Estimating Parameters of Partial Differential Equations with Gradient Matching

Zhongyi Liu

[redacted]

Supervisors:

Benn Macdonald

Dirk Husmeier

Diana Giurghita

Contents

Abstract

Parameter inference in partial differential equations (PDEs) is a problem that many researchers are interested in. The conventional methods suffer from severe computational costs because these method require to solve the PDEs repeatedly by numerical integration. The concept of gradient matching have been proposed in order to reduce the computational complexity, which consists of two steps. First, the data are interpolated with certain smoothing methods. Then, the partial derivatives of the interpolants are calculated and the parameters are optimized to minimize the distance (measured by loss functions) between partial derivatives of interpolants and the PDE systems. In this article, we first studied the parameter inference accuracy of gradient matching based on two simple PDE models. Then the method of gradient matching was used to infer the parameters of PDE models describing cell movement and select the most appropriate model.

Keywords: partial differential equations, gradient matching, Gaussian processes, cell movement, parameter inference

1. Introduction

Partial differential equations (PDEs) are equations that involve multi-variable functions and their partial derivatives. For the function $u(x_1,...,x_n)$, a PDE is an equation of the form

$$f\left(x_1,...,x_n,u,\frac{\partial u}{\partial x_1},...,\frac{\partial u}{\partial x_n},\frac{\partial^2 u}{\partial x_1 \partial x_1},...;\boldsymbol{\theta}\right)=0\ ,\ (1,1)$$

where $\boldsymbol{\theta}$ is the vector of parameters of the PDE.

A wide variety of phenomena, including heat, sound, fluid dynamics or collective movements in cell systems can be described via PDE models. Like other statistical models, the performance of PDE models depend on their parameters. For complex models, most of the parameters cannot be directly measured. Hence, many studies have been focused on inferring the parameters of different partial differential equation (PDE) or ordinary differential equation (ODE) systems using different methods.

In principle, the ordinary techniques of statistical inference can also be used to solve this problem. At first, we can calculate the difference between the real data values and the fitted values calculated from the models and at the parameter levels we choose to get the likelihood function of the data. For a particular parameter set, the PDEs are solved and the solutions are compared to the data. Then, the parameters can be optimized to maximize the likelihood, or be sampled from a corresponding posterior distribution.

Although inference in this way is possible, in practice it suffers from the problem of large computational costs. Complex PDE models usually do not have closed form solutions, so to do the inference, the PDEs must be solved repeatedly using numerical methods, which results in a large computational burden. Besides, the likelihood functions of the parameters are usually not unimodal, but with multiple local optima, so the optimization is also a challenging task, and for Bayesian inference, the sampling procedure would be difficult to converge [10].

To reduce the computational complexity, the method of gradient matching has been

adopted by different authors [6][9-11], mainly on ODE models. The same method can be applied on PDE models as well. The idea of gradient matching mainly consists of the following two steps. First there is a smoothing step, where the data (usually noisy) are interpolated using a certain smoothing method. In a second step, the partial derivatives of the interpolants are calculated, and the parameters of the PDEs are optimized to minimize a certain error metric describing the partial derivatives calculated from the interpolants and the partial derivatives from the PDEs. In this way, we do not need to numerically solve the PDEs to get fitted function values. Instead, we can directly compare the partial derivatives to the PDE models to get estimates of parameters. In this way, the computational costs can be greatly reduced. However, the accuracy of the parameter inference requires further investigation.

In the first part of this study, two PDE models with existing closed form solutions are investigated. Simulated data are produced from the closed form solutions with i.i.d. Gaussian noise being added to simulate to real cases that we cannot avoid all the noises. The data are then smoothed using Gaussian process regression. The derivatives of the interpolants are calculated and the parameters are inferred with the method of gradient matching. The results show that with properly selected covariance functions and hyper-parameters, the parameter estimation using noise-free simulated data can give results which are very close to the true values of parameters we choose in advance. When the noise level increases, the performance becomes poorer, but the medians and IQRs of the point estimates are still within 30% of the true values, showing that the gradient matching method provides a decent approximation to the conventional procedure of parameter estimation of PDE models.

We then use gradient matching method to infer the parameters of several PDE models describing directional cell movements. After estimating the parameters, the likelihood of and model selection criteria are calculated to find the most appropriate model. Among all the three candidate models, the model which describes the underlying concentration of certain attractant to cells as a modified sigmoidal function gives the best explanation of the data.

At the end of the paper, further discussions are made in regard to related results from other studies, possible future works and other concerning issues.

2. Methodology

2.1 Example PDE models

Before moving on to the models which describe the cell movement, it is preferable to give a benchmark on the gradient matching method with some simpler models to see how the method performs. It would be good if the test models we choose have their closed form solutions so that we can generate simulated data from them multiple times with little effort. Guided by this idea, we choose the diffusion equation and the advection-diffusion equation as our start point of the project. Besides, the cell movement models can be regarded as generalisations of these two models so understanding the nature of these two models is critical to our following analyses.

2.1.1 The diffusion model

In physics, the diffusion equation can be used to describe the flux of a substance at a system where there is no mean flow and only the diffusion caused by different concentration across the system. To illustrate, we can imagine dripping a drop of ink into still water, and diffusion equation describes the movement of the color molecules. The equation can be written as:

$$\frac{\partial c(x,t)}{\partial t} = D\frac{\partial^2 c(x,t)}{\partial x^2} \quad , (2.1)$$

where $c(x,t)$ is the concentration of the diffusing substance at location x and time t and $D>0$ is the diffusion coefficient. More generally, D could be a function of x and t and the location x could be multi-dimensional. To reduce the complexity and to get a closed form solution, we only consider this simplest case. From Atkins (1987), the closed form solution of (2.1) is:

$$c(x,t) = \frac{M}{\sqrt{4\pi Dt}} \exp\left(-\frac{x^2}{4Dt} \right) \quad . (2.2)$$

2.1.2 The advection-diffusion equation

The advection-diffusion equation models the flux of a substance at a system where the fluid is moving in a certain direction in addition to the diffusion. Again, we can imagine dripping ink into water, but this time with the water flowing to a certain direction with velocity u .The one-dimensional advection-diffusion equation is written as

$$\frac{\partial c(x,t)}{\partial t} + u\frac{\partial c(x,t)}{\partial x} = D\frac{\partial^2 c(x,t)}{\partial x^2} \quad , (2.3)$$

where the three terms representing the local concentration change with respect to time, the mean flow of the substance, and the effect of diffusion.

From Atkins (1987), the closed form solution of equation (2.3) is:

$$c(x,t) = \frac{M}{\sqrt{4\pi Dt}} \exp\left(-\frac{(x-ut)^2}{4Dt} \right) \quad . (2.4)$$

2.2 PDE models of cell movement

From Ferguson et al. (2016), models used to measure cell movement in this study are all generalisations of one-dimensional advection-diffusion-reaction PDEs of the form:

$$\frac{\partial C(x,t)}{\partial t} = -\frac{\partial}{\partial x}\{a(x,t)C(x,t)\} + \frac{\partial}{\partial x}\{D_C(t)\frac{\partial C(x,t)}{\partial x}\} + \nu C(x,t) \quad . (2.5)$$

where t is time, x is space and $C(x,t)$ is the cell density. The advection coefficient $a(x,t)$ describes the velocity of the directional movement towards higher or lower x, indicated by positive or negative coefficients, respectively. The diffusion coefficient $D_C(t) \geq 0$ describes the rate of the random movement of cells from high-density area to low-density area, and the reaction term $\nu C(x,t)$ describes the exponential growth process of cells at rate $\nu \geq 0$.

In this study, we focused on cell movement data observed in a small time and space region. As a result, the increase in cell population caused by cell division is non-existent, so ν is set to be zero.

Different advection coefficients were investigated in this study. Each advection coefficient represents a different hypothesis for the mechanism of cell movement. Our *constant advection model* assumes that the driver of the directional cell movement is the same across space and time, i.e.:

$$a(x,t) = a \quad , (2.6)$$

Another model we are interested in is the *sigmoidal attractant model*, where the advection coefficient is of the form:

$$a(x,t) = a(t)\frac{\partial A(x,t)}{\partial x} \quad , (2.7)$$

where $A(x,t)$ is the concentration of a chemical attractant and is of the form

$$A(x,t) = \frac{\alpha}{1+exp(-\beta(x-\gamma t))} \quad . (2.8)$$

Its spatial gradient is of the form

$$\frac{\partial A(x,t)}{\partial x} = \frac{\alpha\beta \exp(-\beta(x-\gamma t))}{\{1+\exp(-\beta(x-\gamma t))\}^2} \quad , (2.9)$$

where the parameters α , β and γ can be interpreted as the amplitude, steepness and velocity of the resource concentration, respectively. An illustration of the meaning of the parameters is shown in Figure 1.

The underlying assumption of the sigmoidal attractant model is that initially, the chemical that attracts cells is uniformly distributed across the space. Afterwards, the cells consume the attractant at the location they occupy. Thus, the concentration of the attractant decreases where the cell concentration is high. The effect is named local depletion. After the local attractants are consumed, cells then begin to move

directionally to locations where there were no or fewer cells before, and that is the motivation of directed cell movement. As a result, the concentration of attractant at locations where the cells newly arrived will begin to fall as well. Meanwhile, the attractant concentration behind the front cells gets very low due to local depletion causing the cells remaining at those locations to display random movement. Figure 2 gives an intuitive illustration of this process.

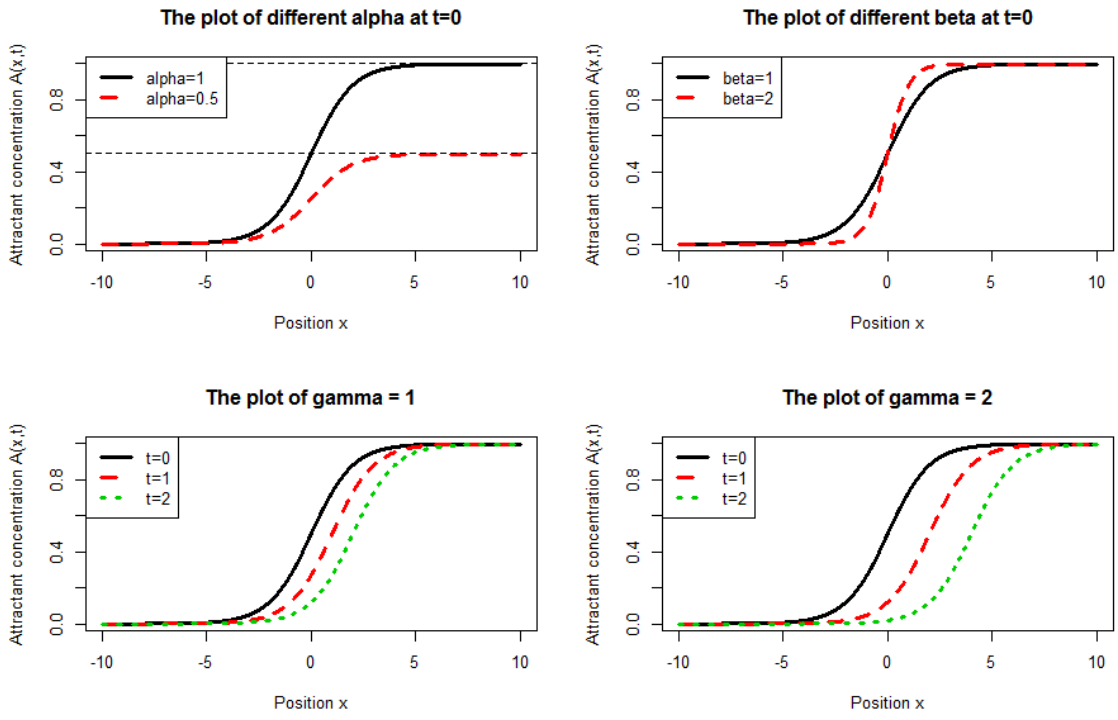


Figure 1. The illustration of the meanings of parameters in the sigmoidal attractant model. Parameter α represents the amplitude, which controls the maximum of the attractant concentration. Parameter β represents the steepness, which controls how rapidly the attractant concentration changes w.r.t. changes of position. And parameter γ represents the velocity, which measures how fast the S-shaped structure of the function moves as time goes.

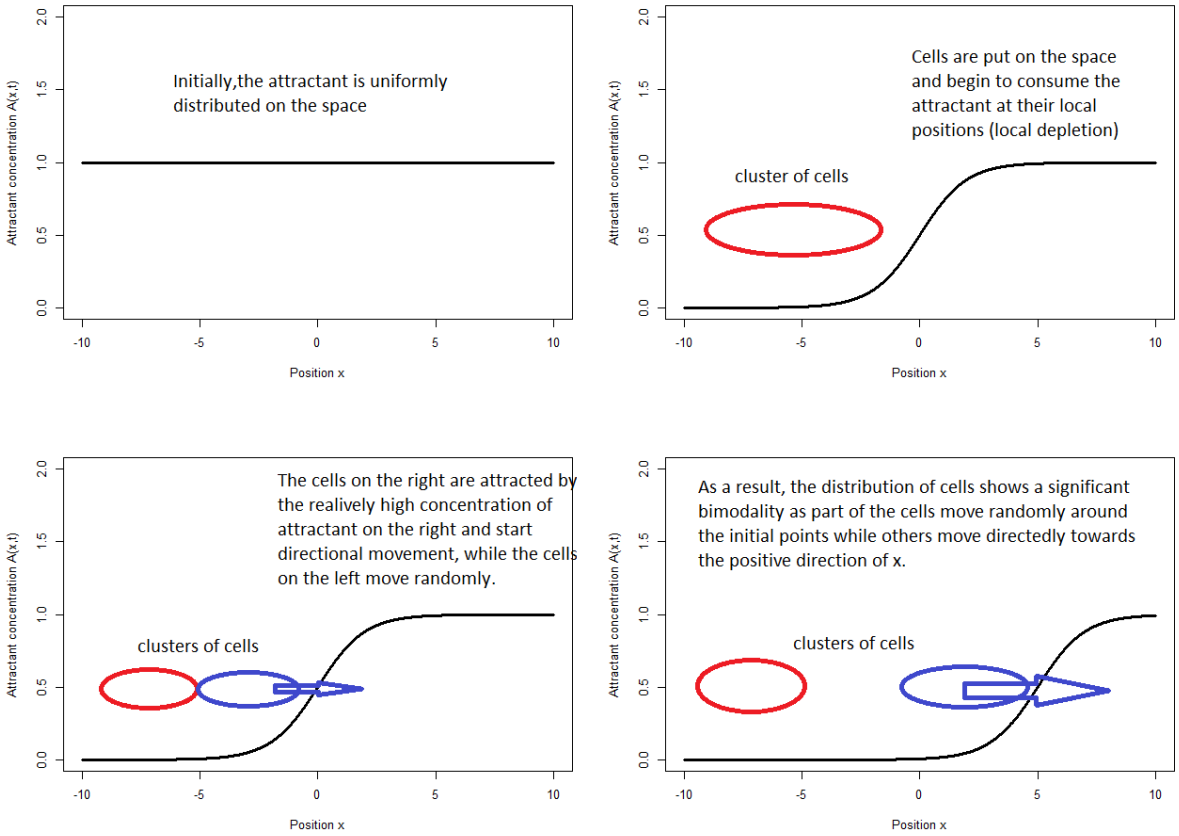


Figure 2. An intuitive illustration of the sigmoidal attractant model, where the y-axis is the concentration of the attractant, and the x-axis is the space. The coloured circles represent the clusters of cells. At first, the distribution of cells was unimodal. Then the cells close to the high attractant concentration area (represented by blue circle) began their directional movement towards the high concentration area. After reaching these area, they consumed the attractant in their new position and continued to move directionally. Meanwhile, other cells (represented by red circle) stayed at their initial positions and moved randomly.

2.3 Gaussian process

2.3.1 Gaussian process regression

In this study, Gaussian process regression is used to interpolant the data. We assume that the noisy observation $y(x,t) = f(x,t) + \varepsilon(x,t)$ consists of an underlying Gaussian process $f(x,t)$ and i.i.d. Gaussian noise $\varepsilon(x,t) \sim N(0,\sigma^2)$, where x represents space and t represents time. A Gaussian process is a collection of random variables, any finite

number of which have a joint Gaussian distribution. From Rasmussen (2006), a Gaussian process is specified by its mean function $m(x)$ and covariance function $k(x,x')$, written as:

$$f(x,t) \sim GP(m(x,t), k((x,t),(x',t'))) \quad . (2.10)$$

For simplicity, the mean function is usually set to be zero. In general, the covariance functions represent some kind of distance or similarity between the two points (x,t) and (x',t'). There are different choices of covariance functions, one of which is the squared exponential kernel:

$$k((x_1,t_1),(x_2,t_2)) = \sigma_f^2 \exp\left(-\frac{(x_1-x_2)^2}{2l_1} - \frac{(t_1-t_2)^2}{2l_2}\right). (2.11)$$

As equation (2.10) shows, the covariance function is controlled by its hyper-parameters, represented by vector $\boldsymbol{\phi}$ .For the squared exponential kernel, $\boldsymbol{\phi} = [\sigma_f, l_1, l_2]^T$. After observing the data, we can estimate the hyper-parameters of the kernel using maximum-likelihood estimation with gradient-based optimization algorithms. If we use $\boldsymbol{y}$ to represent the vector of observed noisy data, $\boldsymbol{f}$ to represent the latent function values from the Gaussian process [equation (2.9)], and $\boldsymbol{\varepsilon}$ for a set of i.i.d. Gaussian noise, then the hyper-parameters of the GP can be optimized as follows. Since

$$\boldsymbol{y} \mid \boldsymbol{f}, \boldsymbol{\phi} \sim N(0, \sigma^2 \boldsymbol{I}) \quad , (2.12)$$

$$\boldsymbol{f} \mid \boldsymbol{\phi} \sim N(0, K), \quad \text{where } K_{ij} = k((x_i,t_i),(x_j,t_j)) \quad (2.13)$$

the likelihood can be calculated from integration

$$p(\boldsymbol{y} \mid \boldsymbol{\phi}) = \int p(\boldsymbol{y} \mid \boldsymbol{f}, \boldsymbol{\phi}) p(\boldsymbol{f} \mid \boldsymbol{\phi}) d\boldsymbol{f} \quad (2.14).$$

Equation (2.14) is the convolution of two Gaussian densities [15] so the integral can be simplified to

$$p(\boldsymbol{y} \mid \boldsymbol{\phi}) = N(0, K + \sigma^2 \boldsymbol{I}) \,. \quad (2.15)$$

So the maximum likelihood estimate of hyper-parameters can be written as

$$\hat{\boldsymbol{\phi}} = \arg\max_{\phi} \log(p(\boldsymbol{y} \mid \boldsymbol{\phi})) = \arg\max_{\phi} \left(-\frac{1}{2} \boldsymbol{y}^T (K + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{y} - \frac{1}{2} \log \left| K + \sigma^2 \boldsymbol{I} \right| \right). \quad (2.16)$$

After inferring the hyper-parameters, the predictions on new data points can be made by first writing down the joint distribution of the observed target values and the function values we want to predict at the test points:

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{f}^* \end{bmatrix} \sim N\left(0, \begin{bmatrix} K(X, X) + \sigma^2 \boldsymbol{I}, K(X, X^*) \\ K(X^*, X) \qquad , K(X^*, X^*) \end{bmatrix} \right). \quad (2.17)$$

Then the corresponding conditional predictive distribution is

$$\boldsymbol{f}^* \mid \boldsymbol{y} \sim N(K(X^*, X)(K(X, X) + \sigma^2 \boldsymbol{I})^{-1} \boldsymbol{y}, K(X^*, X^*) - K(X^*, X)(K(X, X) + \sigma^2 \boldsymbol{I})^{-1} K(X, X^*))$$

(2.18).

2.3.2 Logistic Gaussian process

Logistic Gaussian process is a generalization of Gaussian process described previously. For ordinary Gaussian process regression, the predicted value at a new point of interest follows Gaussian distribution of the form shown by equation (2.18), and thus the possible range of predicted value is $(-\infty, +\infty)$. However, in this cell movement case, what we are interested in is the cell density or the probability density of the cell (which can be calculated by dividing the cell density by the sum of celss) at certain time point and certain location. A probability density function, by its definition, should be non-negative and integrate to 1 on the whole space. Therefore, the method of logistic Gaussian process is adopted to transform the ordinary Gaussian process to valid probability densities. From Riihimaki & Vehtari (2013), the logistic density transform is of the form:

$$p(\boldsymbol{x}) = \frac{\exp(f(\boldsymbol{x}))}{\int_{\nu} \exp(f(\boldsymbol{s}))d\boldsymbol{s}}, \quad (2.19)$$

where ν is the finite region that the unknown distribution is in, $f(\boldsymbol{x})$ is an unconstrained latent function, and in logistic Gaussian process, a GP prior is put on $f(\boldsymbol{x})$,

$$f(\boldsymbol{x}) \sim GP(0, k(\boldsymbol{x}, \boldsymbol{x}')), \quad (2.20)$$

where $k(\boldsymbol{x}, \boldsymbol{x}')$ is the covariance function. Further mathematical details of logistic Gaussian process are discussed in Riihimaki & Vehtari (2013).

2.4 Gradient matching

After the data is interpolated, we can move on to the next step of calculating the gradients of the interpolant. For Gaussian processes, one of the advantages is that the distribution of the gradients can be derived analytically if the kernel is differentiable, and the derivative of a GP is also a GP. From Rasmussen (2006), the covariance function between data points and partial derivatives and the covariance function of different partial derivatives can be written as:

$$\operatorname{cov}\left(f_i, \frac{\partial f_j}{\partial x_{dj}} \right) = \frac{\partial k(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\partial x_{dj}}, \quad \operatorname{cov}\left(\frac{\partial f_i}{\partial x_{di}}, \frac{\partial f_j}{\partial x_{ej}} \right) = \frac{\partial k(\boldsymbol{x}_i, \boldsymbol{x}_j)}{\partial x_{di} \partial x_{ej}}. \quad (2.21)$$

As such, the predictive distribution of the partial derivatives is of the form:

$$\begin{aligned} &\frac{\partial \boldsymbol{f}}{\partial x_d} \sim N(m_d, K_d), \text{ where} \\ &m_d = K'(K)^{-1}\boldsymbol{f}, \quad K_d = K'' - K'(K)^{-1}(K')^T, \text{ where} \\ &K'_{ij} = \operatorname{cov}\left(\frac{\partial f_i}{\partial x_{di}}, f_j \right), \quad K''_{ij} = \operatorname{cov}\left(\frac{\partial f_i}{\partial x_{di}}, \frac{\partial f_j}{\partial x_{dj}} \right) \end{aligned} \quad (2.22)$$

In this study, the means of the predictive distributions are regarded as the estimate of the partial derivatives and are used in the following gradient matching steps.

For smoothing methods whose partial derivatives cannot be derived analytically, we can use numerical methods to get approximated values of the gradients. For a continuous

and (at least) second-order differentiable function $f(x)$, from Taylor's expansion, we can get

$$f(x+a) = f(x) + a\frac{\partial f(x)}{\partial x} + a^2\frac{1}{2!}\frac{\partial^2 f(x)}{\partial x^2} + \ldots (2.23)$$

$$f(x-a) = f(x) - a\frac{\partial f(x)}{\partial x} + a^2\frac{1}{2!}\frac{\partial^2 f(x)}{\partial x^2} + \ldots (2.24)$$

From equation (2.23) – equation (2.24), we can get

$$\frac{\partial f(x)}{\partial x} \approx \frac{f(x+a) - f(x-a)}{2a}. \quad (2.25)$$

From equation (2.23) + equation (2.24), we can get

$$\frac{\partial^2 f(x)}{\partial x^2} = \frac{f(x+a) + f(x-a) - 2f(x)}{a^2}. \quad (2.26)$$

After the partial derivatives are calculated, we can move on to the actual gradient matching steps. The main idea of gradient matching is that for a good interpolant of the data, not only should the fitted values from the interpolant be close to the real data points under certain error metrics, but the estimated partial derivatives should also be close to the real gradients in some ways. As for PDE models, the partial derivatives follow the equation (1.1). So we can put the fitted derivatives into the left hand side of equation (1.1), and measure the difference between the left hand side of equation (1.1) and zero using the following loss function

$$L(\boldsymbol{\theta}) = \frac{\sum_{i=1}^{n}\left(f(\boldsymbol{x}_i, u_i, \frac{\partial u}{\partial \boldsymbol{x}_i}, \ldots; \boldsymbol{\theta}) - 0 \right)^2}{n}. \quad (2.27)$$

The estimates of parameters are then derived by

$$\hat{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta}). \quad (2.28)$$

We can find out that if the PDE in equation (1.1) is a linear one, minimizing the loss

function in equation (2.27) is equivalent to performing a linear regression, with one of the partial derivatives being regarded as the dependent variable and others as the independent variables. For non-linear PDE systems, the loss function can be optimized by a wide variety of optimization algorithms.

To check the accuracy of parameter estimation using gradient matching, we simulate multiple datasets using the closed form solutions of diffusion and advection-diffusion equations [equation (2.2) and (2.4)] for the same real parameter level. For each dataset, gradient matching is conducted and point estimates of parameters are derived. Then the point estimates from different datasets can be shown in a boxplot. Inspecting the median and spread of the boxplot indicates the accuracy of the method.

3. Data

3.1 Diffusion and advection-diffusion equation

The test data were generated from closed form solutions [equation (2.2) and (2.4)] with the following parameters: $D = 2$ for equation (2.2) and $D = 2, u = 3$ for equation (2.4). Multiple datasets were generated, each contained certain number of data points, randomly generated from a certain region of time and space. Further details are discussed in corresponding result parts. To better simulate the reality, independent and identically-distributed Gaussian noises are added to some of the datasets, with the noise level controlled by the signal-to-noise ratio (SNR), which is defined by

$$SNR = \frac{\mathrm{var}(data)}{\sigma_{noise}^2} \quad . (3.1)$$

Further details of the simulated datasets are discussed in corresponding result parts.

3.2 Cell movement data

The cell movement data are collected from a video of a group of *Dictyostelium* amoebae. The *Dictyostelium* cells are added to the center of a dish of agar containing uniform levels of the chemoattractant folate across the whole dish. The cells consume the local

attractant and create a resource gradient. The movement of cells was filmed by a microscope. We made use of eight screen shots from the video at time t=0s, 2s, 4s,...,14s, and for each screenshot, the locations of the cells were extracted using the ImageJ software.

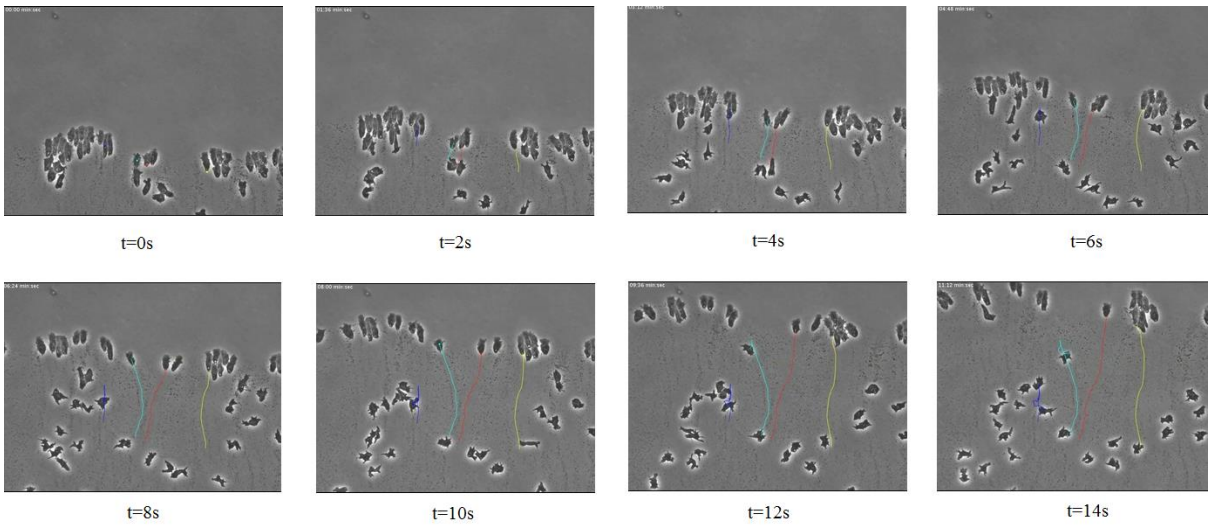


Figure 3. Screenshots from the video of cell movement. As the screenshots show, the behaviour of cells has two different types, a proportion of cells move towards the upper side of the plots, while the rest of the cells move randomly around the lower side.

4. Results

4.1 Gradient matching with closed form solution

If we have the closed form solution of the PDE, we can try to perform gradient matching onto real gradients instead of the gradients of interpolants. Fro example, for the diffusion equation (2.1), if we know that at the point (x_0, t_0), the partial derivatives are:

$$\frac{\partial c}{\partial t}(x_0, t_0) = 6, \frac{\partial^2 c}{\partial x^2}(x_0, t_0) = 3 \quad . (4.1)$$

Then we can put equation (4.1) back into equation (2.1), and within a second we can solve the equation and realize that $D = 2$.

4.2 Gradient matching with numerically calculating the gradients

However, it is usually impractical to get the closed form solution of a PDE or the analytical solutions of gradients. One of the solutions is to approximate the derivatives using numerical methods. To do this for the advection-diffusion equation, for example, we first divide the space and time region of interest into an $n_g \times n_g$ equidistant grid, and we may call the parameter n_g as "the size of the grid". It can be seen that for a fixed region, larger value of n_g represents lower distance between adjacent grid points and better resolution of the grid. After setting up the grid, we calculate the function values for each grid points using closed form solution equation (2.4). Afterwards, the numerical estimates of partial derivatives are calculated using equation (2.25) to (2.26) (the partial derivatives at the points on the border of the grid cannot be calculated in this way so these points are neglected in the following steps). Because equation (2.3) is a linear PDE, the gradient matching can be done via linear regression. From equation (2.3), we derive

$$\frac{\partial c(x,t)}{\partial t} = -u\frac{\partial c(x,t)}{\partial x} + D\frac{\partial^2 c(x,t)}{\partial x^2}. \quad (4.2)$$

Size of the grid n_g	100	50	30	10
Estimate of u	3.01855	3.06129	3.13126	3.38269
Percentage away from real value $u = 3$	0.62%	2.04%	4.38%	12.76%
Estimate of D	1.9828	1.93325	1.82376	0.80258
Percentage away from real value $D = 2$	-0.86%	-3.34%	-8.81%	-59.87%

Table 1. The parameter estimation using gradient matching based on noise free data from the closed form solution of advection-diffusion equation from a grid of space $x \in (0, 2)$ and time $t \in (0.3, 5)$. The result shows that as the size of the grid increases, the estimates are closer to their real values $u = 3$ and $D = 2$.

With partial derivatives being calculated, the parameters can be estimated by fitting a linear model where $\frac{\partial c(x,t)}{\partial t}$ is the dependent variable and $\frac{\partial c(x,t)}{\partial x}$ and $\frac{\partial^2 c(x,t)}{\partial x^2}$ are the independent variables.

For each size of grid n_g's, one dataset was produced. The point estimates of parameters from these datasets are shown in Table 1.It can be seen from the table that with more grid points taken (i.e., narrower distance between adjacent grid points), the estimates become closer to the real value, and even if the size of grid is not very large ($n_g = 30$), the estimates are still within 10% of the real value. However, the biggest disadvantage of conducting gradient matching with unsmoothed data is that this method is not robust against noise at all. To illustrate this, we add some minor noise (SNR=1000) to the data calculated from the closed form solution. The same gradient matching steps are conducted and the results are shown in Table 2.

Size of the grid	100	50	30	10
Estimate of u	0.4389	1.1978	1.989	2.889
Percentage away from real value $u = 3$	-85.37%	-60.07%	-33.70%	-3.70%
Estimate of D	8.269e-06	0.00015	0.0005	0.0158
Percentage away from real value $D = 2$	-100.00%	-99.99%	-99.98%	-99.21%

Table 2. The parameter estimation using gradient matching based on data from the closed form solution of advection-diffusion equation from a grid of space $x \in (0,2)$ and time $t \in (0.3,5)$ with i.i.d. Gaussian noise of SNR=1000 added. The result shows that with such minor noise, the results are no longer close to real values $u = 3$ and $D = 2$.

As Table 2 shows, with minor noise (SNR=1000) the estimates of D based on unsmoothed data become extremely close to zero and the estimates of u become closer

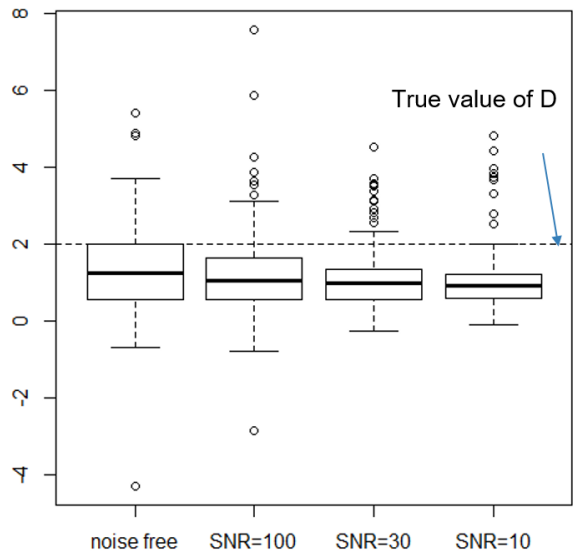
to the true value of u as the resolution becomes worse, which is counter-intuitive. Both of the phenomena can be explained. From equation (2.26), the numerical solution of second-order derivatives contains the numerator denominator a^2, which in this case can be very small because a represents the distance between adjacent grid points. As a result, the numerically calculated second-order derivatives become more sensitive to noise and the estimated diffusion coefficients become rather close to zero because the noise has outweigh the information. As for the advection coefficient, as it is related to the first-order derivative calculated from equation (2.25), since the absolute noise level does not change (i.e. the distributions of noise on the numerator of equation(2.25) do not change), the influence of noise will become less as the distance a becomes larger.

In reality, it is hard for us to get closed form solutions for the PDEs of interest or collect noise-free data. So we have to use some smoothing methods to interpolate the data.

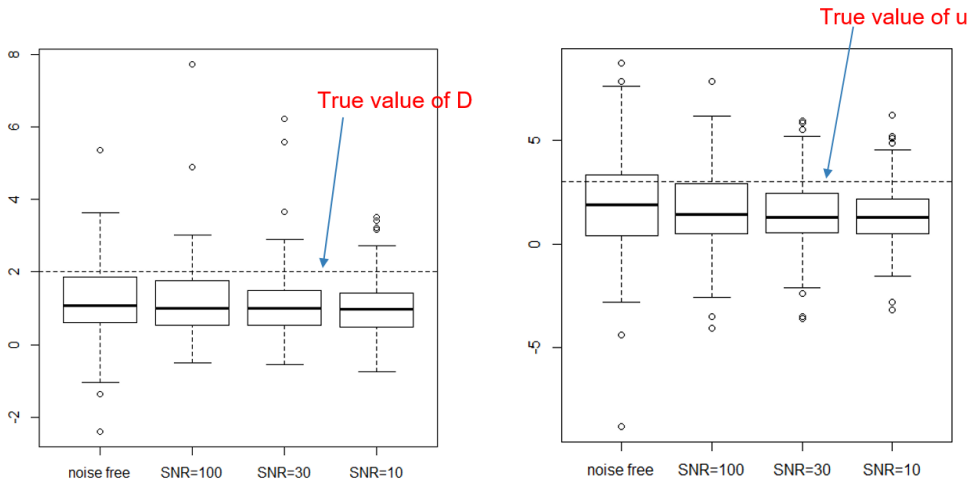
4.3 Gradient matching with Gaussian processes

Using the closed form solutions in equation (2.2) and (2.4), we generated random data points of the two PDE models and then added i.i.d. Gaussian noise to them.

Four noise levels are selected in order to test the effectiveness of Gaussian process regression and gradient matching, and how the results will change as noise enlarges. The noise levels are: "noise-free", $SNR = 100$, $SNR = 30$ and $SNR = 10$. For each noise level, 100 independent datasets were generated. Each data set contains 600 data points, randomly chosen from the time and space domain $t \in (0,2), x \in (0,2)$. Each data set was then interpolated using Gaussian process regression with squared exponential kernel, after which the gradients of the GP are calculated and fitted into the PDE to get point estimates of the parameters. The estimates of parameters are collected and shown in boxplots in Figure 4.



(a)



(b) (c)

Figure 4. The boxplots of point estimates of parameters of diffusion equation [(a)] and advection-diffusion equation [(b) and (c)] using Gaussian process and gradient matching

By examining Figure 4, we can see that for both models and all the parameters, the bulks of the distributions are biased from the real values, and as the noise level increases, the distributions of point estimates become closer to zero and further from the real values. This might suggest that it is harder to extract useful information from data as more noise is included. Although the medians of the point estimates are about 50% away from the

real values of the parameters and most of the IQRs of the boxplots do not contain the real value of parameters, the results do show a positive correlation between the inferred parameters and real values, which encourages us to further explore the method and try to improve its performance.

We can first do this by visual inspection. Figure 5 illustrates the closed form solution of the diffusion equation and three fitted GPs from four datasets with different noise levels (the datasets come from the same set of sampled (x,t) coordinates, where the only difference is the added Gaussian noise). By inspecting these figures, we can come up with the following hypotheses of why the method did not perform well.

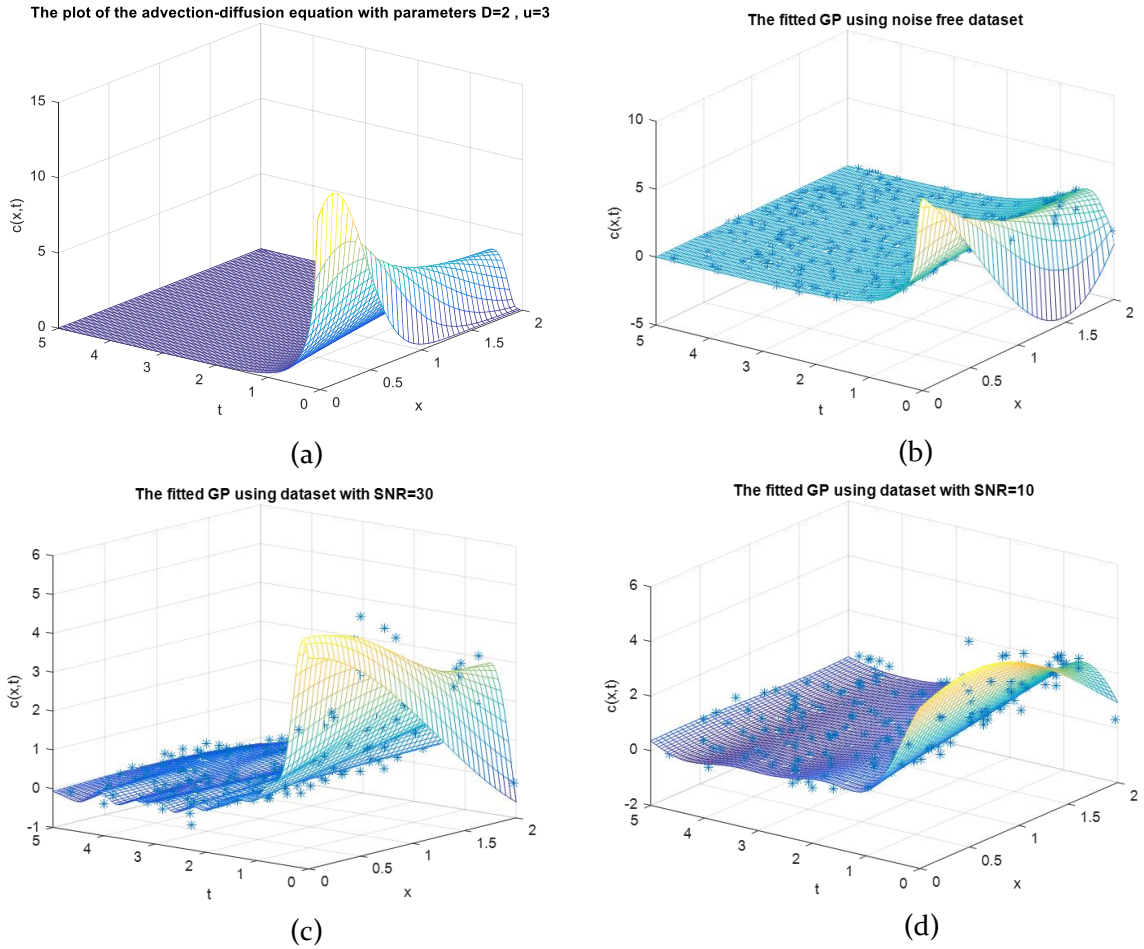


Figure 5. The plot of advection-diffusion equation and three Gaussian process interpolants with different noise levels. As the plot of the closed form solution of the PDE shows, the plot is the sharpest when t is small, and the fitted GPs with squared exponential kernel, cannot accurately capture the sharp peak. Besides, as the noise increases, the interpolants are less close to the original curve.

1. The Gaussian process might not fit the data well. In the space and time region where the closed form solution is relatively flat, the GP interpolants look quite similar to the original plot. But in the region where the function is sharp, the GPs failed to capture the high-value data points.

2. The data might be too sharp at some locations. When the value of time t is small, there are a few data points with relatively high function values, which makes the plot become sharp at these areas. That might increase the difficulty of interpolating.

3. Increased noise levels will worsen the results of estimation. Larger noise will make it more difficult for us to extract useful information from the data. In the worst case, if the noise is very large, we cannot get any useful information from the data.

Respectively, we can come up with potential solutions and see how it can improve the result.

To get a well-fitted GP interpolant, a proper covariance function is needed. Initially, the covariance function of the GP regression is the squared exponential kernel

$$k((x_1,t_1),(x_2,t_2)) = \sigma_f^2 \exp\left(-\frac{(x_1-x_2)^2}{2l_1} - \frac{(t_1-t_2)^2}{2l_2}\right) \quad (4.3)$$

From Rasmussen (2006), the squared exponential kernel is a stationary covariance function, which means that it is a function of $(\mathbf{x}-\mathbf{x}')$ and is invariant to the transforms in the input space. However, after having a look at the plots of the closed form solutions of the diffusion equation and advection-diffusion equation, we will find that both processes are not stationary, and a stationary covariance function like the squared exponential kernel may not be proper in this case. So we considered a non-stationary covariance function, namely the neural network covariance function [also called kernel multilayer perceptron, Rasmussen (2006) provides more details about it]. Figure 6 provides the plot of the predicted GP curve using neural network covariance function with noise-free data simulated from the advection-diffusion equation (with parameter u=3, D=2). At first glance, the curve of the neural network covariance function is more

similar to the original PDE curve (Figure 5(a)) compared to the curve of squared exponential kernel (Figure 5(b)). But if we do the same gradient matching procedure as before using the neural network covariance function, the parameters inferred from this Gaussian distribution is far away from the true values. To figure out the reason, we take a deeper look at the PDE and the fitted GPs.

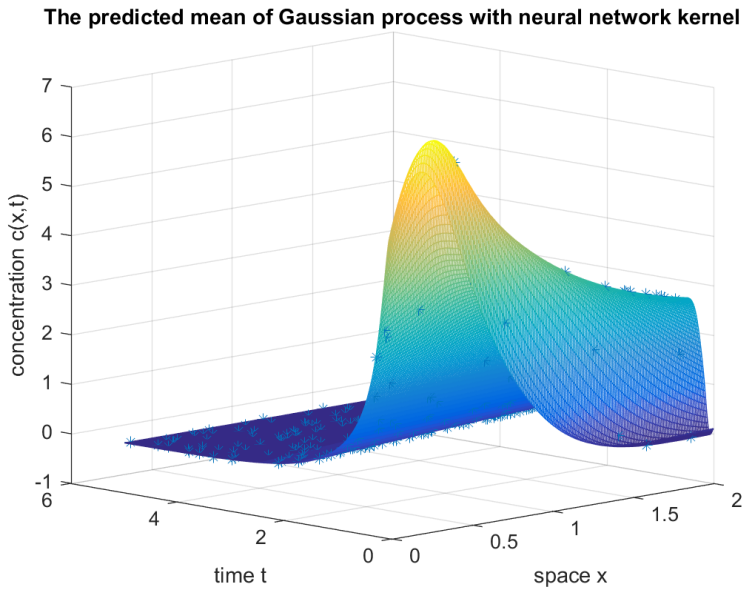
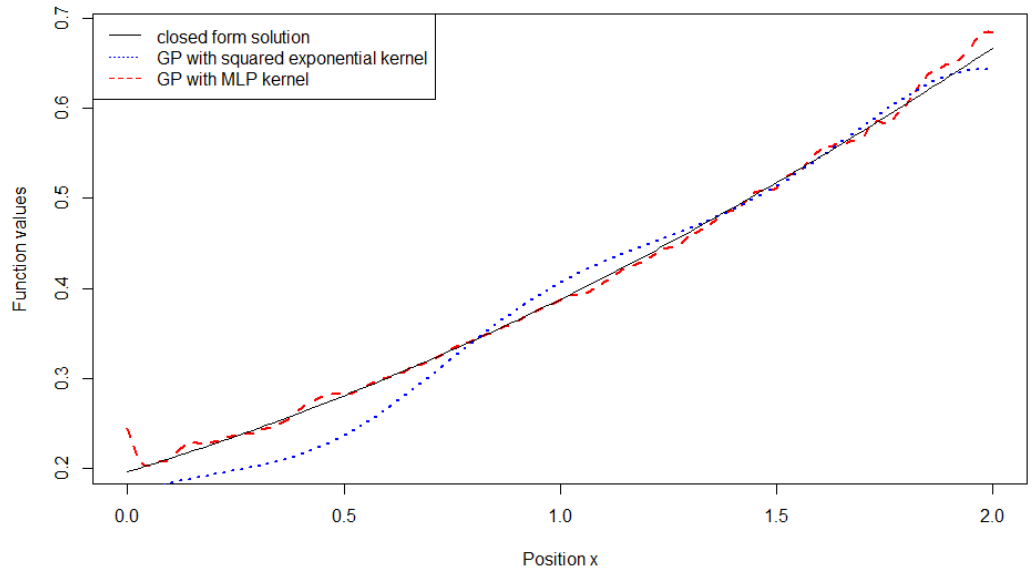
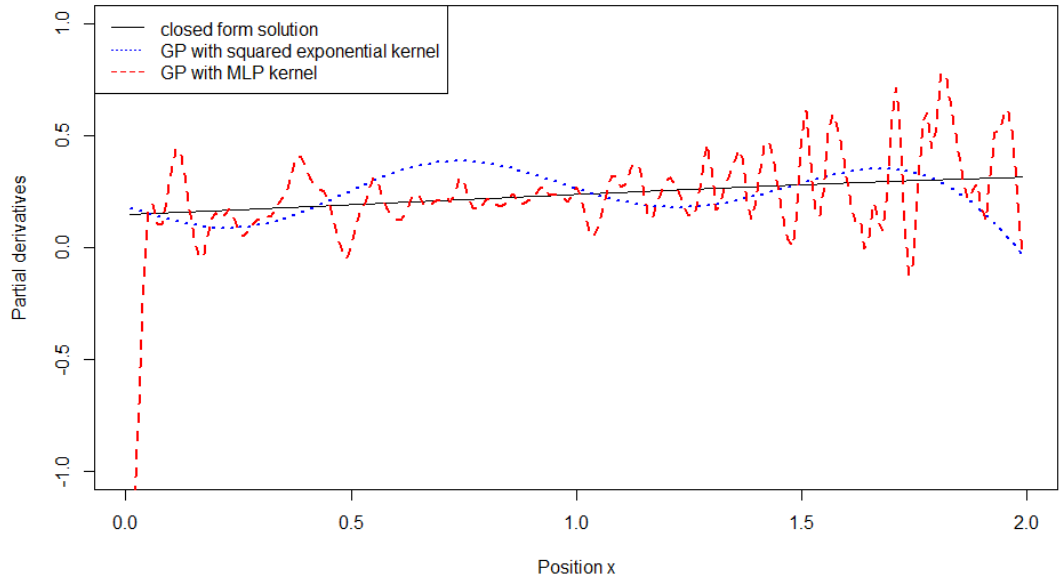


Figure 6. The plot of the predited mean of Gaussian process with neural network kernel, based on simulated dataset from closed form solution of advection-diffusion equation

Figure 7 shows the cross sections of the PDE and fitted GPs at time t=1.8. As the plots show, the original PDE curve is smooth, so is the fitted GP with squared exponential kernel. For the neural network covariance, although the fitted data points are closer to the real values, the curve is relatively rugged. As a result, the partial derivatives (especially the second-order ones) calculated from the GP can be far from the real gradients. And this will lead to the failure of gradient matching as we fit the gradients from the GP back into the original PDE.


closed form solution
GP with squared exponential kernel
GP with MLP kernel
Function values
0.2
0.3
0.4
0.5
0.6
0.7
0.0
0.5
1.0
1.5
2.0
Position x


closed form solution
GP with squared exponential kernel
GP with MLP kernel
Partial derivatives
-1.0
-0.5
0.0
0.5
1.0
0.0
0.5
1.0
1.5
2.0
Position x

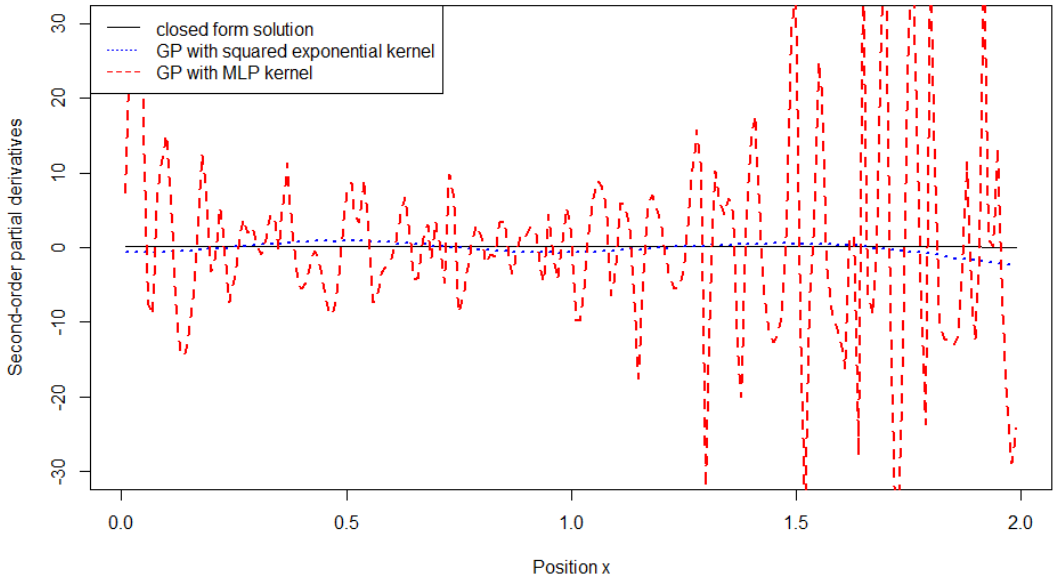


Figure 7. The plot of closed form solution of advection-diffusion equation and two fitted GP curves at t=1.8 (upper) and the plots of corresponding partial derivatives and second-order partial derivatives w.r.t. x (middle and lower).

Another probable reason of the unsatisfying GP fits is that we may not sample sufficient data points from the sharp area of the function where time is low, so we do not capture all the characteristics of the function and that may lead to unsatisfactory GP fits in this region. To try to solve this, instead of sampling from a uniform distribution of time and space, we try to sample more data points in the region where the value of t is small and fewer data points where t is relatively large. However, as is shown in Figure 8, this modification does not provide significant improvement to the results. We take a look at one of the failed cases. As Figure 9 demonstrates, in this case, there is a data point with extremely large function value. As a result, the GP regression with squared exponential kernel becomes rather unstable and the gradient matching will fail.

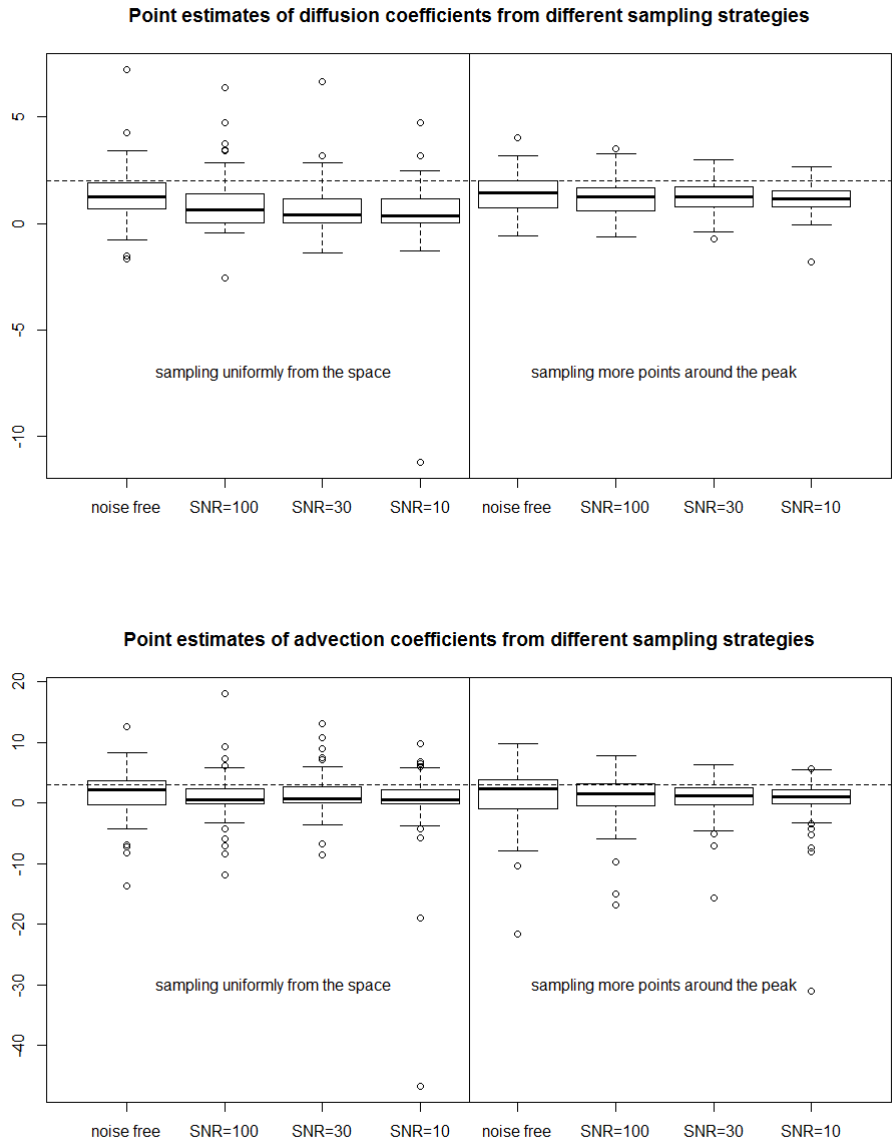


Figure 8. The boxplots of point estimates when more data points are simulated from the peak area (right side of both plots) compared with data generated uniformly from the space of $x \in (0,2), t \in (0,5)$. From the plots, the results are not significantly improved by sampling more points around the peak.

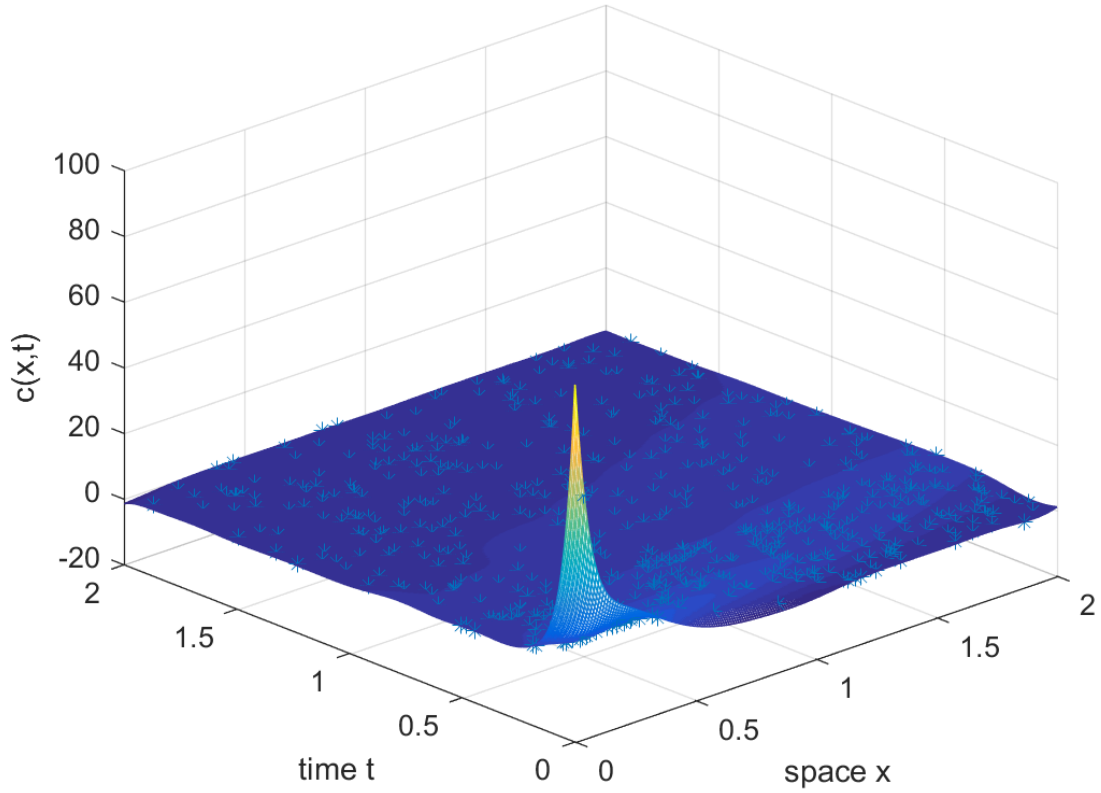


Figure 9. the plot of the fitted mean of Gaussian process from one noise free dataset. We can observe a data point with very large function values, which might be the reason that the gradient matching do not do well in this case. To try to solve this, we consider cutting off the peak area in following steps.

After observing this phenomena, we may try a different modification of data sampling. Instead of sampling more data points from the sharp area, we abandoned sampling from this sharp area by cutting off the time domain. This modification is reasonable from a practical perspective. If we look back at the closed form solutions [equation (2.2) and (2.4)], we may find the following initial conditions underlying the closed form solutions

$$c(x,t=0)=\begin{cases}+\infty, & x=0\\ 0 \quad , & \text{elsewhere}\end{cases} \qquad (4.4)$$

Which is not realistic in practice since we cannot have infinite concentration at one position and zero concentration at any other positions. If we cut off some time domain where t is small $(\text{say}, t\in(0,t_0))$, we can regard the new process as a PDE start at $t=t_0$

with initial condition $c(x,t_0)$, which is continuous and differentiable, and this might be more close to what happened in real world. Besides, as the sharpest part of the function is cut off, it may be more appropriate to use the squared exponential kernel. Although the real process is never stationary, it looks close to stationary in this restricted area, which makes the use of squared exponential kernel appropriate. The boxplots of the point estimates from datasets drawn from different time and space regions are shown in Figure 10.

After inspecting Figure 10, we may conclude that after choosing proper time and space region, the estimates of parameters gathered from gradient matching can be very close to the real values in the noise-free scenario. As the noise gets larger, the estimates become more biased and the variances of the estimates become larger, but the percentages that the medians of the point estimates are away from the real values are no more than 15% even when the noise level is high (SNR=10), and the IQRs of the boxplots are much narrower compared with the results in Figure 4. In general, the method of gradient matching with Gaussian process performs reasonably in estimating the parameters of diffusion equation and advection-diffusion equation, so we can move on to the next step, and try to estimate the parameters of PDE models describe the cell movement.

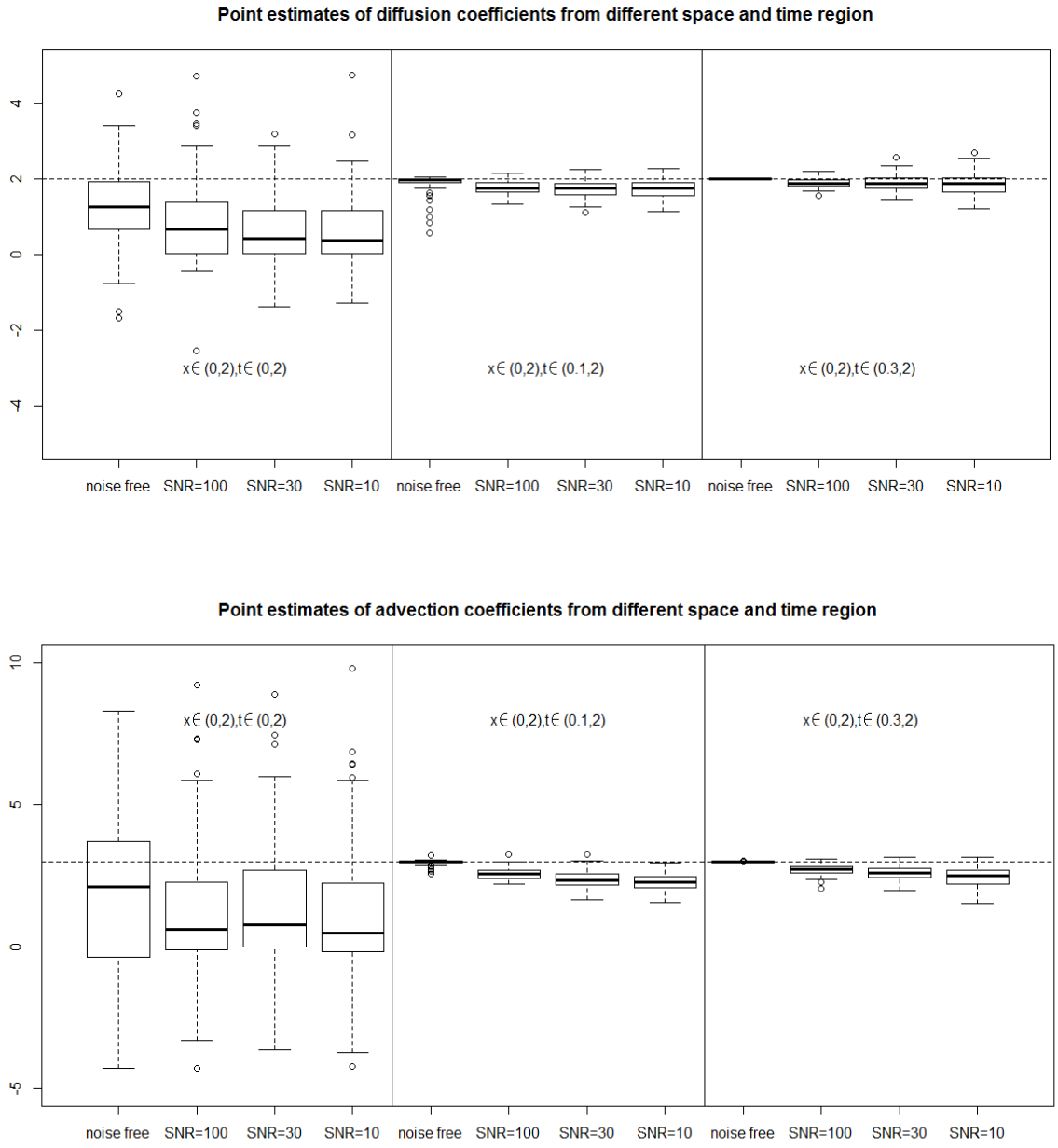


Figure 10. The boxplots of point estimates from gradient matching of parameters in the advection-diffusion equation using GP regression and squared exponential kernel. On both plots, the four boxplots on the left are the boxplots of point estimates inferred from data simulated from the region $x \in (0,2),\ t \in (0,2)$. The boxplots on the middle are from data simulated from the region $x \in (0,2),\ t \in (0.1,2)$,where some of the peak of the function are cut off. And the four boxplots on the right are from data simulated from the region $x \in (0,2),\ t \in (0.3,2)$,where most of the peak of the

function are cut off. From the boxplots, we can find out that after cutting off the peak, the accuracy of point estimates improve a lot. When the data are noise free and the time interval is restricted, the point estimates of parameters are distributed very closely around the true values. When the data are noisy, the biases and spreads of the point estimates are much smaller for the constrained time interval compared with the original time interval. But most of the IQRs of the boxplots still do not contain the true values of parameters, showing that the noises still have large impact on the accuracy of estimation.

4.4 Gradient matching and model selection of cell movement data

The cell position data is first transformed into probability density of cell using a logistic Gaussian process with three different covariance functions, namely the squared exponential kernel, the Matérn covariance function with $\nu = 3/2$ (referred to as "Matérn32" in the following sections) and the Matérn covariance function with $\nu = 5/2$ (referred to as "Matérn52" in the following sections). The Matérn covariance function is of the form

$$C_\nu(d) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}\frac{d}{\rho} \right)^\nu K_\nu \left(\sqrt{2\nu}\frac{d}{\rho} \right) \quad , (4.5)$$

Where $\Gamma(\nu)$ is the gamma function, K_ν is the modified Bessel function of the second kind, ρ and d are non-negative hyper-parameters.

For $\nu = 3/2$,

$$C_{3/2}(d) = \sigma^2 \left(1 + \frac{\sqrt{3}d}{\rho} \right) \exp\left(-\frac{\sqrt{3}d}{\rho} \right) \quad . (4.6)$$

For $\nu = 5/2$,

$$C_{5/2}(d) = \sigma^2 \left(1 + \frac{\sqrt{5}d}{\rho} + \frac{5d^2}{3\rho^2} \right) \exp\left(-\frac{\sqrt{5}d}{\rho} \right). \quad (4.7)$$

The relationship between the probability density of cell (represented by $p(x,t)$) and the

cell density $C(x,t)$ in equation (2.5) can be represented as

$$p(x,t) = \frac{C(x,t)}{n(t)} \quad , (4.8)$$

where $n(t)$ is a function of time describe the total number of cells on the space. In this study, the data were collected in a short time interval and an isolated space, so we can assume that no cells died, were reproduced or moved in and out of the region during the time interval. Thus the function $n(t)$ can be regarded as a constant. As a result, the PDE models for cell density $C(x,t)$ also hold for the probability density $p(x,t)$.

To calculate the logistic GP density, the lgpdens() function from the Matlab package "gpstuff" is called [14]. The space and time domain is separated into grids, and for each grid, the probability density at the center of the grid is estimated via a logistic Gaussian process. After inspecting the data, the grid points are set to be: 400 equidistant points from -100 to 900 for x and {0,2,4,...,14} for t. Figure 11 illustrates the fitted logistic Gaussian process density with the squared exponential kernel. As the figure shows, initially, the cells are concentrated at around x=500. Afterwards, a large majority of cells shows a directional movement towards the negative direction of x axis, with the remaining cells showing some kind of random movement around the start point. After communicating with the data collector, we realized that the data collector used an unconventional coordinate system. To make the figure more intuitive, we perform the following transformation on to the x coordinate

$$x_{new} = -(x_{old} - 500) . \quad (4.9)$$

The illustration of the old and new coordinate system is shown in Figure 12. In the rest of this paper, all the x coordinates refer to the transformed ones. The logistic GP density plots with transformed space coordinates and squared exponential kernel are shown in Figure 13.

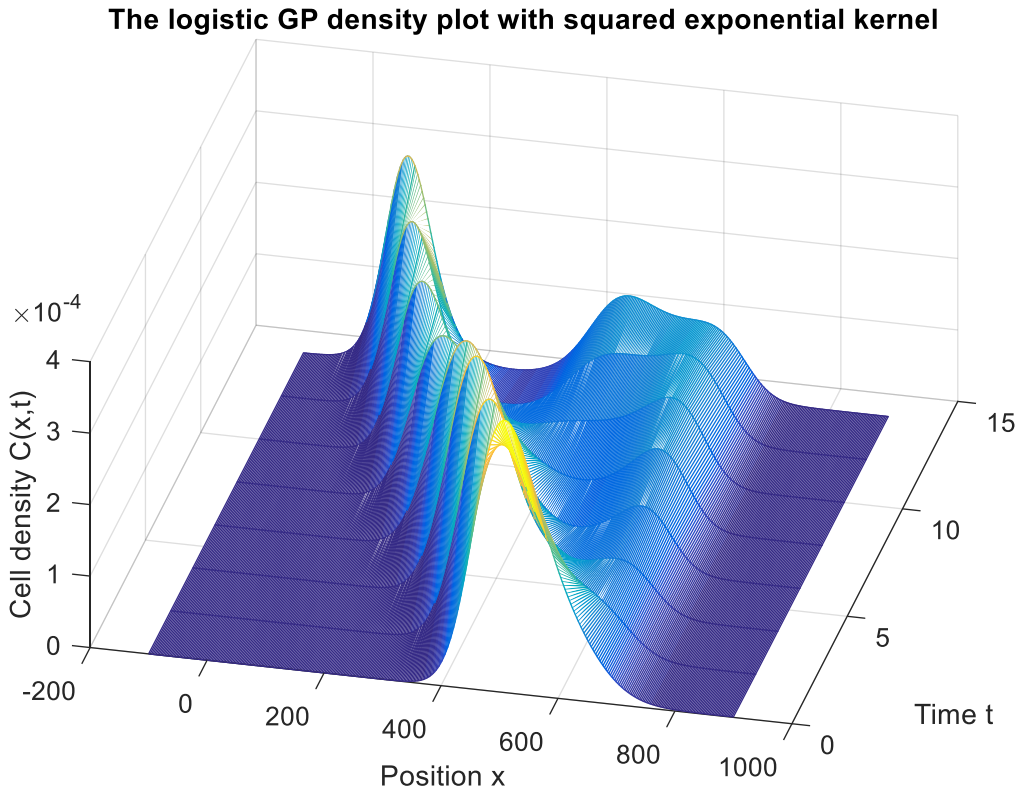


Figure 11. The plot of cell density fitted by logistic Gaussian process with squared exponential kernel. As time goes, the distribution of cells show a significant bimodality. Besides, a large proportion of cells moves form positive x to around zero, which is caused by the unconventional x coordinate. So we decided to modify the data to make the plots more intuitive.

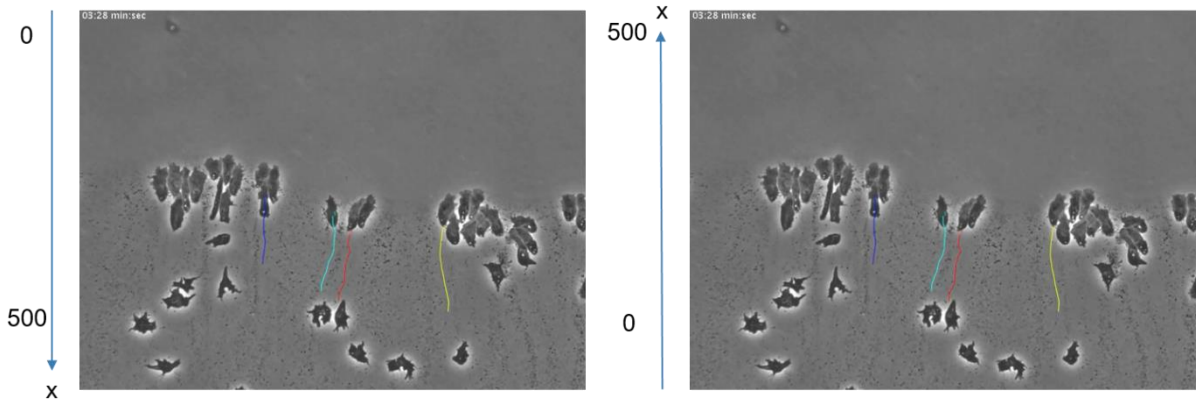


Figure 12. The illustration of original (left) and modified (right) x coordinate systems. The modified coordinate system is easier to understand as the cells start at around x=0 and move towards larger x.

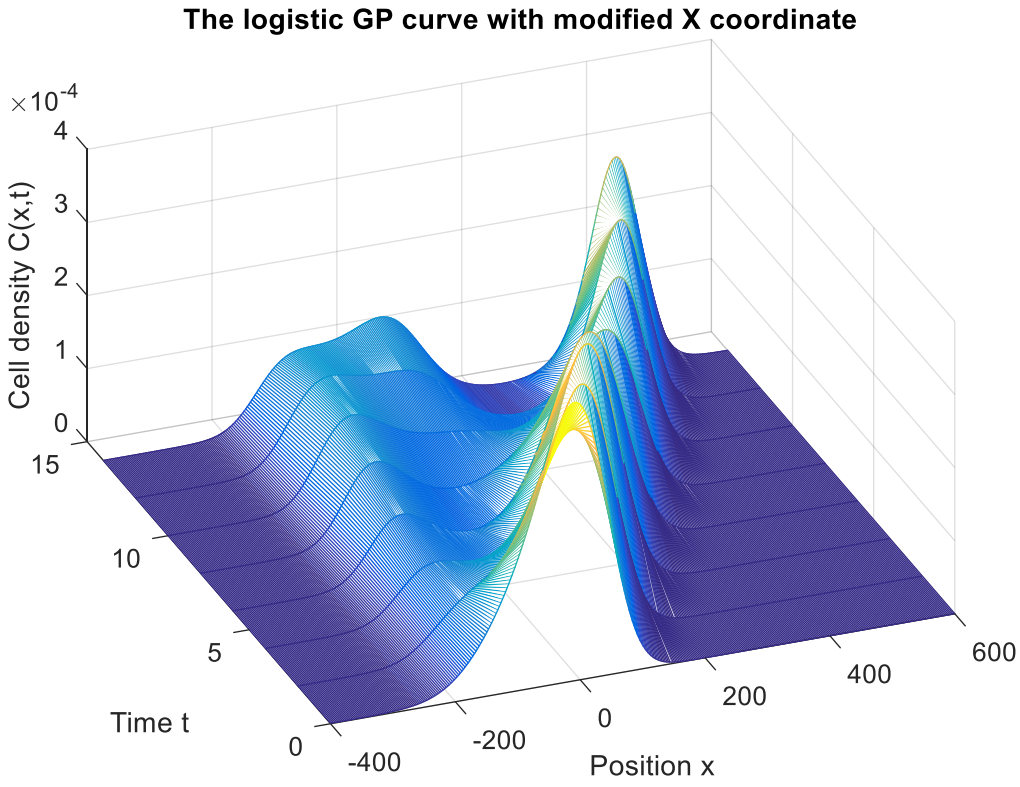


Figure 13. The logistic GP curve with modified coordinate system.

After fitting the logistic Gaussian process, the partial derivatives of the fitted densities are calculated using numerical approximation [equation (2.25) and (2.26)] and the parameters are optimized by minimizing corresponding loss function in equation (2.27). The Nelder–Mead algorithm was used to optimize the loss function. For the sigmoidal attractant model, the loss function is complex and multi-modal, so the algorithm always falls into local optima. To solve this, we tried multiple different start points and chose the best result from these trails as our final result. The inferred parameters for both models are $\hat{u} = 19.6693,\ \hat{D} = 0$ for the constant advection model and $\hat{\alpha} = 7971.94,\ \hat{\beta} = 0.01038,\ \hat{\gamma} = 27.55,\ \hat{D} = 75.0894$ for the sigmoidal attractant model.

After optimizing the parameter estimation, a numerical solver of PDE (Matlab toolbox "pdepe") is called to get the numerical solutions of the PDE models with corresponding estimated parameters. The initial conditions are set to be the fitted logistic GP density at t=0 and boundary conditions are set to be zeros. Figure 14 shows the fitted PDE curves

of the constant advection model and the sigmoidal attractant model. From the plots, we can observe that both models captured the directional cell movement, but not the random movement around the start point. From Figure 13 we can see that the logistic GP interpolant derived from original data shows significant bimodality as the density has two peaks when t is large. However, both fitted models in Figure 14 are unimodal, although the spread of cells becomes large as time goes by in the sigmoidal attractant model, which makes the plot a bit more reasonable than the other model.

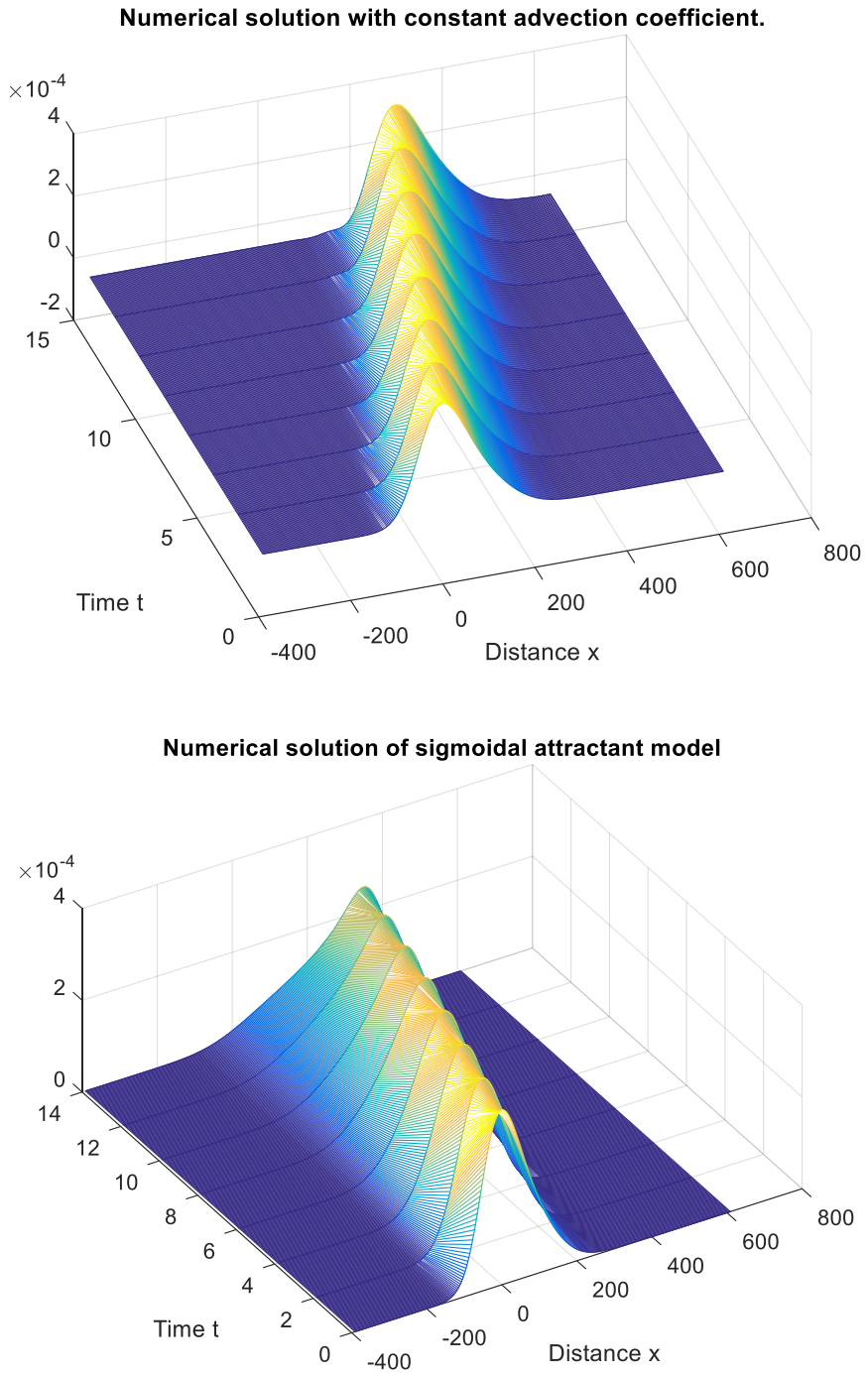


Figure 14. The plots of the numerically solved PDE models of constant advection coefficient [equation (2.6)] and sigmoidal attractant [equation (2.8)]. Both plots do not capture the bimodality of the data, so we decide to modify the sigmoidal attractant model with a "center" parameter to see if it can improve the result

To try to further improve the sigmoidal attractant model, we take a deeper look at the attractant concentration and its spatial derivative [equation (2.8)]. After solving the equation

$$\frac{\partial}{\partial x}\left(\frac{\partial A(x,t)}{\partial x}\right)\Bigg|_{t=0} = 0\,, \quad (4.10)$$

we can see that no matter what the parameter values are, the spatial derivative $\frac{\partial A(x,t)}{\partial x}$ reaches its maximum at $x=0$ given $t=0$, which means that no matter how we perform the parameter estimation, the sigmoidal attractant model is based on the underlying assumption that given $t=0$, the concentration of attractant changes most rapidly at position $x=0$, which is not always the case because the x coordinates are set arbitrarily and we do not have any information about the underlying change of concentration of attractant. Inspired by this, a new parameter μ , representing the "*center*" of the sigmoidal function, is introduced to the sigmoidal attractant model, and the new expression of attractant concentration can be written as:

$$\frac{\partial A(x,t)}{\partial x} = \frac{\alpha\beta\exp(-\beta((x-\mu)-\gamma t))}{\{1+\exp(-\beta((x-\mu)-\gamma t))\}^2}\,. \quad (4.11)$$

We can do the same gradient matching procedure to the new model. The optimization is conducted using the Nelder-Mead algorithm with starting value of α,β,γ being set as the optimized parameter values of the corresponding sigmoidal attractant model without centring and $\mu^{(0)}=0$. The numerical solution of the PDE model with optimized parameters is shown in Figure 15. Using the exponential squared kernel, the parameters inferred from the model sigmoidal attractant model with center parameter are $\hat{\alpha}=5723.07,\ \hat{\beta}=0.01822,\ \hat{\gamma}=31.3990,\ \hat{\mu}=$-63.0440, $\hat{D}=380.03$.

As Figure 15 shows, the fitted curve suggests a clear bimodality similar to the original data. To see if this new model with a center parameter outperforms other models, we need to calculate model selection criteria to select the most appropriate model.

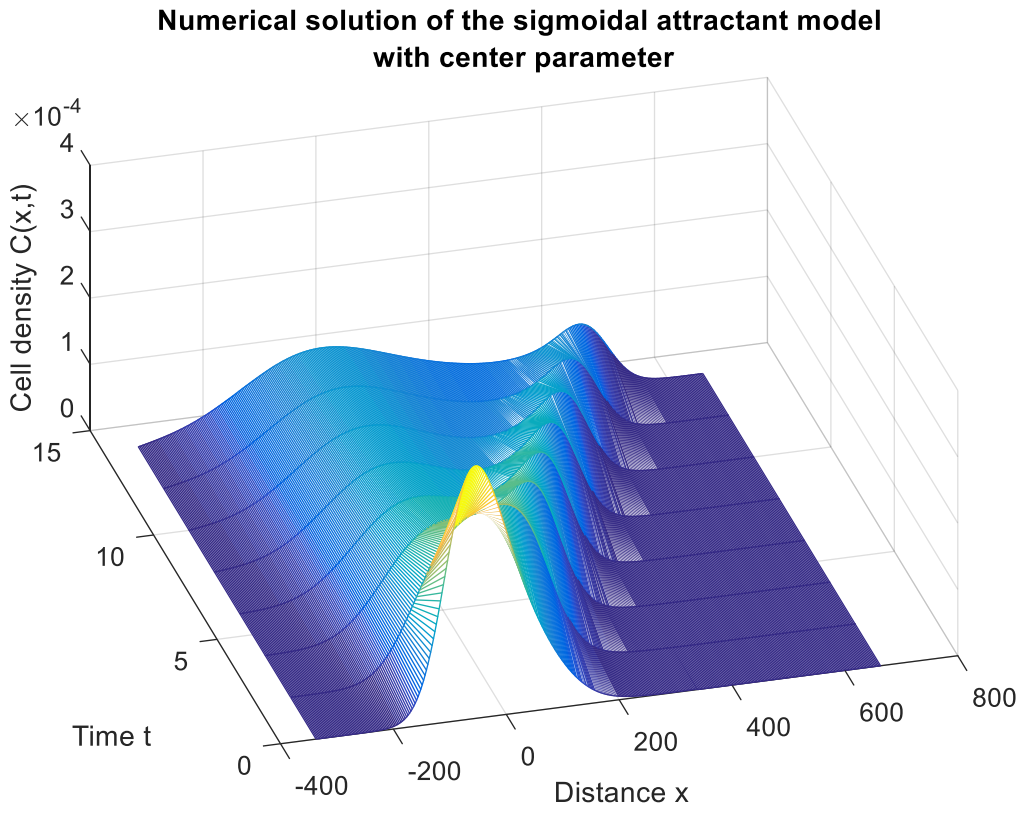


Figure 15. The plot of the sigmoidal attractant model with a new "center" parameter, with parameters being optimized using Nelder-Mead algorithm. The biggest difference of this plot is that the cell density is bimodal.

After the models are fitted, the likelihood of each model is calculated to determine the most appropriate model. As is stated in section 2.2, the function $p(x,t)$ represents the probability density of cells. So the log-likelihood can be calculated as follows.

With a certain model and covariance function, the parameters are estimated using gradient matching. Then, for each data point (x_i, t_i) in the original dataset, the fitted density $\hat{p}(x_i, t_i)$ is calculated by numerically solving the corresponding PDE with the optimized parameters. Afterwards, the log-likelihood is calculated with the equation:

$$\text{log-likelihood} = \sum_{i=1}^{n} \log\left(\hat{p}(x_i, t_i)\right). \quad (4.12)$$

After calculating the log-likelihoods, the AIC and BIC criteria are also derived using

equation (4.13) and (4.14) for model selection,

$$AIC = 2k - 2\log(\hat{L}) \quad (4.13)$$

$$BIC = \log(n)k - 2\log(\hat{L}) \quad (4.14)$$

where k is the number of parameters, n is the number of data points ($n = 325$ in this study) and $\hat{L}$ is the maximum of likelihood function. The results are listed in Table 3. From the AIC and BIC criteria, we can conclude that the sigmoidal attractant model with the "center" parameter is the most appropriate one in modelling the cell movement among all the candidates.

model	no. of parameters	kernel	log likelihood	AIC	BIC
constant advection	2	sexp	-3289.37	6582.746	6590.314
		matern32	-3271.25	6546.493	6554.061
		matern52	-3274.35	6552.708	6560.275
sigmoidal attractant	4	sexp	-3113.73	6235.46	6250.596
		matern32	-3137.16	6282.314	6297.449
		matern52	-3151.08	6310.164	6325.299
sigmoidal attractant with center inferred	5	sexp	-3014.59	6039.171(*)	6058.09(*)
		matern32	-3414.66	6839.314	6858.233
		matern52	-3027.23	6064.453	6083.372

Table 3. The model selection criteria of three models used in this study and three different kernels for each model. (*)'s indicate the minimal values of both criteria. From the table both AIC and BIC criteria suggest that the sigmoidal attractant model with a "center" parameter inferred with the covariance function of logistic Gaussian process being the squared exponential kernel is the most appropriate model among all the candidates.

5. Discussion

In this study, we have mainly inspected the method of gradient matching in estimating the parameters of partial differential equations. The statistical methods of Gaussian process and logistic Gaussian process are used to smooth the data and derive the partial derivatives. Afterwards the point estimates of PDEs are derived by minimizing the loss function. For the simulated datasets, the point estimates are compared with the real values of parameters, and for the cell movement dataset, the PDEs are numerically solved with the estimated parameters. The AIC and BIC criteria for different PDE models are calculated to select the most appropriate model to describe the dataset. Due to the time limitation, there are many questions that remain unsolved and many improvements that can be made in this study and here is a brief discussion of them.

Since the main purpose of introducing gradient matching is to reduce computational complexity, we will first focus on the computational cost of the gradient matching method. Since we do not use Bayesian inference or MCMC sampling, the main source of the computational cost comes from the fitting of the Gaussian processes, the complexity of which is asymptotically $O(N^3)$, where N is the number of data points, and for N no more than 1000, the hyper-parameters of GPs can be inferred by maximum-likelihood estimation within a few seconds using a personal laptop. So the gradient matching method does reduce the computational cost to a great extent. Besides, with properly selected covariance function and region of space and time, the gradient matching method can give point estimates very close to real values using noise-free simulated data. But for data subject to noise, the IQRs of the boxplots parameter estimation do not always contain the real value, which means that this method may not be robust with respect to noise.

One of the disadvantage of the gradient matching procedure used in this study, as stated in Dondelinger et al. (2013), comes from the two-step structure of the method. As we showed in section 2, the hyper-parameters of the GPs we are trying to fit are inferred based on the data alone, without receiving any feedbacks from the partial derivatives,

which makes the results of the parameter inference highly dependent on the quality of the GPs. An example is the attempt to fit the simulated data into a GP with neural network covariance function made in section 4.2. In that case, the predicted mean value from the GP is close to the original data, but the partial derivatives of the GP are not close to the partial derivatives of the real function (see Figure 7). Because of the lack of the feedback, we cannot detect this problem until results that are far away from the true values are made with the inferred hyper-parameters.

A better approach of regularizing the interpolants by the differential equations themselves is first suggested in Ramsay et al. (2007). Based on the approach, Dondelinger et al. (2013) developed the method of adaptive gradient matching based on ODE models using non-parametric Bayesian approach and Gaussian process. In their study, both the hyper-parameters of the GP and the ODE parameters can be jointly inferred from the joint posterior distribution. The method is called "adaptive" because the GP is adapted based on the information feedback from the differential equation systems. As the authors stated, this method can significantly improve the robustness with respect to noise. Due to the limited time, we do not perform the Bayesian inference in this study, but in future studies, we can impose the method of adaptive gradient matching on PDE models and compare with the results from this study to show if the approach works for PDE models as well.

Because of the fact that the likelihood function is not unimodal and has multiple local minima, the parameter optimization would become an NP-hard problem [10], and the sampling procedure in Bayesian inference would be likely to fall in local optima and take a long term to converge. The method of parallel tempering is proposed to tackle the problem of local optima [3][5]. The idea is to run multiple MCMC simulations at different levels of the likelihood in parallel. The power posterior at certain level is

$$p_{\beta^{(j)}}(\theta^{(j)} \mid y) \propto p(\theta^{(j)})\, p(y \mid \theta^{(j)})^{\beta^{(j)}}, \quad (5.1)$$

where $\beta^{(j)}$ is the j-th level. When $\beta^{(j)}$ is low, the posterior is close to the prior which is usually flat, so the parameter positon from sampling can easily move across the region.

As $\beta^{(j)}$ becomes larger, the posterior becomes more rugged. At every MCMC step, two chains of different levels of likelihood are chosen and the parameter locations are swapped. In this way, it is easier for the parameter locations in a MCMC chain to move from the local optimum to somewhere else, and thus makes the sampling easier to converge. The strengths and weaknesses of parallel tempering in gradient matching with ODE models are reviewed in [10] and in future studies, the method can also be used on inferences of PDE models.

As for the model selection in section 4.3, from Table 3 we can see that for the constant advection coefficient model and the sigmoidal attractant model, the AICs and BICs calculated from different covariance functions in the logistic GP are relatively close. However, for the sigmoidal attractant model with center parameter, the AIC and BIC for Matérn32 kernel are large, indicating that using Matérn32 kernel to infer the parameters of this model may not be appropriate. A possible explanation is that the Matérn32 kernel is only first-order differentiable and mismatch might take place as we fit this kernel to a PDE which contains second-order partial derivatives. But generally speaking, we would recommend trying different covariance functions when doing PDE model selection with gradient matching and hopefully this will help to find the most appropriate model and corresponding parameters.

Due to the time limitation, only two fundamental PDE models for chemotaxis and a modification for one of them are examined in this study. Hillen & Painter (2009) provided an overview of past results on modelling the cell movement mechanisms using PDE models. In the future, we can also work on estimating the parameters of these more complicated models using gradient matching and see how the method works.

References

1. Atkins, P. W. (1987). *Physical Chemistry. Third edition.* Oxford press

2. Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer.

3. Calderhead B., Girolami M. (2009). Estimating Bayes factors via thermodynamic integration and population MCMC. *Comput. Stat. Data Anal.* 53, 4028–4045.10.1016/j.csda.2009.07.025

4. Calderhead, B., Girolami, M., & Lawrence, N. D. (2009). Accelerating Bayesian inference over nonlinear differential equations with Gaussian processes. In *Advances in neural information processing systems* (pp. 217-224).

5. Campbell D., Steele R. (2012). Smooth functional tempering for nonlinear differential equation models. *Comput. Stat*. 22, 429–443.10.1007/s11222-011-9234-3

6. Dondelinger F., Filippone M., Rogers S., Husmeier D. (2013). "ODE parameter inference using adaptive gradient matching with Gaussian processes," in *Journal of Machine Learning Research Workshop and Conference Proceedings (JMLR WCP)*: The 16th International Conference on Artificial Intelligence and Statistics (AISTATS), Vol. 31, 216–228.

7. Ferguson, E. A., Matthiopoulos, J., Insall, R. H., & Husmeier, D. (2016). Inference of the drivers of collective movement in two cell types: Dictyostelium and melanoma. *Journal of The Royal Society Interface*, *13*(123), 20160695.

8. Hillen, T., & Painter, K. J. (2009). A user's guide to PDE models for chemotaxis. *Journal of mathematical biology*, *58*(1-2), 183.

9. Macdonald B., Husmeier D. (2015). "Computational inference in systems biology," in *Bioinformatics and Biomedical Engineering: Third International Conference, IWBBIO 2015. Proceedings, Part II. Series: Lecture Notes in Computer Science* (9044) eds. Ortuño F., Rojas I., editors. (Granada: Springer;), 276–288.

10. Macdonald, B., & Husmeier, D. (2015). Gradient Matching Methods for Computational Inference in Mechanistic Models for Systems Biology: A Review and Comparative Analysis. *Frontiers in Bioengineering and Biotechnology*, *3*, 180. http://doi.org/10.3389/fbioe.2015.00180

11. Ramsay J., Hooker G., Campbell D., Cao J. (2007). Parameter estimation for differential equations: a generalized smoothing approach. J. R. Stat. Soc. Series B Stat. Methodol. 69, 741–796.10.1111/j.1467-9868.2007.00610.x

12. Rasmussen C., Williams C. (2006). Gaussian Processes for Machine Learning. Cambridge: MIT Press.

13. Riihimäki, J., & Vehtari, A. (2014). Laplace approximation for logistic Gaussian process density estimation and regression. *Bayesian analysis*, *9*(2), 425-448.

14. Vanhatalo, J., Riihimäki, J., Hartikainen, J., Jylänki, P., Tolvanen, V., & Vehtari, A. (2013). GPstuff: Bayesian modeling with Gaussian processes. *Journal of Machine Learning Research*, *14*(Apr), 1175-1179.

15. Vinga, S. (2004). Convolution integrals of normal distribution functions.

Appendix

R code (see readme.txt for more information)

PDE_GP3.r

```
kernel <- function(X,theta){
  n <- ncol(X)
  sigma.f <- theta[1]
  l1 <- theta[2]
  l2 <- theta[3]
  sigma<-theta[4]
  result <- matrix(0,ncol=n,nrow=n)
  result1 <- matrix(0,ncol=n,nrow=n)
  result2 <- matrix(0,ncol=n,nrow=n)

  result1 <- distance.p(X[1,])
  result2 <- distance.p(X[2,])

  result<-sigma^2*diag(1,nrow=n) + exp(-result1/(2*l1^2)-result2/(2*l2^2))*sigma.f^2
  return(result)
}

distance.p <- function(X){
  n<-length(X)
  matrix(rep(X^2,n),nrow=n,byrow = FALSE) + matrix(rep(X^2,n),nrow=n,byrow = TRUE) -
2*matrix(X,ncol=1)%*%matrix(X,nrow=1)
}
distance.s <- function(X){
  n<-length(X)
  matrix(rep(X,n),nrow=n,byrow = FALSE) - matrix(rep(X,n),nrow=n,byrow = TRUE)
}

kernel.nf <- function(X,theta){
  n <- ncol(X)
  sigma.f <- theta[1]
  l1 <- theta[2]
  l2 <- theta[3]
  result <- matrix(0,ncol=n,nrow=n)
  result1 <- matrix(0,ncol=n,nrow=n)
  result2 <- matrix(0,ncol=n,nrow=n)

  result1 <- distance.p(X[1,])
  result2 <- distance.p(X[2,])
```

```
  result<-exp(-result1/(2*l1^2)-result2/(2*l2^2))*sigma.f^2
  return(result)
}
cov.Yxx.Y <- function(X,theta){
  n <- ncol(X)
  sigma.f <- theta[1]
  l1 <- theta[2]
  l2 <- theta[3]
  sigma<-theta[4]
  result <- -kernel.nf(X,theta)/l1^2+kernel.nf(X,theta)*distance.p(X[1,])/l1^4
  return(result)
}

cov.Yx.Y <- function(X,theta){
  n <- ncol(X)
  sigma.f <- theta[1]
  l1 <- theta[2]
  l2 <- theta[3]
  sigma<-theta[4]

  result <- kernel.nf(X,theta) * distance.s(X[1,]) /(-l1^2)
  return(result)
}
cov.Yxx.Yxx <- function(X,theta){
  n <- ncol(X)
  sigma.f <- theta[1]
  l1 <- theta[2]
  l2 <- theta[3]
  sigma<-theta[4]
  result <- matrix(0,ncol=n,nrow=n)
  for (i in 1:n){
    for (j in 1:n){
      result[i,j] <- sigma.f^2*exp(-(X[1,i]-X[1,j])^2/(2*l1^2)-(X[2,i]-X[2,j])^2/(2*l2^2))*(3*l1^-4-
6*(X[1,i]-X[1,j])^2*l1^-6+(X[1,i]-X[1,j])^4*l1^-8)
    }
  }
  return(result)
}
cov.Yt.Y <- function(X,theta){
  n <- ncol(X)
  sigma.f <- theta[1]
  l1 <- theta[2]
  l2 <- theta[3]
```

```
  sigma<-theta[4]
  result <- kernel.nf(X,theta) * distance.s(X[2,]) /(-l2^2)
  return(result)
}
cov.Yt.Yt <- function(X,theta){
  n <- ncol(X)
  sigma.f <- theta[1]
  l1 <- theta[2]
  l2 <- theta[3]
  sigma<-theta[4]
  result <- matrix(0,ncol=n,nrow=n)
  for (i in 1:n){
    for (j in 1:n){
      result[i,j] <- sigma.f^2*exp(-(X[1,i]-X[1,j])^2/(2*l1^2)-(X[2,i]-X[2,j])^2/(2*l2^2))*(l2^-2-
(X[2,i]-X[2,j])^2*l2^-4)
    }
  }
  return(result)
}
cov.Yxx.Yt <- function(X,theta){
  n <- ncol(X)
  sigma.f <- theta[1]
  l1 <- theta[2]
  l2 <- theta[3]
  sigma<-theta[4]
  result <- matrix(0,ncol=n,nrow=n)
  for (i in 1:n){
    for (j in 1:n){
      result[i,j] <- sigma.f^2*exp(-(X[1,i]-X[1,j])^2/(2*l1^2)-(X[2,i]-X[2,j])^2/(2*l2^2))*(-l1^-
2+(X[1,j]-X[1,i])^2*l1^-4)*(X[2,j]-X[2,i])*l2^-2
    }
  }
  return(result)
}


m1 <- function(X,theta,SC2,cgp)
  (cov.Yt.Y(X,theta))%*%SC2%*%cgp      #Question: is the expression(11) in the paper correct?
K1 <- function(X,theta)
  cov.Yt.Yt(X,theta)-cov.Yt.Y(X,theta)%*%SC2%*%t(cov.Yt.Y(X,theta))
m2 <- function(X,theta,SC2,cgp)
  cov.Yxx.Y(X,theta)%*%SC2%*%cgp
K2 <- function(X,theta)
  cov.Yxx.Yxx(X,theta)-cov.Yxx.Y(X,theta)%*%SC2%*%t(cov.Yxx.Y(X,theta))
```

```
plot(m1(X,thetahat),m2(X,thetahat))
x
x^2>2*D*t
m3 <- function(X,theta,SC2,cgp)
  (rbind(cov.Yxx.Y(X,theta),cov.Yt.Y(X,theta)))%*%SC2%*%cgp
K3 <- function(X,theta)

cbind(rbind(cov.Yxx.Yxx(X,theta),cov.Yxx.Yt(X,theta)),rbind(t(cov.Yxx.Yt(X,theta)),cov.Yt.Yt(X,
theta)))-
rbind(cov.Yxx.Y(X,theta),cov.Yt.Y(X,theta))%*%SC2%*%t(rbind(cov.Yxx.Y(X,theta),cov.Yt.Y(X,t
heta)))
```

A-D.r

```
generate.data.A_D.plural <- function(D,u,SNR,sample,type="1"){
  M <- 10
  x<-matrix(0,300,sample)
  t<-matrix(0,300,sample)
  c<-matrix(0,300,sample)
  y1<-matrix(0,300,sample)
  y2<-matrix(0,300,sample)
  y3<-matrix(0,300,sample)
  var.noise<-numeric(3)
  if (type=="1"){
    for (i in 1:sample){
      x[,i] <- runif(600,0,2)
      t[,i] <- runif(600,0.3,2)
      #X <- rbind(x,t)

      c[,i] <- M/sqrt(4*pi*D*t[,i])*exp(-(x[,i]-u*t[,i])^2/(4*D*t[,i]))
      #plot(x,t)
      var.noise <- SNR^-1*var(c[,i])
      y1[,i] <- c[,i] + rnorm(600,0,sqrt(var.noise[1]))
      y2[,i] <- c[,i] + rnorm(600,0,sqrt(var.noise[2]))
      y3[,i] <- c[,i] + rnorm(600,0,sqrt(var.noise[3]))
    }
  }
  else if (type=="2"){
    for (i in 1:sample){
      x[,i] <- runif(600,0,2)
      t[,i] <- runif(600,0.1,2)
      #X <- rbind(x,t)

      c[,i] <- M/sqrt(4*pi*D*t[,i])*exp(-(x[,i]-u*t[,i])^2/(4*D*t[,i]))
```

```
        #plot(x,t)
        var.noise <- SNR^-1*var(c[,i])
        y1[,i] <- c[,i] + rnorm(600,0,sqrt(var.noise[1]))
        y2[,i] <- c[,i] + rnorm(600,0,sqrt(var.noise[2]))
        y3[,i] <- c[,i] + rnorm(600,0,sqrt(var.noise[3]))
      }
  }
  else if (type=="3"){
    for (i in 1:sample){
      x[,i] <- runif(300,0,2)
      t[,i] <- runif(300,0,2)
      #X <- rbind(x,t)

      c[,i] <- M/sqrt(4*pi*D*t[,i])*exp(-(x[,i]-u*t[,i])^2/(4*D*t[,i]))
      #plot(x,t)
      var.noise <- SNR^-1*var(c[,i])
      y1[,i] <- c[,i] + rnorm(300,0,sqrt(var.noise[1]))
      y2[,i] <- c[,i] + rnorm(300,0,sqrt(var.noise[2]))
      y3[,i] <- c[,i] + rnorm(300,0,sqrt(var.noise[3]))
    }
  }
  else if (type=="special"){
    x<-matrix(0,600,sample)
    t<-matrix(0,600,sample)
    c<-matrix(0,600,sample)
    y1<-matrix(0,600,sample)
    y2<-matrix(0,600,sample)
    y3<-matrix(0,600,sample)
    for (i in 1:sample){
      x[,i] <- c(runif(600,0,2))
      t[,i] <- c(runif(300,0,0.5),runif(300,0.5,2))
      #X <- rbind(x,t)

      c[,i] <- M/sqrt(4*pi*D*t[,i])*exp(-(x[,i]-u*t[,i])^2/(4*D*t[,i]))
      #plot(x,t)
      var.noise <- SNR^-1*var(c[,i])
      y1[,i] <- c[,i] + rnorm(600,0,sqrt(var.noise[1]))
      y2[,i] <- c[,i] + rnorm(600,0,sqrt(var.noise[2]))
      y3[,i] <- c[,i] + rnorm(600,0,sqrt(var.noise[3]))
    }
  }
  return(list(x=x,t=t,y1=y1,y2=y2,y3=y3,c=c))
}
#########################################################################
```

```
dat_1<-generate.data.A_D.plural(2,3,c(100,30,10),100,type = "1")
x<-dat_1$x;t<-dat_1$t;y1<-dat_1$y1;c<-dat_1$c;y2<-dat_1$y2;y3<-dat_1$y3
write.csv(x,"x_1.csv");write.csv(t,"t_1.csv");write.csv(y1,"y1_1.csv");write.csv(c,"c_1.csv");write.csv
(y2,"y2_1.csv");write.csv(y3,"y3_1.csv")

dat_2<-generate.data.A_D.plural(2,3,c(100,30,10),100,type = "2")
x<-dat_2$x;t<-dat_2$t;y1<-dat_2$y1;c<-dat_2$c;y2<-dat_2$y2;y3<-dat_2$y3
write.csv(x,"x_2.csv");write.csv(t,"t_2.csv");write.csv(y1,"y1_2.csv");write.csv(c,"c_2.csv");write.c
sv(y2,"y2_2.csv");write.csv(y3,"y3_2.csv")

dat_3<-generate.data.A_D.plural(2,3,c(100,30,10),100,type = "3")
x<-dat_3$x;t<-dat_3$t;y1<-dat_3$y1;c<-dat_3$c;y2<-dat_3$y2;y3<-dat_3$y3
write.csv(x,"x_3.csv");write.csv(t,"t_3.csv");write.csv(y1,"y1_3.csv");write.csv(c,"c_3.csv");write.cs
v(y2,"y2_3.csv");write.csv(y3,"y3_3.csv")

dat_sp <- generate.data.A_D.plural(2,3,c(100,30,10),100,type = "special")
x<-dat_sp$x;t<-dat_sp$t;y1<-dat_sp$y1;c<-dat_sp$c;y2<-dat_sp$y2;y3<-dat_sp$y3
write.csv(x,"x_sp.csv");write.csv(t,"t_sp.csv");write.csv(y1,"y1_sp.csv");write.csv(c,"c_sp.csv");wri
te.csv(y2,"y2_sp.csv");write.csv(y3,"y3_sp.csv")

##
xt1 <- seq(0,2,0.05)
xt2 <-seq(0,5,0.05)
cxt<-matrix(0,101,41)
for (i in 1:101){
  for(j in 1:41){

    cxt[i,j] <- 10/sqrt(4*pi*-2*xt2[i])*exp(-(xt1[j]-3*xt2[i])^2/(4*-2*xt2[i]))
  }
}
write.csv(cxt,"cxt2.csv",row.names = FALSE)
###
GM <- function(filename,noise.level,type){
  input<-read.csv(filename,header = FALSE)
  D_hat<-numeric(100)
  u_hat<-numeric(100)
  if (type=="1"){
    x<-dat_1$x;t<-dat_1$t;y1<-dat_1$y1;c<-dat_1$c;y2<-dat_1$y2;y3<-dat_1$y3
  }  else if (type=="2"){
    x<-dat_2$x;t<-dat_2$t;y1<-dat_2$y1;c<-dat_2$c;y2<-dat_2$y2;y3<-dat_2$y3
  }  else if (type=="3"){
    x<-dat_3$x;t<-dat_3$t;y1<-dat_3$y1;c<-dat_3$c;y2<-dat_3$y2;y3<-dat_3$y3
  }  else if (type=="special"){
    x<-dat_sp$x;t<-dat_sp$t;y1<-dat_sp$y1;c<-dat_sp$c;y2<-dat_sp$y2;y3<-dat_sp$y3
```

```
  }

  for (i in 1:100){
    print(i)
    thetahat <- as.vector(input[,i])^c(0.5,1,1,0.5)
    K2<-kernel(rbind(x[,i],t[,i]),thetahat)
    Knf <- kernel.nf(rbind(x[,i],t[,i]),thetahat)
    #if (noise.level==0)
    #   SC2 <-solve(Knf)
    #else
    #   SC2 <- solve(K2)
    SC2 <- solve(K2)
    if (noise.level==0)
      cgp<-c[,i]
    else if (noise.level==1)
      cgp<-Knf%*%SC2%*%y1[,i]
    else if (noise.level==2)
      cgp<-Knf%*%SC2%*%y2[,i]
    else if (noise.level==3)
      cgp<-Knf%*%SC2%*%y3[,i]
    #temp<-m3(rbind(x[,i],t[,i]),thetahat,SC2,cgp)
    cgp_t<-m1(rbind(x[,i],t[,i]),thetahat,SC2,cgp)
    cgp_x2<-m2(rbind(x[,i],t[,i]),thetahat,SC2,cgp)
    cgp_x <- cov.Yx.Y(rbind(x[,i],t[,i]),thetahat)%*%SC2%*%cgp
    D_hat[i]<-lm(cgp_t~cgp_x2+cgp_x-1)$coef[1]
    u_hat[i]<- -lm(cgp_t~cgp_x2+cgp_x-1)$coef[2]
  }
  return(cbind(D_hat,u_hat))
}

boxplot(Est_c_1)

Est_c_11 <- GM("theta_c_1.csv",0,"1")
Est_y1_11 <- GM("theta_y1_1.csv",1,"1")
Est_y2_11<- GM("theta_y2_1.csv",2,"1")
Est_y3_11 <- GM("theta_y3_1.csv",3,"1")

Est_c_21 <- GM("theta_c_2.csv",0,"2")
Est_y1_21 <- GM("theta_y1_2.csv",1,"2")
Est_y2_21 <- GM("theta_y2_2.csv",2,"2")
Est_y3_21 <- GM("theta_y3_2.csv",3,"2")

Est_c_31 <- GM("theta_c_3.csv",0,"3")
Est_y1_31 <- GM("theta_y1_3.csv",1,"3")
```

```
Est_y2_31 <- GM("theta_y2_3.csv",2,"3")
Est_y3_31 <- GM("theta_y3_3.csv",3,"3")

Est_c_sp <- GM("theta_c_sp.csv",0,"special")
Est_y1_sp <- GM("theta_y1_sp.csv",1,"special")
Est_y2_sp <- GM("theta_y2_sp.csv",2,"special")
Est_y3_sp <- GM("theta_y3_sp.csv",3,"special")
boxplot(Est_c_31[,1],Est_y1_31[,1],Est_y2_31[,1],Est_y3_31[,1],
        Est_c_sp[,1],Est_y1_sp[,1],Est_y2_sp[,1],Est_y3_sp[,1],
        names = rep(c("noise free","SNR=100","SNR=30","SNR=10"),2),main="Point estimates
of diffusion coefficients from different sampling strategies")
abline(h=2,v=4.5,lty=c(2,1))
text(x=c(2.5,6.5),y=-7,labels=c("sampling uniformly from the space","sampling more points
around the peak"))
boxplot(Est_c_31[,2],Est_y1_31[,2],Est_y2_31[,2],Est_y3_31[,2],
        Est_c_sp[,2],Est_y1_sp[,2],Est_y2_sp[,2],Est_y3_sp[,2],
        names = rep(c("noise free","SNR=100","SNR=30","SNR=10"),2),main="Point estimates
of advection coefficients from different sampling strategies")
abline(h=3,v=4.5,lty=c(2,1))
text(x=c(2.5,6.5),y=-30,labels=c("sampling uniformly from the space","sampling more points
around the peak"))
boxplot(Est_c_31[,1],Est_y1_31[,1],Est_y2_31[,1],Est_y3_31[,1],
        Est_c_21[,1],Est_y1_21[,1],Est_y2_21[,1],Est_y3_21[,1],
        Est_c_11[,1],Est_y1_11[,1],Est_y2_11[,1],Est_y3_11[,1],ylim=c(-5,5),
        names = rep(c("noise free","SNR=100","SNR=30","SNR=10"),3),main="Point estimates
of diffusion coefficients from different space and time region")
abline(h=2,lty=2)
abline(v=c(4.5,8.5))
text(x=c(2.5,6.5,10.5),y=-3,labels=c("x∈(0,2),t∈(0,2)", "x∈(0,2),t∈(0.1,2)", "x∈(0,2),t∈
(0.3,2)"))

boxplot(Est_c_31[,2],Est_y1_31[,2],Est_y2_31[,2],Est_y3_31[,2],
        Est_c_21[,2],Est_y1_21[,2],Est_y2_21[,2],Est_y3_21[,2],
        Est_c_11[,2],Est_y1_11[,2],Est_y2_11[,2],Est_y3_11[,2],ylim=c(-5,10),
        names = rep(c("noise free","SNR=100","SNR=30","SNR=10"),3),main="Point estimates
of advection coefficients from different space and time region")
abline(h=3,lty=2)
abline(v=c(4.5,8.5))
text(x=c(2.5,6.5,10.5),y=8,labels=c("x∈(0,2),t∈(0,2)", "x∈(0,2),t∈(0.1,2)", "x∈(0,2),t∈
(0.3,2)"))


boxplot(Est_c_11[,2],Est_y1_11[,2],Est_y2_11[,2],Est_y3_11[,2],Est_c_21[,2],Est_y1_21[,2],Est_y2_21[,
2],Est_y3_21[,2])#,Est_y1_31[,2],Est_y2_31[,2],Est_y3_31[,2],ylim=c(-10,10)
```

```
abline(h=3,lty=2)

boxplot(Est_c_1[,1],Est_y1_1[,1],Est_y2_1[,1],Est_y3_1[,1],Est_c_2[,1],Est_y1_2[,1],Est_y2_2[,1],Est_y3_2[,1],names = rep(c("noise free","SNR=100","SNR=30","SNR=10"),2),main="Point estimates of Diffusion coefficients")
abline(h=2,lty=2)
legend("topleft",legend = c("t=[0.3,2]","x=[0,2]"));legend("topright",legend = c("t=[0,2]","x=[0.3,2]"))
boxplot(Est_c_1[,2],Est_y1_1[,2],Est_y2_1[,2],Est_y3_1[,2],Est_c_2[,2],Est_y1_2[,2],Est_y2_2[,2],Est_y3_2[,2],names = rep(c("noise free","SNR=100","SNR=30","SNR=10"),2),main="Point estimates of Advection coefficients")
abline(h=3,lty=2)
legend("topleft",legend = c("t=[0.3,2]","x=[0,2]"));legend("topright",legend = c("t=[0,2]","x=[0.3,2]"))
```

GM_closedfrom.r

```
l <- 10
xt1 <- seq(0,2,length.out = l)
xt2 <-seq(0.3,5,length.out = l)
cxt<-matrix(0,l,l)
for (i in 1:l){
   for(j in 1:l){

      cxt[i,j] <- 10/sqrt(4*pi*2*xt2[i])*exp(-(xt1[j]-3*xt2[i])^2/(4*2*xt2[i]))
   }
}
a<-xt1[2]-xt1[1]
b<-xt2[2]-xt2[1]
cxt.noise <- cxt+matrix(rnorm(l^2,0,sqrt(var(as.numeric(cxt))/1000)),l)
gradient <- function(cxt,i,j,a,b){
   nx <- ncol(cxt)
   nt<- nrow(cxt)


   dcdt<- (cxt[i+1,j]-cxt[i-1,j])/(2*b)
   dcdx<- (cxt[i,j+1]-cxt[i,j-1])/(2*a)
   d2cdx2 <- (cxt[i,j+1]+cxt[i,j-1]-2*cxt[i,j])/(a^2)
   return(list(dcdt,dcdx,d2cdx2))
}
gradient2 <- function(cxt,a,b){
   nx <- ncol(cxt)
   nt<- nrow(cxt)
   dcdt <- matrix(NaN,nt,nx)
   dcdx <- matrix(NaN,nt,nx)
```

```
  d2cdx2 <- matrix(NaN,nt,nx)
  for (i in 2:(nt-1)){
    for (j in 2:(nx-1)){
      result<-gradient(cxt,i,j,a,b)
      dcdt[i,j]<-result[[1]]
      dcdx[i,j]<-result[[2]]
      d2cdx2[i,j]<-result[[3]]
    }
  }
  return(list(dcdt=dcdt,dcdx=dcdx,d2cdx2=d2cdx2))
}
re<-gradient2(cxt,a,b)
re$dcdt[10,2]+3*re$dcdx[10,2]-2*re$d2cdx2[10,2]
re2<-gradient2(cxt.noise,a,b)
re2$dcdt[10,20]+3*re2$dcdx[10,20]-2*re2$d2cdx2[10,20]
dcdt<-numeric((l-2)^2)
for (i in 1:(l-2)){
  dcdt[((l-2)*i-(l-3)):((l-2)*i)]<-re2$dcdt[i+1,2:(l-1)]
}
dcdx<-numeric((l-2)^2)
for (i in 1:(l-2)){
  dcdx[((l-2)*i-(l-3)):((l-2)*i)]<-re2$dcdx[i+1,2:(l-1)]
}
d2cdx2<-numeric((l-2)^2)
for (i in 1:(l-2)){
  d2cdx2[((l-2)*i-(l-3)):((l-2)*i)]<-re2$d2cdx2[i+1,2:(l-1)]
}

summary(lm(dcdt~dcdx+d2cdx2-1))
plot(dcdt[-(1:8)],dcdx[-(1:8)])

fittedgp <- read.csv("fittedgp.csv",header = FALSE)
plot((0:200)/100,as.numeric(fittedgp[180,]),type="l")
gr.fittedgp <- gradient2(fittedgp,0.01,0.01)
summary(lm(as.numeric(gr.fittedgp$dcdt)~as.numeric(gr.fittedgp$dcdx)+as.numeric(gr.fittedg
p$d2cdx2)-1))
gr.fittedgp$d2cdx2[200,150]

fittedgpsexp<-read.csv("fittedgpsexp.csv",header=FALSE)
plot(as.numeric(fittedgpsexp[180,]),type="l")
gr.fittedgpsexp<-gradient2(fittedgpsexp,0.01,0.01)
summary(lm(as.numeric(gr.fittedgpsexp$dcdt[-(1:30),])~as.numeric(gr.fittedgpsexp$dcdx[-
(1:30),])+as.numeric(gr.fittedgpsexp$d2cdx2[-(1:30),])-1))
plot(as.numeric(gr.fittedgpsexp$dcdt),as.numeric(gr.fittedgpsexp$d2cdx2))
```

```
par(mfcol=c(1,1))
plot((0:200)/100,as.numeric(fittedgp[181,]),type="l",xlab="Position x",ylab = "Function
values",lwd=2,lty=2,col=2)

lines((0:200)/100,as.numeric(fittedgpsexp[181,]),lwd=2,lty=3,col=4)

cxt_t_1.8 <- 10/sqrt(4*pi*2*1.8)*exp(-((0:200)/100-3*1.8)^2/(4*2*1.8))
lines((0:200)/100,cxt_t_1.8,lwd=1)
legend("topleft",legend=c("closed form solution","GP with squared exponential kernel","GP
with MLP kernel"),lty=c(1,3,2),col=c(1,4,2))

d.fittedgp <- rep(NA,201)
d.fittedsecp <- rep(NA,201)
d.pde <- rep(NA,201)
for (i in 2:200){
   d.fittedsecp[i] <- (fittedgpsexp[181,i+1]-fittedgpsexp[181,i-1])/0.02
   d.fittedgp[i] <- (fittedgp[181,i+1]-fittedgp[181,i-1])/0.02
   d.pde[i] <- (cxt_t_1.8[i+1]-cxt_t_1.8[i-1])/0.02
}
plot((1:199)/100,d.pde[2:200],type="l",xlab="Position x",ylab = "Partial derivatives",ylim=c(-1,1))
lines((1:199)/100,d.fittedsecp[2:200],lwd=2,lty=3,col=4)
lines((1:199)/100,d.fittedgp[2:200],lwd=2,lty=2,col=2)
legend("topleft",legend=c("closed form solution","GP with squared exponential kernel","GP
with MLP kernel"),lty=c(1,3,2),col=c(1,4,2))

dd.fittedgp <- rep(NA,201)
dd.fittedsecp <- rep(NA,201)
dd.pde <- rep(NA,201)
for (i in 2:200){
   dd.fittedsecp[i] <- (fittedgpsexp[181,i+1]+fittedgpsexp[181,i-1]-2*fittedgpsexp[181,i])/0.01^2
   dd.fittedgp[i] <- (fittedgp[181,i+1]+fittedgp[181,i-1]-2*fittedgp[181,i])/0.01^2
   dd.pde[i] <- (cxt_t_1.8[i+1]+cxt_t_1.8[i-1]-2*cxt_t_1.8[i])/0.01^2
}

plot((1:199)/100,dd.pde[2:200],type="l",xlab="Position x",ylab = "Second-order partial
derivatives",ylim=c(-30,30))
lines((1:199)/100,dd.fittedsecp[2:200],lwd=2,lty=3,col=4)
lines((1:199)/100,dd.fittedgp[2:200],lwd=2,lty=2,col=2)
legend("topleft",legend=c("closed form solution","GP with squared exponential kernel","GP
with MLP kernel"),lty=c(1,3,2),col=c(1,4,2))
```

cell_movement.r

```
cell <- read.csv("CSE2D.csv",header=FALSE)

cell <- cell[,c(1,2,3)]
colnames(cell)<-c("x","t","p")
cell$dcdt<-NaN
cell$dcdx<-NaN
cell$d2cdx2<-NaN
a<-cell$x[2]-cell$x[1]
b<-2
for (i in 1:3200){
   if (cell$x[i]!=-100&cell$x[i]!=900&cell$t[i]!=0&cell$t[i]!=14){
      cell$dcdt[i] <- (cell$p[i+400]-cell$p[i-400])/(2*b)
      cell$dcdx[i] <- (cell$p[i+1]-cell$p[i-1])/(2*a)
      cell$d2cdx2[i] <- (cell$p[i+1]+cell$p[i-1]-2*cell$p[i])/(a^2)
   }
}
index<- !is.nan(cell$dcdt)
summary(lm(cell$dcdt[index]~cell$dcdx[index]+cell$d2cdx2[index]-1))
plot(cell$d2cdx2,cell$dcdt)

loss_cell <- function(dcdt,dcdx,d2cdx2,par){
   u <- par[1]
   D2 <- par[2]^2
   sum((dcdt+u*dcdx-D2*d2cdx2)^2)
}
optim(c(18,0),loss_cell,dcdt=cell$dcdt[index],dcdx=cell$dcdx[index],d2cdx2=cell$d2cdx2[index
],method="Nelder-Mead")
```

cell_movement_model2&3.r

```
C322D <- read.csv("C322D.csv",header=FALSE)
C522D <- read.csv("C522D.csv",header=FALSE)
CSE2D <- read.csv("CSE2D.csv",header=FALSE)
colnames(CSE2D) <- c("x","t","50","5","95")
colnames(C322D) <- c("x","t","50","5","95")
colnames(C522D) <- c("x","t","50","5","95")
sigmoidal_attractant <- function(x,t,alpha,beta,gamma){
   alpha * beta * exp( -beta *(x-gamma*t))/(1+exp(-beta*(x-gamma*t)))^2
}
CSE2D$sigmodal<-sigmoidal_attractant(CSE2D$x,CSE2D$t,1,1,1)
CSE2D$x_centered <- CSE2D$x-CSE2D$x[244]
```

```
CSE2D$large_c <- 10^5*CSE2D$`50`
C322D$large_c <- 10^5*C322D$`50`
C522D$large_c <- 10^5*C522D$`50`
plot(CSE2D$x_centered[CSE2D$t==0],CSE2D$`50`[CSE2D$t==0])
dcdt <- function(c,x,t){
  a <- x[2] -x[1]
  b<- 2
  result <- rep(NaN,length(x))
  for (i in 1:3200){
    if (i%%400!=1&i%%400!=0&t[i]!=0&t[i]!=14)
      result[i]<-(c[i+400]-c[i-400])/(2*b)
  }
  return(result)
}
dcdx(CSE2D$sigmodal,CSE2D$x,CSE2D$t)

dcdx <- function(c,x,t){
  a <- x[2] -x[1]
  b<- 2
  result <- rep(NaN,length(x))
  for (i in 1:3200){
    if (i%%400!=1&i%%400!=0&t[i]!=0&t[i]!=14)
      result[i]<-(c[i+1]-c[i-1])/(2*a)
  }
  return(result)
}
d2cdx2 <- function(c,x,t){
  a <- x[2] -x[1]
  b<- 2
  result <- rep(NaN,length(x))
  for (i in 1:3200){
    if (i%%400!=1&i%%400!=0&t[i]!=0&t[i]!=14)
      result[i]<-(c[i+1]+c[i-1]-2*c[i])/(a^2)
  }
  return(result)
}

Loss_function <- function(par,A,xx,tt){

  alpha <- par[1]
  beta<-par[2]
  gamma<-par[3]
  D<-abs(par[4])
  center<-par[5]
```

```
   c<-A
   x<-xx-center
   t<-tt
   sigmoid <- sigmoidal_attractant(x,t,alpha,beta,gamma)
   dc_dt <- dcdt(c,x,t)
   dc_dx <- dcdx(sigmoid*c,x,t)
   d2c_dx2 <- d2cdx2(c,x,t)
   return(sum((dc_dt[!is.nan(dc_dt)] + dc_dx[!is.nan(dc_dx)] - D *
d2c_dx2[!is.nan(d2c_dx2)])^2))
}
Loss_function(CSE2D$`50`,CSE2D$x,CSE2D$t,c(0,0,0,0))
par_optim <- optim(par=c(7930.70859250    , -0.01044872 , -27.50366735 ,
78),fn=Loss_function,A=CSE2D$large_c,xx=CSE2D$x,tt=CSE2D$t,method="Nelder-
Mead",control = list(maxit=10000,trace=1))

sexp_c <- optim(par=c(5723.15598817 , - 0.01822074 , -31.39842450,
75,563.04604896),fn=Loss_function,A=CSE2D$large_c,xx=CSE2D$x,tt=CSE2D$t,method="Nelder-Mead",control = list(maxit=10000,trace=1))


m32_b <- optim(par=c(971.93712923    , -0.01038666 , -27.55155552    ,
75.08944061),fn=Loss_function,A=C322D$large_c,xx=C322D$x,tt=C322D$t,method="Nelder-
Mead",control = list(maxit=10000,trace=1))

m32_c <- optim(par=c(9.108263e+03 ,-8.571173e-03 ,-2.837372e+01 , 5.993772e-05 ,
4.438400e+02),fn=Loss_function,A=C322D$large_c,xx=C322D$x,tt=C322D$t,method="Nelder-
Mead",control = list(maxit=10000,trace=1))

m52_b <- optim(par=c(930.70859250    , -0.01044872 , -7.50366735 ,
75),fn=Loss_function,A=C522D$large_c,xx=C522D$x,tt=C522D$t,method="Nelder-
Mead",control = list(maxit=10000,trace=1))

m52_c <- optim(par=c(5723.06928480    , -0.01822045,    -31.39900051,    380.03181225,
563.04398174),fn=Loss_function,A=C522D$large_c,xx=C522D$x,tt=C522D$t,method="Nelder-
Mead",control = list(maxit=10000,trace=1))
```

Likelihood_and_AIC.r

```
likelihood_grid <- data_c$V1
for (i in 1:324){
   for (j in (i+1):325){
      if (likelihood_grid[i]==likelihood_grid[j])
```

```
        likelihood_grid[j]<- -10^4
   }
}
likelihood_grid<-likelihood_grid[likelihood_grid!=-10^4]
likelihood_grid<-sort(likelihood_grid)
write.csv(likelihood_grid,"grid.csv")

likelihood_sexp <- read.csv("likelihood_sexp.csv",header = FALSE)
l_sexp_a <- read.csv("likelihood_sexp_a.csv",header = FALSE)
l_sexp_b <- read.csv("likelihood_sexp_b.csv",header = FALSE)

l_m32_a <- read.csv("likelihood_m32_a.csv",header = FALSE)
l_m32_b <- read.csv("likelihood_m32_b.csv",header = FALSE)
l_m32_c <- read.csv("likelihood_m32_c.csv",header = FALSE)

l_m52_a <- read.csv("likelihood_m52_a.csv",header = FALSE)
l_m52_b <- read.csv("likelihood_m52_b.csv",header = FALSE)
l_m52_c <- read.csv("likelihood_m52_c.csv",header = FALSE)

lgp_likelihood <- function(x,t,density){
   for (i in 1:241){
      if (abs(likelihood_grid[i]-x<10^-6))
         xi <- 243-i
   }
   ti <- 0.5*t+1
   return(abs(as.numeric(density[ti,xi])))
}

log_likelihood_sexp_c <- 0
for (i in 1:325){
   log_likelihood_sexp_c<- log_likelihood_sexp_c +
log(lgp_likelihood(x=data_c$V1[i],t=data_c$V2[i],likelihood_sexp))
}
log_likelihood_sexp_c

log_likelihood_sexp_a <- 0
for (i in 1:325){
   log_likelihood_sexp_a<- log_likelihood_sexp_a +
log(lgp_likelihood(x=data_c$V1[i],t=data_c$V2[i],l_sexp_a))
}

log_likelihood_sexp_b <- 0
for (i in 1:325){
```

```
   log_likelihood_sexp_b<- log_likelihood_sexp_b +
log(lgp_likelihood(x=data_c$V1[i],t=data_c$V2[i],l_sexp_b))
}

log_likelihood_m32_a <- 0
for (i in 1:325){
   log_likelihood_m32_a<- log_likelihood_m32_a +
log(lgp_likelihood(x=data_c$V1[i],t=data_c$V2[i],l_m32_a))
}
log_likelihood_m32_b <- 0
for (i in 1:325){
   log_likelihood_m32_b<- log_likelihood_m32_b +
log(lgp_likelihood(x=data_c$V1[i],t=data_c$V2[i],l_m32_b))
}
log_likelihood_m32_c <- 0
for (i in 1:325){
   log_likelihood_m32_c<- log_likelihood_m32_c +
log(lgp_likelihood(x=data_c$V1[i],t=data_c$V2[i],l_m32_c))
}

log_likelihood_m52_a <- 0
for (i in 1:325){
   log_likelihood_m52_a<- log_likelihood_m52_a +
log(lgp_likelihood(x=data_c$V1[i],t=data_c$V2[i],l_m52_a))
}
log_likelihood_m52_b <- 0
for (i in 1:325){
   log_likelihood_m52_b<- log_likelihood_m52_b +
log(lgp_likelihood(x=data_c$V1[i],t=data_c$V2[i],l_m52_b))
}
log_likelihood_m52_c <- 0
for (i in 1:325){
   log_likelihood_m52_c<- log_likelihood_m52_c +
log(lgp_likelihood(x=data_c$V1[i],t=data_c$V2[i],l_m52_c))
}

l_l_s <-
c(log_likelihood_sexp_a,log_likelihood_m32_a,log_likelihood_m52_a,log_likelihood_sexp_b,lo
g_likelihood_m32_b,

log_likelihood_m52_b,log_likelihood_sexp_c,log_likelihood_m32_c,log_likelihood_m52_c)
AICs <- 2*c(2,2,2,4,4,4,5,5,5)-2*l_l_s

BICs <- log(325)*c(2,2,2,4,4,4,5,5,5)-2*l_l_s
```

```
write.csv(cbind(l_l_s,AICs,BICs),"AIC.csv")
```