Efficient mechanism discovery for cell invasion with uncertainty quantification

Daniel J. VandenHeuvel[1], Christopher C. Drovandi[1], and Matthew J. Simpson[1]

[1]School of Mathematical Sciences, Queensland University of Technology, Brisbane, Queensland 4001, Australia

March 9, 2022

Abstract

Parameter estimation for biological processes is often a difficult problem and depends significantly on the present data. We introduce a framework which utilises Gaussian processes to discovery the mechanisms underlying delay, diffusion, and reaction in a cell invasion process. These Gaussian processes are leveraged using bootstrapping to provide uncertainty quantification for learned mechanisms and to solutions of the corresponding partial differential equations. This framework is efficient and easily parallelisable, and can be easily extended to other problems. We demonstrate our methods on a scratch assay experiment, demonstrating how simply we can experiment with different functional forms and perform hypotheses on underlying mechanisms, such as whether delay is present in the cell invasion process. All code and data to reproduce this work is available at https://github.com/DanielVandH/EquationLearning.jl.

1 Author summary

In this work we extend previously developed equation learning methods such as physics-informed and biologically-informed neural networks to introduce uncertainty quantification. Our process is efficient and applicable to problems with nonlinear mechanisms and experiments where only sparse noisy data is available. We demonstrate our methods on a scratch assay experiment and show the underlying mechanisms can be learned, providing confidence intervals for functional forms and for solutions to partial differential equations.

2 Introduction

Estimating parameters from biological experiments using experimental data is a difficult problem. The data is often sparse and the underlying mechanisms governing the biology are complex, such as those governing cell migration and proliferation [2, 3]; see for example the experiments in Figure 2 performed by Jin et al. [1]. In addition to being sparse, the collected data is often noisy and the equipment used in the experiment may not be enough to capture the desired effects. These issues together make it difficult to understand the laws governing the biological process of interest as the sparsity of the data gives little information for estimating these laws, and the noise embedded in the data implies that any estimates that could be made must come with significant uncertainty. In this work, we propose an equation learning framework for efficiently discovering the functional forms of individual biological process within an experiment from sparse noisy data. Our method naturally provides uncertainty in these learned functional forms through confidence intervals. We illustrate our approach using data from a scratch assay experiment which studied the dependence of an initial density of cells on the migration and proliferation of cells.

Equation learning allows us to gain deeper insight into the underlying biological processes, and make use of available data to discover the structure of a governing model. Equation learning was first introduced by Brunton et al. [6] in the context of ordinary differential equations (ODEs), and then extended to partial different equations (PDEs) by Rudy et al. [7] who demonstrate their methods using data simulated according to a PDE, showing how spatiotemporal data and some assumed general form of a PDE can be used to learn the dynamics of an underlying model. Raissi et a. [9] developed physics informed neural networks for learning the structure of nonlinear PDEs from data, although they only consider results using synthetic data similarly to Rudy et al. [7]. Recent developments have shown the efficacy of equation learning methods applied to actual data. Lagergren et al. [8] extended the concept of a physics informed neural network to that of a biologically informed neural network for equation learning, and demonstrate their methods on actual data from an experiment from Jin et al. [1]. Chen et al. [10] directly extend on the approach by Raissi et al. [9] to make their results more interpretable, and also consider the data by Jin et al. [1] along with many other models.

In our work we extend on these ideas by implementing an approach which replaces the neural networks of the above references with a Gaussian process, and introduce uncertainty quantification using a parametric bootstrapping approach. Gaussian processes have been successfully used by other authors for learning differential equations. Wang et al. [33] and Heinonen et al. [35] use Gaussian processes to learn

ODEs, with Wang et al. jointly sampling function values and derivatives to avoid issues with evaluating derivatives numerically. Chen et al. [34, 37] use Gaussian processes to learn the functional forms for the coefficients in nonlinear PDEs. Bajaj et al. [36] extend on the above work and on the above physics informed neural networks, using Gaussian processes to make the equation learning process more robust. Our contribution differs from these references in that we specifically consider sparse data, and we enable the use of uncertainty quantification for learned results. We illustrate the work using data from Jin et al. [1], described in more detail in the discussion section, which describes the invasion of cells and their density u as a function of space x and time t.

The structure of the paper is as follows. In Section 3 we show the results for a delay-reaction-diffusion model, comparing results between a Fisher-Kolmogorov model, Porous-Fisher model, and a quadratic diffusion model. In Section 5 we discuss these results and the conclusions that can be made, and make recommendations for future work. Section 6 considers our methods in greater detail, outlining how we use Gaussian processes to learn the equations and details about optimisation and bootstrapping. Section 7 finishes with final conclusions and remarks. Additional details are presented in the supplementary material, and code for reproducing the work is available with the Julia package at `https://github.com/DanielVandH/EquationLearning.jl`.

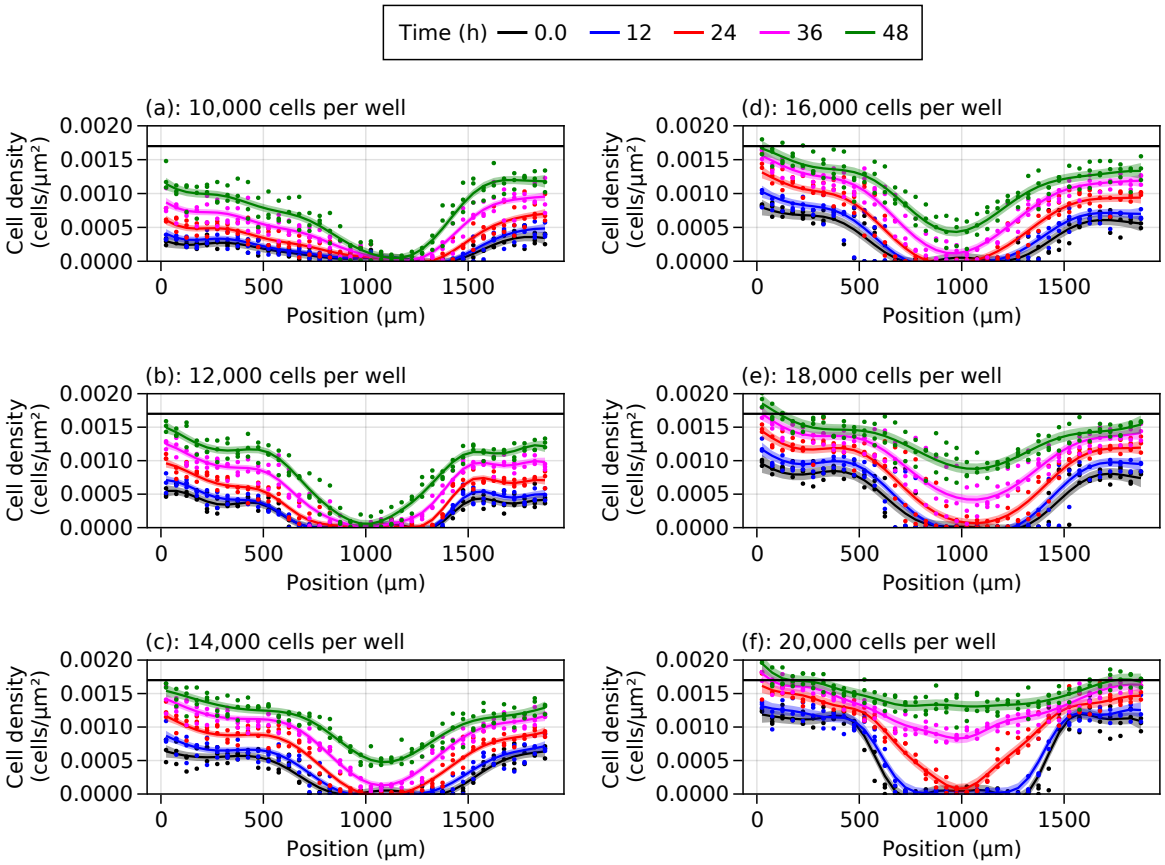
3 Results

In this section we show how well the Gaussian processes fit our data, and the results for our equation learning process on several models for some of the datasets from Jin et al. [1]. All plots in this section and the following are produced in Julia using the `Makie` package [38].

3.1 Gaussian processes

In Figure 1 we show Gaussian processes fitted to the data from Jin et al. [1] along with corresponding confidence intervals around each curve. These curves are fit using the `GaussianProcesses` package in Julia [19]. We give more details on the mathematical definition of this Gaussian process and how they are fitted in Section 5. These Gaussian processes are two-dimensional, being fit to both the spatial and temporal data. In Figure 2 we show the data in a space-time diagram to show the relationship between space x and time t on the density values u. The results show how the evolution of the density depends on the initial density. It is this data that is used to fit the Gaussian processes and sample from them to

Figure 1: Fitted Gaussian processes. Fitted Gaussian processes to the data from Jin et al. [1] with (a) 12,000 cells and (b) 20,000 cells per well. The shaded regions show the confidence intervals around the mean curves shown by the solid line. The points show the data.



allow for equation learning, as described in Section 5.

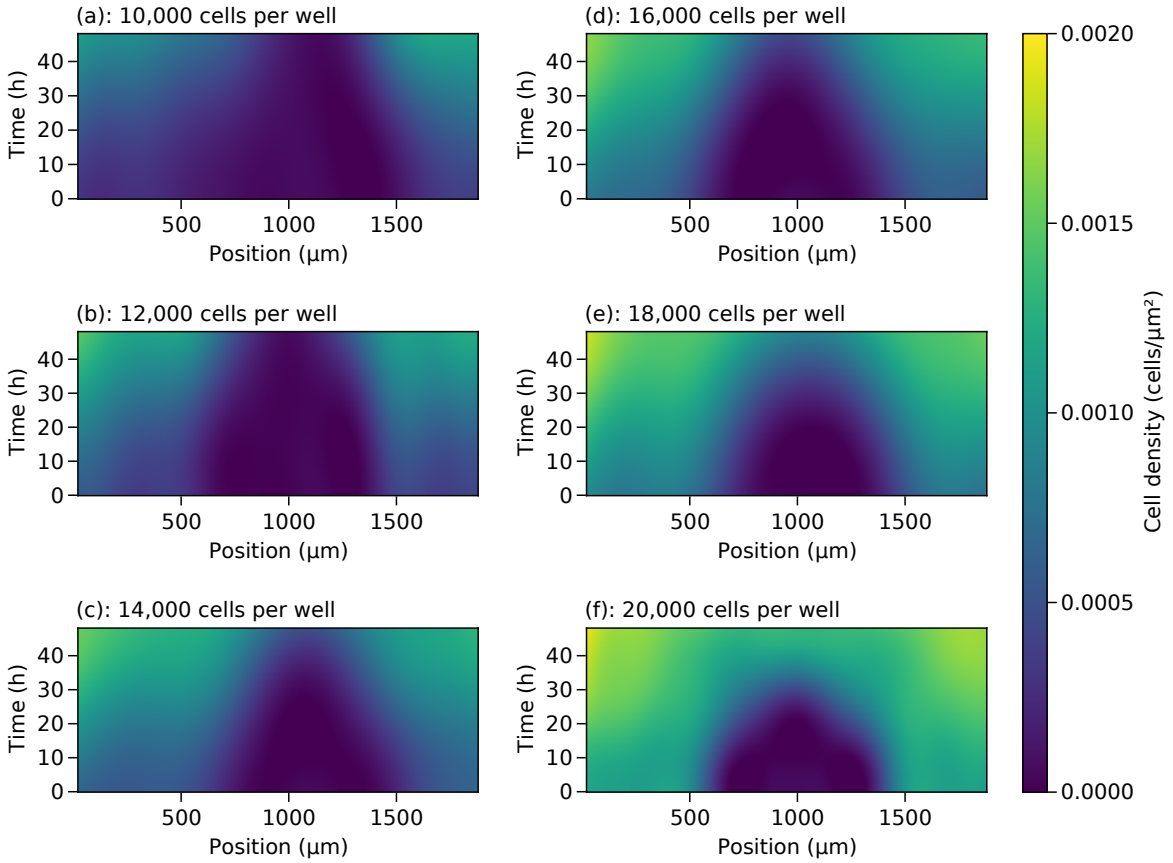
3.2 Equation learning

Now we describe some of the equation learning results. We only consider models fit to the data by Jin et al. [1] in this section; results for simulated data are described in the supplementary material. Let (x_i, t_j) describe the data point from Jin et al. [1] at the ith spatial point and jth time, e.g. $i = 4$ and $j = 2$ would correspond to point $x = 175\,\mu\text{m}$ along the well at time $t = 12\,\text{h}$. We then let u_{ij} be the density at this point (x_i, t_j). The equations we are interested in take the form

$$\frac{\partial u(x,t)}{\partial t} = T(t;\boldsymbol{\alpha})\left[\frac{\partial}{\partial x}\left(D(u(x,t);\boldsymbol{\beta})\frac{\partial u}{\partial x}\right) + R(u(x,t);\boldsymbol{\gamma})\right], \tag{1}$$

where T, D, and R are the delay, diffusion, and reaction functions, respectively. We suppose that these functions are parametrised by vectors $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, and $\boldsymbol{\gamma}$, respectively, that specify their functional forms — it

Figure 2: Space-time diagrams. Space-time diagrams from the fitted Gaussian processes to the data from Jin et al. [1] with (a) 12,000 cells and (b) 20,000 cells per well.



is these coefficients that need to be learned. The procedure used for estimating these coefficients relies on drawing samples from the Gaussian processes and using nonlinear least squares, as described in Section 5.

We first consider fitting a Fisher-Kolmogorov model to the data sets from Jin et al. [1] with 12,000 cells per well and 20,000 cells per well. The assumed forms of T, D, and R are thus

$$T(t;\boldsymbol{\alpha}) = \frac{1}{1+\exp(-\alpha_1-\alpha_2 t)}, \quad D(u;\boldsymbol{\beta}) = \beta_1, \quad \text{and} \quad R(u;\boldsymbol{\gamma}_1) = \gamma_1 u\left(1-\frac{u}{K}\right), \tag{2}$$

where $K = 1.7\times10^{-3}$ cells/μm^2 is the cell carrying capacity reported by Jin et al. [1]. This delay function is the same one used by [8]. We use 100 bootstrap iterations for each model.

4 Discussion

5 Methods

All procedures discussed below are implemented in the Julia language [18]. The data and code are available in a Julia package at https://github.com/DanielVandH/EquationLearning.jl. The following subsections describe (i) the data, (ii) the procedure for fitting the Gaussian processes, (iii) the equation learning procedure, (iv) the implementation of the parametric bootstrap for our work, and (v) the numerical procedure used to solve the underlying differential equations.

5.1 Scratch assay data

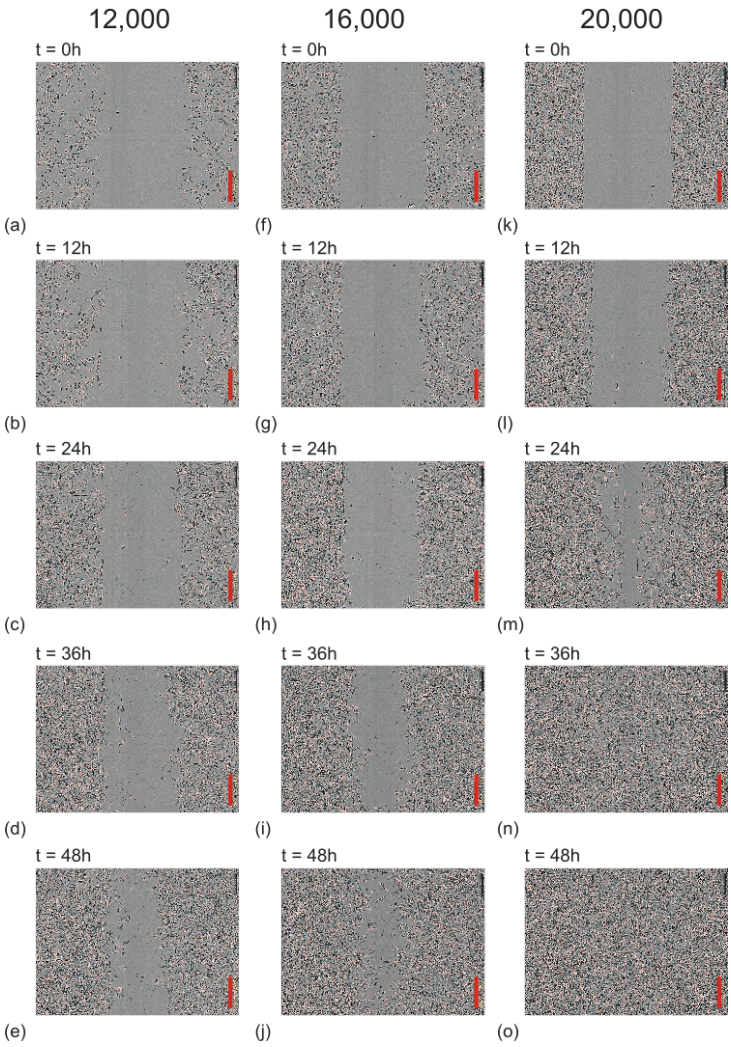
The methods we introduce in this paper are illustrated using the data set by Jin et al. [1]. In this experiment, Jin et al. [1] perform a monolayer scratch assay using the IncuCyte ZOOM™ system (Essen BioScience, MI USA). With a prepared set of cells distributed uniformly over wells, they use a WoundMaker™ (Essen BioScience) to create a scratch in the middle of the well, as can be seen in the early stages of the experiment shown in Figure 3. They then take measurements within each region between the blue vertical lines shown in Figure 4, giving measurements of cell densities $u(x, t)$ measured in cells/μm^2 at $x = 25, 75, \ldots, 1925$ μm at times $t = 0, 12, 24, 36, 48$ h. The carrying capacity K is estimated at $K = 1.7 \times 10^{-3}$ cells/μm^2 by measuring the density of cells in the blue rectangular regions in Figure 4. Given this estimate, we do not estimate K in this work and leave it fixed at $K = 1.7 \times 10^{-3}$ cells/μm^2, similarly to Jin et al. [1] and Lagergren et al. [8].

A plot of the data for each initial cell density and over $t = 0, 12, 24, 36, 48$ h is given in Figure 5. We see that most of the experiments do not reach capacity, and the rate of increase for the cell density is dependent on the initial cell density. Note that in this work we only focus on the data in (b) and (f) for the 12,000 and 20,000 cells per well data, respectively.

5.2 Fitting Gaussian processes

Suppose we have some spatial grid of N points $x_1, \ldots, x_N$, and a temporal grid of M points $t_1, \ldots, t_M$, where $x_1 < \cdots < x_N$ and $t_1 < \cdots < t_M$. For each point (x_i, t_j) on the spatiotemporal grid defined by

Figure 3: Scratch assay data. Images from the scratch assay experiments performed by Jin et al. [1] at 12,000, 16,000, and 20,000 cells per well in the first, second, and third columns, respectively. In each image, the red scale bar corresponds to 300 µm. The title of each image corresponds to the time of the measurement since the start of the experiment.



these points we associate a cell density $u(x_i, t_j) = u_{ij}$. We then define

$$\mathbf{u} = \begin{bmatrix} u_{11} \\ u_{21} \\ \vdots \\ u_{N1} \\ u_{12} \\ \vdots \\ u_{NM} \end{bmatrix} \in \mathbb{R}^{NM} \quad \text{and} \quad \mathbf{X} = \begin{bmatrix} x_1 & x_2 & \cdots & x_N & x_1 & \cdots & x_N \\ t_1 & t_1 & \cdots & t_1 & t_2 & \cdots & t_M \end{bmatrix} \in \mathbb{R}^{2 \times NM}. \tag{3}$$

Figure 4: Data setup and measuring cell carrying capacity. The data is split into columns at $x = 25, 75, \ldots, 1925\,\mu\text{m}$ as shown in (a). The cells are measured by focusing on subregions, such as shown in the yellow rectangle in (a), and counted using Adobe Photoshop to identify cells as shown in (b), giving a number of identified cells as indicated in (c). The carrying capacity is measured by counting the number of cells at $t = 48\,\text{h}$ in the blue rectangles shown in (d), divided by the area of these regions.

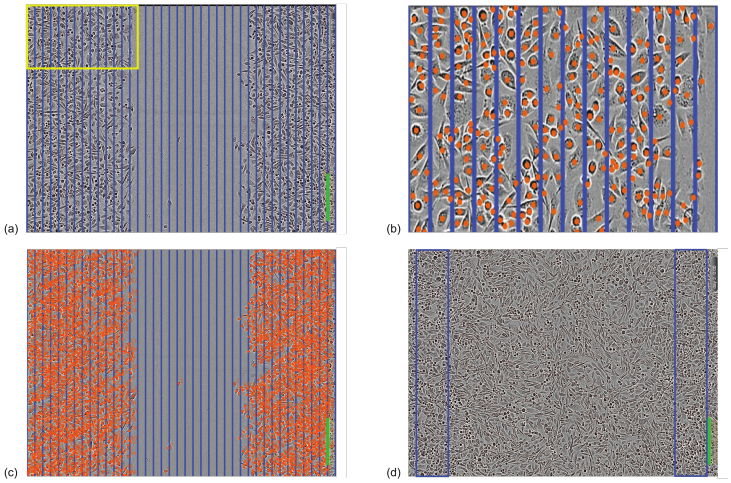
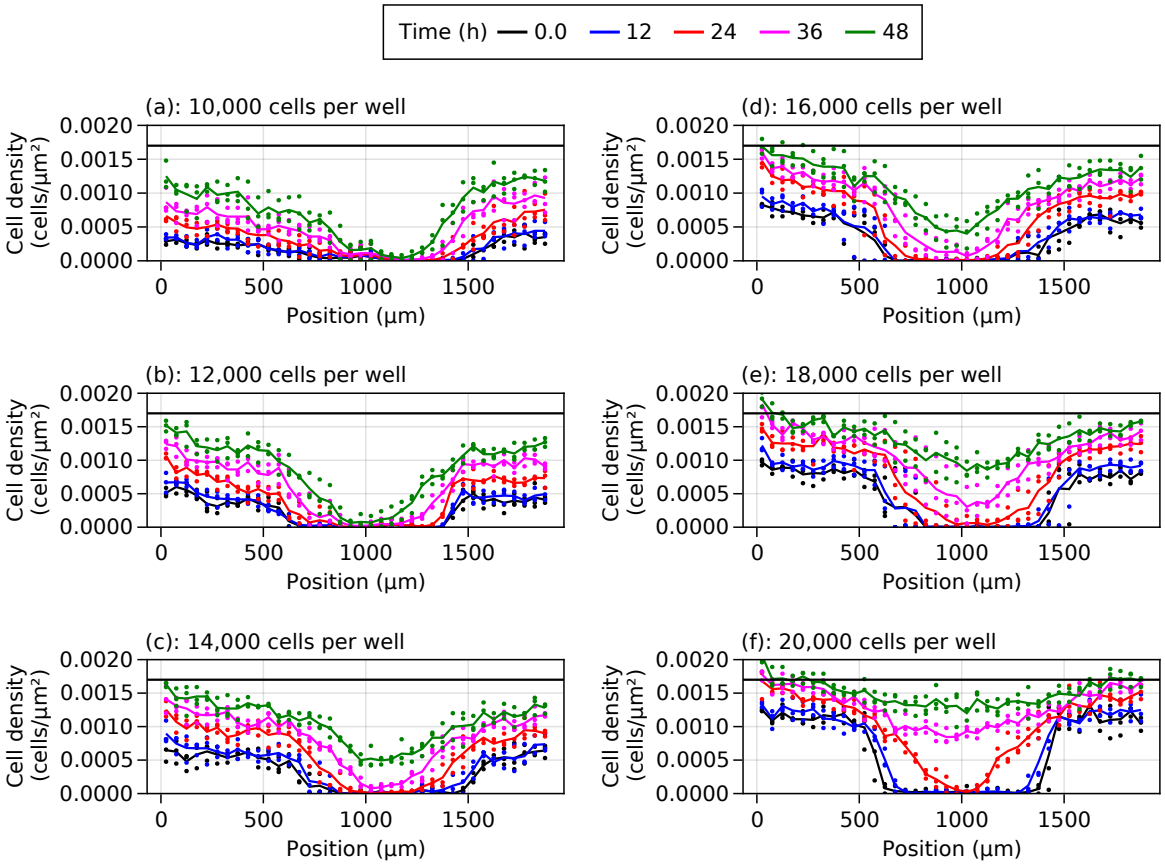


Figure 5: Scratch assay data. The scratch assay data from Jin et al. [1] for each initial cell density. The horizontal line in each plot is the estimated carrying capacity $K = 1.7 \times 10^{-3}\,\text{cells}/\mu\text{m}^2$. The points shown are the data and the lines show the average value over the three replications at each point and time.



Following Rasmussen et al. [17], we assume that we have some data $\mathbf{u} = \mathbf{u}_* + \sigma_n \mathbf{z}$ where $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, and we then define the Gaussian process

$$\mathbf{u}_* \mid \mathbf{X}, \mathbf{u}, \mathbf{X}_* \sim \mathcal{N}\left(\overline{\mathbf{u}}_*, \mathbb{C}\mathrm{ov}(\mathbf{u}_*)\right), \tag{4}$$

where $\mathbf{X}_* \in \mathbb{R}^{2 \times n_x n_t}$ contains new points defined over our original grid at $x_1^*, \ldots, x_{n_x}^*$ and $t_1^*, \ldots, t_{n_t}^*$ in space and time, respectively, aligned in a similar manner to the points (x_i, t_j) in $\mathbf{X}$ in Equation (3). The vector $\mathbf{u}_* \in \mathbb{R}^{n_x n_t \times 1}$ is a normal random variable for the values of the cell densities at these corresponding points. The mean and covariance for this Gaussian process are given by [17, Page 16]

$$\overline{\mathbf{u}}_* = \mathbf{K}(\mathbf{X}_*, \mathbf{X}) \left[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}_{NM}\right]^{-1} \mathbf{u}, \tag{5}$$

$$\mathbb{C}\mathrm{ov}(\mathbf{u}_*) = \mathbf{K}(\mathbf{X}_*, \mathbf{X}_*) - \mathbf{K}(\mathbf{X}_*, \mathbf{X}) \left[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}_{NM}\right]^{-1} \mathbf{K}(\mathbf{X}, \mathbf{X}_*), \tag{6}$$

where $\mathbf{I}_{NM}$ is the NM-square identity matrix, σ_n is the standard deviation of the noise in the data, and $\mathbf{K}(\mathbf{P}, \mathbf{Q})$ denotes the kernel matrix between $\mathbf{P}$ and $\mathbf{Q}$, with the (i, j)th entry giving the covariance between the ith column of $\mathbf{P}$, $\mathbf{p}_i$, and the jth column of $\mathbf{Q}$, $\mathbf{q}_j$, written as $k(\mathbf{p}_i, \mathbf{q}_j)$. There are many possible choices for the kernel k [17, Chapter 4] and in this paper we will only consider the squared-exponential kernel,

$$k(\mathbf{p}_i, \mathbf{q}_j) = \sigma_f^2 \exp\left\{-\frac{1}{2} \left(\mathbf{p}_i - \mathbf{q}_j\right)^\mathsf{T} \mathbf{\Lambda}^{-1} \left(\mathbf{p}_i - \mathbf{q}_j\right)\right\}, \tag{7}$$

where σ_f is the standard deviation of the noise-free data and $\mathbf{\Lambda} = \mathrm{diag}(\ell_1^2, \ell_2^2)$, where ℓ_1 is the spatial length scale and ℓ_2 is the temporal length scale. These hyperparameters σ_n, σ_f, ℓ_1, and ℓ_2 are estimated in Julia [18] using the `GaussianProcesses` package developed by Fairbrother et al. [19]. This package estimates the hyperparameters using maximum likelihood estimation with the `Optim` package by Mogensen et al. [20] which uses the L-BFGS algorithm to optimise the likelihood. This algorithm requires initial estimates for each hyperparameter. We choose rough bounds for each parameter and then use the `LatinHypercubeSampling` package [39] to generate w initial estimates for each parameters. For the resulting r sets of hyperparameters from each optimisation, we choose the set which gives the greatest marginal likelihood. We use $w = 50$ in this work and we take $\ell_1 \in [10^{-4}, 1]$, $\ell_2 \in [10^{-4}, 1]$, $\sigma_f \in [10^{-1}, 2\,\mathbb{SD}(u)]$, and $\sigma_n \in [10^{-5}, 2\,\mathbb{SD}(u)]$ for the rough bounds that define the hypercube, where $\mathbb{SD}(u)$ denotes an estimate of the standard deviation of the density data u. These length scale bounds are used as we scale the data between 0 and 1, as discussed below. We note that the final optimised hyperparameters are not enforced to stay within these bounds.

We remark that the expression for the mean in Equation (5) omits an extra noise term that would be

used to estimate the noisy data [17, Section 2.3] since we are interested only in the underlying function describing the cell density profile as a function of space and time rather than the actual values of the values of the data when considering noise.

In this work we choose the x_i^* to be 80 equally spaced points between x_1 and x_N, and the t_j^* are 75 equally spaced points between t_1 and t_M. We note that it is possible that a uniform spacing for these points may not be ideal, and future work may consider for example choosing these points optimally to pick up the most important information from the density data while minimising the number of points, $n_x n_t$, that have to be used. With these choices of x_i^* and t_j^*, the optimisation procedure was unstable due to the difference in spatial and temporal scales. To stabilise the estimates, we normalised the data to be between 0 and 1, defining

$$x_i^\dagger = \frac{x_i^* - x_i}{x_N - x_1} \quad \text{and} \quad t_j^\dagger = \frac{t_j^* - t_1}{t_N - t_1}, \quad i = 1, \ldots, n_x,\, j = 1, \ldots, n_t. \tag{8}$$

5.3 Estimates function values and derivatives

Following the fitting of the Gaussian process, we need estimates for the cell density and the corresponding derivatives. Consider our Gaussian process Equation (4), written as

$$\mathbf{u} \sim \mathcal{GP}\left(\mathbf{0}, \mathbf{K}(\mathbf{X}, \mathbf{X})\right). \tag{9}$$

Using the property

$$Lf(\mathbf{x}) \sim \mathcal{GP}(0, L_1 L_2 k(\mathbf{x}, \mathbf{x}')) \quad \text{if} \quad f(\mathbf{x}) \sim \mathcal{GP}(0, k(\mathbf{x}, \mathbf{x}')), \tag{10}$$

where L is a linear differential operator and L_i denotes L applied to the ith argument of $k(\mathbf{x}, \mathbf{x}')$ [40], and using similar ideas to those used by Wang et al. [33], we can show that, assuming as we did in Equation (4) that $\mathbf{u} = \mathbf{u}_* + \sigma_n \mathbf{z}$ with $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$,

$$\begin{bmatrix} \mathbf{u}_* \\ \frac{\partial \mathbf{u}_*}{\partial t^\dagger} \\ \frac{\partial u_*}{\partial x^\dagger} \\ \frac{\partial^2 \mathbf{u}_*}{\partial x^{\dagger 2}} \end{bmatrix} \mid \mathbf{X}, \mathbf{u}, \mathbf{X}_* \sim \mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\Sigma}\right), \tag{11}$$

where

$$\boldsymbol{\mu} = \begin{bmatrix} \mathbb{C}\text{ov}(\mathbf{u}_*, \mathbf{u}) \\ \mathbb{C}\text{ov}\left(\frac{\partial \mathbf{u}_*}{\partial t^\dagger}, \mathbf{u}\right) \\ \mathbb{C}\text{ov}\left(\frac{\partial \mathbf{u}_*}{\partial x^\dagger}, \mathbf{u}\right) \\ \mathbb{C}\text{ov}\left(\frac{\partial^2 \mathbf{u}_*}{\partial x^{\dagger 2}}, \mathbf{u}\right) \end{bmatrix} \left[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}_{NM}\right]^{-1} \mathbf{u}, \tag{12}$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} \mathbb{C}\text{ov}(\mathbf{u}_*, \mathbf{u}_*) & \mathbb{C}\text{ov}\left(\mathbf{u}_*, \frac{\partial \mathbf{u}_*}{\partial t^\dagger}\right) & \mathbb{C}\text{ov}\left(\mathbf{u}_*, \frac{\partial \mathbf{u}_*}{\partial x^\dagger}\right) & \mathbb{C}\text{ov}\left(\mathbf{u}_*, \frac{\partial^2 \mathbf{u}_*}{\partial x^{\dagger 2}}\right) \\ \mathbb{C}\text{ov}\left(\frac{\partial \mathbf{u}_*}{\partial t^\dagger}, \mathbf{u}_*\right) & \mathbb{C}\text{ov}\left(\frac{\partial \mathbf{u}_*}{\partial t^\dagger}, \frac{\partial \mathbf{u}_*}{\partial t^\dagger}\right) & \mathbb{C}\text{ov}\left(\frac{\partial \mathbf{u}_*}{\partial t^\dagger}, \frac{\partial \mathbf{u}_*}{\partial x^\dagger}\right) & \mathbb{C}\text{ov}\left(\frac{\partial \mathbf{u}_*}{\partial t^\dagger}, \frac{\partial^2 \mathbf{u}_*}{\partial x^{\dagger 2}}\right) \\ \mathbb{C}\text{ov}\left(\frac{\partial \mathbf{u}_*}{\partial x^\dagger}, \mathbf{u}_*\right) & \mathbb{C}\text{ov}\left(\frac{\partial \mathbf{u}_*}{\partial x^\dagger}, \frac{\partial \mathbf{u}_*}{\partial t^\dagger}\right) & \mathbb{C}\text{ov}\left(\frac{\partial \mathbf{u}_*}{\partial x^\dagger}, \frac{\partial \mathbf{u}_*}{\partial x^\dagger}\right) & \mathbb{C}\text{ov}\left(\frac{\partial \mathbf{u}_*}{\partial x^\dagger}, \frac{\partial^2 \mathbf{u}_*}{\partial x^{\dagger 2}}\right) \\ \mathbb{C}\text{ov}\left(\frac{\partial^2 \mathbf{u}_*}{\partial x^{\dagger 2}}, \mathbf{u}_*\right) & \mathbb{C}\text{ov}\left(\frac{\partial^2 \mathbf{u}_*}{\partial x^{\dagger 2}}, \frac{\partial \mathbf{u}_*}{\partial t^\dagger}\right) & \mathbb{C}\text{ov}\left(\frac{\partial^2 \mathbf{u}_*}{\partial x^{\dagger 2}}, \frac{\partial \mathbf{u}_*}{\partial x^\dagger}\right) & \mathbb{C}\text{ov}\left(\frac{\partial^2 \mathbf{u}_*}{\partial x^{\dagger 2}}, \frac{\partial^2 \mathbf{u}_*}{\partial x^{\dagger 2}}\right) \end{bmatrix}^{-1}$$
$$- \begin{bmatrix} \mathbb{C}\text{ov}(\mathbf{u}_*, \mathbf{u}) \\ \mathbb{C}\text{ov}\left(\frac{\partial \mathbf{u}_*}{\partial t^\dagger}, \mathbf{u}\right) \\ \mathbb{C}\text{ov}\left(\frac{\partial \mathbf{u}_*}{\partial x^\dagger}, \mathbf{u}\right) \\ \mathbb{C}\text{ov}\left(\frac{\partial^2 \mathbf{u}_*}{\partial x^{\dagger 2}}, \mathbf{u}\right) \end{bmatrix} \left[\mathbf{K}(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}_{NM}\right]^{-1} \begin{bmatrix} \mathbb{C}\text{ov}(\mathbf{u}, \mathbf{u}_*) & \mathbb{C}\text{ov}\left(\mathbf{u}, \frac{\partial \mathbf{u}_*}{\partial t^\dagger}\right) & \mathbb{C}\text{ov}\left(\mathbf{u}, \frac{\partial \mathbf{u}_*}{\partial x^\dagger}\right) & \mathbb{C}\text{ov}\left(\mathbf{u}, \frac{\partial^2 \mathbf{u}_*}{\partial x^{\dagger 2}}\right) \end{bmatrix}. \tag{13}$$

Note that these derivatives are on the scale of the data that we put between 0 and 1 in Equation (8). They can be put onto the correct scale by dividing by the denominators in Equation (8), for example the temporal derivatives $\partial/\partial t^\dagger$ are related to $\partial/\partial t$ by $\partial/\partial t = [1/(t_M - t_1)]\partial/\partial t^\dagger$ and similarly $\partial^2/\partial x^2 = [1/(x_N - x_1)^2]\partial^2/\partial t^{\dagger 2}$. These covariances in Equation (12) and Equation (13) are computed using the property in Equation (10), for example

$$\mathbb{C}\text{ov}\left(\mathbf{u}, \frac{\partial \mathbf{u}_*}{\partial t^\dagger}\right) = \left\{ \frac{\partial}{\partial t_j^\dagger} k\left(x_i, t_i, x_j^\dagger, t_j^\dagger\right) \right\}_{\substack{i=1,\ldots,NM, \\ j=1,\ldots,n_x n_t}} \tag{14}$$

and

$$\mathbb{C}\text{ov}\left(\frac{\partial^2 \mathbf{u}_*}{\partial x^{\dagger 2}}, \frac{\partial \mathbf{u}_*}{\partial t^\dagger}\right) = \left\{ \frac{\partial^3}{\partial x_i^{\dagger 2} \partial t_j^\dagger} k\left(x_i^\dagger, t_i^\dagger, x_j^\dagger, t_j^\dagger\right) \right\}_{i,j=1,\ldots,n_x n_t}, \tag{15}$$

with k as defined in Equation (7). The values of the hyperparameters used, σ_n, σ_f, ℓ_1, and ℓ_2, are taken to be those obtained from the optimisation process discussed in the last section for the Gaussian process Equation (4).

To now use Equation (11) to sample the function values and corresponding derivatives, let $\mathbf{U}$ be the random vector on the left of Equation (11). We then write $\mathbf{U} = \boldsymbol{\mu} + \mathbf{L}\mathbf{z}$, where $\mathbf{L}$ is the Cholesky factor of $\boldsymbol{\Sigma}$ such that $\mathbf{L}\mathbf{L}^\mathsf{T} = \boldsymbol{\Sigma}$, and $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Drawing a sample $\mathbf{z}$ from $\mathcal{N}(\mathbf{0}, \mathbf{I})$ thus constitutes a draw from $\mathbf{U}$ from which we can obtain the function values and derivatives. All Cholesky decompositions are

computed using the `cholesky` function from the `LinearAlgebra` package in Julia. For these Cholesky factors we need to ensure that they are symmetric positive definite. To do so we follow the approach by Chen et al. [34]. In particular, let us write the diagonal entries of the matrix $\boldsymbol{\Sigma}$ in Equation (13) as $\text{diag}(\boldsymbol{\Sigma}) = \text{diag}(\boldsymbol{\Sigma}_{11}, \boldsymbol{\Sigma}_{22}, \boldsymbol{\Sigma}_{33}, \boldsymbol{\Sigma}_{44})$, where each block is of size $n_x n_t \times n_x n_t$; for example, $\boldsymbol{\Sigma}_{22}$ corresponds to the covariance matrix for $\partial \mathbf{u}_* / \partial t^\dagger$. We then set $\tau_{ii} = \text{tr}(\boldsymbol{\Sigma}_{ii})$ for $i = 1, 2, 3, 4$, where $\text{tr}(\mathbf{A})$ denotes the trace of the matrix $\mathbf{A}$, and define the nugget terms

$$\eta_1 = \eta, \quad \eta_2 = \frac{\tau_{22}}{\tau_{11}}\eta, \quad \eta_3 = \frac{\tau_{33}}{\tau_{11}}\eta, \quad \text{and} \quad \eta_4 = \frac{\tau_{44}}{\tau_{11}}\eta, \tag{16}$$

where η is chosen to be 10^{-5} in this paper. We then define $\boldsymbol{\eta} = \text{diag}(\eta_1 \mathbf{I}_{n_x n_t}, \eta_2 \mathbf{I}_{n_x n_t}, \eta_3 \mathbf{I}_{n_x n_t}, \eta_4 \mathbf{I}_{n_x n_t})$ and replace $\boldsymbol{\Sigma}$ with $\boldsymbol{\Sigma} + \boldsymbol{\eta}$. This procedure for regularising $\boldsymbol{\Sigma}$ to ensure it remains symmetric positive definite allows the nugget terms to adapt to the scales of the corresponding derivative term relative to the function values.

5.4 Equation learning and nonlinear optimisation

In this section we assume that we have already drawn some function values $\mathbf{u}_*$ and derivatives $\partial \mathbf{u}_* / \partial t$, $\partial \mathbf{u}_* / \partial x$, and $\partial^2 \mathbf{u}_* / \partial x^2$, where the derivatives have been transformed onto the original scale rather than the unit interval. These values are obtained over the gridpoints $(x_i^\dagger, t_j^\dagger)$ for $i = 1, \ldots, n_x$ and $j = 1, \ldots, n_t$. We can use these values to estimate each term in Equation (1), where we expand the flux term and write for $i = 1, \ldots, n_x$ and $j = 1, \ldots, n_t$,

$$\frac{\partial u_*(x_i^\dagger, t_j^\dagger)}{\partial t} = T\left(t_j^*; \boldsymbol{\alpha}\right) \left[\frac{\mathrm{d}D\left(u_*(x_i^\dagger, t_j^\dagger); \boldsymbol{\beta}\right)}{\mathrm{d}u} \left(\frac{\partial u_*(x_i^\dagger, t_j^\dagger)}{\partial x} \right)^2 + D\left(u_*(x_i^\dagger, t_j^\dagger); \boldsymbol{\beta}\right) \frac{\partial^2 u_*(x_i^\dagger, t_j^\dagger)}{\partial x^2} + R\left(u_*(x_i^\dagger, t_j^\dagger); \boldsymbol{\gamma}\right) \right]. \tag{17}$$

Note that Equation (17) requires that D is differentiable at all the gridpoints.

To estimate the parameters $\boldsymbol{\theta} = (\boldsymbol{\alpha}^\mathsf{T}, \boldsymbol{\beta}^\mathsf{T}, \boldsymbol{\gamma}^\mathsf{T})^\mathsf{T}$ we need to first define a loss function to be optimised. Following Lagergren et al. [8], we define two loss functions based on Equation (17) and differences between a numerical solution of Equation (1) and the actual data. We define $\widehat{u}_{ij}$ to be an estimate of the cell density u at (x_i, t_j) for $i = 1, \ldots, N$ and $j = 1, \ldots, M$, and $\widehat{u}_{ij}^* = u_*(x_i^\dagger, t_j^\dagger)$ is the estimate from the Gaussian process for the cell density, and similarly for the function values. We then define the two loss

functions:

$$\mathcal{L}_{\text{PDE}}(\boldsymbol{\theta}) = \frac{1}{n_x n_t} \sum_{i=1}^{n_x} \sum_{j=1}^{n_t} \left\{ \frac{\partial \widehat{u}_{ij}^*}{\partial t} - T(t_j^*; \boldsymbol{\alpha}) \left[\frac{\mathrm{d}D\left(\widehat{u}_{ij}^*; \boldsymbol{\beta}\right)}{\mathrm{d}u} \left(\frac{\partial \widehat{u}_{ij}^*}{\partial x} \right)^2 + D\left(\widehat{u}_{ij}^*; \boldsymbol{\beta}\right) \frac{\partial^2 \widehat{u}_{ij}^*}{\partial x^2} + R\left(\widehat{u}_{ij}^*; \boldsymbol{\gamma}\right) \right] \right\}^2, \tag{18}$$

$$\mathcal{L}_{\text{GLS}}(\boldsymbol{\theta}) = \frac{1}{NM} \sum_{i=1}^{N} \sum_{j=1}^{M} \left(\frac{\widehat{u}_{ij} - u_{ij}}{\sigma_n} \right)^2. \tag{19}$$

These values $\widehat{u}_{ij}$ are obtained by solving Equation (1) at the given parameter values $\boldsymbol{\theta}$. To solve the differential equations we use the Julia package `DifferentialEquations` [28] as described later. Note that we divide by σ_n to make the residual $\widehat{u}_{ij} - u_{ij}$ have constant variance, assuming our Gaussian process model holds, motivated by the generalised least squares (GLS) residual developed by Lagergren et al. [8, 41]. To counteract the fact that $\mathcal{L}_{\text{PDE}}$ and $\mathcal{L}_{\text{GLS}}$ are on quite different scales, we take the logarithm of each and define the loss function $\mathcal{L}(\boldsymbol{\theta}) = \log[\mathcal{L}_{\text{PDE}}(\boldsymbol{\theta})] + \log[\mathcal{L}_{\text{GLS}}(\boldsymbol{\theta})]$, allowing the optimiser to more easily converge with these smaller function values. We then obtain estimates $\widehat{\boldsymbol{\theta}} = (\widehat{\boldsymbol{\alpha}}^\mathsf{T}, \widehat{\boldsymbol{\beta}}^\mathsf{T}, \widehat{\boldsymbol{\gamma}}^\mathsf{T})^\mathsf{T}$ by solving the optimisation problem

$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \mathbb{R}^{a+d+r}}{\operatorname{argmin}} \, \mathcal{L}(\boldsymbol{\theta}), \tag{20}$$

assuming that $\boldsymbol{\alpha} \in \mathbb{R}^a$, $\boldsymbol{\beta} \in \mathbb{R}^d$, and $\boldsymbol{\gamma} \in \mathbb{R}^r$. To ensure that the found parameter values lead to physically realistic functional forms, we set the loss to be infinite if there are any negative delay or diffusion function values, or if the integral of the reaction function over $[0, u_M]$, where u_M is the maximum density value, is negative. This integral $\int_0^{u_M} R(u)\,\mathrm{d}u$ is computed numerically using Gauss-Legendre quadrature, computing the Gauss-Legendre nodes using the `FastGaussQuadrature` package [45] which implements the method by Bogaert [46]. We use the `Optim` package in Julia [20] to solve this optimisation problem with the L-BFGS algorithm. The `Optim` package leverages the `ForwardDiff` package [42] for computing gradients and Hessians of $\mathcal{L}$ with respect to $\boldsymbol{\theta}$.

The optimisation problem Equation (20) can be very sensitive to the initial values used for each parameter, especially for $\boldsymbol{\beta}$ which is not surprising given the results observed by Jin et al. [1] and convergence problems identified by Kaltenbacher et al. [32] in their fixed point approach to recovering nonlinear terms in reaction-diffusion equations. To overcome this problem we choose a range of initial values within some hypercube based on individual bounds for each parameter. Similar to our optimisation procedure for fitting the Gaussian processes, the Julia package `LatinHypercubeSampling` [39] is used for this purpose. In this work we only use multiple initial values for say 10 bootstrap iterations (with bootstrapping defined in the next section) with say five initial sets of parameters chosen. Using these 10 results we obtain rough estimates for each parameter θ_i in $\boldsymbol{\theta}$, $i = 1, \ldots, a + d + r$, which we then

use to scale the parameters such that $\theta_i = \mathcal{O}(1)$ for each $i = 1, \ldots, a + d + r$. This scaling puts all parameters on the same scale so that multiple initial values are not typically required, and we may simply start each parameter at $\theta_i = 1$ for $i = 1, \ldots, a + d + r$. This scaling also helps the convergence of the algorithm in general since relative scaling between parameters is a crucial factor in the optimiser's performance [25, Section 7.5]. To scale further, in this work we put the data from Jin et al. [1] in units of mms for space and days for time so that the coefficients are closer together in magnitude than they would be in the original units. If it does turn out that more restarts are required, one could use for example the high-confidence stopping rule of Dick et al. [26] and use random restarts around the value $\theta_i = 1$ for each $i = 1, \ldots, a + d + r$.

5.5 Bootstrapping

Now we discuss how we use the above the results to perform parametric bootstrapping, providing us with uncertainty quantification for our learned coefficients, functional forms, and solutions to PDEs. We start with the Gaussian process Equation (9) and assume that $\boldsymbol{\mu}$ and $\mathbf{L}$, the Cholesky factor of $\boldsymbol{\Sigma}$, have already been computed. We then draw B random variates $\mathbf{z}^{(b)}$, $b = 1, \ldots, B$, where each $\mathbf{z}^{(b)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$. Note that $\boldsymbol{\mu}$ and $\mathbf{L}$ only have to be computed once. We can then compute $\mathbf{U}^{(b)} = \boldsymbol{\mu} + \mathbf{L}\mathbf{z}^{(b)}$ for each $b = 1, \ldots, B$, giving B sets of parameter vectors $\widehat{\boldsymbol{\theta}}^{(b)}$ that are solutions to the optimisation problem Equation (20), noting that each $\mathbf{U}^{(b)}$ leads to a different set of function values and derivatives. These results in turn lead to B sets of parameter vectors $\widehat{\boldsymbol{\alpha}}^{(b)}$, $\widehat{\boldsymbol{\beta}}^{(b)}$, and $\widehat{\boldsymbol{\gamma}}^{(b)}$, for $b = 1, \ldots, B$. In this work we mostly use $B = 100$. This process could easily be parallelised as the parameter estimation for each b is independent of the other bootstrap iterations.

The B samples for each individual parameter θ_i, $i = 1, \ldots, a+d+r$, defines a sampling distribution for that parameter, which we can then use to extract quantiles and density estimates. The `KernelDensity` package in Julia is used for obtaining kernel density estimates for each parameter. We obtain 95% confidence intervals using Julia's `quantile` function at levels 2.5% and 97.5%, giving us a sense for the uncertainty in each parameter.

In addition to being able to obtain confidence intervals for parameter estimates, we can obtain confidence intervals for the functional forms themselves by simply computing the corresponding B curves for each learned functional form over some grid of values, and then taking the quantiles at each point. For example, if we had $D(u; \boldsymbol{\beta})$ we could evaluate this function for each $\boldsymbol{\beta}^{(b)}$, $b = 1, \ldots, B$, over some range of values $[u_m, u_M]$ of u, say $[0, K]$. These evaluations provide B samples for the diffusion curve on

this range. We then take each point used in $[u_m, u_M]$ and compute the 2.5% and 97.5% quantiles, along with the mean of the samples at this point. We can then plot these values to give the mean value of the function at each u, and a ribbon plot can be used to demonstrate the uncertainty around these function values. Similar ideas are obtained for obtaining confidence intervals for the PDE solutions.

5.6 Numerical solution of the partial differential equations and uncertainty

Here we describe the procedure for numerically solving Equation (1). We note that while the boundary conditions used for the data considered in this paper are

$$\frac{\partial u(0,t)}{\partial x} = \frac{\partial u(1900,t)}{\partial x} = 0, \tag{21}$$

following Jin et al. [1], to allow for generality in future problems we consider

$$\begin{aligned} \frac{\partial u}{\partial t} &= T(t)\left[\frac{\partial}{\partial x}\left(D(u)\frac{\partial u}{\partial x}\right) + R(u)\right] && a < x < b,\ t > 0, \\ u(x,0) &= f(x) && a \le x \le b, \\ a_0 u(a,t) - b_0 \frac{\partial u}{\partial x} &= c_0 && t > 0, \\ a_1 u(b,t) + b_1 \frac{\partial u}{\partial x} &= c_1 && t > 0, \end{aligned} \tag{22}$$

with $b_0, b_1 \neq 0$. The boundary conditions in Equation (21) correspond to $a_0 = a_1 = c_0 = c_1 = 0$ and $b_0 = b_1 = 1$ with $a = 0$ and $b = 1900$.

The discretisation of Equation (22) proceeds by using a vertex-centred finite volume method [27]. We place N_p nodes $x_1, \ldots, x_{N_p}$ over some mesh (with these x_i being distinct from the data discussed), with $a = x_1 < x_2 < \cdots < x_{N_p - 1} < x_{N_p} = b$. We assume that these points are equally spaced so that $\Delta x = x_{i+1} - x_i$ for $i = 1, \ldots, N_p - 1$ is constant. In particular, we set $x_j = a + (j-1)\Delta x$ with $\Delta x = (b-a)/(N_p - 1)$ for $j = 1, \ldots, N_p$. We then set

$$w_i = \begin{cases} x_1 & i = 1, \\ \frac{1}{2}(x_{i-1} + x_i) & i = 2, \ldots, N_p, \end{cases} \quad e_i = \begin{cases} \frac{1}{2}(x_i + x_{i+1}) & i = 1, \ldots, N_p - 1, \\ x_{N_p} & i = N_p, \end{cases} \quad V_i = \begin{cases} \frac{1}{2}\Delta x & i \in \{1, N_p\}, \\ \Delta x & i = 2, \ldots, N_p - 1. \end{cases} \tag{23}$$

We then define the control volume averages

$$\bar{u}_i(t) = \frac{1}{V_i}\int_{w_i}^{e_i} u(x,t)\,\mathrm{d}x \quad \text{and} \quad \bar{R}_i(t) = \frac{1}{V_i}\int_{w_i}^{e_i} R(u(x,t))\,\mathrm{d}x\,, \quad i = 1,\ldots,N_p,\ t > 0. \tag{24}$$

With these definitions, Equation (22) becomes

$$\frac{\mathrm{d}\bar{u}_i}{\mathrm{d}t} = T(t)\left\{\frac{1}{V_i}\left[D(u(e_i,t))\frac{\partial u(e_i,t)}{\partial x} - D(u(w_i,t))\frac{\partial u(w_i,t)}{\partial x}\right] + \bar{R}_i\right\}, \quad i = 1,\ldots,N_p,\ t > 0. \tag{25}$$

We then make the following approximations and let $u_i = u(x_i,t)$, $R_i = R(u_i)$, and $D_i = D(u_i)$:

$$\bar{u}_i(t) \approx u_i, \quad \bar{R}_i(t) \approx R_i, \quad i = 2,\ldots,N_p, \tag{26}$$

$$D(u(w_i,t)) \approx \frac{D_{i-1}+D_i}{2}, \quad i = 2,\ldots,N_p, \quad D(u(e_i,t)) = \frac{D_i + D_{i+1}}{2}, \quad i = 1,\ldots,N_p-1, \tag{27}$$

$$\frac{\partial u(w_i,t)}{\partial x} \approx \frac{u_i - u_{i-1}}{\Delta x}, \quad i = 2,\ldots,N_p, \quad \frac{\partial u(e_i,t)}{\partial x} \approx \frac{u_{i+1}-u_i}{\Delta x}, \quad i = 1,\ldots,N_p-1. \tag{28}$$

We thus obtain the following system of differential equations that approximates Equation (22):

$$\begin{aligned}
\frac{\mathrm{d}u_1}{\mathrm{d}t} &= T(t)\left\{\frac{2}{\Delta x}\left[\left(\frac{D_1+D_2}{2}\right)\left(\frac{u_2-u_1}{\Delta x}\right) - \frac{a_0}{b_0}D_1u_1\right] + \frac{2c_0}{b_0\Delta x}D_1 + R_1\right\},\\
\frac{\mathrm{d}u_i}{\mathrm{d}t} &= T(t)\left\{\frac{1}{2\Delta x^2}\left[(D_i+D_{i+1})(u_{i+1}-u_i) - (D_{i-1}+D_i)(u_i-u_{i-1})\right] + R_i\right\}, \quad i = 2,\ldots,N_p-1,\\
\frac{\mathrm{d}u_{N_p}}{\mathrm{d}t} &= T(t)\left\{\frac{2}{\Delta x}\left[-\frac{a_1}{b_1}D_{N_p}u_{N_p} - \left(\frac{D_{N_p-1}+D_{N_p}}{2}\right)\left(\frac{u_{N_p}-u_{N_p-1}}{\Delta x}\right)\right] + \frac{2c_1}{b_1\Delta x}D_{N_p} + R_{N_p}\right\}.
\end{aligned} \tag{29}$$

It is easy to extend Equation (29) to problems with Dirichlet boundary conditions by modifying the equations for $\mathrm{d}u_1/\mathrm{d}t$ and $\mathrm{d}u_{N_p}/\mathrm{d}t$ appropriately. We solve this system of differential equations in Equation (29) using the Julia package `DifferentialEquations` [28] which computes Jacobians using the `ForwardDiff` package [42].

We now discuss the choice of initial condition $u(x,0) = f(x)$ used for obtaining the initial values for solving Equation (29). One choice is to follow Jin et al. [1] and simply fit a linear spline through the averaged density data at time $t = 0$. This fitting can be accomplished using the implementation of the Dierckx Fortran library [30] in Julia available from the `Dierckx` package.

One other choice is to allow the initial condition to be sampled rather than fixed based on the experimental data, noting that we expect there to be significant noise in the data considered in this paper. Simpson et al. [31] demonstrated in the context of population growth that estimating an initial

condition rather than fixing it based on observed data leads to significantly improved results, and it turns out that for this data set by Jin et al. [1] we observe similar improvements, with better fitting curves and variability that captures the data.

The initial conditions can be sampled as follows. Suppose we are solving Equation (1) with parameters $\widehat{\boldsymbol{\theta}}^{(b)}$ from the bth bootstrap iteration. These parameters will correspond to some densities $\mathbf{u}_*^{(b)}$. We can then consider the initial condition to be those densities corresponding to $t = 0$. For these densities, we note that the Gaussian processes may give negative values, and we therefore need to replace any negative densities with zero. Extensions to this work may implement constrained Gaussian processes to prohibit any negative values [44], using for example the truncated Gaussian process framework developed by Da Veiga et al. [43]. To allow for a larger mesh for the PDE grid than the $n_x \times n_t$ grid used for bootstrapping, we can then use a linear spline to fit through these densities and then evaluate the spline over the PDE mesh. This evaluation defines an initial condition corresponding to the bth bootstrap iteration which can then be used to solve the PDE.

Regardless of the choice of initial condition, we end up with B solutions to our PDE over the spatial mesh and at some times. In particular, at each point (x_i, t_j) on our grid we have B solutions $u^{(b)}(x_i, t_j)$ for $b = 1, \ldots, B$, which can then be used to obtain the 2.5% and 97.5% quantiles at each point. We can therefore present confidence intervals for our solution curves, thus enabling us to quantify the uncertainty not only in the functional forms, but also in the solutions themselves.

6 Acknowledgements

DJV acknowledges funding support from the QUT Centre for Data Science. CCD acknowledges support from the Australian Research Council Future Fellowship Award FT210100260.

References

[1] Jin W, Shah ET, Penington CJ, McCue SW, Chopin LK, Simpson MJ. Reproducibility of scratch assays is affected by the initial degree of confluence: Experiments, modelling and model selection. Journal of Theoretical Biology. 2016; 390:136–145.

[2] Pijuan J, Barceló C, Moreno DF, Maiques O, Sisó P, Marti RM, Macià A, Panosa A. In vitro cell migration, invasion, and adhesion assays: from cell imaging to data analysis. Frontiers in Cell and Developmental Biology. 2019; 7:107.

[3] Veiseh O, Kievit FM, Ellenbogen RG, Zhang M. Cancer cell invasion: treatment and monitoring opportunities in nanomedicine. Advanced Drug Delivery Reviews. 2011; 8:582–596.

[4] Jeong H, Tombor B, Albert R, Oltvai ZN, Barabási, A-L. The large-scale organization of metabolic networks. Nature. 2000; 407:651–654.

[5] Dhar PK, Giuliani A. Laws of biology: why so few? Systems and Synthetic Biology. 2010; 4:7–13.

[6] Brunton SL, Proctor JL, Kutz JN. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. Proceedings of the National Academy of Sciences. 2016; 113:3932–3937.

[7] Rudy SH, Brunton SL, Proctor JL, Kutz JN. Data-driven discovery of partial differential equations. Science Advances. 2017; 3:e1602614.

[8] Lagergren JH, Nardini JT, Baker RE, Simpson MJ, Flores KB. Biologically-informed neural networks guide mechanistic modeling from sparse experimental data. 2020; 16:e1008462.

[9] Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics. 2019; 378:686–707.

[10] Chen Z, Liu Y, Hao S. Physics-informed learning of governing equations from scarce data. Nature Communications. 2021; 12.

[11] Lima-Mendez G, van Helden J. The powerful law of the power law and other myths in network biology. Molecular Biosystems. 2009; 12:1482–1493.

[12] Martina-Perez S, Simpson MJ, Baker RE. Bayesian uncertainty quantification for data-driven equation learning. Proceedings of the Royal Society A. 477:20210426.

[13] Graf O, Flores P, Protopapas P, Pichara K. Uncertainty quantification in neural differential equations. arXiv preprint arXiv:2111.04207.2021.

[14] Abdar M, Pourpanah F, Hussain S, Rezazadegan D, Liu L, Ghavamzadeh M, Fieguth P, Cao X, Khosravi A, Acharya UR, Makarenkov V. A review of uncertainty quantification in deep learning: Techniques, applications and challenges. Information Fusion. 2021; 76:243–297.

[15] Binois M, Gramacy RB, Ludkovski M. Practical heteroscedastic Gaussian process modeling for large simulation experiments. Journal of Computational and Graphical Statistics. 2018; 27:808–821.

[16] Duvenaud D. Automatic model construction with Gaussian processes (Doctoral dissertation, University of Cambridge).

[17] Rasmussen CE, Williams CKI. Gaussian Processes for Machine Learning. Cambridge: MIT Press; 2006.

[18] Bezanson J, Edelman A, Karpinski S, Shah VB. Julia: A Fresh Approach to Numerical Computing. SIAM Review. 2017; 59:65–98.

[19] Fairbrother J, Nemeth C, Rischard M, Brea J, Pinder T. GaussianProcesses.jl: A Nonparametric Bayes package for the Julia Language. arXiv:1812.09064v2 [Preprint]. 2019 [cited 2022 January 19].

[20] Mogensen PK, Riseth AN. Optim: A mathematical optimization package for Julia. Journal of Open Source Software. 2018; 3:615.

[21] Kroese DK, Botev ZI, Taimre T, Vaisman R. Data Science and Machine Learning: Mathematical and Statistical Methods. Boca Raton: CRC Press; 2019.

[22] Sauer T. Numerical Analysis. Hoboken: Pearson Education; 2012.

[23] Kochenderfer MJ, Wheeler TA. Algorithms for Optimization. Cambridge: MIT Press; 2019.

[24] Jin W, Shah ET, Penington CJ, McCue SW, Maini PK, Simpson MJ. Logistic proliferation of cells in scratch assays is delayed. Bulletin of Mathematical Biology. 2017; 79:1028–1050.

[25] Gill PE, Murray W, Wright MH. Practical Optimization. San Diego: Academic Press; 1997.

[26] Dick T, Wong E, Dann C. How many random restarts are enough? Technical Report; 2014.

[27] Versteeg HK, Malalasekera W. An Introduction to Computational Fluid Dynamics. Harlow: Prentice Hall; 2007.

[28] Rackauckas C, Nie Q. DifferentialEquations.jl - A performant and and feature-rich ecosysem for solving differential equations in Julia. The Journal of Open Research Software. 2017; 5.

[29] Hindmash AC, Brown PN, Grant KE, Lee SL, Serban R, Shumaker DE, Woodward CS. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. ACM Transactions on Mathematical Software. 2005; 31:363–396.

[30] Dierckx P. Curve and Surface Fitting with Splines. Oxford: Oxford University Press; 1993.

[31] Simpson MJ, Browning AP, Warne DJ, Maclaren OJ, Baker RE. Parameter identifiability and model selection for sigmoid population growth models. Journal of Theoretical Biology. 2022; 535:110998.

[32] Kaltenbacher B, Rundell William. On the identification of a nonlinear term in a reaction-diffusion equation. Inverse Problems. 2019; 35:115007.

[33] Wang H, Zhou X. Explicit estimation of derivatives from data and differential equations by Gaussian process regression. International Journal for Uncertainty Quantification. 2021; 11:41–57.

[34] Chen Y, Hosseini B, Owhadi H, Stuart AM. Solving and learning nonlinear PDEs with Gaussian processes. Journal of Computational Physics. 2021; 447:110668.

[35] Heinonen M, Yildiz C, Mannerström H, Intosalmi J, Lähdesmäki H. Learning unknown ODE models with Gaussian processes. International Conference on Machine Learning. 2018; 1959–1968.

[36] Bajaj C, McLennan L, Andeen T, Roy A. Robust learning of physics informed neural networks. arXiv preprint arXiv:2110.13330.

[37] Raissi M, Perdikaris P, Karniadakis GE. Numerical Gaussian processes for time-dependent and nonlinear partial differential equations. SIAM Journal on Scientific Computing. 2018; 40:A172–A198.

[38] Danisch S, Krumbiegel J. Makie.jl: Flexible high-performance data visualisation for Julia. Journal of Open Source Software. 2021; 6:3349.

[39] Urquhart M, Ljungskog E, Sebben S. Surrogate-based optimisation using adaptively scaled radial basis functions. Applied Soft Computing.2020; 88:106050.

[40] Grapel T. Solving noisy linear operator equations by Gaussian processes: Application to ordinary and partial differential equations. International Conference on Machine Learning. 2003; 3:234–241.

[41] Lagergren JH, Nardini JT, Lavigne GM, Rutter EM, Flores KB. Learning partial differential equations for biological transport models from noisy spatio-temporal data. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences. 202; 476:20190800.

[42] Revels J, Lubin M, Papamarkou T. Forward-mode automatic differentiation in Julia. arXiv preprint arXiv:1607.07892.

[43] Da Veiga S, Marrel Amandine. Gaussian process modeling with inequality constraints. Annales Faculté Sci. Toulouse. 2012; 21:529–555.

[44] Swiler L, Gulian M, Frankel A, Safta C, Jakeman J. A survey of constrained Gaussian process regression: Approaches and implementation challenges. Journal of Machine Learning for Modeling and Computing. 2020; 1:119–156.

[45] Townsend A. http://www.github.com/ajt60gaibb/FastGaussQuadrature.jl/. GitHub Repository. 2015.

[46] Bogaert I. Iteration-free computation of Gauss-Legendre quadrature nodes and weights. SIAM Journal on Scientific Computing. 2014; 36:A1008–A1026.

7 Supplementary Material

7.1 Simulation studies

In this section we demonstrate some of our methods on simulated data. We start with some PDE

$$\frac{\partial u}{\partial t} = T(t;\boldsymbol{\alpha})\left[\frac{\partial}{\partial x}\left(D(u;\boldsymbol{\beta})\frac{\partial u}{\partial x}\right) + R(u;\boldsymbol{\gamma})\right]. \tag{30}$$

We set the $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$, and $\boldsymbol{\gamma}$ vectors to be some specified values. To generate data from Equation (30) we start by fitting a Gaussian process to the data by Jin et al. [1] at $t = 0$, giving a one-dimensional Gaussian process in space. The procedure used for fitting this Gaussian process is the same as was used for the two-dimensional case discussed in the methods section. We then place N_p equally spaced points between $x = 25\,\mu\text{m}$ and $x = 1875\,\mu\text{m}$ and, using these points, obtain the mean vector $\boldsymbol{\mu}$ for the Gaussian process at these points using the `predict_f` function from the `GaussianProcesses` package [19]. This mean vector defines the initial condition, where we replace any negative values with zeros. We then solve the PDE up to $t = 48\,\text{h}$, saving the solution values at $t = 0, 12, 24, 36, 48\,\text{h}$. The PDEs are solved using the approach discussed in he methods section. We then add to each solution value, u^*_{ij}, some noise $\sigma_n z_{ij}$ where $z_{ij} \sim \mathcal{N}(0,1)$ and σ_n is the estimated standard deviation of the noise from the fitted Gaussian process. Any negative values introduced from these new values $u^*_{ij} + \sigma_n z_{ij}$ are replaced with zero. To make the data sparse we then take a small sample of these points, say 150 total data points. This procedure gives some data at the points $x_1, \ldots, x_N$ and times $t_1, \ldots, t_M$, with the point (x_i, t_j) corresponding to a density value u_{ij}; this setup exactly matches that described in the paper and is thus representative. We note that for all studies discussed below, the bootstrapping procedure is completed in units of mm and day rather than in µm and h, but we still report the values in these more interpretable latter units.

7.1.1 Study I: Fisher-Kolmogorov model

We start by considering a Fisher-Kolmogorov model, using the data from Jin et al. [1] with 10,000 cells per well for simulating the data. Our exact PDE Equation (30) takes the form

$$\frac{\partial u}{\partial t} = 310\frac{\partial^2 u}{\partial x^2} + 0.044u\left(1 - \frac{u}{K}\right), \tag{31}$$

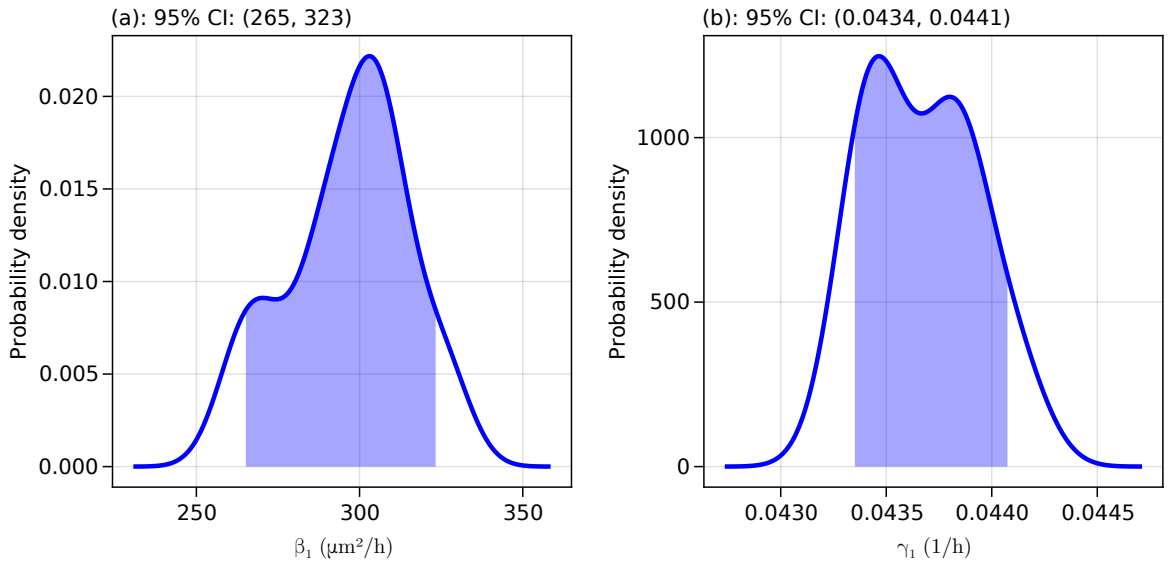
where $K = 1.7 \times 10^{-3}$ cells/µm^2. In particular, $T(t;\boldsymbol{\alpha}) \equiv 1$, $D(u;\boldsymbol{\beta}) \equiv 310$, and $R(u;\boldsymbol{\gamma}) \equiv 0.044u(1 - u/K)$ with $\boldsymbol{\alpha} = 1.0$, $\boldsymbol{\beta} = 310$, and $\boldsymbol{\gamma} = 0.044$. These values are taken from the results given by Jin et al. [1]. Since we have no delay in this problem we do not estimate $\boldsymbol{\alpha}$, and for the parameter estimation we will try and learn the functions $D(u;\boldsymbol{\beta}) = \beta_1$ and $R(u;\boldsymbol{\gamma}) = \gamma_1$. Note that $[\beta_1] = \mu\text{m}^2/\text{h}$ and $[\gamma_1] = 1/\text{h}$, where $[x]$ denotes the units of x.

To demonstrate how we rescale the parameters, we first consider 10 bootstrap iterations with four optimisation restarts for each iteration. Our bounds for generating values for β_1 and γ_1 are $[41.6667, 20833.33]$ and $[0.0250, 0.0625]$, respectively (recalling that these bounds do not refer to constraints in the optimisatio problem). Obtaining the estimates for β_1 and γ_1 and the density estimates, we obtain Figure 6. We use these densities to re-scale our problem, leading to the definitions

$$D(u;\boldsymbol{\beta}) = 323\widehat{\beta}_1 \quad \text{and} \quad R(u;\boldsymbol{\gamma}) = 0.044\widehat{\gamma}_1\left(1 - \frac{u}{K}\right), \tag{32}$$

where we have based these scales on the end point of the reported confidence intervals. The true values of $\widehat{\beta}_1$ and $\widehat{\gamma}_1$ are now 1.0333 and 1, respectively, now putting the parameters on the same scale.

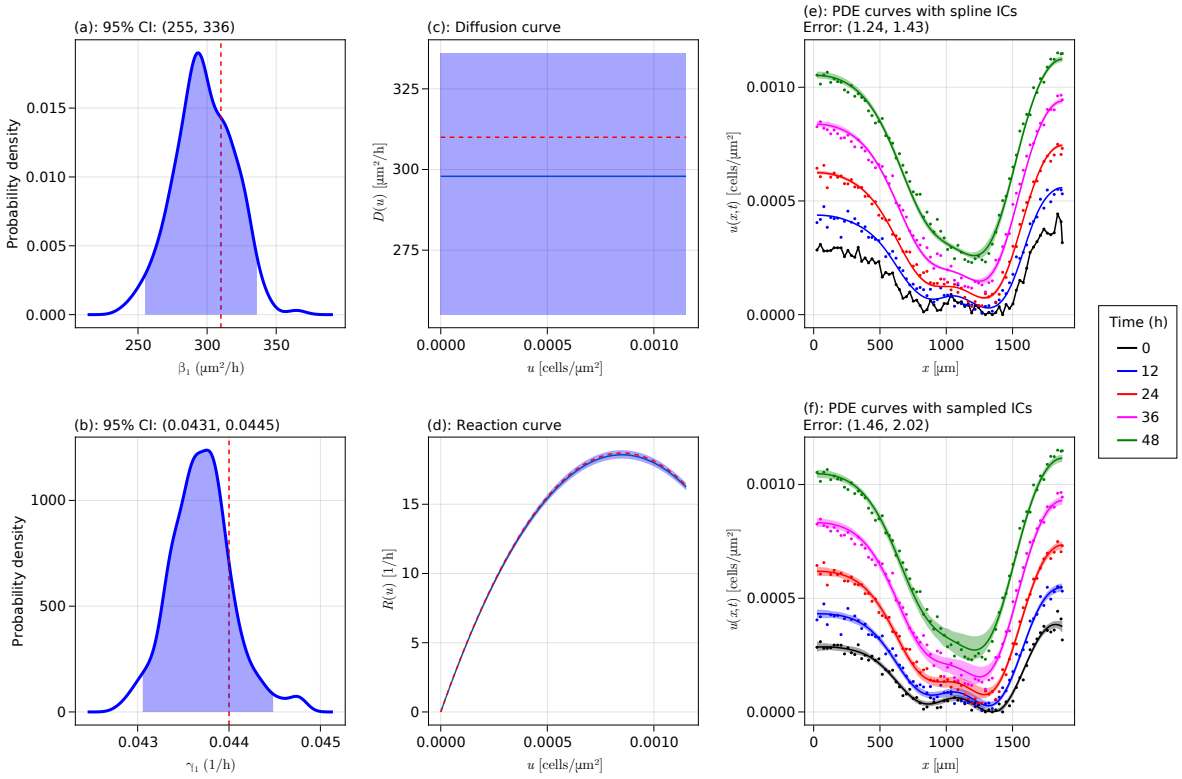
Figure 6: Initial density results for the simulated Fisher-Kolmogorov model. Initial density results for the model Equation (31) without any scaling for (a) diffusion and (b) reaction.



Now we perform the procedure on the scaled problem with 200 bootstrap iterations and three optimiser restarts. The final results are shown in Figure 7. We see that we have successfully recovered the true parameter values, with the true parameter values and functional forms lying within their respective confidence intervals. The PDE results show a close match with the data with little uncertainty, although

the model which uses a spline initial condition in (e) has too little uncertainty to capture the data points. The solutions which sample the initial conditions, shown in (f), do a much better job at capturing these points.

Figure 7: Final results for the simulated Fisher-Kolmogorov model. Final results for parameters, functional forms, and PDE solutions for the model Equation (31). In the density plots, the dashed red line is at the true parameter value, and in the curve plots the dashed red curve shows the true curve.



7.1.2 Study II: Porous-Fisher model with delay

Here we consider a Porous-Fisher model with delay, using the data from Jin et al. [1] with 20,000 cells per well for simulating the data. Our exact PDE Equation (30) takes the form

$$\frac{\partial u}{\partial t} = \frac{1}{1+\exp(4.0651-0.4166t)}\left\{\frac{\partial}{\partial x}\left[2900\left(\frac{u}{K}\right)\frac{\partial u}{\partial x}\right] + 0.0951u\left(1-\frac{u}{K}\right)\right\}, \tag{33}$$

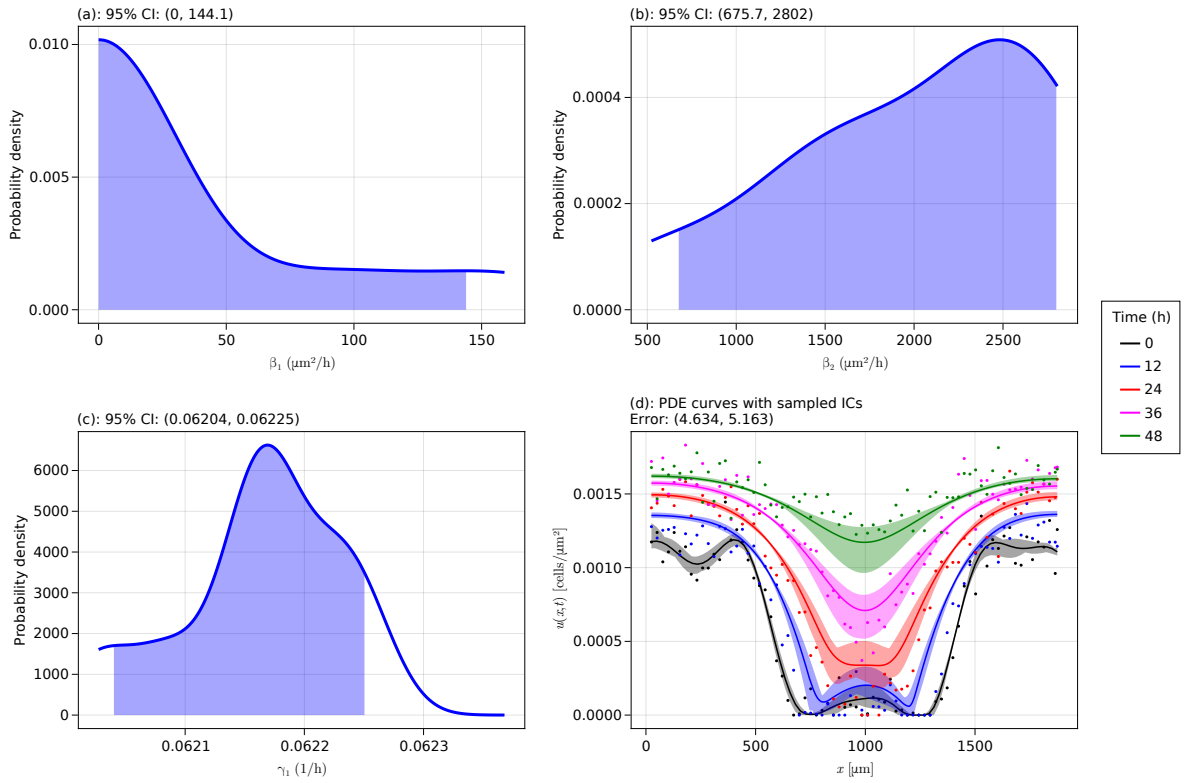
where again $K = 1.7\times10^{-3}$ cells/μm^2. In particular, $T(t;\boldsymbol{\alpha}) \equiv 1/[1+\exp(4.0651-0.4166t)]$, $D(u;\boldsymbol{\beta}) \equiv 2900(u/K)$, and $R(u;\boldsymbol{\gamma}) \equiv 0.0951u(1-u/K)$, where $\boldsymbol{\alpha} = (-4.0651, 0.4166)^\mathsf{T}$, $\boldsymbol{\beta} = 2900$, and $\boldsymbol{\gamma} = 0.0951$. These values are based on those given by Lagergren et al. [8]. For estimating the parameters, we start with assuming the functions take the form

$$T(t;\boldsymbol{\alpha}) = 1.0, \quad D(u;\boldsymbol{\beta}) = \beta_1 + \beta_2\left(\frac{u}{K}\right), \quad R(u;\boldsymbol{\gamma}) = \gamma_1 u\left(1-\frac{u}{K}\right). \tag{34}$$

Notice that we assume that there is no delay, and we add an extra β_1 term to $D(u;\boldsymbol{\beta})$. We aim to see how this misspecification of the model affects the results. Note that $[\beta_1] = [\beta_2] = \mu m^2/h$ and $[\gamma_1] = 1/h$, where a unit of 1 means the variable is unitless. We sample the initial values for the optimiser in the intervals $\beta_1 \in [41.6667, 20833.33]$, $\beta_2 \in [41.6667, 20833.33]$, and $\gamma_1 \in [0.0250, 0.1458]$, with 10 bootstrap iterations to start with and four optimiser restarts.

In Figure 8 we show the results for this initial set of iterations. We notice several problems. Firstly, the density for β_1 in (a) is concentrated at 0, correctly suggesting that this term need not be in the model. Secondly, (b) does not show a well-defined peak for β_2 so we have poor identification of this parameter, which is correct since the confidence interval does not even contain the true value of β_2. The reaction parameter γ_1's density in (c) does have a more well-defined peak, but does not contain the true value in its confidence interval. In fact, the confidence value is much closer to the value reported by Jin et al. [1] for the same model without any delay and $\beta_1 = 0$. The last problem is shown in (d): the curves are initially above the actual data points, and then for later times falls below these data points — this problem suggests the presence of delay.

Figure 8: Initial results for the simulated Porous-Fisher model. Initial density and PDE results for the model Equation (33) when misspecifying the model as Equation (34) without any scaling for (a, b) diffusion and (c) reaction.



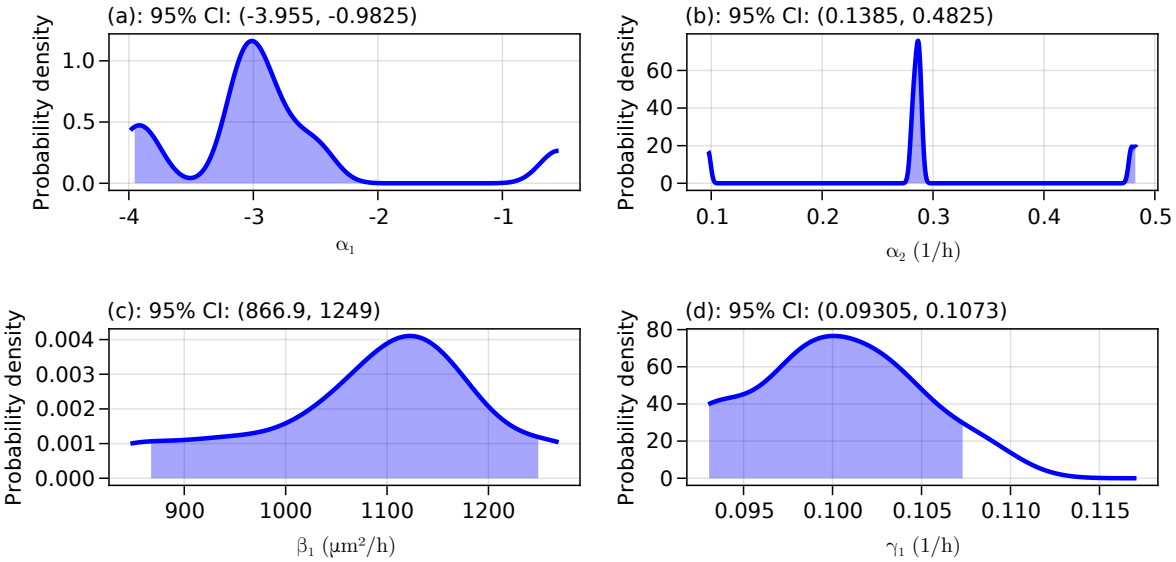
This sequence of problems naturally leads us to the correct conclusion: we can set $\beta_1 = 0$ and we

need to include delay in the model. In particular, our new model takes the form

$$T(t;\boldsymbol{\alpha}) = \frac{1}{1+\exp(-\alpha_1-\alpha_2 t)}, \quad D(u;\boldsymbol{\beta}) = \beta_1\left(\frac{u}{K}\right), \quad R(u;\boldsymbol{\gamma}) = \gamma_1 u\left(1-\frac{u}{K}\right). \tag{35}$$

We again start with 10 bootstrap iterations and four optimiser restarts, sampling $\alpha_1 \in [-10, 0]$, $\alpha_2 \in [0, 0.4167]$, $\beta_1 \in [41.6667, 20833.33]$, and $\gamma_1 \in [0.0250, 0.1458]$. The revised results are shown in Figure 9. The β_1 value is not exactly centred around the true value $\beta_1 = 2900$, but this may be a consequence of the optimiser struggling to decouple $T(t;\boldsymbol{\alpha})D(u;\boldsymbol{\beta})$ into separate functions. From these densities we select the scales -3.013, 0.286, 1122.632, and 0.1 for α_1, α_2, β_1, and γ_1, respectively; these scales are chosen according to the modes of the densities in Figure 9.

Figure 9: Revised initial results for the simulated Porous-Fisher model. Revised density results for the model Equation (33) when correctly specifying the functional forms for delay, diffusion, and reaction, and not scaling parameter estimates.



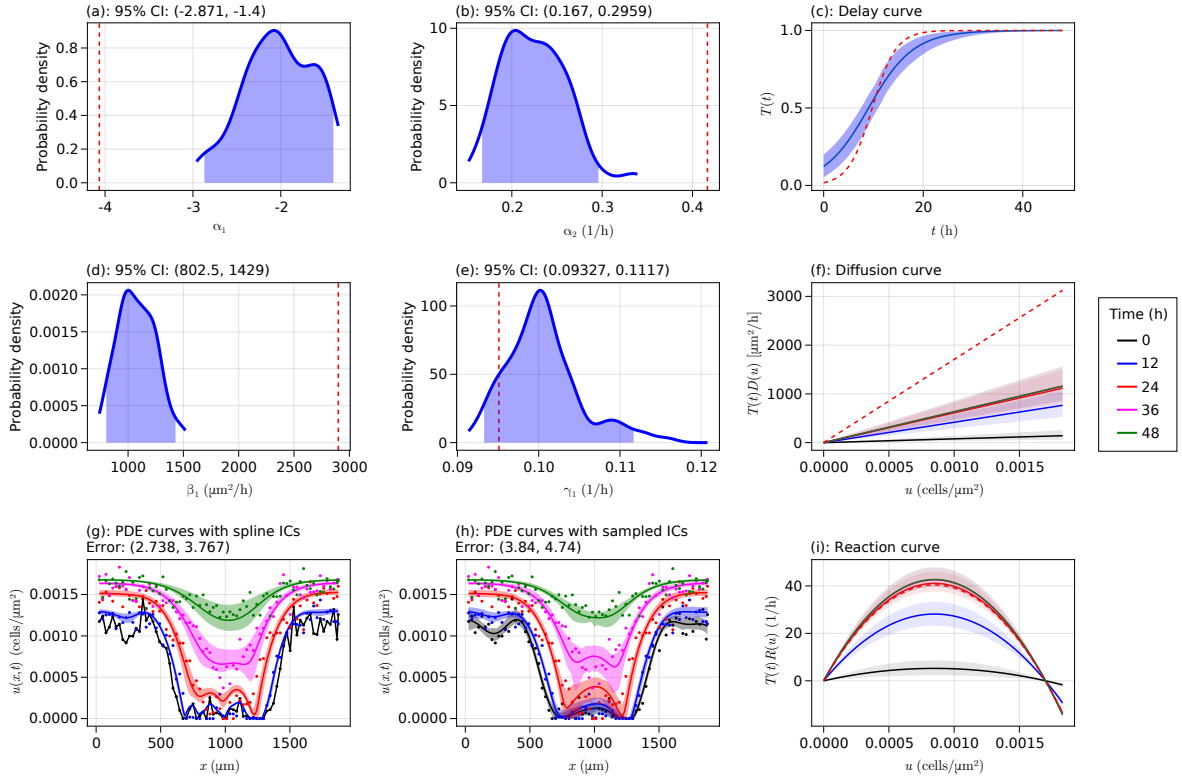
In the scaled model, we have

$$T(t;\boldsymbol{\alpha}) = \frac{1}{1+\exp(3.013\widehat{\alpha}_1 - 0.286\widehat{\alpha}_2 t)}, \quad D(u;\boldsymbol{\beta}) = 1122.632\left(\frac{u}{K}\right), \quad R(u;\boldsymbol{\gamma}) = 0.1\widehat{\gamma}_1\left(1-\frac{u}{K}\right). \tag{36}$$

We now perform the estimation procedure again, performing 100 bootstrap iterations and three optimiser restarts; we had initially selected no restarts but increased this amount after noticing that the densities were bimodal. The results are in Figure 10. We see that we have not recovered the true parameter values for the delay and diffusion coefficients, which is not surprising given that diffusion is already difficult to estimate to begin with, and we are only making it more difficult by coupling it with $T(t)$. The delay curve in (c) does still show a reasonable match to the true curve. In (f) we see that the diffusion curve is

much higher than our estimated curve. The reaction curve in (i) has been perfectly recovered. Despite these issues with delay and diffusion, we see in (g) and (h) that the learned solutions of the PDE capture the data quite well, especially when we recognise that the data has had a significant amount of noise introduced into it. Overall, these results are satisfactory, and demonstrate that, while it is difficult to decouple the product $T(t)D(u)$ to learn $T(t)$ and $D(u)$ separately, the mechanisms we learn are still reasonable models of the underlying biology.

Figure 10: Final results for the simulated Porous-Fisher model. Final results for parameters, functional forms, and PDE solutions for the model equation Equation (33). In the density plots, the dashed red line is at the true parameter value, and in the curve plots the dashed red curve shows the true curve. For these curve plots we show, for the estimated curves, the products $T(t)D(u)$ and $T(t)R(u)$ for (f) and (i), respectively, rather than only $D(u)$ and $R(u)$.



7.1.3 Study III: Quadratic diffusion model

Now we consider a model with quadratic diffusion and a logistic reaction term, using the data from Jin et al. [1] with 12,000 cells per well for simulating the data. Our exact PDE Equation (30) takes the form

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left\{\left[300 + 1700\left(\frac{u}{K}\right) + 3200\left(\frac{u}{K}\right)^2\right]\frac{\partial u}{\partial x}\right\} + 0.06u\left(1 - \frac{u}{K}\right), \tag{37}$$

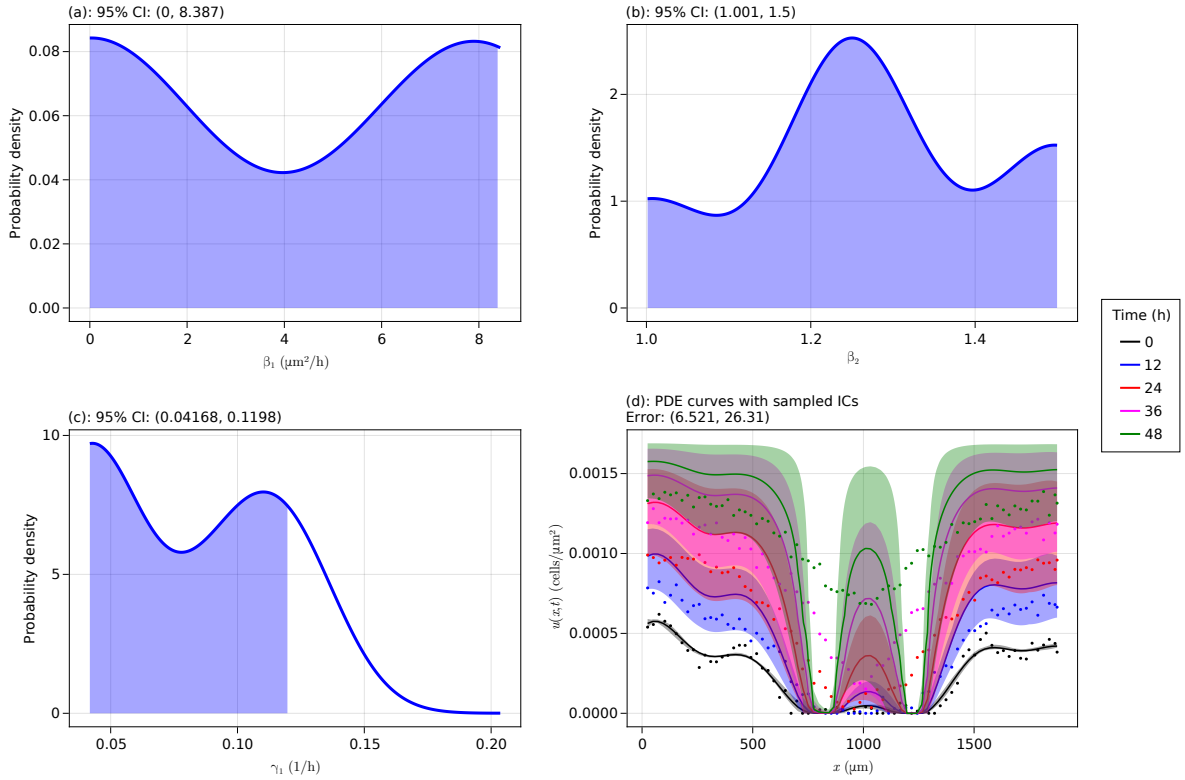
again using $K = 1.7 \times 10^{-3}$ cells/μm^2. In particular, $T(t;\boldsymbol{\alpha}) \equiv 1$, $D(u;\boldsymbol{\beta}) \equiv 300 + 1700(u/K) + 3200(u/K)^2$, and $R(u;\boldsymbol{\gamma}) \equiv 0.06u(1-u/K)$, where $\boldsymbol{\alpha} = 1$, $\boldsymbol{\beta} = (300, 1700, 3200)^\intercal$, and $\boldsymbol{\gamma} = 0.06$. These

values are based on those given by Jin et al. [1] and Lagergren et al. [8]. To demonstrate how leaving a term out can affect the results, we start by attempting to learn the functions

$$D(u;\boldsymbol{\beta}) = \beta_1 \left(\frac{u}{K}\right)^{\beta_2}, \quad R(u;\boldsymbol{\gamma}) = \gamma_1 u \left(1 - \frac{u}{K}\right). \tag{38}$$

Notice that the diffusion mechanism being assumed is that used in a Porous-Fisher model with a generalised exponent (similar to the exponent m used by Lagergren et al. [8]). Note that $[\beta_1] = \mu\text{m}^2/\text{h}$, $[\beta_2] = 1$, and $[\gamma_1] = 1/\text{h}$. We start with 10 bootstrap iterations and five optimiser restarts, sampling $\beta_1 \in [41.667, 3333.33]$, $\beta_2 \in [1, 2]$, and $\gamma_1 \in [0.041667, 0.145833]$.

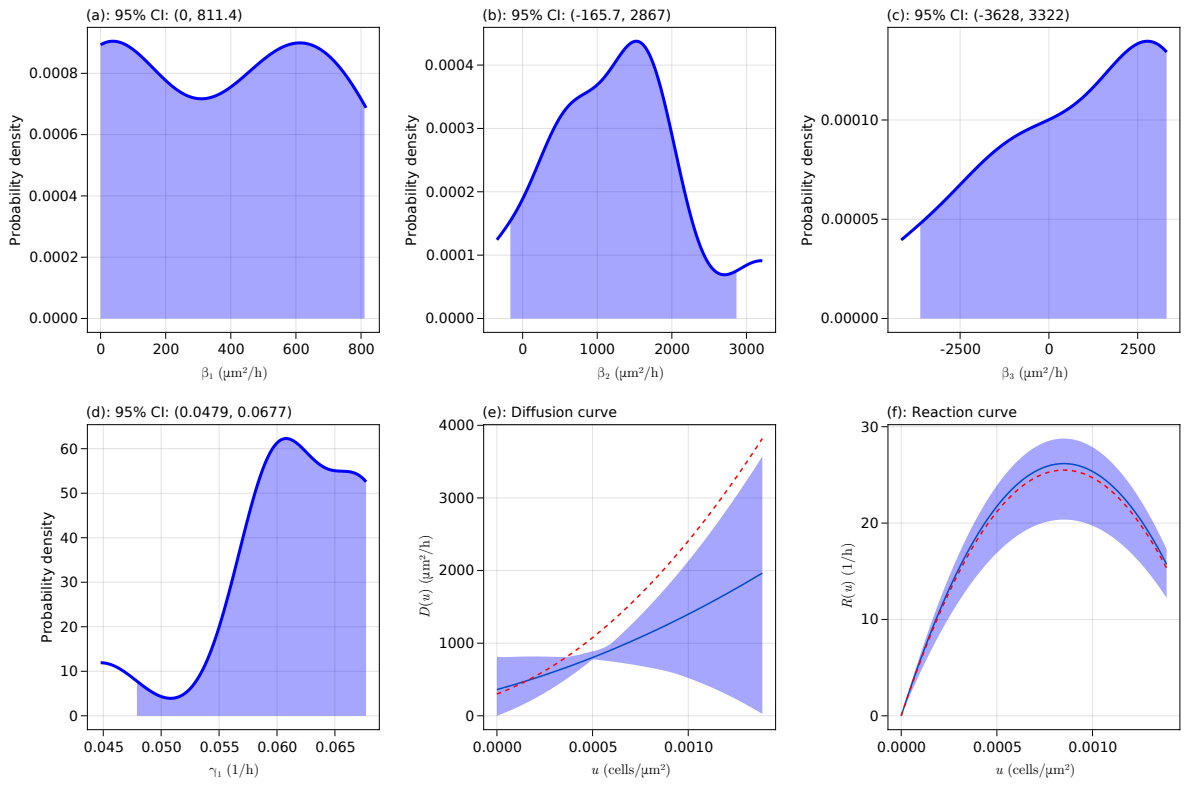
Figure 11: Initial results for the simulated quadratic diffusion model. Initial density and PDE results for the model Equation (37) when misspecifying the model as Equation (38) without any scaling for (a, b) diffusion and (c) reaction.



The results for the model Equation (38) are shown in Figure 11. We see that there is no clear mode for β_1, showing that the optimiser has failed to identify a suitable value. The exponent β_2 does have a mode around $\beta_2 = 1.3$ which is reasonably well-defined, but there are some other peaks at the ends. In (d) we see that the solutions are clearly not a good fit, with the curves failing to match the evolution of the density in the centre of the domain. These issues in (d) suggest a clear issue with the diffusion model, although the endpoints are captured well. These issues thus suggest the following modifications to $D(u;\boldsymbol{\beta})$: Since we can capture the exponents well, this form of model may be close to the correct

model, so we keep terms in the form u/K; since the exponent estimate seems to be trying to reach different exponents, we try a polynomial solution. Thus, we try $D(u;\boldsymbol{\beta}) = \beta_1 + \beta_2(u/K) + \beta_3(u/K)^2$, which we know is the correct model but is the natural evolution for our model choice based on the issues in Figure 11. We sample each $\beta_j \in [0.001, 0.08]$ for $j = 1, 2, 3$ and $\gamma_1 \in [0.041667, 0.145833]$, and we again do 10 bootstrap iterations with five optimiser restarts.

Figure 12: Revised initial results for the simulated quadratic diffusion model. Revised density and curve results for the model Equation (37) when correctly specifying the functional forms for diffusion, and not scaling parameter estimates.

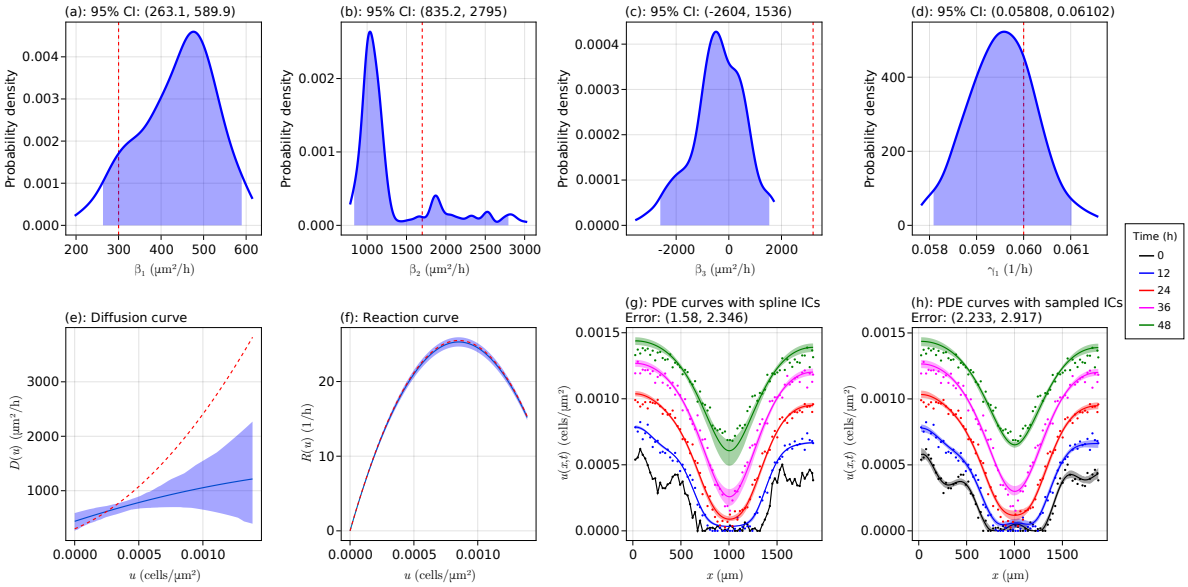


The new results are shown in Figure 12. We see that β_1 has three extrema, with one around $\beta_1 = 0$. The density for β_2 in (b) does have a mode around 1500, and β_3 in (c) has a mode around 2779. For β_1 it is not clear what we should take, so we instead average between the two modes which gives an approximate value of 300, which does happen to be the true parameter value. For γ_1 in (d) we take the mode around 0.060, which is again the true value. For the learned functional forms in (e) and (f), we see that we can recover the reaction curve, though with more uncertainty than in the previous models, but the diffusion curve is more difficult to recover. This result in (e) is not surprising given our previous discussion about issues with estimating diffusion, and because our parameter values are on very different scales. To now improve these estimates we take the new models

$$D(u;\boldsymbol{\beta}) = 300\widehat{\beta}_1 + 1500\widehat{\beta}_2\left(\frac{u}{K}\right) + 2779\widehat{\beta}_3\left(\frac{u}{K}\right) \quad \text{and} \quad R(u;\boldsymbol{\gamma}) = 0.06\widehat{\gamma}_1 u\left(1 - \frac{u}{K}\right). \tag{39}$$

We perform 200 bootstrap iterations with no optimiser restarts for each iteration.

Figure 13: Final results for the quadratic diffusion model. Final results for parameters, functional forms, and PDE solutions for the model equation Equation (37). In the density plots, the dashed red line is at the true parameter value, and in the curve plots the dashed red curve shows the true curve.



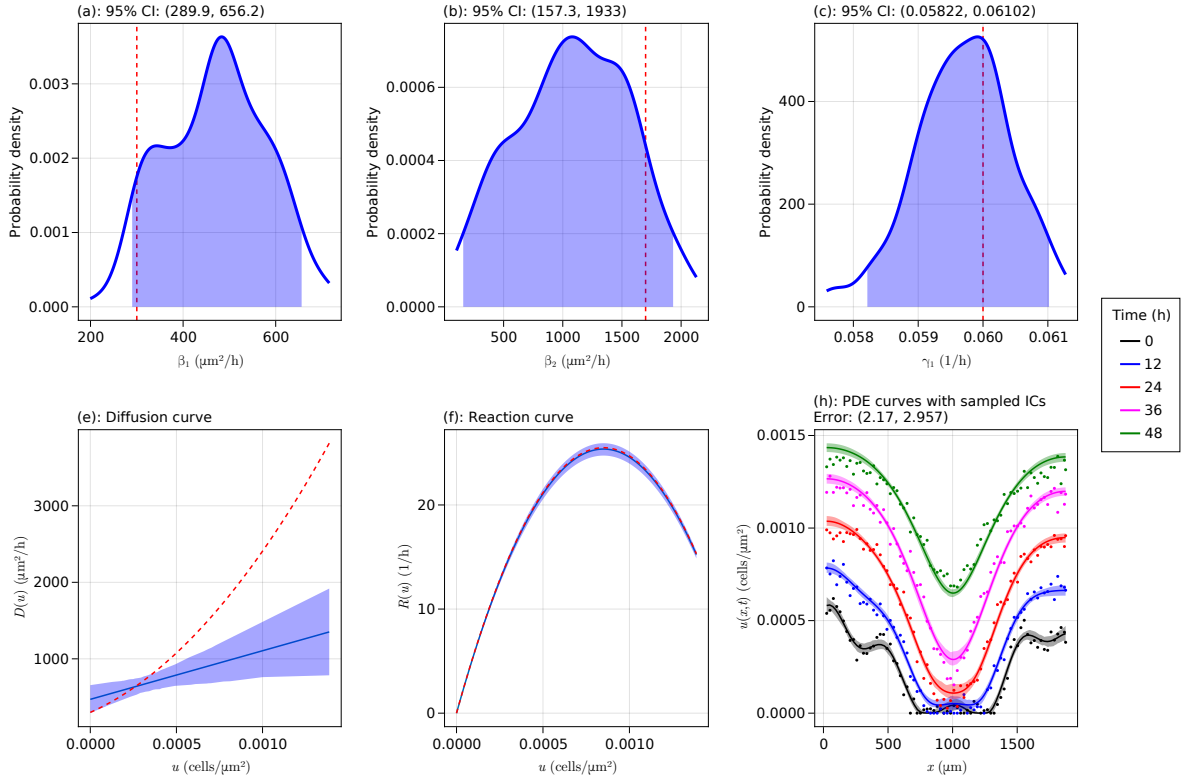
The results are shown in Figure 13. We see that we have recovered all parameters except β_3, and this failure to recover the quadratic term is clear in (e) which shows that the curve is more like an affine diffusion function. Despite learning the wrong diffusion curve, we see in (g) and (h) that the final model is still reasonable, suggesting that linear diffusion is good enough. Given that the confidence interval for β_3 in (c) contains 0 so that we fail to reject the null hypothesis $H_0 \colon \beta_3 = 0$, and since the curve in (e) is essentially linear, let us try fitting a model with β_3. The new results for this affine diffusion model are shown in Figure 14. See that we have decreased the mean error slightly for the solutions in (h), and we have successfully recovered the remaining parameter estimates.

7.2 Basis function approach for linear problems

In this section we present an alternative to the nonlinear optimisation problem in Equation (20) for problems without delay and whose diffusion and reaction processes can be represented as linear combinations of individual basis functions,

$$D(u;\boldsymbol{\beta}) = \sum_{k=1}^{d} \beta_k \phi_k(u) \quad \text{and} \quad R(u;\boldsymbol{\gamma}) = \sum_{k=1}^{r} \gamma_k \psi_k(u), \tag{40}$$

Figure 14: Final results for the affine diffusion model. Final results for parameters, functional forms, and PDE solutions for the model equation Equation (37) with the quadratic term removed. In the density plots, the dashed red line is at the true parameter value, and in the curve plots the dashed red curve shows the true curve.



where the basis functions ϕ_k and ψ_k, for each $k = 1, \ldots, d$ and $k = 1, \ldots, r$, respectively, are differentiable. With these functions Equation (17) becomes

$$\frac{\partial u(x_i^\dagger, t_j^\dagger)}{\partial t} = \sum_{k=1}^{d} \left[\phi_k' \left(u(x_i^\dagger, t_j^\dagger) \right) \left(\frac{\partial u(x_i^\dagger, t_j^\dagger)}{\partial x} \right)^2 + \phi_k \left(u(x_i^\dagger, t_j^\dagger) \right) \frac{\partial^2 u(x_i^\dagger, t_j^\dagger)}{\partial x^2} \right] \beta_k + \sum_{k=1}^{r} \gamma_k \psi_k \left(u(x_i^\dagger, t_j^\dagger) \right), \tag{41}$$

where primes denote derivatives in u. Denoting $u(x_i^\dagger, t_j^\dagger)$ by u_{ij}, we then define the matrices $\mathbf{A}_1 \in \mathbb{R}^{n_x n_t \times d}$ and $\mathbf{A}_2 \in \mathbb{R}^{n_x n_t \times r}$ such that

$$[\mathbf{A}_1]_{(i,j),k} = \phi_k(u_{ij}) \frac{\partial^2 u_{ij}}{\partial x^2} + \phi_k'(u_{ij}) \left(\frac{\partial u_{ij}}{\partial x} \right)^2 \quad \text{and} \quad [\mathbf{A}_2]_{(i,j),k} = \psi_k(u_{ij}), \tag{42}$$

respectively, where $[\mathbf{A}_\ell]_{(i,j),k}$ denotes the entry in $\mathbf{A}_\ell$ in the kth column in the row corresponding to the point $(x_i^\dagger, t_j^\dagger)$ for $i = 1, \ldots, n_x$, $j = 1, \ldots, n_t$, and $\ell = 1, 2$. Next, define $\mathbf{A} = [\mathbf{A}_1 \ \mathbf{A}_2] \in \mathbb{R}^{n_x n_t \times (d+r)}$ and $\boldsymbol{\theta} = (\boldsymbol{\beta}^\mathsf{T}, \boldsymbol{\gamma}^\mathsf{T})^\mathsf{T}$. Given this form we obtain the matrix system

$$\mathbf{A}\boldsymbol{\theta} = \frac{\partial \mathbf{u}_*}{\partial x}. \tag{43}$$

Equation (43) can then be solved for $\boldsymbol{\theta}$ using the backslash operator in Julia [18]. This approach could be especially useful for obtaining initial scale estimates for parameters, for example, rather than dealing with the much slower nonlinear optimisation method on an unscaled problem to start with.

We note that while this approach could be suitable for say a Porous-Fisher PDE of the form

$$\frac{\partial u}{\partial t} = \beta_1 \frac{\partial}{\partial x}\left[\left(\frac{u}{K}\right)\frac{\partial u}{\partial x}\right] + \gamma_1 u\left(1 - \frac{u}{K}\right), \tag{44}$$

it would not be suitable if there were any nonlinear delay or if we had for example the PDE

$$\frac{\partial u}{\partial t} = \frac{\partial}{\partial x}\left\{\left[\beta_1 \mathrm{e}^{-\beta_2 u}\left(\frac{u}{K}\right)\right]\frac{\partial u}{\partial x}\right\} + \gamma_1 u\left(1 - \frac{u}{K}\right) \tag{45}$$

since the diffusion term cannot be represented as a linear combination of basis functions. We could still approximate this problem such that the functions are linear in the parameters, for example by temporarily fixing β_2 at a value, say $\beta_2 = 1$, and using the above approach to estimate β_1. We could then fix β_1 at this found value and estimate $\mathrm{e}^{-\beta_2 u} \approx 1 - \beta_2 u$ and estimate β_2. This process of fitting β_1 and then β_2 separately could be repeated, similarly to a standard predictor-corrector method, if desired. These approximations would give rough scales of the estimates that could be used to rescale the parameters and make the nonlinear optimisation problem in Equation (20) significantly faster.

Let us compare the results from this approach to those given in our quadratic diffusion model in Section 7.1.3, writing

$$D(u;\boldsymbol{\beta}) = \beta_1\phi_1(u) + \beta_2\phi_2(u) + \beta_3\phi_3(u) \quad \text{and} \quad R(u;\boldsymbol{\beta}) = \gamma_1\psi_1(u), \tag{46}$$

where $\phi_1(u) \equiv 1$, $\phi_2(u) \equiv u/K$, $\phi_3(u) \equiv (u/K)^2$, and $\psi_1(u) \equiv u(1 - u/K)$. The final results are shown in Figure 15 with 200 bootstrap iterations. The results are very similar to those in Figure 13, and again we fail to recover the quadratic model and instead find an affine diffusion function. One feature we do observe is that in (b) the density is now unimodal, and that in (g) and (h) the errors are large. The fact that the error between the data and the PDE solutions makes sense, if we recall that the solution

to Equation (43) is equivalent to solving the least squares problem

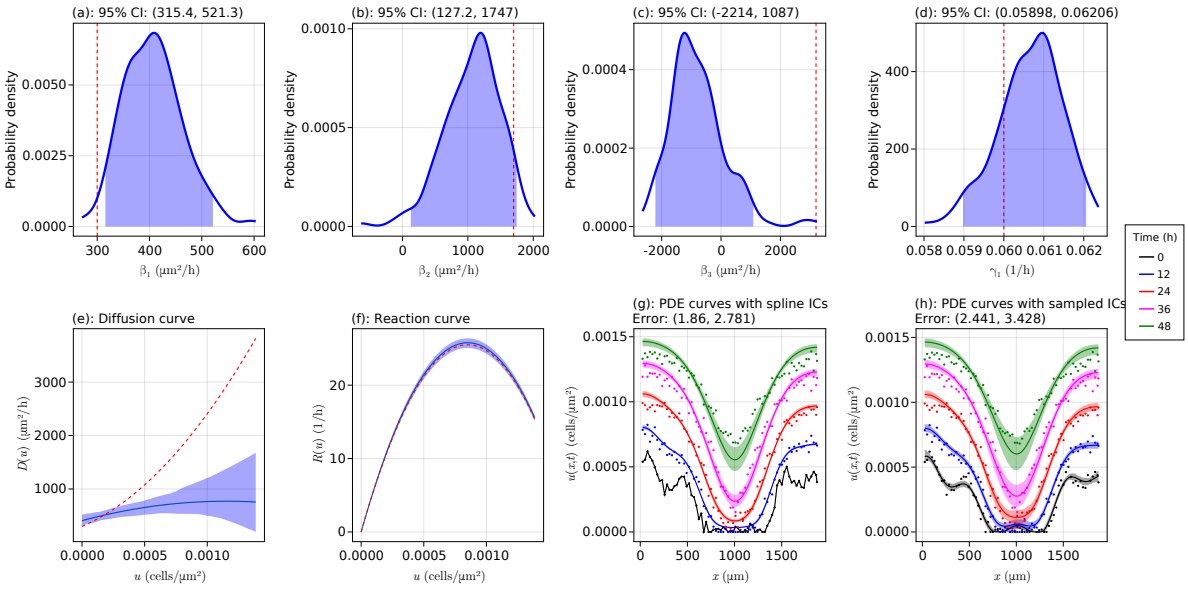
$$\widehat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}\in\mathbb{R}^{d+r}}{\operatorname{argmin}}\, \mathcal{L}(\boldsymbol{\theta}), \tag{47}$$

$$\mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^{n_x}\sum_{j=1}^{n_t}\left\{\frac{\partial u(x_i^\dagger, t_j^\dagger)}{\partial t} - \left(\sum_{k=1}^{d}\left[\phi_k'\left(u(x_i^\dagger, t_j^\dagger)\right)\left(\frac{\partial u(x_i^\dagger, t_j^\dagger)}{\partial x}\right)^2 + \phi_k\left(u(x_i^\dagger, t_j^\dagger)\right)\frac{\partial^2 u(x_i^\dagger, t_j^\dagger)}{\partial x^2}\right]\beta_k\right. \tag{48}$$

$$\left. + \sum_{k=1}^{r}\gamma_k\psi_k\left(u(x_i^\dagger, t_j^\dagger)\right)\right)\right\}^2, \tag{49}$$

where the summand is the difference between the terms on both sides of Equation (41). See that this optimisation problem is the same as minimising only $\mathcal{L}_{\text{PDE}}$ in Equation (18) and not considering $\mathcal{L}_{\text{GLS}}$ in Equation (19). Therefore, the parameter estimation is focusing only on our sampled values without any direct consideration for the actual data as would be accomplished through $\mathcal{L}_{\text{GLS}}$. This failure to capture the data during the bootstrapping procedure using the basis function approach demonstrates one downside. Another downside can be seen when considering (h) in Figure 15: there is too much uncertainty in the centre of the solution curves, and the curves start too far away from the data at the endpoints, in comparison to the results in Figure 13(h). This issue again stems from the fact that we are not considering the data through $\mathcal{L}_{\text{GLS}}$.

Figure 15: Results for the quadratic diffusion model using the basis function approach. Results for parameters, functional forms, and PDE solutions for the model equation Equation (37) using the basis function approach. In the density plots, the dashed red line is at the true parameter value, and in the curve plots the dashed red curve shows the true curve.



The above discussion allows us to make the following conclusions around this method. The approach is extremely fast relative to the main approach which requires solving Equation (20), for example taking only

two minutes to do the 200 bootstrap iterations required in Equation (15) compared to the approximate 30 minutes for the results in Figure 13, which we note does not include the time required for finding the appropriate parameter scales. Nevertheless, the approach can lead to greater error in the PDE solutions due to not directly considering the data when estimating the parameters. We therefore recommend this method as an efficient way to obtain scales for the parameters when the problem is linear, or for example as a quick way to see if delay may be needed in the problem (for example by inspecting the PDE solutions as we did in Figure 8(d) to see if the curves show delay). Once these scales are found, the coefficients should be rescaled and the nonlinear optimisation problem in Equation (20) should be used to give the final results. Future work could for example augment Equation (49) with the loss function $\mathcal{L}_{\text{PDE}}$ or some other term that regularises the loss function in a way that considers the data in a more direct manner, although this may not lead to a simple matrix system as in Equation (43).

7.3 Data thresholding

In this section we discuss a method for improving the quality of data used for estimating the parameters $\boldsymbol{\theta}$ by removing data points that violate certain conditions. We suppose we have some data $\mathbf{u}_*$ and corresponding derivatives $\partial\mathbf{u}_*/\partial t$, $\partial\mathbf{u}_*/\partial x$, and $\partial^2\mathbf{u}_*/\partial x^2$. We then define some threshold parameters $0 \leq \tau_1, \tau_2 < 1$, where $\tau = 0$ corresponds to no thresholding and $\tau = 1$ means all points are removed. To motivate what conditions we should use to discard data, let us consider the space-time diagrams in Figure 2. We see that there are some regions where there are very little cells so that the density is almost zero, and there are some regions where the density evolves very slowly in time. These values would pollute the large sum in Equation (20), and so it would be useful to remove them as they do not contribute any valuable information. Thus, we define the following three conditions that must be satisfied for a data point to be used, where $m = \min_{i,j} |\widehat{u}_{ij}|$, $M = \max_{i,j} |\widehat{u}_{ij}|$, $m' = \min_{i,j} |\partial_t \widehat{u}_{ij}|$, and $M' = \max_{i,j} |\partial_t \widehat{u}_{ij}|$, where $i = 1, \ldots, n_x$ and $j = 1, \ldots, n_t$:

1. $m\tau_1 \leq |\widehat{u}_{ij}| \leq M(1 - \tau_1)$;
2. $m'\tau_2 \leq |\partial_t \widehat{u}_{ij}| \leq M'(1 - \tau_2)$;
3. $\widehat{u}_{ij} \geq 0$.

We note that this last condition is needed since some sampled values may go below zero. With the first two conditions we can control for low density regions and for cells that are approaching the carrying capacity, as discussed above. Those data points that do not satisfy all of these conditions are not included

when computing Equation (18).

To now explore these threshold conditions, we explore its impact on the models in Equation (31) and Equation (33). We will use 50 bootstrap iterations for each pair (τ_1, τ_2) and no optimiser restarts, and we will scale the parameters such that the true parameter values are 1.0. The errors that we report from these experiments are based on the PDE solutions with the sampled initial conditions, and we only report the mean error.