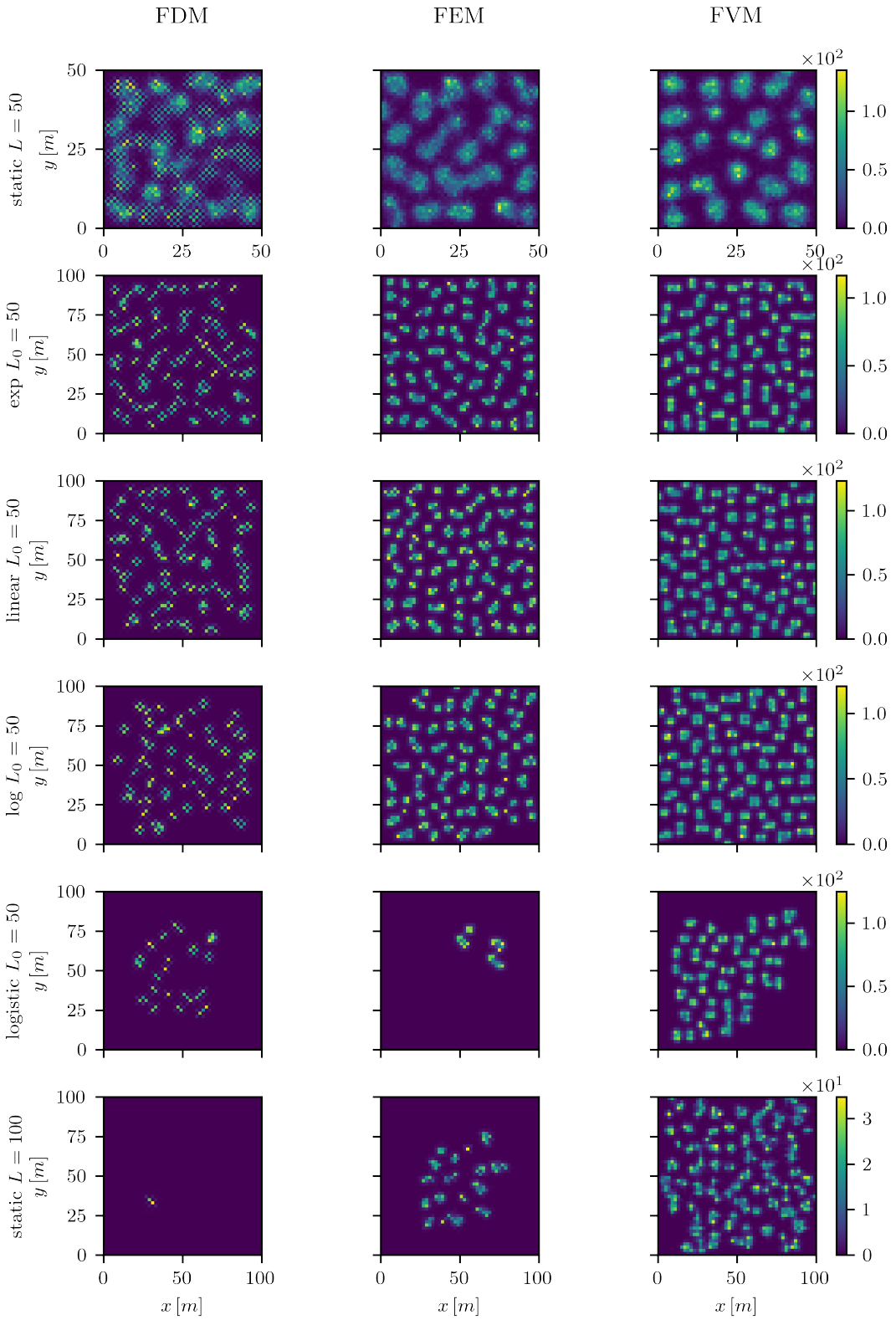


Figure 3.11: Results from simulations of the Gray-Scott system, showing the state of the simulations at $T = 5000\,s$. We use the following reaction rates: $D_u = 0.14\,m^2 s^{-1}$, $D_v = 0.06\,m^2 s^{-1}$, $k_1 = 2 \times 10^{-5}\,s^{-1}$, $k_2 = 7.0\,s^{-1}$, $k_3 = 0.035\,s^{-1}$ and $k_4 = 0.1\,s^{-1}$.

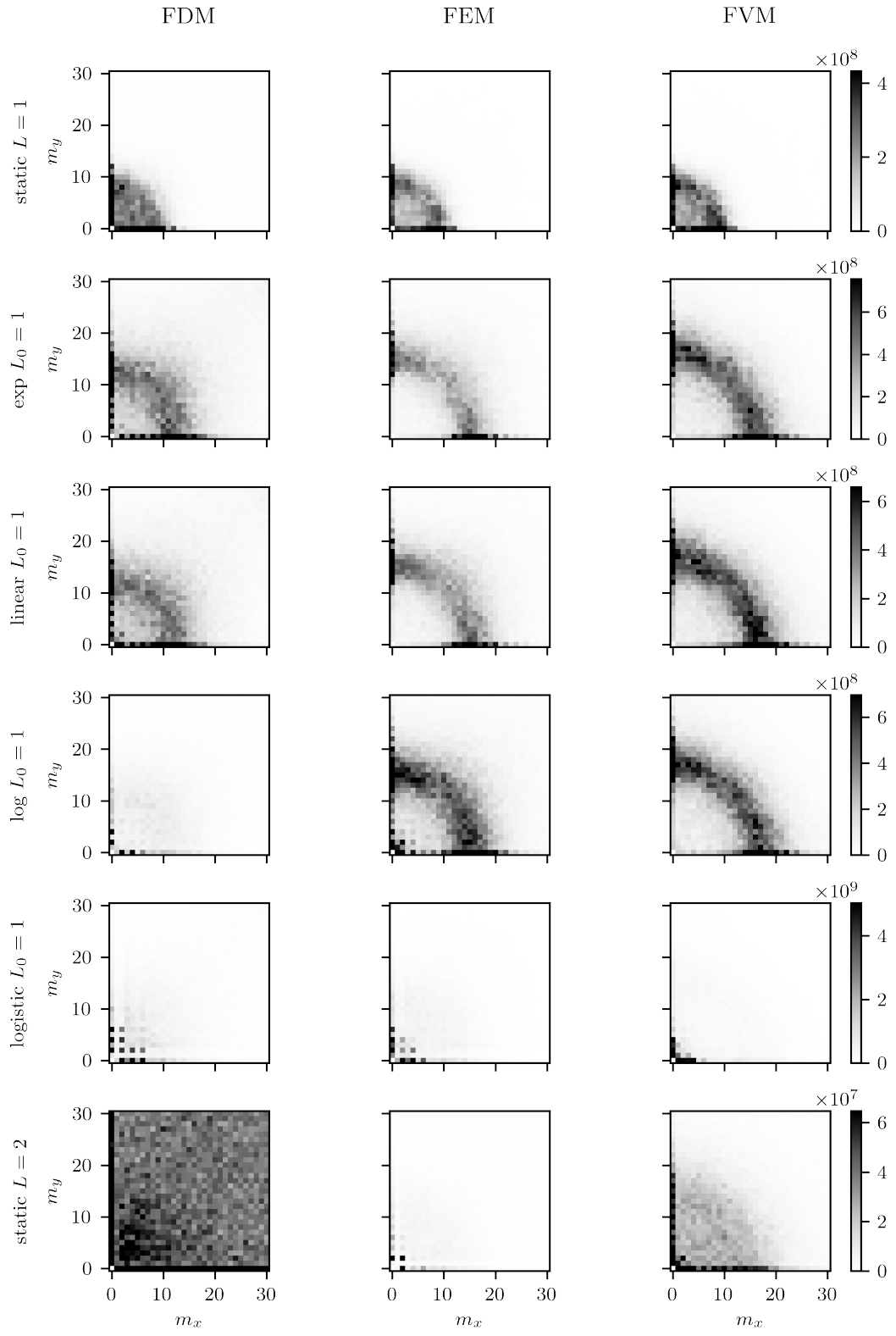


Figure 3.12: Power spectra of simulations of the Gray-Scott system at $T = 5000\,s$ averaged over 100 simulations. We use the following reaction rates: $D_u = 0.14\,m^2 s^{-1}$, $D_v = 0.06\,m^2 s^{-1}$, $k_1 = 2 \times 10^{-5}\,s^{-1}$, $k_2 = 7.0\,s^{-1}$, $k_3 = 0.035\,s^{-1}$ and $k_4 = 0.1\,s^{-1}$.

3.3 Discussion

In this chapter, we extended the RDME framework discussed in Chapter 2 to model domain growth for RD systems on a Cartesian mesh. We looked at different functions that can be used to describe domain growth and we compared simulation results of various RD systems for each domain growth function. The domain growth functions we considered were exponential, linear, logarithmic and logistic. While comparing domain growth functions, we also explored how the different methods of derivation of the jump rates affect simulation results when domain growth is incorporated into the model.

Similarly to Chapter 2, we first looked at a diffusion-only system, where we compared the errors resulting from comparing stochastic simulations to an analytic solution. We picked the growth rates for the domain growth functions such that the domain width, $L(t)$, would be double its initial value at the end of a simulation. The error plots showed that simulations that used FVM-derived jump rates had the best agreement with the analytic solution. The simulations that used FDM- and FEM-derived jump rates had very similar errors to each other, which were noticeably large compared to errors from simulations with FVM-derived jump rates.

Next, we analysed the results of simulating the Schnakenberg system on a growing domain. The different derivations of diffusive jump rates did not seem to affect the patterns formed, similar to the results from Chapter 2. What did make a difference, however, was the growth of the domain – peaks in chemical concentration split over the course of the simulations as the domain grew with a given growth function. Through our simulations we showed that when the domain grows logistically, the dominant wavemode has a higher value than the other growth functions used, throughout the simulations. However, we could also observe that the dominant wavemodes in simulations with logistic

domain growth were lower than the dominant wavemodes in simulations on a static domain that was as large as the size of the domain at the end of simulations with domain growth.

We also looked at the Gray-Scott system that models pattern formation; it gave very different results compared to the Schnakenberg system. Firstly, the choice of derivation of the jump rates heavily influenced simulation results. For some domain growth functions no pattern appeared when FDM- or FEM-derived jump rates were used. Secondly, domain growth in some cases promoted pattern formation and in others, specifically logistic domain growth, it stymied pattern formation.

The results in this chapter demonstrate just how much of an effect domain growth can have on RD systems, specifically on models of pattern formation. Furthermore, we observed further evidence of how important the choice of jump rate derivation is for simulation results. Though many results in this chapter echoed the results from Chapter 2, we saw how domain growth added more interesting cases.

So far in this thesis, we have considered the domain to be discretised by a Cartesian mesh into a rectangular compartments. This limits, to a certain degree, the geometry of the domain. The logical next step is to extend the work we presented thus far to unstructured meshes, so that simulations can be run on domains of more complex shape.

Chapter 4

The RDME framework on unstructured meshes with domain growth

Thus far in this work, for simplicity, we have considered a square domain discretised by square or rectangular compartments. However, square compartments limit the domains on which we can simulate. In this chapter we generalise how we discretise the domain of a RD system by extending our approach to use unstructured meshes. We describe different the derivations of the jump rates given an unstructured mesh, and we explore the impact of different methods using a range of simulation studies. This generalisation allows us to consider the effects of complex domain shapes on pattern formation. Since the aim is to build on the work presented in the previous chapter, we also consider the effects of domain growth on simulations using unstructured meshes.

First, in Section 4.1.1, we introduce the types of meshes we use in this chapter and then, in Section 4.1.2, we outline the algorithms used to generate the jump rates for unstructured

meshes. Second, in Section 4.2, we show simulation results comparing two methods of derivation of the jump rates for unstructured meshes alongside with the investigation of different functions that describe domain growth.

4.1 Mathematical techniques

In this section we describe the mathematical objects and algorithms that are needed for the simulations in this chapter. First, we describe the method we use for domain discretisation. Then we describe the derivations and the corresponding algorithms for the jump rates on unstructured meshes.

4.1.1 Meshes and dual meshes

One of the most common ways to discretise a domain is using a Delauney triangulation [24, 49]. It is often favoured due to the nice properties that the Delauney triangulation has for numerical schemes such as the finite element method (FEM) and finite volume method (FVM). The particular property that is of main interest to us is that all the non-diagonal entries in the stiffness matrix are positive, which results in positive jump rates when using the FEM [19, 46]. Hence, we exclusively use a Delauney triangulation to discretise domains.

First, let $V = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_I\}$ be a set of points in $\mathbb{R}^2$, where we assume that these points are not all collinear. Next, let E be the set of $\binom{I}{2}$ edges between the points in V. A triangulation of the set of points V is a planar straight line graph $G(V, E')$, such that E' is a maximal subset of E such that no two edges of E' intersect each other at any points other than their endpoints [37]. A Delauney triangulation is a special type of triangulation, which has the property that no circumcircle of any triangle in the triangulation contains

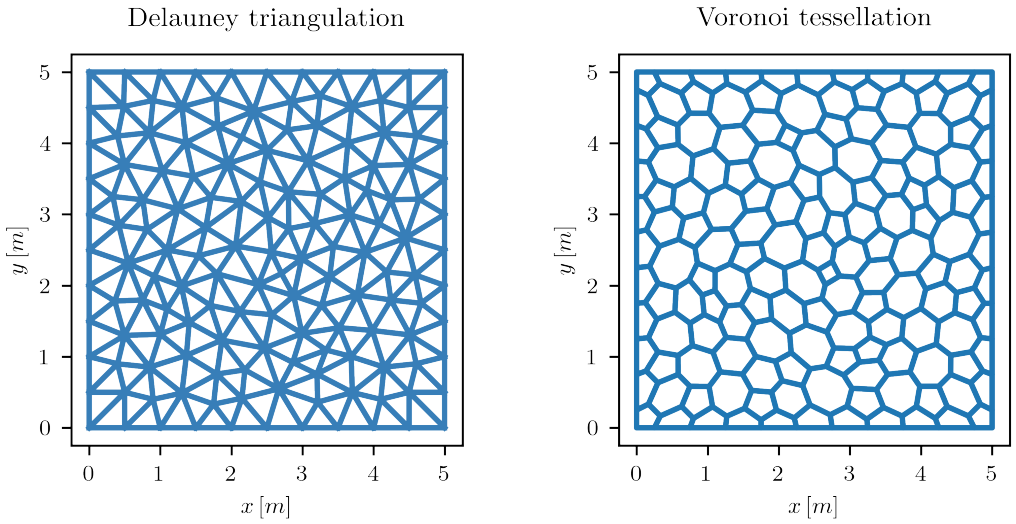


Figure 4.1: Examples of meshes discussed Section 4.1.1. On the left a Delauney triangulation and on the right its dual, Voronoi tessellation.

any of the points in V in its interior [2].

A Delauney triangulation gives us a mesh that connects the centre of each compartment, C_i, and is used to calculate the jump rates between adjacent compartments. However, we also need to determine the boundaries of each compartment. In this respect, the Delauney triangulation has another useful property – that it is related to its Voronoi tessellation, which partitions a domain into regions, that we use to define the boundaries of each compartment. The Voronoi tessellation is called the dual of the Delauney triangulation, where the Voronoi tessellation has a point for each triangle in the Delauney triangulation [2, 49]. Figure 4.1 shows an example of a square domain $[0, 5\,m]^2$ triangulated using a Delauney triangulation, together with the corresponding Voronoi tessellation.

4.1.2 Jump rates on unstructured meshes

We have established the Delauney triangulation and Voronoi tessellation in the previous section, and we use them in this section to derive jump rates. Much like in Chapter 2,

we use numerical schemes to derive the rates of molecules jumping between adjacent compartments. Since we build on the work presented in the previous chapter, we consider a generalised framework for growing domains. Note that we cannot use the finite difference method (FDM) or first exit time (FET) method to do this: the FDM does not extend to unstructured meshes and, as discussed in Chapter 3, the FET method does not extend to growing domains. Hence we focus on the derivations of jump rates using the FEM and the FVM numerical schemes. We note that the derivation of the jump rates on a growing domain shown in Chapter 3 is valid for unstructured meshes and follows in exactly the same way.

As stated in the Section 4.1.1, we use Delauney triangulations and Voronoi tessellations for simulations on unstructured meshes. The FEM can be used on other types of meshes, however, if the mesh is not of the Delauney type negative jump rates can arise. Specifically, when the sum of angles opposite of an edge is greater than π radians, as shown in Figure 4.2, negative jump rates arise [19]. As we use the jump rates to calculate probabilities, and the probability of an event cannot be negative, we must use a mesh that only gives non-negative jump rates, namely a Delauney triangulation.

In the derivation of the jump rates using the FVM we instead require the Voronoi tessellation. However, given the Voronoi tessellation we also have the Delauney triangulation, as one is the dual of the other, which means that the edges of the Delauney triangulation arise from the perpendicular bisectors of neighbouring circumcentres of the compartments.

In both derivations we start with the diffusion equation, much like in Chapter 2,

$$\frac{\partial p(\mathbf{x},t)}{\partial t} = D\nabla^2 p(\mathbf{x},t) \quad \text{for} \quad \mathbf{x} \in \Omega, \tag{4.1}$$

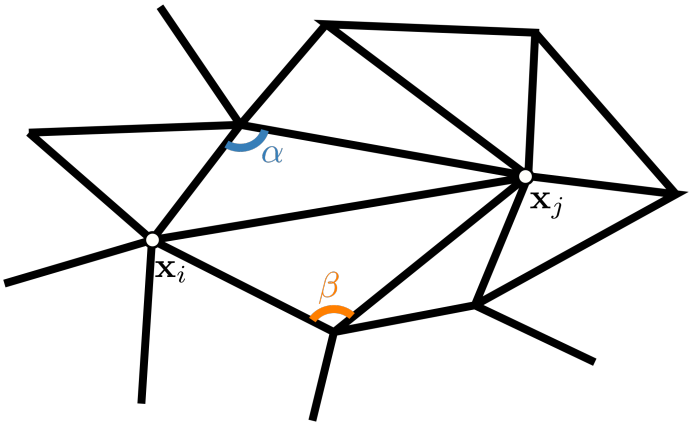


Figure 4.2: Schematic diagram of a part of a non-Delauney mesh, where the sum of the opposite angles, α and β, is greater than π radians, and so would yield a negative jump rate between x_i and x_j.

describing the probability density of a molecule to be at position $\mathbf{x}(t)$ at time t, with initial condition

$$p(\mathbf{x}, 0) = p_0(\mathbf{x})\,, \tag{4.2}$$

and Neumann boundary conditions

$$(\mathbf{n} \cdot \nabla)p = 0 \quad \text{for} \quad \mathbf{x} \in \partial\Omega\,. \tag{4.3}$$

4.1.2.1 Finite element method

We multiply Equation (4.1) by a test function $w(\mathbf{x})$, which is a function in Sobolev space [18] (*i.e.* is bounded and has bounded first partial derivatives), and integrate over the whole domain Ω:

$$\begin{aligned} \int_\Omega \frac{\partial p}{\partial t} w \,\mathrm{d}x\,\mathrm{d}y\,, &= D\int_\Omega w\nabla^2 p \,\mathrm{d}x\,\mathrm{d}y \\ &= D\int_\Omega \left[\nabla\cdot(w\nabla p) - \nabla p\cdot\nabla w\right] \mathrm{d}x\,\mathrm{d}y. \end{aligned} \tag{4.4}$$

Now we use the divergence theorem to give

$$\int_\Omega \frac{\partial p}{\partial t} w \,\mathrm{d}x\,\mathrm{d}y = D\int_{\partial\Omega} \mathbf{n}\cdot[w\nabla p]\,\mathrm{d}s - D\int_\Omega \nabla p\cdot\nabla w\,\mathrm{d}x\,\mathrm{d}y, \tag{4.5}$$

where $\mathrm{d}s$ is the line element of the domain boundary, $\partial\Omega$, and $\mathbf{n}$ is the normal vector to the domain boundary at the point $\mathbf{x}$. Due to the Neumann boundary conditions, the first term on the right-hand side of Equation (4.5) vanishes and we are left with

$$\int_\Omega \frac{\partial p}{\partial t} w \,\mathrm{d}x\,\mathrm{d}y = -D\int_\Omega \nabla p\cdot\nabla w\,\mathrm{d}x\,\mathrm{d}y. \tag{4.6}$$

Next we express the functions p, $\partial p/\partial t$ and w in terms of the basis functions φ_j,

$$\begin{aligned} w(x,y) &= \varphi_j(x,y), & (4.7)\\ p(x,y,t) &\approx \sum_{i=1}^{I} \tilde{p}_i(t)\varphi_i(x,y), & (4.8)\\ \frac{\partial p}{\partial t} &\approx \sum_{i=1}^{I} \frac{\mathrm{d}\tilde{p}_i}{\mathrm{d}t}(t)\varphi_i(x,y), & (4.9) \end{aligned}$$

where the φ_i needs to satisfy the following condition:

$$\varphi_j\left(x_i,y_i\right) = \begin{cases} 1 & \text{if} \quad i=j; \\ 0 & \text{if} \quad i\neq j. \end{cases} \tag{4.10}$$

Substituting Equations (4.7), (4.8) and (4.9) into Equation (4.6) gives, for each j,

$$\sum_{i=1}^{I}\left(\int_\Omega \varphi_i\varphi_j\,\mathrm{d}x\,\mathrm{d}y\right)\frac{\mathrm{d}\tilde{p}_i}{\mathrm{d}t} = -\sum_{i=1}^{I}\left(\int_\Omega D\nabla\varphi_i\cdot\nabla\varphi_j\,\mathrm{d}x\,\mathrm{d}y\right)\tilde{p}_i, \tag{4.11}$$

which can be expressed as a matrix equation

$$M\frac{\mathrm{d}\tilde{\mathbf{p}}}{\mathrm{d}t} = -N\tilde{\mathbf{p}}\,, \tag{4.12}$$

where

$$\begin{aligned}
\tilde{\mathbf{p}} &= (\tilde{p}_1, \tilde{p}_2, \ldots, \tilde{p}_I), && (4.13)\\
M_{ij} &= \int_\Omega \varphi_i\,\varphi_j\,\mathrm{d}x\,\mathrm{d}y, && (4.14)\\
N_{ij} &= \int_\Omega D\nabla\varphi_i\cdot\nabla\varphi_j\,\mathrm{d}x\,\mathrm{d}y. && (4.15)
\end{aligned}$$

Unlike in the FEM derivation in Chapter 2, we cannot assume translational invariance or any rotational symmetries for a given mesh. As a result, we need an algorithm to calculate the entries of matrices M and N. Algorithm 3 shows how the derivation of both matrices proceeds computationally. To establish the jump rates between C_i and C_j when $i \neq j$, we calculate

$$\lambda_{ij} = \frac{-N_{ij}}{\sum_{j=1}^{I} M_{ij}}\,. \tag{4.16}$$

Given that we are working with linear finite elements, no quadrature rule is used and the integrals are calculated per compartment exactly.

4.1.2.2 Finite volume method

Now we derive the jump rates using the FVM numerical scheme. Much like in the FVM derivation in Chapter 2, we start by integrating Equation (4.1) over the area of compartment C_i, which is defined as the area of the Voronoi cell around point $\mathbf{x}_i$. Hence, we

Algorithm 3 Algorithm to generate entries of matrices M and N in the FEM derivation of the jump rates.

Let $i \in \{1, \ldots, I\}$ be an index of a point in the mesh and let φ_i be the corresponding basis function of the linear Lagrange finite element *i.e.* piecewise linear polynomials that satisfy Equation (4.10).
for i in $\{1, \ldots, I\}$ **do**
Determine the set B_i which consists of the neighbouring vertices of the vertex i, *i.e.* there exists an edge that both vertices share. Notice that this set also includes the vertex i.
for j in B_i **do**
Let E be the set of elements in the triangulation such that both vertex i and vertex j is one of their vertices. Then
$M_{ij} = \sum_{e \in E} \int_e \varphi_i \, \varphi_j \, \mathrm{d}x \, \mathrm{d}y$,
$N_{ij} = \sum_{e \in E} \int_e D \nabla \varphi_i \cdot \nabla \varphi_j \, \mathrm{d}x \, \mathrm{d}y$.
end for
end for

have

$$\int_{C_i} \frac{\partial p}{\partial t} \, \mathrm{d}x \, \mathrm{d}y = \int_{C_i} \nabla^2 p \, \mathrm{d}x \, \mathrm{d}y \, . \tag{4.17}$$

Next, we use the divergence theorem to arrive at

$$\int_{C_i} \frac{\partial p}{\partial t} \, \mathrm{d}x \, \mathrm{d}y = \int_{\partial C_i} \mathbf{n} \cdot \nabla p \, \mathrm{d}s \, , \tag{4.18}$$

where $\mathrm{d}s$ is a line element of the compartment boundary, ∂C_i, and $\mathbf{n}$ is the normal vector to the compartment boundary at the point $\mathbf{x}$. We continue by separating the integral on the right-hand side of Equation (4.18) into integrals along segments Γ_{ij} of the boundary ∂C_i so that

$$\int_{C_i} \frac{\partial p}{\partial t} \, \mathrm{d}x \, \mathrm{d}y = \sum_{j \sim i} \int_{\Gamma_{ij}} \mathbf{n}_{ij} \cdot \nabla p \, \mathrm{d}s \, , \tag{4.19}$$

where $\Gamma_{ij} = \partial C_i \cap \partial C_j$ and $j \sim i$ means compartments neighbouring C_i. We approximate the integral on the right-hand side of Equation (4.19) by a two-point flux approximation,

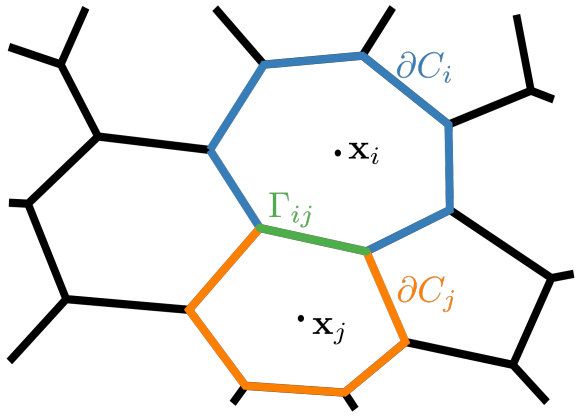


Figure 4.3: Schematic diagram for the FVM derivation of the jump rates on an unstructured mesh.

which is first-order accurate [49]

$$A_i \frac{\partial p_i}{\partial t} = \sum_{j \sim i} \frac{p_j - p_i}{|\mathbf{x}_i - \mathbf{x}_j|} \gamma_{ij} \,, \tag{4.20}$$

where γ_{ij} is the length of the segment Γ_{ij}. Figure 4.3 shows a schematic diagram of the quantities involved in the derivation outlined in this section.

Similar to the FEM derivation in this chapter, we can no longer rely on any symmetries of the mesh that discretises the domain Ω, and so we state the computational algorithm we use to generate the jump rates for any given Delauney triangulation and its corresponding Voronoi tessellation. Algorithm 4 shows how to calculate the jump rates between all the adjacent compartments. We note that the areas of the compartments in the Voronoi tessellation, A_i in Algorithm 4, can be calculated using the shoelace formula [36]. Furthermore, the distance d_{ij} is the distance between points in the Delauney triangulation, while γ_{ij} is the distance between the relevant points in the Voronoi tessellation.

Algorithm 4 Algorithm to generate the jump rates, λ_{ij}, using the FVM.

Let $i \in \{1, \dots, I\}$ be an index of a point in the mesh
for i in $\{1, \dots, I\}$ **do**
  Determine the set B_i which consists of the neighbouring vertices to the vertex i (excluding i)
  for j in B_i **do**
    Let A_i be the area of the compartment i in the Voronoi tessellation (C_i).
    Let d_{ij} be the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$.
    Let γ_{ij} be the length of the boundary between compartments i and j.
    Then $\lambda_{ij} = \dfrac{\gamma_{ij}}{A_i d_{ij}}$.
  end for
end for

4.2 Case studies

In this section, we explore effects of using different meshes on the RD systems we considered thus far, and the effects of the FEM- and FVM-derived jump rates on simulations results. As in the previous chapter, we also consider different domain growth functions and how they affect simulation results in conjunction with the domain being discretised by non-square compartments. Initially, we consider a square domain of size $\Omega_t = [0, L(t)\, m]^2$, as we have done so far in this thesis. However, the main motivation for using shapes other than squares to discretise a domain of a RD system is to allow for complex domain shapes that are more suited to model a specific tissue or organism, and so we also consider other domain shapes, such as an annulus. The structure of this section is very similar to the structure of the previous chapter, in that we first consider a diffusion-only system and look at the errors between the simulation results and the associated analytic solution. Next, we look at the Schnakenberg system of reactions and the effects of using unstructured meshes on the simulation output. We consider static and growing domains at the same time so we can compare the simulation results for both cases.

4.2.1 Diffusion only

As in Section 2.2.1 and Section 3.2.1, let us look first at a RD system where molecules only diffuse on the domain $\Omega_t = [0, L(t)\,m]^2$. The domain is discretised by an unstructured mesh and we compare the simulations on that mesh with simulation results from Section 3.2.1, where we use a structured mesh. Both structured and unstructured meshes can be seen in Figure 4.4. When generating the unstructured mesh, we choose the average compartment size such that it is very close to the compartment size in the structured mesh. Figure 4.5 shows the histogram of compartment sizes in the unstructured mesh compared to the value of compartment size in the structured mesh we use in this section. The value of compartment size for the structured mesh, 0.0625, was chosen such that it lies approximately in the middle of the distribution of compartment sizes for the unstructured mesh, which has the mean 0.0585.

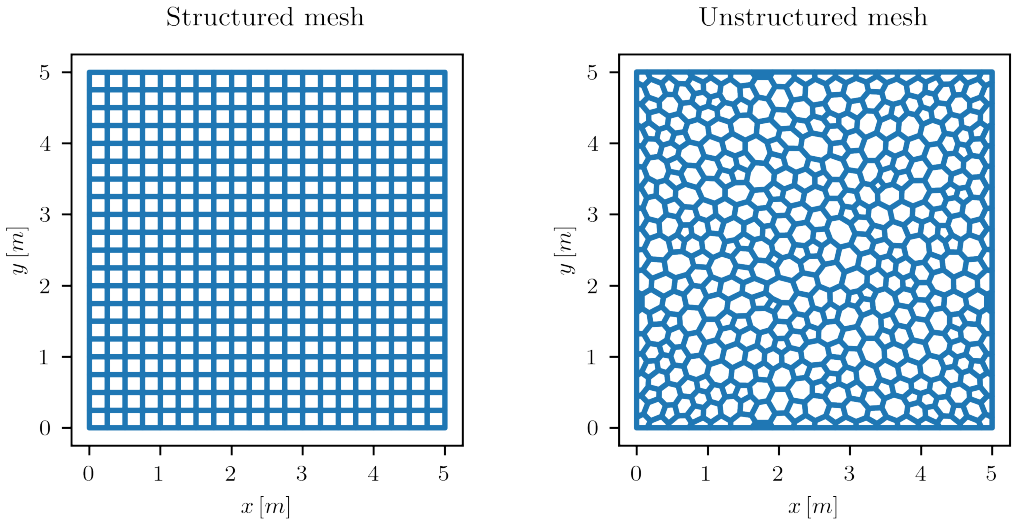


Figure 4.4: On the left is the structured mesh we considered in Section 3.2.1 and on the right is the unstructured mesh we consider in Section 4.2.1.

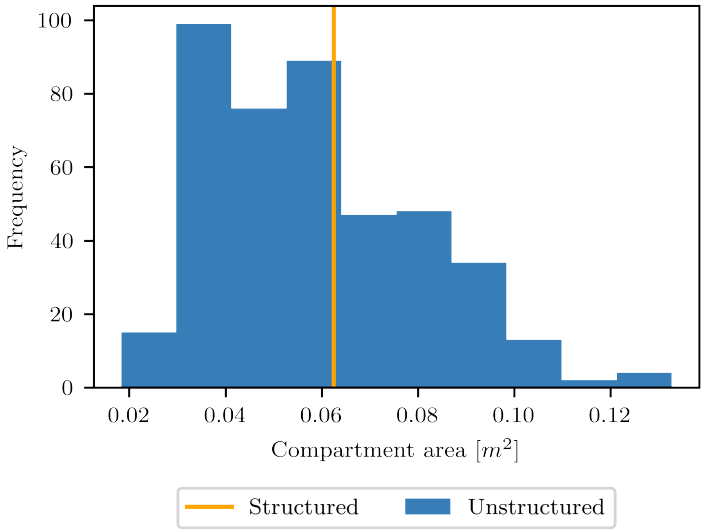


Figure 4.5: Histogram of compartment areas for the unstructured mesh used in this section alongside an orange vertical line which is placed at the value of compartment area for the structured mesh used in this section.

We compare the result of a diffusion-only simulation on an unstructured mesh to the corresponding analytic solution in Figure 4.6. The simulation result appears to agree well with the analytic solution. To be able to compare the different methods of derivation of the jump rates in this context, we again use the error function described by Equation (2.59).

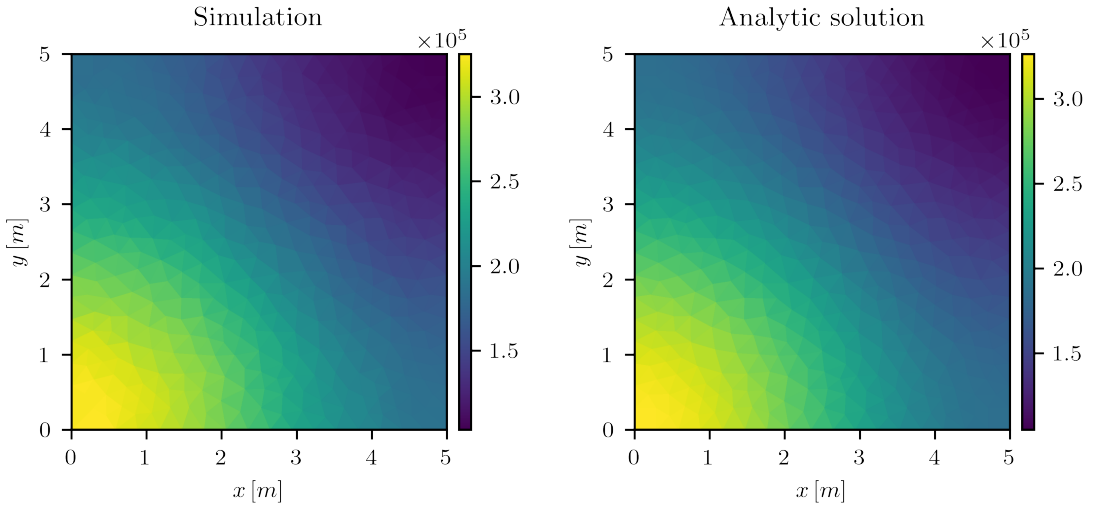


Figure 4.6: Heat maps of simulation output and analytic solution to Equation (2.60) at $T = 5.0$. The domain, $\Omega = [0, 5\,m]^2$, is discretised by an unstructured mesh and 5×10^6 molecules placed in the compartment next to the origin. FVM-derived jump rates are used for the stochastic simulation.

Figure 4.7 shows how the errors of simulations are affected by the domain discretised by unstructured meshes compared to simulations on a domain discretised by structured (Cartesian) meshes. The errors in simulations with FEM-derived jump rates on a domain discretised by an unstructured mesh are higher compared to simulations with FVM-derived jump rates, much like in Sections 2.2.1 and 3.2.1 where a Cartesian mesh was used to discretise the domain. Also, the errors are significantly higher for simulations on an unstructured mesh compared to simulations on a structured mesh when FEM-derived jump rates are used.

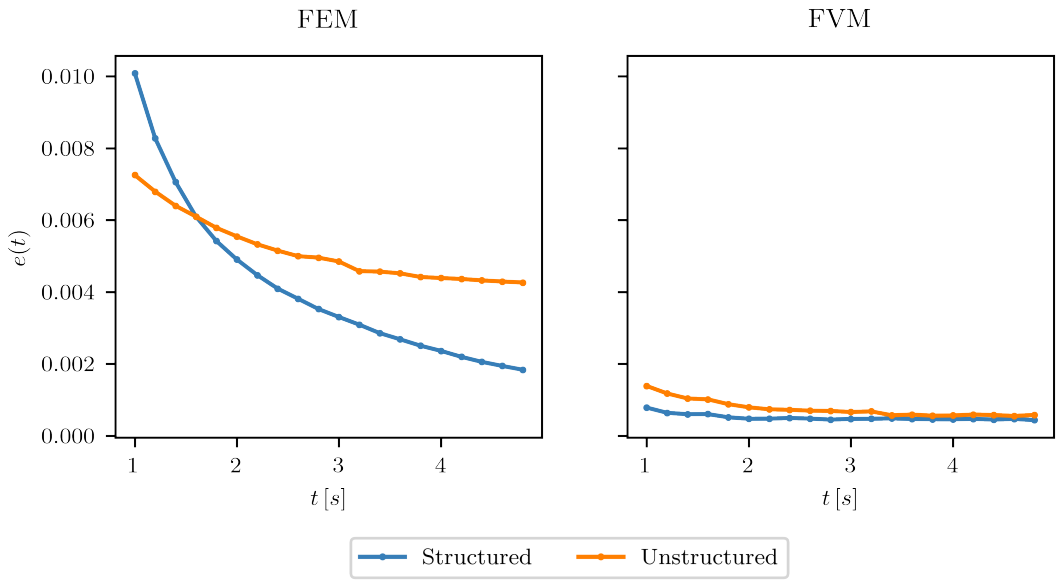


Figure 4.7: Plots of the errors, $e(t)$, on structured and unstructured meshes. In the plots on the left-hand side, we consider errors from simulations using FEM-derived jump rates, while on the right-hand side we consider errors from simulations using FVM-derived jump rates. The diffusion coefficient for the simulations was $D = 1\,m^2 s^{-1}$ and 5×10^6 molecules are initially placed in the bottom-right compartment.

4.2.1.1 Including domain growth

We also investigate the effects of domain growth on unstructured meshes. Similarly to Chapter 3, we consider the domain growth functions seen in Figure 4.8. Figure 4.9 shows how domain growth affects simulation results for a range of growth functions, and it appears that domain growth has similar effects on unstructured meshes as on structured meshes. The errors are higher for the static domain case of size $L = 10\,m$, compared to other cases, but this again is as expected because the number of compartments used is the same in all simulations, and as a result the discretisation of the domain is more coarse.

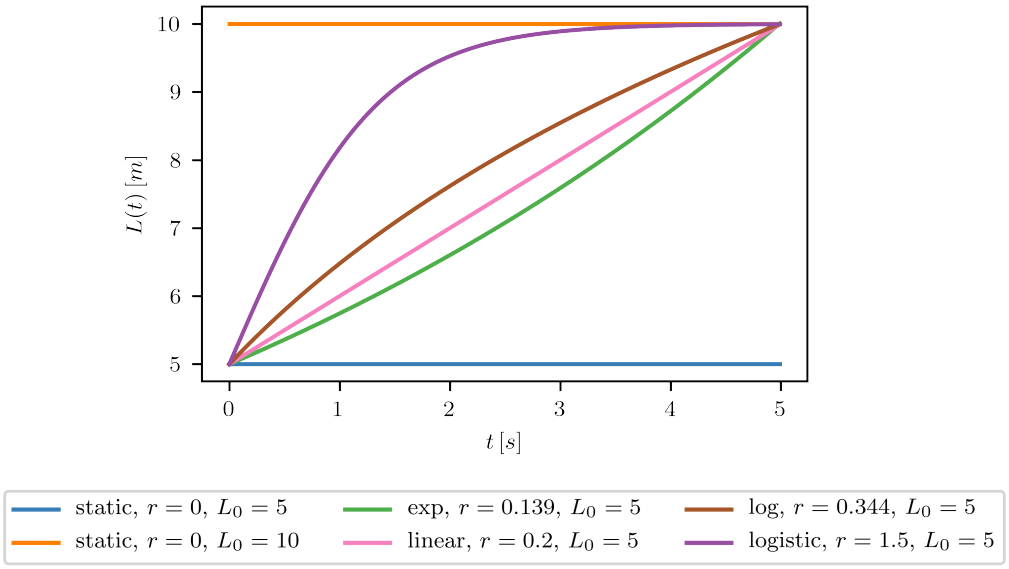


Figure 4.8: Plot of the domain growth functions we consider in Section 4.2.1. We consider two static cases: $L_0 = 5\,m$ and $L_0 = 10\,m$. We chose the growth rate, r, such that for simulations where the domain is growing, the domain size starts off at one static case under consideration ($L_0 = 5\,m$) and by the end of the simulation is the same size as the other static case ($L_0 = 10\,m$).

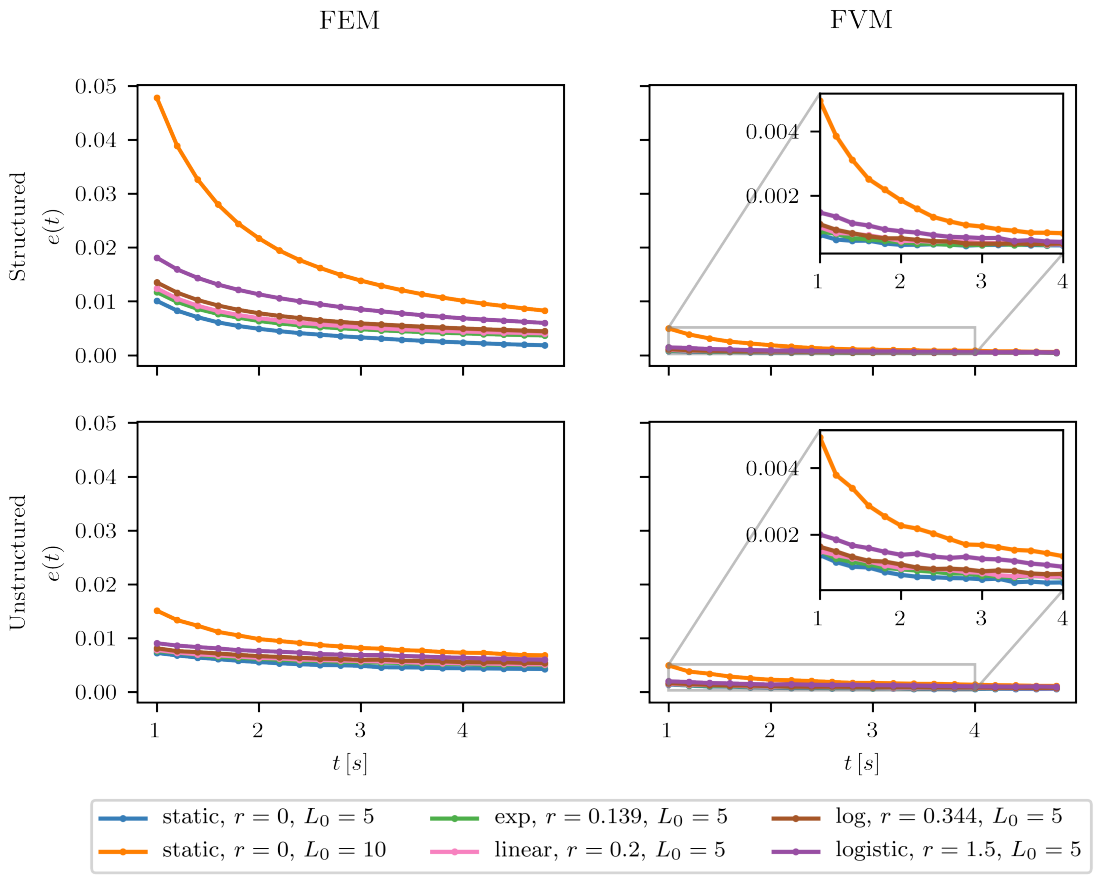


Figure 4.9: Plots of the errors, $e(t)$, on structured and unstructured meshes. In each plot we consider both static and growing domain cases. In the plots on the left-hand side, we consider errors from simulations using FEM-derived jump rates, while on the right-hand side we consider errors from simulations using FVM-derived jump rates. In the top row, we consider simulations run on domains discretised by a Cartesian mesh, while in the bottom row we consider unstructured meshes. The diffusion coefficient for the simulations was $D = 1\,m^2 s^{-1}$ and 5×10^6 molecules were initially placed in the bottom-right compartment.

4.2.2 Schnakenberg kinetics

Now we use the Schnakenberg system of reactions to investigate the effects of modelling choices on a model of pattern formation. First, we consider a square domain for simulations of the Schnakenberg RD system and compare simulations with different domain growth functions that use either FEM- or FVM-derived jump rates. We then compare the power spectra of many simulations to determine any trends in the simulation results.

Second, we extend our exploration to consider other domain shapes, such as circular domains.

When we consider domain growth in this section, we look at the usual set of functions we considered in the previous section and in Chapter 3, that being exponential, linear, logarithmic and logistic, alongside static domains. Figure 4.10 shows the domain growth functions and the rates we use to investigate the effects of domain growth on simulations when the domain is discretised by unstructured meshes.

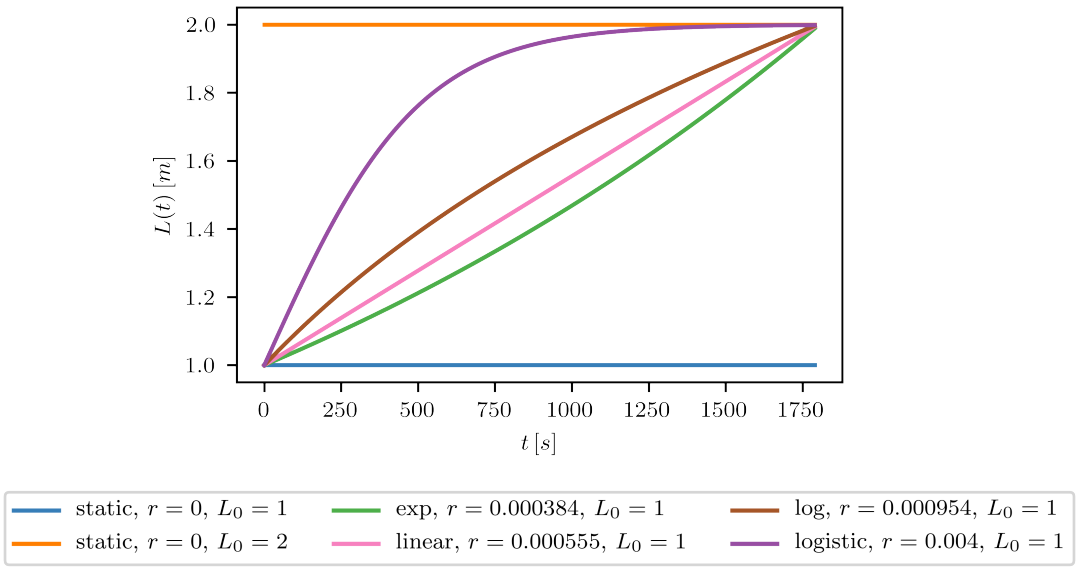


Figure 4.10: Plot of domain growth functions considered in Section 4.2.2.

4.2.2.1 Square domain

For comparison with results in Section 3.2.2, we first consider a square domain, $\Omega_t = [0, L(t)\,m]^2$, except now the domain is discretised by an unstructured mesh. Figure 4.11 shows examples of simulation results at time $t = 1800\,s$ for different domain growth functions where either FEM- or FVM-derived jump rates were used. We can see that there is no discernable difference between simulations that used FEM- or FVM-derived jump rates, at least in these plots. What we can see though is that the number of spots

is higher when the domain is growing or static with $L = 2\,m$.

To better quantify the differences between the methods of derivation of the jump rates and the differences between domain growth functions we look at the power spectra of the simulations we used in the previous two chapters. Figure 4.12 shows the power spectra that result from averaging over 100 simulations, as well as plots of the possible wavemodes predicted by LSA, where we used the size of the domain at the end of the simulations in our calculations. The first thing we see is that method of derivation of the jump rates seems to have no discernable effect on the power spectra, whichever domain growth function is used. The most striking aspect of the plots of the power spectra is how very specific wavemodes are dominant across all types of domain growth function, which is in contrast with our results in Chapter 3, where the power spectra plots (when averaged over 100 simulations) displayed a range of dominant wavemodes a certain distance from the origin. Moreover, the wavemodes that are excited when the domain is static and of size $L = 1\,m$ are outside of the range of possible wavemodes predicted by LSA (when $L = 1\,m$), but are the same wavemodes that are present in other simulations, whether the domain is growing or not.

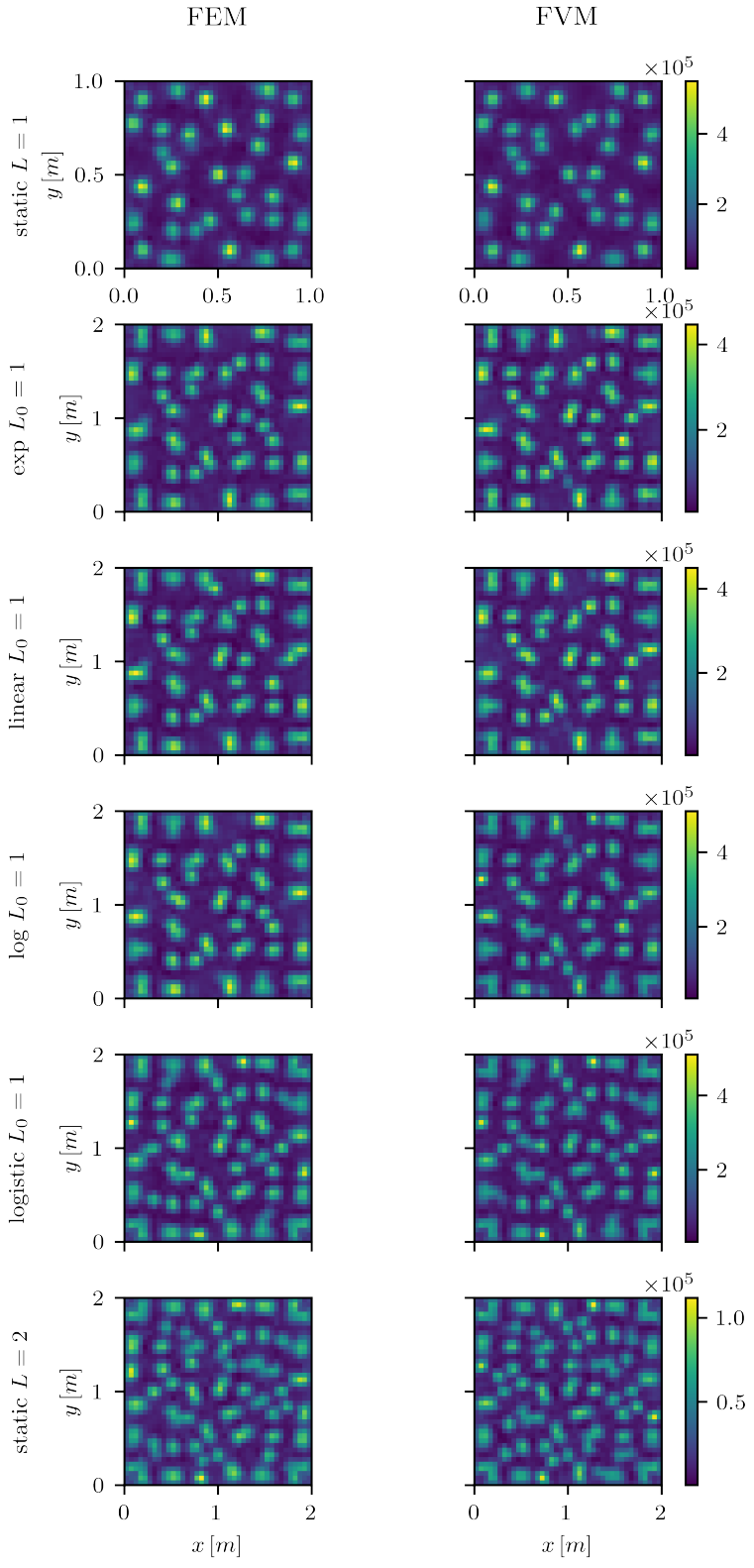


Figure 4.11: Results from simulations of the Schnakenberg system, showing the state of the simulations at $T = 1800\,s$ with the following parameters: $D_u = 1 \times 10^{-5}\,m^2 s^{-1}$, $D_v = 0.001\,m^2 s^{-1}$, $k_1 = 1.0\,m^{-2} s^{-1}$, $k_2 = 0.02\,s^{-1}$, $k_3 = 1 \times 10^{-6}\,m^4 s^{-1}$, $k_4 = 3.0\,m^{-2} s^{-1}$.

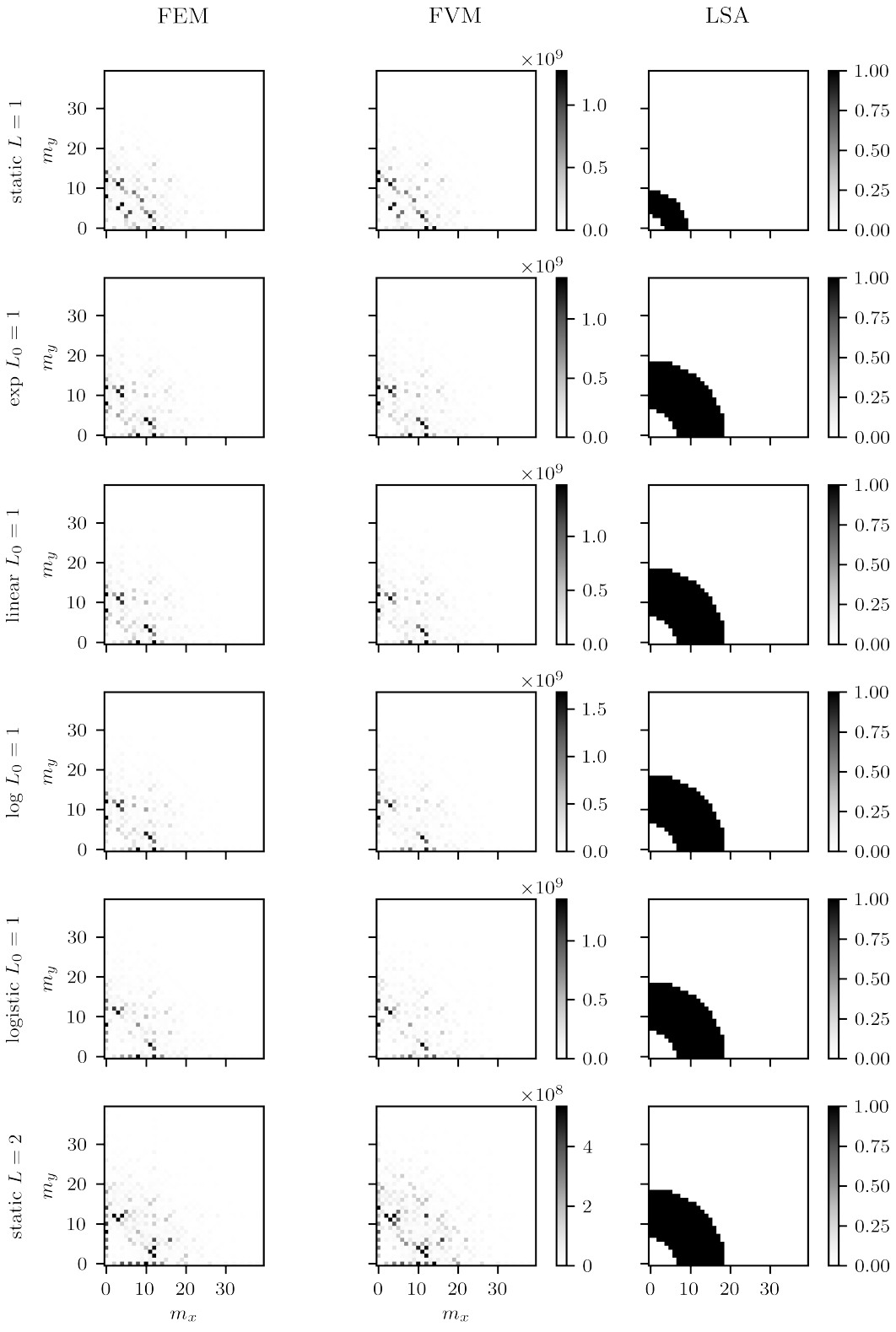


Figure 4.12: Power spectra of simulations of the Schnakenberg system at $T = 1800\,s$ averaged over 100 simulations with the following parameters: $D_u = 1 \times 10^{-5}\,m^2s^{-1}$, $D_v = 0.001\,m^2s^{-1}$, $k_1 = 1.0\,m^{-2}s^{-1}$, $k_2 = 0.02\,s^{-1}$, $k_3 = 1 \times 10^{-6}\,m^4s^{-1}$, $k_4 = 3.0\,m^{-2}s^{-1}$.

We hypothesise that the wavemodes are associated with the specific mesh chosen for the simulations. To see how different meshes affect the power spectra, we ran simulations on two different meshes and calculated the averaged power spectra. Figure 4.13 shows the meshes used in the top row and the corresponding power spectra below each mesh. We see that the excited wavemodes take only a sparse range of values, unlike for the power spectra seen in Chapter 3. Furthermore, the wavemodes in both power spectra plots in Figure 4.13 are outside of the range of possible wavemodes predicted by LSA (which predicts that wavemodes should not be higher than 10 in either x- or y-directions). Our results from Chapter 2 indicate that LSA is reliable for simulations on structured meshes with FEM- and FVM-derived jump rates. However, the results presented in this chapter demonstrate that for unstructured meshes LSA is unreliable as a predictor of the range of potential patterns.

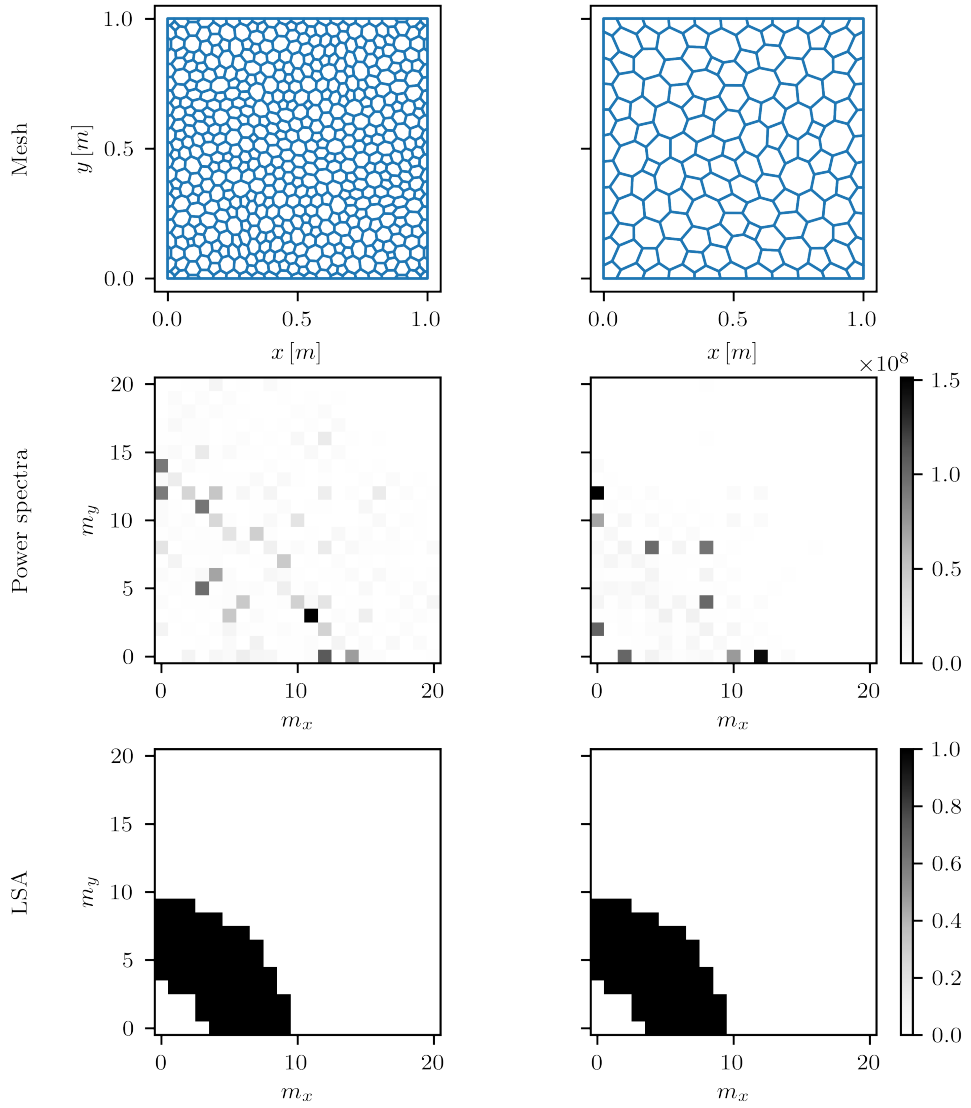


Figure 4.13: Comparison of power spectra of simulations of the Schnakenberg system at $T = 1800\,s$ averaged over 100 simulations with the following parameters: $D_u = 1 \times 10^{-5}\,m^2s^{-1}$, $D_v = 0.001\,m^2s^{-1}$, $k_1 = 1.0\,m^{-2}s^{-1}$, $k_2 = 0.02\,s^{-1}$, $k_3 = 1 \times 10^{-6}\,m^4s^{-1}$, $k_4 = 3.0\,m^{-2}s^{-1}$.

4.2.2.2 Other domain shapes

One of the main motivations of this chapter was to develop and explore the use of unstructured meshes to discretise a given domain so that we could simulate models on complicated domains. In this section, we showcase one example of a non-square domain, and how by changing a parameter associated with the domain shape we can influence the

patterns formed.

The shape we consider is an annulus, which is described by an inner radius, r_i, and outer radius, r_o. Figure 4.14 shows the shape of the domain as we increase r_i and how pattern formation is affected. When $r_i = 0.1$, the patterns formed look a lot like those when the domain is square. However, as we increase the inner radius to $r_i = 0.25$, the spots become ordered into two layers: one adjacent to the inner boundary and the other adjacent to the outer boundary. When we look at the patterns formed when $r_i = 0.3$ in Figure 4.14, we see just one layer of spots closely packed together. Finally, the right-most plot in Figure 4.14 shows that when the width of the annulus is 0.1, *i.e.* the inner radius is $r_i = 0.4$, there is only one layer of spots, but the spacing between spots has increased compared to the $r_i = 0.3$ case. The example of pattern formation on an annulus highlights just how much of an impact the shape of the domain can have on pattern formation on top of all the other modelling decisions we discussed so far in this work.

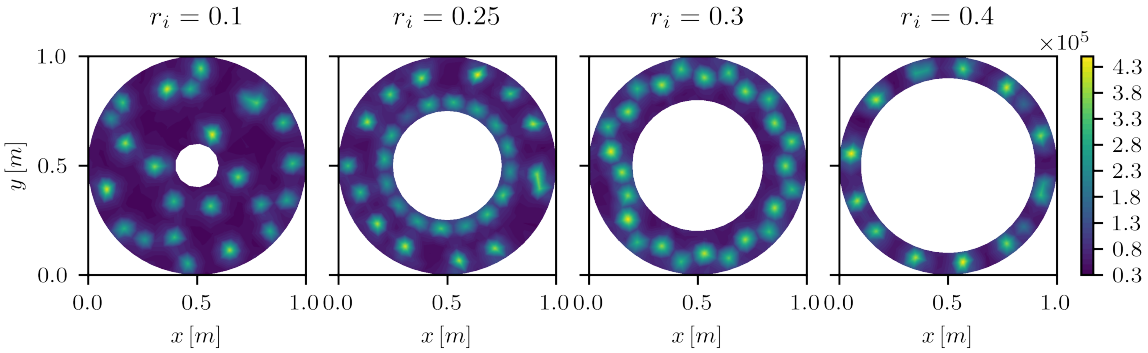


Figure 4.14: Examples of simulations of the Schnakenberg RD system on annulus domain centered on $\mathbf{x} = (0.5, 0.5)$ with outer radius $r_o = 0.5$ and varying the inner radius. The following parameters were used: $D_u = 1 \times 10^{-5}\, m^2 s^{-1}$, $D_v = 0.001\, m^2 s^{-1}$, $k_1 = 1.0\, m^{-2} s^{-1}$, $k_2 = 0.02\, s^{-1}$, $k_3 = 1 \times 10^{-6}\, m^4 s^{-1}$, $k_4 = 3.0\, m^{-2} s^{-1}$ and $T = 1800\, s$.

4.3 Discussion

In this chapter, we introduced an approach to discretise the domain using unstructured meshes and used FEM and FVM methods to derive appropriate jump rates within the RDME framework. We also used the approach derived in Chapter 3 to investigate the effects of domain growth when the domain is discretised by an unstructured mesh. Similarly to Chapter 3 we looked at the following domain growth functions: exponential, linear, logistic and logarithmic. We used diffusion-only and the Schnakenberg systems to investigate the effects of unstructured discretisations of the domain, the shape of the domain, the method of derivation of the jump rates and growth of the domain on simulation results.

We first considered a diffusion-only system and looked at the error metric introduced in Chapter 2 to compare the results of simulations with the analytic solution of the corresponding macroscale PDE. We only compared simulations with FEM- and FVM-derived jump rates, as the FDM does not extend to unstructured meshes and the FET, as discussed in Chapter 3, does not extend to simulations on a growing domain. The results in this chapter mirrored the results from the previous two chapters, where the simulations with FVM-derived jump rates had smaller error compared to those with FEM-derived jump rates.

When we considered the Schnakenberg RD system, we were able to see just how much of an effect the choice of the mesh had. The power spectra when the domain was discretised by an unstructured mesh bear very little resemblance to the power spectra discussed in Chapter 3 for the Schnakenberg system, having only relatively few wavemodes present. Moreover, the wavemodes present in the plots of the power spectra were consistently present irrespective of whether the domain was growing or was static, which indicated

that those wavemodes were specific to the mesh. When we used a different mesh to discretise the domain, we saw different wavemodes in the power spectra plots, further indicating that certain wavemodes are associated with each choice of mesh. Lastly, we looked at an annulus domain where we varied the inner radius and we saw how the shape of the domain affected patterns formed.

We also attempted to investigate the effects of using unstructured meshes on the Gray-Scott RD system. However, we could not get any patterns to form under any conditions when an unstructured mesh was used to discretise the domain. Based on our results from Chapter 3, the Gray-Scott RD system was more affected by modelling choices compared to the Schnakenberg system. Hence, it could be that Gray-Scott system is too sensitive for patterns to form on unstructured meshes. We leave further exploration of this observation for future work.

Chapter 5

An efficient computational implementation of the RDME framework

As seen so far in this thesis, many modelling choices in the RDME framework can have a significant effect on the simulation results. However, there is one more aspect that can have even stronger effects and that is if the software used to generate results has a fault, more commonly known as a bug. Hence, we dedicate a chapter to describing the software engineering principles used, and scientific software packages developed during this project to simulate RDME models.

Reproducibility is one of the most fundamental aspects of science. Yet, at the time of writing this thesis, we are living in a reproducibility crisis. More than 70% of scientists have, at some point, failed to reproduce their own or other scientists' results [3]. Many fields in science rely on software, to either analyse the results of experiments or produce *in-silico* results, and it is very often in the development and use of this software that

reproducibility challenges arise [54]. Taking this in mind, reproducibility was of high priority over the course of this project.

Two ways to assure reproducibility are through the development of open-source software and use of unit testing. A software is considered open-source if all the source code is publicly available, so that if it is sufficiently well documented anyone can use it and understand how it works, if need be. Unit-testing is a software development methodology that ensures that each individual component of a software package works as expected. To make sure that the results presented in this thesis are reproducible, the software packages that were developed and used to generate the results presented in Chapters 2–4 are open-source and include unit tests in their repositories. In this chapter, we present the main characteristics of each piece of software, showcase how they work in practice and how they are tested.

This chapter is divided into two sections: Section 5.1 is concerned with the software developed for simulations using the RDME framework; and Section 5.2 is concerned with the software developed for generating unstructured meshes.

Parts of the work presented in this chapter have been published in the peer reviewed journal *Journal of Open Source Software*. The full reference is Bartmanski, B. J., and Baker, R. E. (2020). StoSpa2: A C++ software package for stochastic simulations of spatially extended systems. Journal of Open Source Software, 5(50), 2293. The software is freely available for download at https://github.com/BartoszBartmanski/StoSpa2.

5.1 StoSpa2

As discussed in Chapter 1, the CME and RDME frameworks are used to model various phenomena in fields such as chemistry, epidemiology and systems biology. We developed

StoSpa2 in order to facilitate easy and fast simulations of systems modelled using the CME and RDME. There are some alternative software packages that can be used, for example, URDME [15] and MesoRD [21], however these require the installation of many other packages, often called dependencies, that can be a barrier to installation and use; URDME has proprietary software as dependencies (MATLAB and COMSOL), while MesoRD has a large number of dependencies that make it challenging to install. StochSS [16] is another relevant software package, and it provides a docker container, a stand-alone version of a package, and a web interface. However, docker containers require special privileges to run, which not every user may have, and the web interface does not allow high-throughput execution of simulations. Hence, we developed *StoSpa2*, which relies upon as few other packages as possible and is designed to be easily installed. Furthermore, our software can simulate dynamics on a uniformly growing domain using the Extrande method [66], while the alternatives cannot.

For efficiency, we chose *StoSpa2* to be written in C++ programming language, and for ease of access we made made it available through Python programming language. The Python interface was achieved through the use of Pybind11 software package. Hence, we have the benefits of using a compiled language, which results in fast execution times, together with the ease of use of the Python programming language. To make sure *StoSpa2* is easy to use, we limit the number of packages that *StoSpa2* depends on, *i.e.* the number of dependencies is kept to a minimum, such that the installation procedure is straightforward. Furthermore, when developing *StoSpa2* we made no assumptions about the shape of the domain or the shapes of the compartments making up the domain, and as a result simulations can be performed on domains discretised by both structured and unstructured meshes, as shown in Figure 5.1 and in the results presented in Chapters 2–4. Lastly, to

make the software user-friendly, the application programming interface (API) has been designed with simplicity in mind and all the details about the API can be found at the *StoSpa2* documentation website (https://stospa2.readthedocs.io).

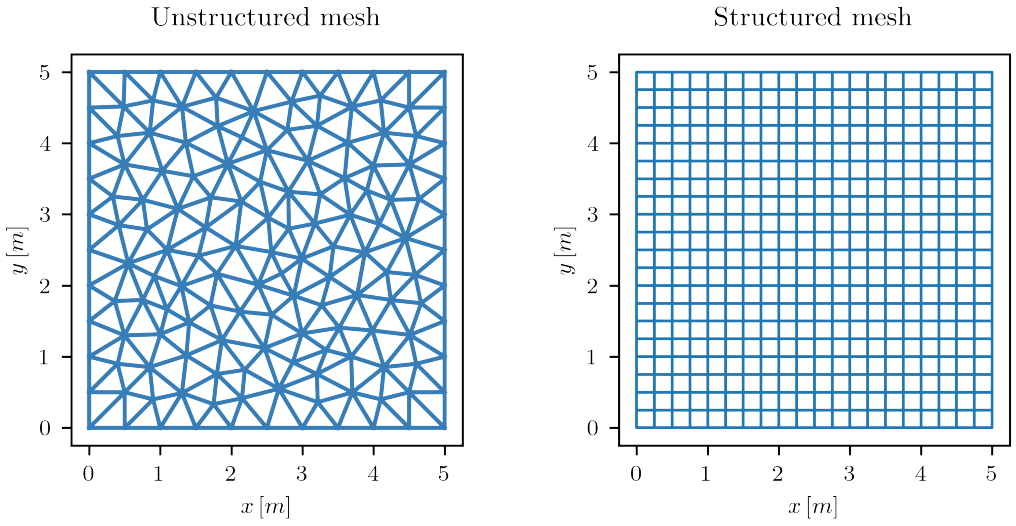


Figure 5.1: Examples of meshes that can be used within *StoSpa2* for RDME simulations. On the left-hand side is an example of an unstructured mesh, while on the right-hand side is an example of a structured mesh.

As we have discussed, unit-testing is a software development methodology that promotes reproducibility and should be used as often as possible. So that we practice what we preach, the code for *StoSpa2* includes unit-testing. However, those tests need to be executed to check whether all the components work properly. To do so on a regular basis we use the continuous integration platform TravisCI (https://travis-ci.org/), which executes the unit-tests to make sure that any changes in the code-base do not break any functionality of *StoSpa2*, by executing the unit-tests in different computing environments any time changes are submitted to the repository for *StoSpa2*. Both the installation and the functionality are tested on Linux and OSX operating systems.

The code for *StoSpa2* is hosted on the code hosting website GitHub and specifically in the following repository: https://github.com/BartoszBartmanski/StoSpa2. Making

the code for *StoSpa2* open-source ensures our work is reproducible and will help other scientists wishing to use the CME or RDME frameworks to model a RD system.

To showcase the ease of use of *StoSpa2*, we consider four RD systems and the code that can be used to run simulations for those systems. We also describe the unit-tests done on the package *StoSpa2*.

5.1.1 CME example

Let us consider the following chemical reaction

$$\mathrm{U} \xrightarrow{k} \emptyset, \tag{5.1}$$

which occurs at some rate $k\,s^{-1}$ on a one-dimensional domain $\Omega = [0, 10\,m]$. We can simulate this chemical system with the following code. We offer a more substantial commentary after the code.

```
# Import pystospa.
import pystospa as ss


# Create voxel object.
# Arguments: vector of number of molecules, size of the voxel.
initial_num = [100]  # number of molecules of species U
domain_size = 10.0  # size of the domain in m
v = ss.Voxel(initial_num, domain_size)


# Create reaction object.
```

```
# Arguments: reaction rate, propensity func, stoichiometry vector.
k = 1.0
propensity = lambda num_mols, area : num_mols[0]
stoch = [-1]
r = ss.Reaction(k, propensity, stoch)


# Add a reaction to a voxel.
v.add_reaction(r)


# Pass the voxel with the reaction(s) to the simulator object.
s = ss.Simulator([v])
# Run the simulation.
# Arguments: path to output file, time step, number of steps.
s.run("cme_example.dat", 0.01, 500)
```

The first line of code in the above example makes sure that we can use the *StoSpa2* classes. In the main function we first define the Voxel object that will represent the domain of the system:

```
initial_num = [100]  # number of molecules of species U
domain_size = 10.0  # size of the domain in m
v = ss.Voxel(initial_num, domain_size)
```

where we place 100 molecules of species U into a domain of size $10.0\,m$. The size of the domain does not matter in this example, but it is required by the package, as it was designed for simulations of the RDME. In the next segment of the code we create the lambda function that represents the propensity function. A lambda function within

Python is a function that is defined in a concise way. The Reaction object with a rate of $k = 1.0\,s^{-1}$, the propensity function *propensity* and the stoichiometry vector which decreases the number of molecules by one any time that the decay reaction happens are created in the following lines of code:

```
k = 1.0

propensity = lambda num_mols, area : num_mols[0]

stoch = [-1]

r = ss.Reaction(k, propensity, stoch)
```

Note that, the reaction propensity function in this case would be kU with U being the number of molecules of U, whereas the above lambda function returns just the number of molecules. This interface was chosen to allow flexibility with the expression of the propensity function, and so any function that takes two arguments and returns a double is a valid parameter for the Reaction object. The lambda functions for the reaction propensities have to take two arguments: a list and a double. The list represents the number of molecules and the double represents the area of a voxel. We then pass the Reaction object to the Voxel:

```
v.add_reaction(r)
```

Finally, we run the simulation, which saves the output into a file every 0.01 s for 500 steps:

```
s = ss.Simulator([v])

s.run("cme_example.dat", 0.01, 500)
```

We can see an example output of a simulation in Figure 5.2. By using the *run* function of the Simulator class, we do not see every reaction, but that can be done by using the *step* function of the Simulator class.

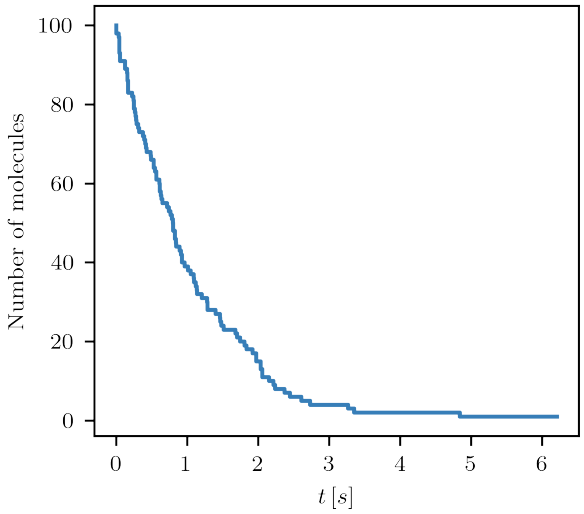


Figure 5.2: Example of a simulation output for a system modelled using CME framework. A single species of molecules decays at rate $k\,s^{-1}$.

5.1.2 RDME example

For a simple example of a simulation within a RDME framework, we consider a one-dimensional domain $\Omega = [0, 10\,m]$, discretised into 10 compartments and we model diffusion as a jump process between neighbouring compartments, as we have done thus far in this thesis:

$$U_1 \overset{d}{\rightleftharpoons} U_2 \overset{d}{\rightleftharpoons} U_3 \overset{d}{\rightleftharpoons} \ldots \overset{d}{\rightleftharpoons} U_{10}\,, \tag{5.2}$$

where U_i is the number of molecules in voxel i. The Python code to simulate this system, where the jump rate between adjacent compartments is $d = 1.0\,s^{-1}$, is as follows:

```
import pystospa as ss


# Create a vector of voxel objects.
# Voxel arguments: vector of number of molecules, size of a compartment.
initial_num = [10000]
```

```
voxel_size = 1.0  # size of a voxel in m
domain = [ss.Voxel(initial_num, voxel_size)]
# We add nine voxels with no molecules.
for i in range(9):
    domain.append(ss.Voxel([0], voxel_size))

# Create and add the reaction objects.
d = 1.0  # diffusion rate
propensity = lambda num_mols, length : num_mols[0]
stoch = [-1]
for i in range(9):
    # Add diffusion jump to the right from voxel i to voxel i+1.
    domain[i].add_reaction(ss.Reaction(d, propensity, stoch, i+1))
    # Add diffusion jump to the left from voxel i+1 to voxel i.
    domain[i+1].add_reaction(ss.Reaction(d, propensity, stoch, i))

# Pass the voxels with the reaction(s) to the simulator object.
s = ss.Simulator(domain)

# Run the simulation.
# Arguments: path to output file, time step, number of steps.
s.run("rdme_example.dat", 0.01, 200)
```

We start by initialising the Voxel objects representing the compartments that make up the domain of the system. First, we set the variables that define the number of molecules

in the left-most compartment and the size of every compartment. Then, we initialise the number of molecules and the size of each compartment as follows:

```
initial_num = [10000]
voxel_size = 1.0  # size of a voxel in m
domain = [ss.Voxel(initial_num, voxel_size)]
# We add nine voxels with no molecules.
for i in range(9):
    domain.append(ss.Voxel([0], voxel_size))
```

We add reactions to the voxels, where we assume that the compartments are ordered by their position on the x-axis. When adding the diffusion reactions, we have one additional parameter in the Reaction class constructors, namely *diffusion_idx*, which is the index of the neighbouring compartment into which a molecule jumps if a diffusion reaction happens:

```
d = 1.0  # diffusion rate
propensity = lambda num_mols, length : num_mols[0]
stoch = [-1]
for i in range(9):
    # Add diffusion jump to the right from voxel i to voxel i+1.
    domain[i].add_reaction(ss.Reaction(d, propensity, stoch, i+1))
    # Add diffusion jump to the left from voxel i+1 to voxel i.
    domain[i+1].add_reaction(ss.Reaction(d, propensity, stoch, i))
```

And finally, as in Section 5.1.1, we run the simulation with the *Simulator* class instance by passing the vector of Voxel objects to it and calling the *Simulator* class run function

```
# Pass the voxels with the reaction(s) to the simulator object.
s = ss.Simulator(domain)


# Run the simulation.
# Arguments: path to output file, time step, number of steps.
s.run("rdme_example.dat", 0.01, 200)
```

which takes the path to a file where to save the data, followed by the size of the time-step and the number of steps to take to finish the simulation. The state of the simulation, initially and at the final time point, is shown in Figure 5.3 where the molecules diffuse.

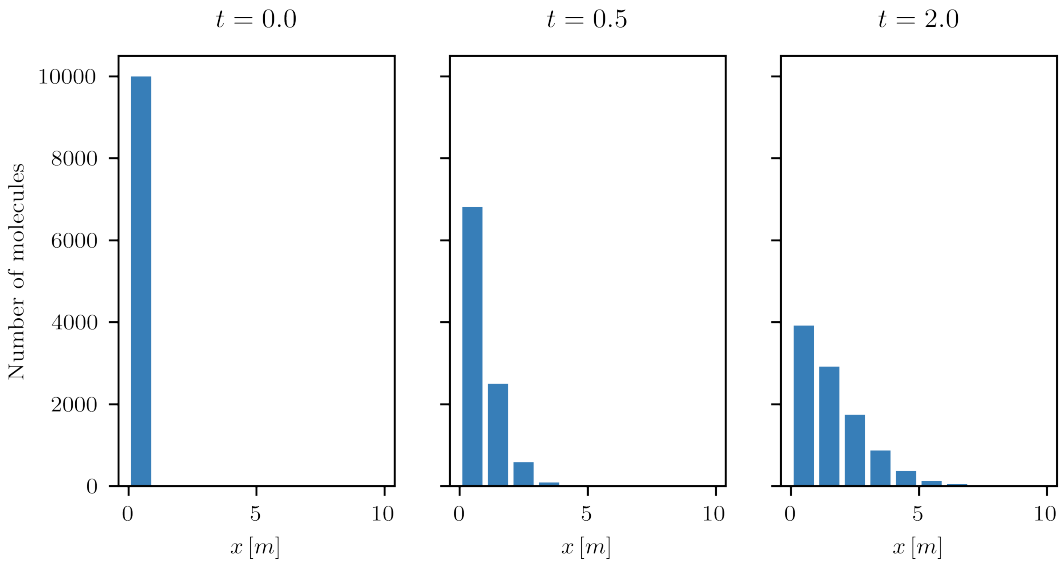


Figure 5.3: Example of a simulation output at times $t = 0.0$, $t = 0.5$ and $t = 2.0$ for a system modelled using the RDME framework. Initially all the molecules are placed in the left-most compartment and the molecules jump between the compartments at rate $d = 1.0m^2s^{-1}$.

5.1.3 Growing domain example

Next, we look at an example of incorporating uniform domain growth into a simulation. Similarly to Section 5.1.2, we consider a one-dimensional domain discretised into 10 com-

partments. However, the domain is growing exponentially with rate parameter r, such that the domain $\Omega = [0, L_0 e^{rt}\, m]$. As in the previous example, we model diffusion as a jump process between neighbouring compartments:

$$U_1 \overset{d}{\rightleftharpoons} U_2 \overset{d}{\rightleftharpoons} U_3 \overset{d}{\rightleftharpoons} \ldots \overset{d}{\rightleftharpoons} U_{10}\,, \tag{5.3}$$

where U_i is the number of molecules in voxel i. The Python code to simulate this system, where the jump rate between adjacent compartments is $d = 1.0\, s^{-1}$ and the growth rate of the domain is $r = 0.4 s^{-1}$, is as follows:

```
import pystospa as ss


# Create a vector of voxel objects.
# Voxel arguments: vector of number of molecules, size of the voxel
initial_num = [10000]
voxel_size = 1.0
growth = lambda t : np.exp(0.4 * t)
domain = [ss.Voxel(initial_num, voxel_size, growth)]
# We add nine voxels with no molecules
for i in range(9):
    domain.append(ss.Voxel([0], voxel_size, growth))


# Create and add the reaction objects
d = 1.0  # diffusion rate
propensity = lambda num_mols, length : num_mols[0]
stoch = [-1]
```

```
for i in range(9):
    # Add diffusion jump to the right from voxel i to voxel i+1
    domain[i].add_reaction(ss.Reaction(d, propensity, stoch, i+1))
    # Add diffusion jump to the left from voxel i+1 to voxel i
    domain[i+1].add_reaction(ss.Reaction(d, propensity, stoch, i))

# Pass the voxels with the reaction(s) to the simulator object
s = ss.Simulator(domain)

# Run the simulation.
# Arguments: path to output file, time step, number of steps
s.run("growth_example.dat", 0.01, 200)
```

The main difference between the above code and the code in the example from Section 5.1.2 is that we add a lambda function for the growth of the domain to the constructor of the voxels:

```
ss.Voxel(initial_num, voxel_size, growth)
```

where the variable *growth* is defined by a lambda function:

```
growth = lambda t : np.exp(0.4 * t)
```

The output from this simulation is shown in Figure 5.4, where we compare the simulation on a growing domain (bottom row) with the simulation from Section 5.1.2, which is on a static domain (top row). We see how, even though the simulations look very similar at $t = 0.5\,s$, by the end of the simulation, the molecules have reached the point $x = 10\,m$ in the simulation on a growing domain, unlike for the static case.

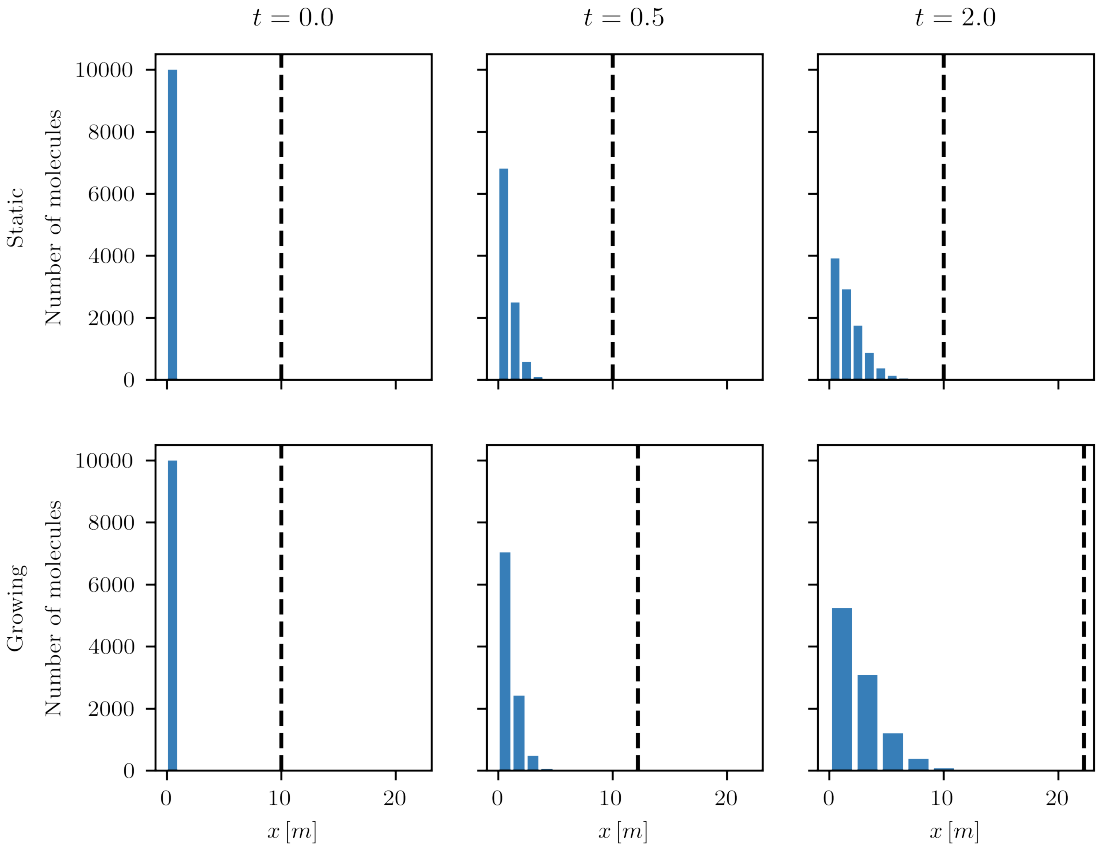


Figure 5.4: Comparison of the output from simulations of molecules diffusing on static and growing domains. The dashed vertical lines show the right boundary of the domain for both static and growing domains.

5.1.4 Schnakenberg example

Now let us consider a more complex example, one involving two species and a range of chemical reactions, specifically the Schnakenberg system from Section 2.2.4. Similarly, to Section 2.2.4, we start by describing the Schnakenberg system, which is composed of two species, U and V, that undergo diffusion (with rates D_u and D_v, respectively) on the domain $\Omega = [0, 1\,m]^2$ with Neumann boundary conditions, discretised into 40

compartments. In addition, we have the following reactions

$$\emptyset \xrightarrow{k_1} \mathrm{U}, \quad \mathrm{U} \xrightarrow{k_2} \emptyset, \quad 2\mathrm{U} + \mathrm{V} \xrightarrow{k_3} 3\mathrm{U}, \quad \emptyset \xrightarrow{k_4} \mathrm{V}, \tag{5.4}$$

with the corresponding propensity functions

$$a_i^1(\mathbf{u}(t), \mathbf{v}(t)) = k_1 A(t), \tag{5.5}$$

$$a_i^2(\mathbf{u}(t), \mathbf{v}(t)) = k_2 u_i(t), \tag{5.6}$$

$$a_i^3(\mathbf{u}(t), \mathbf{v}(t)) = \frac{k_3}{A^2(t)} u_i(t)(u_i(t) - 1)v_i(t), \tag{5.7}$$

$$a_i^4(\mathbf{u}(t), \mathbf{v}(t)) = k_4 A(t). \tag{5.8}$$

The Python code for this system is as follows:

```
n, l = 40, 1.0  # num voxels, domain size
h = l/n  # voxel size


# We define the reaction rates
du, dv = 1e-5, 0.001
k1, k2, k3, k4 = 0.02, 1.0/h, 0.000001 * h**2, 3.0/h


# We initialise the number of molecules in each compartment
domain = [ss.Voxel([200, 75], h) for i in range(n)]


# We define the propensities for the jump reactions
prop_diff1 = lambda num_mols, length : num_mols[0]
prop_diff2 = lambda num_mols, length : num_mols[1]
```

```
# And we add jump reactions
for i in range(len(domain)-1):
    # We create the reaction objects for jump reactions
    r_right1 = ss.Reaction(du/h**2, prop_diff1, [-1, 0], i+1)
    r_right2 = ss.Reaction(dv/h**2, prop_diff2, [0, -1], i+1)
    r_left1 = ss.Reaction(du/h**2, prop_diff1, [-1, 0], i)
    r_left2 = ss.Reaction(dv/h**2, prop_diff2 [0, -1], i)

    # Add diffusion jump to the right from voxel i to
    # voxel i+1 for each species
    domain[i].add_reaction(r_right1)
    domain[i].add_reaction(r_right2)
    # Add diffusion jump to the left from voxel i+1 to
    # voxel i for each species
    domain[i+1].add_reaction(r_left2)
    domain[i+1].add_reaction(r_left2)

# We define the propensities for other reactions in the system
prop_decay = lambda num_mols, length : num_mols[0]
prop_prod = lambda num_mols, length : length
prop_s = lambda num_mols, length : \
num_mols[0]*(num_mols[0]-1)*num_mols[1] / (length**2)
```

```
    # We add the rest of the reactions to each voxel
    for i in range(len(domain)):
        domain[i].add_reaction(ss.Reaction(k1, prop_decay, [-1, 0]))
        domain[i].add_reaction(ss.Reaction(k2, prop_prod, [1, 0]))
        domain[i].add_reaction(ss.Reaction(k4, prop_prod, [0, 1]))
        domain[i].add_reaction(ss.Reaction(k3, prop_s, [1, -1]))

    # Run the simulation.
    # Arguments: path to output file, time step, number of steps
    s = ss.Simulator(domain)
    s.run("schnakenberg_example.dat", 10, 180)
```

where the first two of lines are used to specify the size of each compartment, the jump rates and the reaction rates. After defining the rates, we create the Voxel objects that represent the domain with 200 molecules of species U and 75 molecules of species V in each compartment:

```
    domain = [ss.Voxel([200, 75], h) for i in range(n)]
```

much like in the previous two examples. The next step is to add the jump reactions:

```
    # We define the propensities for the jump reactions
    prop_diff1 = lambda num_mols, length : num_mols[0]
    prop_diff2 = lambda num_mols, length : num_mols[1]

    # And we add jump reactions
    for i in range(len(domain)-1):
```

```
    # We create the reaction objects for jump reactions
    r_right1 = ss.Reaction(du/h**2, prop_diff1, [-1, 0], i+1)
    r_right2 = ss.Reaction(dv/h**2, prop_diff2, [0, -1], i+1)
    r_left1 = ss.Reaction(du/h**2, prop_diff1, [-1, 0], i)
    r_left2 = ss.Reaction(dv/h**2, prop_diff2 [0, -1], i)

    # Add diffusion jump to the right from voxel i to
    # voxel i+1 for each species
    domain[i].add_reaction(r_right1)
    domain[i].add_reaction(r_right2)
    # Add diffusion jump to the left from voxel i+1 to
    # voxel i for each species
    domain[i+1].add_reaction(r_left2)
    domain[i+1].add_reaction(r_left2)
```

where the main difference to previous examples in this chapter is that we have two species, hence the stoichiometry vectors for each compartment are of size two. After that we add other reactions within the system, such as the production and decay reactions:

```
# We define the propensities for other reactions in the system
prop_decay = lambda num_mols, length : num_mols[0]
prop_prod = lambda num_mols, length : length
prop_s = lambda num_mols, length : \
num_mols[0]*(num_mols[0]-1)*num_mols[1] / (length**2)

# We add the rest of the reactions to each voxel
```

```
for i in range(len(domain)):
    domain[i].add_reaction(ss.Reaction(k1, prop_decay, [-1, 0]))
    domain[i].add_reaction(ss.Reaction(k2, prop_prod, [1, 0]))
    domain[i].add_reaction(ss.Reaction(k4, prop_prod, [0, 1]))
    domain[i].add_reaction(ss.Reaction(k3, prop_s, [1, -1]))
```

and finally we run the simulation and save the output:

```
s = ss.Simulator(domain)
s.run("schnakenberg_example.dat", 10, 180)
```

We see the output from this simulation in Figure 5.5, where we get the expected peaks in the number of molecules.

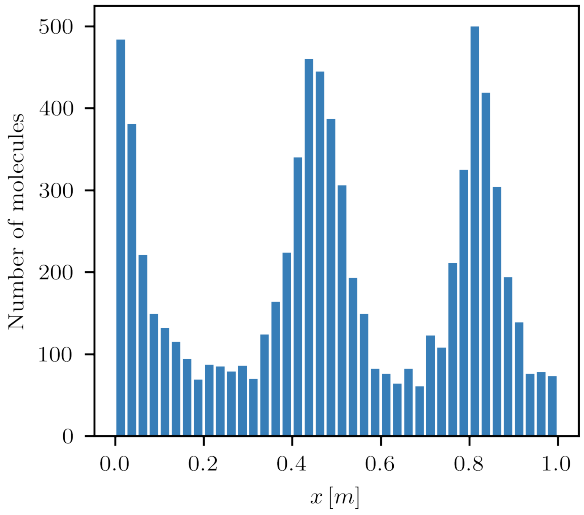


Figure 5.5: The state of the simulation at $T = 1800$ of the Schnakenberg system with the following reaction rates: $D_u = 1 \times 10^{-5}\, m^2 s^{-1}$, $D_v = 1 \times 10^{-3}\, m^2 s^{-1}$, $k_1 = 1\, s^{-1}$, $k_2 = 0.02\, s^{-1}$, $k_3 = 1 \times 10^{-6}\, s^{-1}$ and $k_4 = 3\, s^{-1}$.

5.1.5 Unit-testing

As mentioned at the beginning of Section 5.1, there are three classes within *StoSpa2*: Reaction, Voxel and Simulation. Each of these classes has a range of tests to check that

every function within each of the classes does what it is supposed to. The tests, much like the classes within the software itself, are divided into three classes, one for each class.

Firstly, we look at the tests for the Reaction class:

```
class TestReaction(unittest.TestCase):


    def test_constructor(self):
        # Create a Reaction object
        r = pystospa.Reaction(0.0, lambda x,y : 1.0, [0])


        # Check that Reaction object attributes have been set correctly
        self.assertEqual(r.get_rate(), 0)
        self.assertEqual(r.get_propensity([10], 1.0), 0.0)


    def test_member_functions(self):
        # Create a Reaction object
        r = pystospa.Reaction(0.0, lambda x,y : 1.0, [0])


        # Check that reaction rate can be changed
        r.set_rate(1.55)
        self.assertEqual(r.get_rate(), 1.55)


        # Check that the correct propensity is returned
        self.assertEqual(r.get_propensity([10], 1.0), 1.55)
```

which begin with checks that the constructor sets all the values correctly. The next set of

tests, contained in the *test_member_functions* function, check that other functions of the Reaction class work correctly as well, such as functions that set and give to the user the value of the rate of a reaction.

Secondly, we have the tests for the Voxel class. These tests, much like the tests for the Reaction class, are split into two sections, tests for the constructor and tests for the other functions:

```
class TestVoxel(unittest.TestCase):


    def test_constructor(self):


        # Create a Voxel object

        v = pystospa.Voxel([10], 1.0)


        # Check that the Voxel attributes are as expected

        self.assertEqual(len(v.get_molecules()), 1)

        self.assertEqual(v.get_molecules()[0], 10)

        self.assertEqual(v.get_voxel_size(), 1.0)


    def test_member_functions(self):


        # Create a Voxel object

        v = pystospa.Voxel([10], 1.0)

        r = pystospa.Reaction(1.5, lambda x,y : x[0], [-1])
```

```
        # Check that a Reaction object has been added correctly
        v.add_reaction(r)
        rs = v.get_reactions()
        self.assertEqual(len(rs), 1)
        self.assertEqual(rs[0].get_rate(), 1.5)
        self.assertEqual(v.get_total_propensity(), 15)
```

where, after checks on the constructor, we check functions within the Voxel class relating to reactions, such as that of calculating the total propensity, work correctly.

Lastly, we check that the *Simulator* class, which iterates through the steps in the SSA. Similarly to the previous tests shown in this section, the first set of tests check that all the variables are set correctly by the constructor

```
class TestSimulator(unittest.TestCase):
    def test_constructor(self):

        # Create a Simulator object
        v = pystospa.Voxel([10], 1.0)
        v.add_reaction(pystospa.Reaction(1.5, lambda x,y : x[0], [-1]))
        s = pystospa.Simulator([v])

        # Check that the Simulator attributes are as expected
        self.assertEqual(s.get_time(), 0.0)
        self.assertEqual(s.get_voxels()[0].get_molecules()[0], 10)

    def test_member_functions(self):
```

```
        # Create a Simulator object
        v = pystospa.Voxel([10], 1.0)
        v.add_reaction(pystospa.Reaction(1.5, lambda x,y : x[0], [-1]))
        s = pystospa.Simulator([v])

        # Check setting seed for generating random number
        s.set_seed(153)
        self.assertEqual(s.get_seed(), 153)

        # Check a single step in SSA works
        s.step()
        self.assertGreater(s.get_time(), 0)
        self.assertEqual(s.get_voxels()[0].get_molecules()[0], 9)

        # Check that getting to specific time point works
        s.advance(1.0)
        self.assertGreater(s.get_time(), 1.0)
```

followed by the tests relating to the SSA.

In this section, we focused on the software we developed to efficiently run simulations of RD systems using CME and RDME frameworks. We followed with examples of the use of *StoSpa2* to showcase the simplicity of running simulations using our software. Another software package developed over the course of this project was to generate Voronoi tessellation given Delauney triangulations.

5.2 DualMesh

As we have discussed in Chapter 4, Voronoi tessellations and Delauney triangulations play an important role in discretising a domain Ω into compartments so that we can use the RDME framework to model a RD system. The Delauney-Voronoi framework allows us to define the compartment boundaries, through the Voronoi tessellation of the compartment boundaries and the way the molecules jump between adjacent compartments through the Delauney triangulation. Hence, both Voronoi tessellation and Delauney triangulation are needed when modelling a RD system in the RDME framework for unstructured meshes. In this section we discuss the workflow developed to generate the Delauney triangulations and their corresponding Voronoi tessellations, using another software package that we developed. Then we describe how they are used in simulations using the RDME framework.

Algorithm 5 Algorithm for generating a Voronoi tessellation given a Delauney triangulation

Let P_V and S_V be the set of vertices and set of polygons in the Voronoi tessellation, respectively.
Get the Delauney triangulation generated by *pygmsh*.
Let P_D and S_D be the set of vertices and set of triangles in the Delauney triangulation, respectively.
for Point p in P_D **do**
    Determine all the triangles in S_D which contain p.
    Calculate the center for each triangle in S_D which contains p.
    Add those centers to P_V and the indices of those points to S_V.
end for

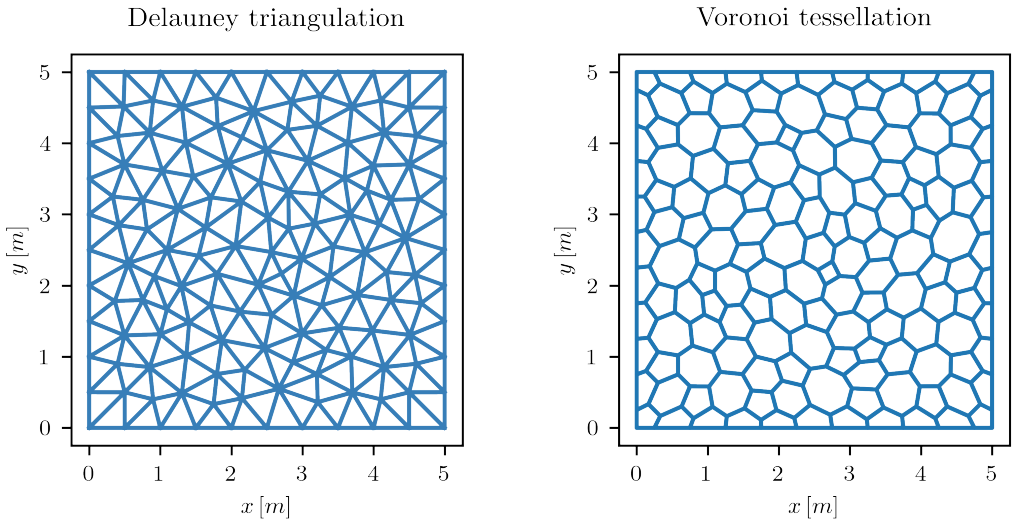


Figure 5.6: On the left: Delauney triangulation generated using *gmsh*[25] and *pygmsh*. On the right: Voronoi tessellation generated using *DualMesh* Python package based on the Delauney mesh shown on the left.

When setting up a simulation on unstructured domain, we first generate a Delauney triangulation using Gmsh [25] and the Python package *pygmsh*. Next, we need the associated Voronoi tessellation, which is where we use the Python package we developed called *DualMesh*. Algorithm 5 shows how the main component of *DualMesh* package works. To run a simulation on an unstructured mesh we need three steps:

1. generate a Delauney triangulation using *pygmsh* package;

2. generate a corresponding Voronoi tessellation using our *Dualmesh* package;

3. calculate the jump rates by using either finite volume method (FVM) or finite element method (FEM);

after which a simulation can be run using our *StoSpa2* package. As an example, we show how to run a simulation of molecules diffusing on an unstructured mesh. The first step is to generate a Delauney triangulation using the package *pygmsh* using the following code:

```
import pygmsh


l, lcar = 5, 0.5


geom = pygmsh.built_in.Geometry()
geom.add_rectangle(*[0, l]*2, 0, lcar=lcar)
msh = pygmsh.generate_mesh(geom, msh_filename="square.msh")
```

where we create a geometry object, then create a square within that geometry object, and finally create a mesh. We save the mesh into a file to be able to reuse it in other simulations. The next step is to generate a dual mesh:

```
import dualmesh as dm
import meshio


msh = meshio.read("square.msh")
d_msh = dm.get_dual(msh, True)
```

where we first open the file of the Delauney triangulation and then we generate the dual mesh, *i.e.* the Voronoi tessellation, using our *Dualmesh* package. The last step is to calculate the jump rates given the Delauney triangulation from step 1 and, if we use the FVM, we have

```
import meshio
import numpy as np
import dualmesh as dm
```

```
def get_neigh_elems(mesh, index):
    return np.where(mesh.cells_dict["triangle"] == index)[0]


msh = meshio.read("square.msh")
dual = dm.get_dual(msh, True)
num_verts = len(msh)
matrix = np.zeros([num_verts]*2)


for i in range(num_verts):
    n_v = np.setdiff1d(get_neigh_verts(mesh, i), [i], True)
    if len(n_v) > 0:
        v_i = dm.get_area(\
        dual.points[self.dual.cells_dict["polygon"][i], :2])
        d_ij = np.linalg.norm(\
        msh.points[n_v] - msh.points[i], axis=1)


        dual_idxs = np.array([\
        np.intersect1d(*dual.cells_dict["polygon"][[i, x]]) for x in n_v])
        gamma_ij = np.linalg.norm(\
        np.diff(dual.points[dual_idxs], axis=1), axis=2).reshape(-1)
        matrix[i, n_v] = gamma_ij / (v_i * d_ij)


np.savetxt("jump_rates.dat", matrix)
```

After those three steps are done, we are in a position to run a simulation:

```
import dualmesh as dm

imort meshio

import numpy as np

import pystospa as ss



# We get the meshes from previous steps

msh = meshio.read("square.msh")

d_msh = dm.get_dual(msh, True)


voxel_sizes = [dm.get_area(d_msh.points[x, :2])\

 for x in d_msh.cells_dict["polygon"]]

n = msh.points.shape[0]

mols = np.zeros(n).astype(int)

mols[0] = 5000000


# We create an array of compartments making up the domain

domain = [ss.Voxel([m], a) for m, a in zip(mols, voxel_sizes)]


# We add the diffusion reactions

lap = np.loadtxt("jump_rates.dat")

for v_idx, j_idx in zip(*np.where(lap > 0)):

    r = ss.Reaction(lap[v_idx, j_idx], lambda x, y : x[0], [-1], j_idx)

    domain[v_idx].add_reaction(r)
```

```
# We run the simulatio and save the data
s = ss.Simulator(domain)
s.run("output.dat", 0.01, 500)
```

As with *StoSpa2*, we made this software package open-source and available on Github (https://github.com/BartoszBartmanski/DualMesh) platform and the Python package index website (https://pypi.org/project/dualmesh/). Unit tests are also included and use the Python-native unit-testing framework. As a result, all the meshes and dual meshes present in Chapter 4 are reliable and can easily be reproduced.

Chapter 6

Discussion

In order to take into account the inherent noise that is prevalent in biology, stochastic models are used to describe RD systems. However, there can be some ambiguity in how diffusion is described in these stochastic models. In this work we have analysed different modelling choices in the context of the RDME framework and provided open source software packages to simulate RDME models on both static and growing domains with both structured and unstructured meshes. In this chapter, we discuss the implications, limitations and possible extensions of our work.

6.1 Chapter 2

In Chapter 2 we introduced the different methods of derivations of the diffusive jump rates: three based on widely used numerical schemes: finite difference method (FDM), finite element method (FEM), finite volume method (FVM); and one based on first exit time (FET). We compared these methods of derivation of the jump rates on different systems on a square domain that is discretised using a Cartesian mesh. The systems that we took into account included diffusion-only, zeroth-, first- and second-order reactions,

as well as the Schnakenberg kinetics, which can give rise to pattern formation via a diffusion-driven instability. We compared all the methods of derivation of the jump rates to determine differences between them and how changes in implementation parameters affected their accuracy.

The simulations which used FVM-derived jump rates were the closest to the analytical solution of the mean-field PDEs for the diffusion-only systems and systems with zeroth- and first-order reactions, followed by simulations with FDM- and FEM-derived jump rates. The simulations with FET-derived jump rates generally had the worst agreement. The disagreement between analytical solutions of the mean-field PDEs and the simulations with FET-derived jump rates is likely due to the fact that the FET-method requires approximating the FET distribution with an exponential distribution with the same mean.

For RD systems with second-order reactions, we explored the extent to which the stationary distribution of a model is impacted by the choice of discretisation. We observed similar distributions from simulations with FDM-, FEM- and FVM-derived jump rates, however, the stationary distribution based on simulations with FET-derived jump rates was shifted noticeably to the left. Finally, we did not observe significant differences between simulations that used different methods of derivation of the jump rates for the Schnakenberg kinetics when the FDM, FEM and FVM were used; since all three methods produced very similar power spectra. On the other hand, the power spectra of simulations of the Schnakenberg kinetics were affected by choice of a parameter which controlled the frequency of diagonal jump rates when using FET-derived jump rates.

6.2 Chapter 3

In Chapter 3 we then extended the framework introduced in Chapter 2 to include domain growth. We focused on isotropic domain growth though the methods presented are easily extensible to more general growth functions. We compared different functions that describe domain growth when using FDM-, FEM- and FVM-derived jump rates. The FET-derived jump rates could not be used because the mean of the FET, as required to approximate the FET distribution using an exponential distribution, is undefined on the growing domain.

Domain growth did not have any noticeable effect on the differences between simulations and the analytic solution of the mean-field PDEs when we considered a diffusion-only system. However, there were substantial differences between simulations with differently derived the jump rates when we considered the Schnakenberg and Gray-Scott kinetics, both of which admit pattern formation via a diffusion-driven instability. In the Schnakenberg case, we observed spots forming irrespective of the domain growth function and method of derivation of the jump rates. Moreover, the size of the spots was only affected by the growth rate – the faster the growth the smaller the spots at the end of the simulations. On the other hand, when we considered Gray-Scott kinetics, we observed that the simulation results are significantly affected by both the method of derivation of the jump rates, as well as the choice of domain growth function. When the domain was growing fast enough, namely with logistic growth, none or barely any spots formed in simulations with FDM- and FEM-derived jump rates, while spots were able to form in simulations with FVM-derived jump rates.

6.3 Chapter 4

In Chapters 2 and 3 we used only rectangles to discretise the domain, which limited the possible geometries that could be considered. To mitigate this issue, in Chapter 4 we introduced a type of unstructured mesh, referred to as a Voronoi mesh, which enabled the simulation of stochastic RD models on complex domain shapes. When simulating RD models on Voronoi meshes, we used only the FEM and FVM methods of derivation of the jump rates, as the FDM does not extend easily to unstructured meshes. Also, we did not consider the FET derivation, because the aim was to also consider domain growth on unstructured meshes.

In a diffusion-only system, we obtained similar conclusions as in the previous chapter – the simulations with FVM-derived jump rates had better agreement with the analytic solution of the mean-field PDEs compared to simulations with FEM-derived jump rates. However, for the Schnakenberg kinetics we saw quite stark differences compared to previous chapters. In the previous chapters, either the method of derivation of the jump rates or the choice of domain growth function or both had effects on the patterns we observed. When considering unstructured Voronoi meshes, the method of derivation of the jump rates and the choice of the domain growth function did not appear to have a noticeable effect on the patterns formed. Instead, we observed that the biggest effect came from the particular choice of Voronoi mesh. To further investigate this, we compared Voronoi meshes with different average compartment size and we observed different averaged power spectra depending on the choice of the mesh. We also note that all of the existing packages that we attempted to use to calculate the jump rates on unstructured meshes, such as Dune or FEniCS, were not helpful to our purposes as they did not specify which entries in the matrix representing discretised Laplacian corresponded to which entries in the mesh.

6.4 Chapter 5

Lastly, in Chapter 5, we described the software packages specifically developed for the simulations in this thesis. First, we outlined *StoSpa2*, which is used for all simulations. We designed *StoSpa2* for ease of use and easy application to complicated problems, such as complex domain shapes. Unlike a lot of other packages, *StoSpa2* has almost no dependencies, making it easy to install and use with any computing environment. Also, *StoSpa2* can simulate models on deterministically growing domains, which other packages for stochastic simulation, to our knowledge, cannot. Second, we introduced *Dualmesh*, which we used in Chapter 4 to generate relevant meshes. While there exist software packages for generating meshes, such as *gmsh*, these generate Delauney triangulations. However, the FVM of derivation of the jump rates on unstructured meshes requires both Delauney and Voronoi meshes, which is why we developed *Dualmesh*. Other packages that calculate a Voronoi tessellation do exist, though they calculate the Voronoi tessellation on an infinite plane, which is not useful for the purpose of this thesis.

Both of these packages were developed using high-standard software development practices, *i.e.* unit-testing, continuous integration and clear and extensive documentation. Unit-testing is a way to make sure that a software package does what it is expected to do, by checking individual components. On the other hand, continuous integration makes sure that any changes to the software do not break any functionality by regularly executing the unit-tests. We used both practices for both software packages to make sure they are of a high standard. Lastly, by making both software packages open-source, we hope that the work contained in this thesis can be easily extended. These practices are important because they enable verification of results and facilitate extensions of the research, by ensuring that other researchers do not have to redevelop the same tools.

6.5 Future work

In Chapters 2–4, we analysed the effects of using different methods of derivation of the jump rates on static and growing domains, discretised by structured and unstructured meshes, for a range of reaction kinetics. We have shown that different modelling choices affect simulation results, in some cases very strongly. This motivates further analytical investigation, since most of our results are derived from simulations alone. In addition, there exist many other ways to extend the work in this thesis and we outline them below.

We focused only on two sets of reaction kinetics, the Schnakenberg and Gray-Scott kinetics, that are capable of generating patterns via a diffusion-driven instability. A clear next step would be to explore other sets of reaction kinetics to evaluate the extent to which our findings generalise. For example, we observed that for certain models of pattern formation on growing domains, the derivation method of the jump rates affected pattern formation and for others it did not. It could be investigated if certain properties of the reaction system, such as shape of the domain or eigenvalues of the system, determine if the derivation method of the jump rates affects pattern formation.

When modelling domain growth, we considered only one specific type, namely isotropic domain growth, and a clear next step is to look at other models of domain growth, such as apical growth, or more general forms of non-isotropic growth. Furthermore, when modelling domain growth, the compartments by which the domain is discretised can quickly become too large for the assumptions underlying the RDME to be valid. To tackle this issue, compartment splitting would need to be added to the RDME framework, as in [4]. After this is implemented, the choices behind compartment splitting, such as how the molecules in the original compartment are divided between the daughter compartments, should also be investigated.

Another extension of the work is to couple the RDME framework with a cell-based model, *e.g.* a vertex-based model of tissue development, to develop a multi-scale model. The resulting framework will be able to model different scales of a biological system: movement of biochemical signalling molecules using the RDME framework, and cell growth, division and rearrangements using the vertex-based model.

Finally, we remark that we did not consider different meshes with the same average compartment size since the software package that we used for mesh generation can only create a unique mesh for a given mean compartment size. To mitigate this issue, another software package that allows for finer control of mesh generation needs to be found or developed, such that the output from this package can be used for simulations in *StoSpa2*. In addition, in order to allow *StoSpa2* to simulate models of cell division or to refine the mesh, *StoSpa2* would need to be extended in such a way as to allow compartments to split into two daughter compartments.

6.6 Concluding remarks

For differential equations, such as ODEs and PDEs, an entire research area is focused on studying the effects of different discretisations on the simulation results. However, this line of thought of questioning model choices and their effects on simulation results is not commonly considered for stochastic models despite the impact it can have on model predictions.

Given the potential of mathematical models to become predictive tools, it is important to understand how modelling choices affect model predictions and how to quantify the uncertainty that comes when using a specific model. It is by doing both of these analyses that one can trust model predictions and shed light on biological mechanisms.

Appendix A

Simulation of stochastic reaction-diffusion models on static domains

A.1 Finite element method derivation

Here we derive the elements of the matrices M and N in the FEM derivation of jump rates. We have

$$M\frac{\partial\tilde{\mathbf{p}}}{\partial t} = -N\tilde{\mathbf{p}}, \tag{A.1}$$

where $M_{ij} = \int_\Omega \varphi_i\varphi_j \mathrm{d}\omega$ and $N_{ij} = \int_\Omega \nabla\varphi_i \cdot \nabla\varphi_j \mathrm{d}\omega$. We wish to find the non-zero elements of the matrices M and N. As we are interested in deriving the jump rates for a non-boundary compartment, we will assume that the element (indexed by i) of matrices M and N is also not on the boundary of the FEM discretisation. Given that the function φ_i is zero outside of its neighbourhood, the neighbourhood being $[x_i-\kappa h, x_i+\kappa h]\times[y_i-h, y_i+h]$, then we only have nine elements to calculate in both matrices M and N for the ith row.

The nine elements in the ith row being due to the eight adjacent points and from itself.

We defined the function φ_i as

$$\varphi_i(x,y) = \begin{cases} \left(1 - \dfrac{x - x_i}{\kappa h}\right)\left(1 - \dfrac{y - y_i}{h}\right) & \text{for} \quad \mathbf{x} \in [x_i, x_i + \kappa h] \times [y_i, y_i + h], \\ \left(1 + \dfrac{x - x_i}{\kappa h}\right)\left(1 - \dfrac{y - y_i}{h}\right) & \text{for} \quad \mathbf{x} \in [x_i - \kappa h, x_i] \times [y_i, y_i + h], \\ \left(1 + \dfrac{x - x_i}{\kappa h}\right)\left(1 + \dfrac{y - y_i}{h}\right) & \text{for} \quad \mathbf{x} \in [x_i - \kappa h, x_i] \times [y_i - h, y_i], \\ \left(1 - \dfrac{x - x_i}{\kappa h}\right)\left(1 + \dfrac{y - y_i}{h}\right) & \text{for} \quad \mathbf{x} \in [x_i, x_i + \kappa h] \times [y_i - h, y_i], \\ 0 & \text{otherwise.} \end{cases} \tag{A.2}$$

Before we proceed with the calculations of the elements of matrices M and N, we need to determine the gradient of the function φ_i, as the integrals in the elements of the matrix N involve gradient of the function φ. The partial derivatives with respect to x and y are

as follows

$$\frac{\partial \varphi_i}{\partial x}(x,y) = \begin{cases} -\frac{1}{\kappa h}\left(1 - \dfrac{y-y_i}{h}\right) & \text{for} \quad \mathbf{x} \in [x_i, x_i+\kappa h] \times [y_i, y_i+h], \\ \frac{1}{\kappa h}\left(1 - \dfrac{y-y_i}{h}\right) & \text{for} \quad \mathbf{x} \in [x_i-\kappa h, x_i] \times [y_i, y_i+h], \\ \frac{1}{\kappa h}\left(1 + \dfrac{y-y_i}{h}\right) & \text{for} \quad \mathbf{x} \in [x_i-\kappa h, x_i] \times [y_i-h, y_i], \\ -\frac{1}{\kappa h}\left(1 + \dfrac{y-y_i}{h}\right) & \text{for} \quad \mathbf{x} \in [x_i, x_i+\kappa h] \times [y_i-h, y_i], \\ 0 & \text{otherwise,} \end{cases} \tag{A.3}$$

and

$$\frac{\partial \varphi_i}{\partial y}(x,y) = \begin{cases} -\frac{1}{h}\left(1 - \dfrac{x-x_i}{\kappa h}\right) & \text{for} \quad \mathbf{x} \in [x_i, x_i+\kappa h] \times [y_i, y_i+h], \\ -\frac{1}{h}\left(1 + \dfrac{x-x_i}{\kappa h}\right) & \text{for} \quad \mathbf{x} \in [x_i-\kappa h, x_i] \times [y_i, y_i+h], \\ \frac{1}{h}\left(1 + \dfrac{x-x_i}{\kappa h}\right) & \text{for} \quad \mathbf{x} \in [x_i-\kappa h, x_i] \times [y_i-h, y_i], \\ \frac{1}{h}\left(1 - \dfrac{x-x_i}{\kappa h}\right) & \text{for} \quad \mathbf{x} \in [x_i, x_i+\kappa h] \times [y_i-h, y_i]. \\ 0 & \text{otherwise.} \end{cases} \tag{A.4}$$

We have discretised the domain Ω, such that there are n_x compartments in the x-direction and n_y compartments in the y-direction. Hence, the index of the compartment

above compartment i is $i + n_x$ and the index of the compartment below is $i - n_x$. For simplicity, let us denote (1) to be the area $[x_i, x_i + \kappa h] \times [y_i, y_i + h]$, (2) to be the area $[x_i - \kappa h, x_i] \times [y_i, y_i + h]$, (3) to be the area $[x_i - \kappa h, x_i] \times [y_i - h, y_i]$ and (4) to be the area $[x_i, x_i + \kappa h] \times [y_i - h, y_i]$.

Now we can calculate the non-zero elements of row i of matrix M, which are

$$\begin{aligned}
\int_\Omega \varphi_i^2 \mathrm{d}\omega &= \int_{(1)} \left(1 - \frac{x - x_i}{\kappa h}\right)\left(1 - \frac{y - y_i}{h}\right)\left(1 - \frac{x - x_i}{\kappa h}\right)\left(1 - \frac{y - y_i}{h}\right) \mathrm{d}x\,\mathrm{d}y + \\
&\quad \int_{(2)} \left(1 + \frac{x - x_i}{\kappa h}\right)\left(1 - \frac{y - y_i}{h}\right)\left(1 + \frac{x - x_i}{\kappa h}\right)\left(1 - \frac{y - y_i}{h}\right) \mathrm{d}x\,\mathrm{d}y + \\
&\quad \int_{(3)} \left(1 + \frac{x - x_i}{\kappa h}\right)\left(1 + \frac{y - y_i}{h}\right)\left(1 + \frac{x - x_i}{\kappa h}\right)\left(1 + \frac{y - y_i}{h}\right) \mathrm{d}x\,\mathrm{d}y + \\
&\quad \int_{(4)} \left(1 - \frac{x - x_i}{\kappa h}\right)\left(1 + \frac{y - y_i}{h}\right)\left(1 - \frac{x - x_i}{\kappa h}\right)\left(1 + \frac{y - y_i}{h}\right) \mathrm{d}x\,\mathrm{d}y \\
&= \frac{4\kappa h^2}{9}, \qquad \text{(A.5)}
\end{aligned}$$

$$\begin{aligned}
\int_\Omega \varphi_i \varphi_{i+1} \mathrm{d}\omega &= \int_{(1)} \left(1 - \frac{x - x_i}{\kappa h}\right)\left(1 - \frac{y - y_i}{h}\right)\left(\frac{x - x_i}{\kappa h}\right)\left(1 - \frac{y - y_i}{h}\right) \mathrm{d}x\,\mathrm{d}y + \\
&\quad \int_{(4)} \left(1 - \frac{x - x_i}{\kappa h}\right)\left(1 + \frac{y - y_i}{h}\right)\left(\frac{x - x_i}{\kappa h}\right)\left(1 + \frac{y - y_i}{h}\right) \mathrm{d}x\,\mathrm{d}y \\
&= \frac{\kappa h^2}{9}, \qquad \text{(A.6)}
\end{aligned}$$

$$\begin{aligned}
\int_\Omega \varphi_i \varphi_{i+n_x+1} \mathrm{d}\omega &= \int_{(1)} \left(1 - \frac{x - x_i}{\kappa h}\right)\left(1 - \frac{y - y_i}{h}\right)\left(\frac{x - x_i}{\kappa h}\right)\left(\frac{y - y_i}{h}\right) \mathrm{d}x\,\mathrm{d}y \\
&= \frac{\kappa h^2}{36}, \qquad \text{(A.7)}
\end{aligned}$$

$$\begin{aligned}\int_\Omega \varphi_i\varphi_{i+n_x}\mathrm{d}\omega &= \int_{(1)}\left(1-\frac{x-x_i}{\kappa h}\right)\left(1-\frac{y-y_i}{h}\right)\left(1-\frac{x-x_i}{\kappa h}\right)\left(\frac{y-y_i}{h}\right)\mathrm{d}x\,\mathrm{d}y + \\ &\quad \int_{(2)}\left(1+\frac{x-x_i}{\kappa h}\right)\left(1-\frac{y-y_i}{h}\right)\left(1+\frac{x-x_i}{\kappa h}\right)\left(\frac{y-y_i}{h}\right)\mathrm{d}x\,\mathrm{d}y \\ &= \frac{\kappa h^2}{9}. \end{aligned} \tag{A.8}$$

By symmetry of the discretisation the other elements in the ith row of matrix M are

$$\int_\Omega \varphi_i\varphi_{i-1}\mathrm{d}\omega = \frac{\kappa h^2}{9}, \tag{A.9}$$

$$\int_\Omega \varphi_i\varphi_{i-n_x}\mathrm{d}\omega = \frac{\kappa h^2}{9}, \tag{A.10}$$

$$\int_\Omega \varphi_i\varphi_{i+n_x-1}\mathrm{d}\omega = \frac{\kappa h^2}{36}, \tag{A.11}$$

$$\int_\Omega \varphi_i\varphi_{i-n_x-1}\mathrm{d}\omega = \frac{\kappa h^2}{36}, \tag{A.12}$$

$$\int_\Omega \varphi_i\varphi_{i-n_x+1}\mathrm{d}\omega = \frac{\kappa h^2}{36}. \tag{A.13}$$

As we will use the lumped mass matrix method [70] on the matrix M, the resulting matrix $\tilde{M}$ will be a diagonal matrix, with the non-zero element in its ith row being the sum of all elements in that row. Therefore, the ith row non-zero element of matrix $\tilde{M}$ is κh^2.

We still need to calculate all the non-zero elements of matrix N. Using the same arguments as when calculating the non-zero elements of matrix M, we need only calculate

four integrals, specifically

$$\begin{aligned}
\int_\Omega |\nabla\varphi_i|^2 \,\mathrm{d}\omega &= \int_{(1)} \left(-\frac{1}{\kappa h}\right)\left(1-\frac{y-y_i}{h}\right)\left(-\frac{1}{\kappa h}\right)\left(1-\frac{y-y_i}{h}\right)\mathrm{d}x\,\mathrm{d}y + \\
&\quad \int_{(1)} \left(1-\frac{x-x_i}{\kappa h}\right)\left(-\frac{1}{h}\right)\left(1-\frac{x-x_i}{\kappa h}\right)\left(-\frac{1}{h}\right)\mathrm{d}x\,\mathrm{d}y + \\
&\quad \int_{(2)} \left(\frac{1}{\kappa h}\right)\left(1-\frac{y-y_i}{h}\right)\left(\frac{1}{\kappa h}\right)\left(1-\frac{y-y_i}{h}\right)\mathrm{d}x\,\mathrm{d}y + \\
&\quad \int_{(2)} \left(1+\frac{x-x_i}{\kappa h}\right)\left(-\frac{1}{h}\right)\left(1+\frac{x-x_i}{\kappa h}\right)\left(-\frac{1}{h}\right)\mathrm{d}x\,\mathrm{d}y + \\
&\quad \int_{(3)} \left(\frac{1}{\kappa h}\right)\left(1+\frac{y-y_i}{h}\right)\left(\frac{1}{\kappa h}\right)\left(1+\frac{y-y_i}{h}\right)\mathrm{d}x\,\mathrm{d}y + \\
&\quad \int_{(3)} \left(1+\frac{x-x_i}{\kappa h}\right)\left(\frac{1}{h}\right)\left(1+\frac{x-x_i}{\kappa h}\right)\left(\frac{1}{h}\right)\mathrm{d}x\,\mathrm{d}y + \\
&\quad \int_{(4)} \left(-\frac{1}{\kappa h}\right)\left(1+\frac{y-y_i}{h}\right)\left(-\frac{1}{\kappa h}\right)\left(1+\frac{y-y_i}{h}\right)\mathrm{d}x\,\mathrm{d}y + \\
&\quad \int_{(4)} \left(1-\frac{x-x_i}{\kappa h}\right)\left(\frac{1}{h}\right)\left(1-\frac{x-x_i}{\kappa h}\right)\left(\frac{1}{h}\right)\mathrm{d}x\,\mathrm{d}y \\
&= \frac{4\kappa}{3}+\frac{4}{3\kappa}, \qquad \text{(A.14)}
\end{aligned}$$

$$\begin{aligned}
\int_\Omega (\nabla\varphi_i)\cdot(\nabla\varphi_{i+1})\mathrm{d}\omega &= \int_{(1)} \left(-\frac{1}{\kappa h}\right)\left(1-\frac{y-y_i}{h}\right)\left(\frac{1}{\kappa h}\right)\left(1-\frac{y-y_i}{h}\right)\mathrm{d}x\,\mathrm{d}y + \\
&\quad \int_{(1)} \left(1-\frac{x-x_i}{\kappa h}\right)\left(-\frac{1}{h}\right)\left(\frac{x-x_i}{\kappa h}\right)\left(-\frac{1}{h}\right)\mathrm{d}x\,\mathrm{d}y + \\
&\quad \int_{(4)} \left(-\frac{1}{\kappa h}\right)\left(1+\frac{y-y_i}{h}\right)\left(\frac{1}{\kappa h}\right)\left(1+\frac{y-y_i}{h}\right)\mathrm{d}x\,\mathrm{d}y \\
&\quad \int_{(4)} \left(1-\frac{x-x_i}{\kappa h}\right)\left(\frac{1}{h}\right)\left(\frac{x-x_i}{\kappa h}\right)\left(\frac{1}{h}\right)\mathrm{d}x\,\mathrm{d}y \\
&= \frac{\kappa}{3}-\frac{2}{3\kappa}, \qquad \text{(A.15)}
\end{aligned}$$

$$\begin{aligned}
\int_\Omega (\nabla\varphi_i)\cdot(\nabla\varphi_{i+n_x+1})\mathrm{d}\omega &= \int_{(1)} \left(-\frac{1}{\kappa h}\right)\left(1-\frac{y-y_i}{h}\right)\left(\frac{1}{\kappa h}\right)\left(\frac{y-y_i}{h}\right)\mathrm{d}x\,\mathrm{d}y + \\
&\quad \int_{(1)} \left(-\frac{1}{\kappa h}\right)\left(1-\frac{y-y_i}{h}\right)\left(\frac{1}{\kappa h}\right)\left(\frac{y-y_i}{h}\right)\mathrm{d}x\,\mathrm{d}y \\
&= -\frac{\kappa}{6}-\frac{1}{6\kappa}, \qquad \text{(A.16)}
\end{aligned}$$

$$\begin{aligned}\int_\Omega (\nabla\varphi_i)\cdot(\nabla\varphi_{i+n_x})\mathrm{d}\omega &= \int_{(1)}\left(-\frac{1}{\kappa h}\right)\left(1-\frac{y-y_i}{h}\right)\left(-\frac{1}{\kappa h}\right)\left(\frac{y-y_i}{h}\right)\mathrm{d}x\,\mathrm{d}y + \\ &\quad \int_{(1)}\left(1-\frac{x-x_i}{\kappa h}\right)\left(-\frac{1}{h}\right)\left(1-\frac{x-x_i}{\kappa h}\right)\left(\frac{1}{h}\right)\mathrm{d}x\,\mathrm{d}y + \\ &\quad \int_{(2)}\left(\frac{1}{\kappa h}\right)\left(1-\frac{y-y_i}{h}\right)\left(\frac{1}{\kappa h}\right)\left(\frac{y-y_i}{h}\right)\mathrm{d}x\,\mathrm{d}y + \\ &\quad \int_{(2)}\left(1+\frac{x-x_i}{\kappa h}\right)\left(-\frac{1}{h}\right)\left(1+\frac{x-x_i}{\kappa h}\right)\left(\frac{1}{h}\right)\mathrm{d}x\,\mathrm{d}y \\ &= \frac{1}{3\kappa}-\frac{2\kappa}{3}. \end{aligned} \tag{A.17}$$

Hence, by symmetry, the other non-zero terms of row i are

$$\int_\Omega (\nabla\varphi_i)\cdot(\nabla\varphi_{i-1})\mathrm{d}\omega = \frac{\kappa}{3}-\frac{2}{3\kappa}, \tag{A.18}$$

$$\int_\Omega (\nabla\varphi_i)\cdot(\nabla\varphi_{i-n_x})\mathrm{d}\omega = \frac{1}{3\kappa}-\frac{2\kappa}{3}, \tag{A.19}$$

$$\int_\Omega (\nabla\varphi_i)\cdot(\nabla\varphi_{i+n_x-1})\mathrm{d}\omega = -\frac{\kappa}{6}-\frac{1}{6\kappa}, \tag{A.20}$$

$$\int_\Omega (\nabla\varphi_i)\cdot(\nabla\varphi_{i-n_x-1})\mathrm{d}\omega = -\frac{\kappa}{6}-\frac{1}{6\kappa}, \tag{A.21}$$

$$\int_\Omega (\nabla\varphi_i)\cdot(\nabla\varphi_{i-n_x+1})\mathrm{d}\omega = -\frac{\kappa}{6}-\frac{1}{6\kappa}. \tag{A.22}$$

Using the above non-zero elements of the matrices M and N, we can calculate the jump rates for the FEM case.

A.2 First exit time method derivation

To derive the jump rates using the FET method, we first have to solve

$$\frac{\partial p}{\partial t} = D\Delta p \quad \text{for} \quad \mathbf{x}\in\omega_{i,j}. \tag{A.23}$$

Due to the translational invariance of this PDE, we can solve it on the domain $[0, 2\kappa h] \times [0, 2h]$, which has the following solution, when solved using separation of variables,

$$p(x,y,t) = \frac{1}{\kappa h^2} \sum_{k=1}^{\infty} \sum_{j=1}^{\infty} (-1)^{j+k} \sin\left(\frac{(2k-1)\pi x}{2\kappa h}\right) \sin\left(\frac{(2j-1)\pi y}{2h}\right) \times \exp\left(-\frac{\pi^2 D t}{4\kappa^2 h^2}\left(\kappa^2(2j-1)^2 + (2k-1)^2\right)\right). \tag{A.24}$$

From this solution we can calculate the survival probability using the equation

$$S(t) = \mathbb{P}\left[\tau \geqslant t\right] = \int_{\omega_{i,j}} p(\mathbf{x}, t)\, \mathrm{d}\omega. \tag{A.25}$$

Hence, we have

$$\begin{aligned} S(t) &= \int_0^{2\kappa h} \int_0^{2h} p(\mathbf{x}, t)\, \mathrm{d}x\, \mathrm{d}y \\ &= \frac{1}{\kappa h^2} \int_0^{2\kappa h} \int_0^{2h} \sum_{k=1}^{\infty} \sum_{j=1}^{\infty} \sin\left(\frac{(2k-1)\pi x}{2\kappa h}\right) \sin\left(\frac{(2j-1)\pi y}{2h}\right) \\ &\quad \cdot \times \exp\left(-\frac{\pi^2 D t}{4\kappa^2 h^2}\left(\kappa^2(2j-1)^2 + (2k-1)^2\right)\right) \\ &= \frac{16}{\pi^2} \sum_{k=1}^{\infty} \sum_{j=1}^{\infty} \frac{(-1)^{j+k}}{(2k-1)(2j-1)} \exp\left(-\frac{\pi^2 D t}{4\kappa^2 h^2}\left(\kappa^2(2j-1)^2 + (2k-1)^2\right)\right). \end{aligned} \tag{A.26}$$

To calculate λ_0 we use Equations (2.54) and (2.55) from the main text to arrive at

$$\lambda_0 = \frac{\pi^4 D}{64\kappa^2 h^2 \sum_{k=1}^{\infty} \sum_{j=1}^{\infty} \frac{(-1)^{j+k}}{(2k-1)(2j-1)(\kappa^2(2j-1)^2 + (2k-1)^2)}}. \tag{A.27}$$

To calculate the probabilities of jumping in each direction, θ_n, we use Equation (A.24) and Equation (2.58) from the main text to derive

$$\theta_1 = \sum_{k=1}^{\infty}\sum_{j=1}^{\infty} \frac{8(-1)^{k+1}}{\pi^2(\kappa^2(2j-1)^2+(2k-1)^2))} \frac{2k-1}{2j-1} \sin\left(\frac{(2j-1)\pi\beta}{2}\right), \tag{A.28}$$

$$\theta_2 = \sum_{k=1}^{\infty}\sum_{j=1}^{\infty} \frac{4(-1)^{j+k}}{\pi^2(\kappa^2(2j-1)^2+(2k-1)^2))} \left\{ \frac{2k-1}{2j-1}\left(\sin\left(\pi(\beta j - \frac{\beta}{2}+j)\right)+1\right) + \frac{2j-1}{\kappa^2(2k-1)}\left(\sin\left(\pi(\beta k - \frac{\beta}{2}+k)\right)+1\right)\right\}, \tag{A.29}$$

$$\theta_3 = \sum_{k=1}^{\infty}\sum_{j=1}^{\infty} \frac{8(-1)^{j+1}\kappa^2}{\pi^2(\kappa^2(2j-1)^2+(2k-1)^2))} \frac{2j-1}{2k-1} \sin\left(\frac{(2k-1)\pi\beta}{2}\right). \tag{A.30}$$

A.3 Analytic solution

To compare the effects of using different methods of derivation of the diffusive jump rates, we compare the results of stochastic simulations with a large number of molecules with analytic solutions of the PDEs stated in the main text. In this section we shall state the analytical solutions we used for those comparisons. The derivations of these analytic solutions simply requires the use of separation of variables, hence, we omit the derivations. Furthermore, we denote

$$u_0 = 5 \times 10^6/A, \tag{A.31}$$

such that the initial condition used in many instances in the main text can be written as follows

$$u(\mathbf{x},0) = \begin{cases} u_0 & \text{for } 0<x<\kappa h,\ 0<y<h, \\ 0 & \text{otherwise.} \end{cases} \tag{A.32}$$

First, we state the analytic solution to Equation (2.60) in the main text:

$$\begin{aligned} u(\mathbf{x},t) &= \frac{u_0\kappa h^2}{L^2} \\ &\quad + \sum_{n=1}^{\infty} \frac{2u_0 h}{n\pi L} \sin\left(\frac{n\pi\kappa h}{L}\right) \cos\left(\frac{n\pi x}{L}\right) \exp\left(-Dt\left(\frac{n\pi}{L}\right)^2\right) \\ &\quad + \sum_{m=1}^{\infty} \frac{2u_0\kappa h}{m\pi L} \sin\left(\frac{m\pi h}{L}\right) \cos\left(\frac{m\pi y}{L}\right) \exp\left(-Dt\left(\frac{m\pi}{L}\right)^2\right) \\ &\quad + \sum_{n=1}^{\infty}\sum_{m=1}^{\infty} \frac{4u_0}{nm\pi^2 L^2} \sin\left(\frac{n\pi\kappa h}{L}\right) \sin\left(\frac{m\pi h}{L}\right) \cos\left(\frac{m\pi x}{L}\right) \cos\left(\frac{n\pi x}{L}\right) \\ &\qquad\qquad \times \exp\left(-Dt\left(\frac{(m^2+n^2)\pi^2}{L^2}\right)\right). \end{aligned} \tag{A.33}$$

Next, we state the analytic solutions to Equations (2.67)–(2.69) in the main text. We shall use $\tilde{u}(\mathbf{x},t)$ to denote the analytical solution stated above, *i.e.* Equation (A.33). Then the analytical solution to Equation (2.67) in the main text is

$$u(\mathbf{x},t) = \tilde{u}(\mathbf{x},t) + k_1 t, \tag{A.34}$$

and the solution to Equation (2.68) in the main text is

$$u(\mathbf{x},t) = \tilde{u}(\mathbf{x},t) e^{k_{-1}t}, \tag{A.35}$$

and, finally, the analytical solution to Equation (2.69) in the main text is

$$u(\mathbf{x},t) = \tilde{u}(\mathbf{x},t) e^{k_{-1}t} + \frac{k_1}{k_{-1}}\left(1 - e^{k_{-1}t}\right). \tag{A.36}$$

When evaluating any of the formulas above numerically, we truncate the sum to the first million terms.

A.4 Zeroth- and first-order reactions

In this section, we show the errors when we include zeroth- and first-order reactions. If we compare Figure A.1 with Figure 2.11, we see that addition of zeroth- and first-order reactions has little effect on the errors.

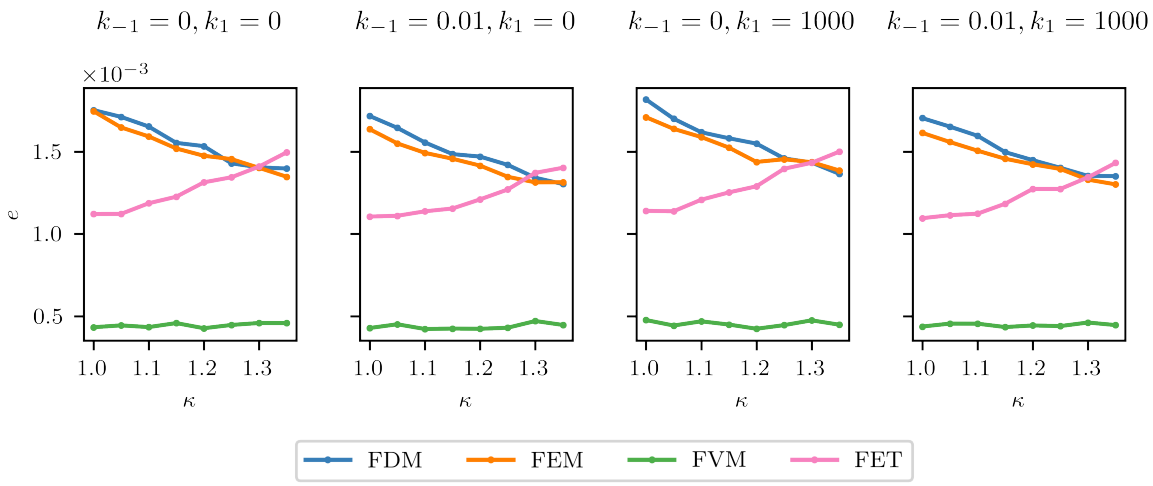


Figure A.1: Plot of the error at time $T = 5\,s$ as a function of κ, the compartment aspect ratio, for simulations that used FDM-, FEM-, FVM- and FET-derived jump rates. The domain $\Omega = [0, 5\,m]^2$ is discretised into $20 \times 20\kappa$ compartments and 5×10^6 molecules are initially placed in the compartment next to the origin. Parameters are: $D = 1\,m^2 s^{-1}$, $\alpha = 0.7$, $\beta = 0.5$. The rate constant for the decay reaction is denoted by k_{-1}, while the rate constant for the production reaction is denoted by k_1.

Appendix B

Growing domains and Cartesian meshes

B.1 Analytic solution

As we have calculated the analytical solution for the static domain case, so we must do the same for the growing domain case. In Section 3.2.1 we compare the results of the stochastic simulations with the analytical solution. As before, for brevity, we simply state the analytical solution (for derivation, we direct the reader to [59])

$$
\begin{aligned}
u(\mathbf{x},t) &= \frac{u_0\kappa h^2}{L_0^2}\exp\left(-2\log\left(L(t)\right)\right) \\
&+ \sum_{n=1}^{\infty}\frac{2u_0 h}{n\pi L_0}\sin\left(\frac{n\pi\kappa h}{L_0}\right)\cos\left(\frac{n\pi x}{L_0}\right)\exp\left(-\left(n\pi\right)^2\int_0^t\frac{D}{L^2(t')}\mathrm{d}t' - 2\log\left(L(t)\right)\right) \\
&+ \sum_{m=1}^{\infty}\frac{2u_0\kappa h}{m\pi L_0}\sin\left(\frac{m\pi h}{L_0}\right)\cos\left(\frac{m\pi y}{L_0}\right)\exp\left(-\left(m\pi\right)^2\int_0^t\frac{D}{L^2(t')}\mathrm{d}t' - 2\log\left(L(t)\right)\right) \\
&+ \sum_{n=1}^{\infty}\sum_{m=1}^{\infty}\frac{4u_0}{nm\pi^2 L_0^2}\sin\left(\frac{n\pi\kappa h}{L_0}\right)\sin\left(\frac{m\pi h}{L_0}\right)\cos\left(\frac{m\pi x}{L_0}\right)\cos\left(\frac{n\pi x}{L_0}\right) \\
&\qquad \times\exp\left(-(n^2+m^2)\pi^2\int_0^t\frac{D}{L^2(t')}\mathrm{d}t' - 2\log\left(L(t)\right)\right),
\end{aligned}
\tag{B.1}
$$

where $L(t)$ is the width of the domain $\Omega = [0, L(t)]^2$ at time t.

Bibliography

[1] T. Alarcón, H. M. Byrne, and P. K. Maini. A cellular automaton model for tumour growth in inhomogeneous environment. *Journal of Theoretical Biology*, 225(2):257–274, 2003.

[2] F. Aurenhammer, R. Klein, and D. Lee. *Voronoi diagrams and Delaunay triangulations*. World Scientific Publishing Company, 2013.

[3] M. Baker. 1,500 scientists lift the lid on reproducibility. *Nature*, 533:452–454, 2016.

[4] R. E. Baker, C. A. Yates, and R. Erban. From microscopic to macroscopic descriptions of cell migration on growing domains. *Bulletin of Mathematical Biology*, 72(3):719–762, 2010.

[5] S. G. Baker and B. S. Kramer. Systems biology and cancer: Promises and perils. 106(2):410–413, 2011.

[6] W. J. Blake, M. Kærn, C. R. Cantor, and J. J. Collins. Noise in eukaryotic gene expression. *Nature*, 422(6932):633–637, 2003.

[7] E. H. Bussell, C. E. Dangerfield, C. A. Gilligan, and N. J. Cunniffe. Applying optimal control theory to complex epidemiological models to inform real-world disease

management. *Philosophical Transactions of the Royal Society B: Biological Sciences*, 374(1776):20180284, 2019.

[8] H. M. Byrne and M. A. J. Chaplain. Growth of nonnecrotic tumors in the presence and absence of inhibitors. *Mathematical Biosciences*, 130(2):151–181, 1995.

[9] H. M. Byrne and M. A. J. Chaplain. Growth of necrotic tumors in the presence and absence of inhibitors. *Mathematical Biosciences*, 135(2):187–216, 1996.

[10] E. J. Crampin. *Reaction-diffusion patterns on growing domains.* PhD thesis, University of Oxford, 2000.

[11] E. J. Crampin, W. W. Hackborn, and P. K. Maini. Pattern formation in reaction-diffusion models with nonuniform domain growth. *Bulletin of Mathematical Biology*, 64(4):747–769, 2002.

[12] J. A. Davies. Do different branching epithelia use a conserved developmental mechanism? *Bioessays*, 24(10):937–948, 2002.

[13] M. Doi. Second quantization representation for classical many-particle system. *Journal of Physics A: Mathematical and General*, 9(9):1465, 1976.

[14] M. Doi. Stochastic theory of diffusion-controlled reaction. *Journal of Physics A: Mathematical and General*, 9(9):1479, 1976.

[15] B. Drawert, S. Engblom, and A. Hellander. Urdme: a modular framework for stochastic simulation of reaction-transport processes in complex geometries. *BMC Systems Biology*, 6(1):76, 2012.

[16] B. Drawert, A. Hellander, B. Bales, D. Banerjee, G. Bellesia, B. J. Daigle Jr, G. Douglas, M. Gu, A. Gupta, S. Hellander, et al. Stochastic simulation service: bridging

the gap between the computational expert and the biologist. *PLOS Computational Biology*, 12(12), 2016.

[17] J. Elf and M. Ehrenberg. Fast evaluation of fluctuations in biochemical networks with the linear noise approximation. *Genome Research*, 13(11):2475–2484, 2003.

[18] H. C. Elman, D. J. Silvester, and A. J. Wathen. Finite elements and fast iterative solvers: with applications in incompressible fluid dynamics. *Secretary*, 2010.

[19] S. Engblom, L. Ferm, A. Hellander, and P. Lötstedt. Simulation of stochastic reaction-diffusion processes on unstructured meshes. *SIAM Journal on Scientific Computing*, 30(6):2709–2733, 2008.

[20] R. Erban, J. S. Chapman, and P. K. Maini. A practical guide to stochastic simulations of reaction-diffusion processes. *arXiv*, pages 24–29, 2007.

[21] D. Fange, A. Mahmutovic, and J. Elf. Mesord 1.0: Stochastic reaction-diffusion simulations in the microscopic limit. *Bioinformatics*, 28(23):3155–3157, 2012.

[22] M. B. Flegg. Smoluchowski reaction kinetics for reactions of any order. *SIAM Journal on Applied Mathematics*, 76(4):1403–1432, 2016.

[23] C. W. Gardiner, K. J. McNeil, D. F. Walls, and I. S. Matheson. Correlations in stochastic theories of chemical reactions. *Journal of Statistical Physics*, 14(4):307–331, 1976.

[24] P. L. George and V. N. Kaliakin. Automatic mesh generation; application to finite element methods. *Journal of Engineering Mechanics*, 119(3):643–644, 1993.

[25] C. Geuzaine and J. Remacle. Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 79(11):1309–1331, 2009.

[26] A. Gierer and H. Meinhardt. A theory of biological pattern formation. *Kybernetik*, 12(1):30–39, 1972.

[27] D. T Gillespie. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22(4):403–434, 1976.

[28] C. M. Guldberg and P. Waage. Studies concerning affinity. *Journal of Chemical Education*, 63:1044, 1986.

[29] J. Hattne, D. Fange, and J. Elf. Stochastic reaction-diffusion simulation with MesoRD. *Bioinformatics*, 21(12):2923–2924, 2005.

[30] S. Hellander and L. Petzold. Reaction rates for reaction-diffusion kinetics on unstructured meshes. *The Journal of Chemical Physics*, 146(6):064101, 2017.

[31] S. A. Isaacson. The reaction-diffusion master equation as an asymptotic approximation of diffusion to a small target. *SIAM Journal on Applied Mathematics*, 70(1):77–111, 2009.

[32] S. A. Isaacson. A convergent reaction-diffusion master equation. *Journal of Chemical Physics*, 139(5), 2013.

[33] S. A. Isaacson, D. M. McQueen, and C. S. Peskin. The influence of volume exclusion by chromatin on the time required to find specific DNA binding sites by diffusion. *Proceedings of the National Academy of Sciences*, 108(9):3815–3820, 2011.

[34] M. U. G. Kraemer, N. R. Faria, R. C. Reiner, N. Golding, B. Nikolay, et al. Spread of yellow fever virus outbreak in Angola and the Democratic Republic of the Congo 2015–16: a modelling study. *The Lancet Infectious Diseases*, 17(3):330–338, 2017.

[35] T. G. Kurtz. Limit theorems for sequences of jump markov processes approximating ordinary differential processes. *Journal of Applied Probability*, 1971.

[36] Antti Laaksonen. *Guide to Competitive Programming*. Springer, 2017.

[37] D. Lee and B. J. Schachter. Two algorithms for constructing a Delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242, 1980.

[38] D. Longo and J. Hasty. Dynamics of single-cell gene expression. *Molecular Systems Biology*, 2(1):64, 2006.

[39] P. Lötstedt and L. Meinecke. Simulation of stochastic diffusion via first exit times. *Journal of Computational Physics*, 300:862–886, 2015.

[40] A. Madzvamuse. Time-stepping schemes for moving grid finite elements applied to reaction-diffusion systems on fixed and growing domains. *Journal of Computational Physics*, 214(1):239–263, 2006.

[41] A. Madzvamuse, E. A. Gaffney, and P. K. Maini. Stability analysis of non-autonomous reaction-diffusion systems: The effects of growing domains. *Journal of Mathematical Biology*, 61(1):133–164, 2010.

[42] A. Madzvamuse and P. K. Maini. Velocity-induced numerical solutions of reaction-diffusion systems on continuously growing domains. *Journal of Computational Physics*, 225(1):100–119, 2007.

[43] A. Madzvamuse, R. D. K. Thomas, P. K. Maini, and A. J. Wathen. A numerical approach to the study of spatial pattern formation in the ligaments of arcoid bivalves. *Bulletin of Mathematical Biology*, 64(3):501–530, 2002.

[44] P. K. Maini, K. J. Painter, and H. N. P. Chau. Spatial pattern formation in chemical and biological systems. *Journal of the Chemical Society - Faraday Transactions*, 93(20):3601–3610, 1997.

[45] N. Masuda and L. E. C. Rocha. A Gillespie algorithm for non-Markovian stochastic processes. 60(1):95–115, 2016.

[46] L. Meinecke. *Stochastic simulation of multiscale reaction-diffusion models via first exit times*. PhD thesis, Acta Universitatis Upsaliensis, 2016.

[47] L. Meinecke, S. Engblom, A. Hellander, and P. Lötstedt. Analysis and design of jump coefficients in discrete stochastic diffusion models. *SIAM Journal on Scientific Computing*, 38(1):A55–A83, 2016.

[48] L. Meinecke and P. Lötstedt. Stochastic diffusion processes on Cartesian meshes. *Journal of Computational and Applied Mathematics*, 294:1–11, 2016.

[49] I. D. Mishev. Finite volume methods on Voronoi meshes. *Numerical Methods for Partial Differential Equations: An International Journal*, 14(2):193–212, 1998.

[50] O. Morgan. How decision makers can use quantitative approaches to guide outbreak responses. 374(1776), 2019.

[51] J. D. Murray. How the leopard gets its spots. *Scientific American*, 258(3):80–87, 1988.

[52] J. D. Murray. *Mathematical biology: II. Spatial models and biomedical applications.* Springer-Verlag New York Incorporated, 2001.

[53] K. J. Painter, P. K. Maini, and H. G. Othmer. Stripe formation in juvenile pomacanthus explained by a generalized turing mechanism with chemotaxis. *Proceedings of the National Academy of Sciences*, 96(10):5549–5554, 1999.

[54] J. M. Perkel. Challenge to scientists: does your ten-year-old code still run? *Nature*, 584(7822):656–658, 2020.

[55] J. M. Raser and E. K. O'Shea. Control of stochasticity in eukaryotic gene expression. *Science*, 304(5678):1811–1814, 2004.

[56] S. Redner and J. R. Dorfman. A guide to first-passage processes. *American Journal of Physics*, 70(11):1166–1166, 2002.

[57] J. Schnakenberg. Simple chemical reaction systems with limit cycle behaviour. *Journal of Theoretical Biology*, 81(3):389–400, 1979.

[58] D. Schnoerr, G. Sanguinetti, and R. Grima. Approximation and inference methods for stochastic biochemical kinetics—a tutorial review. *Journal of Physics A: Mathematical and Theoretical*, 50(9):093001, 2017.

[59] M. J. Simpson and R. E. Baker. Exact calculations of survival probability for diffusion on growing lines, disks, and spheres: The role of dimension. *Journal of Chemical Physics*, 143(9):94109, 2015.

[60] S. Smith and R. Grima. Spatial stochastic intracellular kinetics: A review of modelling approaches. *Bulletin of Mathematical Biology*, 81(8):2960–3009, 2019.

[61] J. C. Strikwerda. *Finite difference schemes and partial differential equations.* SIAM, 2004.

[62] R. N. Thompson, J. E. Stockwin, R. D. van Gaalen, J. A. Polonsky, Z. N. Kamvar, P. A. Demarsh, E. Dahlqwist, S. Li, E. Miguel, T. Jombart, J. Lessler, S. Cauchemez, and A. Cori. Improved inference of time-varying reproduction numbers during infectious disease outbreaks. *Epidemics*, 29:100356, 2019.

[63] C. Tickle. Making digit patterns in the vertebrate limb. *Nature Reviews Molecular Cell Biology*, 7(1):45–53, 2006.

[64] A. M. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 237(641):37–72, 1952.

[65] N. G. Van Kampen. *Stochastic processes in physics and chemistry.*, volume 1. Elsevier, 1992.

[66] M. Voliotis, P. Thomas, R. Grima, and C. G. Bowsher. Stochastic simulation of biomolecular networks in dynamic environments. *PLOS Computational Biology*, 12(6):e1004923, 2016.

[67] T. E. Woolley, R. E. Baker, E. A. Gaffney, and P. K. Maini. Influence of stochastic domain growth on pattern nucleation for diffusive systems with internal noise. *Physical Review E*, 84(4):24–29, 2011.

[68] T. E. Woolley, R. E. Baker, E. A. Gaffney, and P. K. Maini. Power spectra methods for a stochastic description of diffusion on deterministically growing domains. *Physical Review E*, 84(2), 2011.

[69] T. E. Woolley, R. E. Baker, E. A. Gaffney, and P. K. Maini. Stochastic reaction and diffusion on growing domains: understanding the breakdown of robust pattern formation. *Physical Review E*, 84(4):46216, 2011.

[70] S. R. Wu. Lumped mass matrix in explicit finite element method for transient dynamics of elasticity. *Computer Methods in Applied Mechanics and Engineering*, 195(44-47):5983–5994, 2006.