|epcc|

Bayesian Inference Using Sequential Monte-Carlo Algorithm for Dynamic System Models

Chaolin Han

August 18, 2020

MSc in High Performance Computing

The University of Edinburgh

Year of Presentation: 2020

Abstract

Zebrafish can regenerate the spinal cord after injury, and possibly recover swimming function. The regeneration is a complex biological process with many interacting parts. We modelled the dynamics of two cell and two molecules that participate in the regeneration, according to the existing experimental findings[1], and wished to explore uncertainties in their parameter estimates.

As an Approximate Bayesian Computation (ABC) method, ABC SMC (Sequential Monte-Carlo) method was implemented on the proposed models to approximate the posteriors the parameters. We conducted some preliminary experiments to study the ability of inference and robustness of the algorithm before we performed the inference of parameters and model comparison using the same inference framework. The result shows well-inferred parameters and a preferred model with high model probability.

The algorithm implementation shows a decent scaling-up performance; however, problems with local optimum and some other uncertainties were also found. Suggestions on the implementation options and ABC SMC settings were made to avoid these problems.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Zebrafish can regenerate its spinal cord after injury; tissue regeneration observed at the injury site is a complex biological process with interacting parts and dynamical feedback between different types of cells and molecules. These cells and molecules are involved in the inflammation and repair dynamically, and a complex interacting network can be drawn according to it.

Mathematical models and techniques can help us to resolve the complexity in this dynamic system; however, how to derive the parameter estimates of the models remains a challenging work. In this dissertation, our interest lies in the modelling of the regeneration process that happens around the lesion site, especially in the parameter estimates and comparison of possible models. According to available researches [1], a mathematical model was first proposed, but the addressed interactions remain to be precisely defined, and quantitatively examined in experiments. We want to find the best parameter sets of the model using the published data, and ranking models under specific metrics and possibly find the best-fit model. In these steps, models are expected to retain mathematical and biological significance regarding the complex interactions, and differential equations are suitable for this.

For the parameter estimation tasks, optimisation-based methods usually have trouble in determining global optima from local optimal, and noise in the observed data may lead to over-fitting. Compared to optimisation-based methods, Bayesian inference techniques can help to relieve these two concerns[2]. Moreover, approximate inference methods (Approximate Bayesian Computation, ABC) make likelihood-free inference possible, which is suitable for our task as writing down likelihood as a function expression for the dynamic system can be hard.

Such methods introduce computation-intensive sampling-and-test patterns, where a massive number of samples are drawn and followed by complex computations. As the sampling is independent of each other in a certain period, the parallel implementation of the algorithm becomes feasible and is expected to accelerate the parameter inference on a large scale when using HPC resources.

This dissertation mainly focuses on the implementations and experiments of parameter estimation using ABC SMC (sequential Monte-Carlo) method, and discussions on the results. Additionally, performance experiments illustrated the strong-scaling performance of the algorithm. Chapter 2 talks about the background information and theories, Chapter 3 shows how the mathematical models are derived from hypothesis, and Chapter 4 describes the ABC SMC implementations and experiments on the models in detail. A brief scaling-up performance experiment is included in Chapter 5, and Chapter 6 summarises the conclusions and states the future works on this topic.

Chapter 2

Background

2.1 Zebrafish spinal cord regeneration

Compared to mammals, zebrafish can regenerate the spinal cord after injury, and possibly recover swimming function. They are widely used in regenerative ability researches due to this ability. Experiments discovered that although being cut off, the axonal of nerve cells can regenerate and bridge across the lesion site, which critically determined whether zebrafish can regain the swimming function [3]. Tsarouchas et al. [1] studied the complex interactions that happen at the lesion site. The inflammation affects the regeneration: if it is promoted, it accelerates the axonal regeneration; if it is inhibited, it reduces the regeneration[1]. Further experiments showed that two kinds of cytokines, il-1β and tnf-α, are related the inflammation, and they are dynamically controlled by immunes cells, i.e. immune cells produce and control the cytokines-mediated inflammation [1]. Cytokines are small proteins, work as messengers to pass signal across different cells, which are similar to hormones and neurotransmitter.

An interaction map (Figure 2.1) can be drawn according to the evidences found in experiments.[1]. After the injury, a massive immune response was observed, with the increasing numbers of several kinds of cells. This dynamical system consists of complex interactions and dynamical feedback between cells and cytokines. For example, il-1β promotes macrophages' increase. As a primary source, more macrophages leads to more expression of tnf-α mRNA.

Consequently, tnf-α and il-1β plays determinative roles in the axonal regeneration. il-1β promotes the regeneration at early stages, but later has an inhibition effect; tnf-α promotes the regeneration all the time. The presence of the cytokines also affects the number of immune cells, while as sources, more immune cells produce more cytokines. Interactions between different types of cells are also considered in the system (e.g., the number of neutrophils is affected by macrophages).

This dynamic system is suitable for mathematical modelling, where a set of equations can be written to represent the dynamics of different objects, e.g. each type of cells and

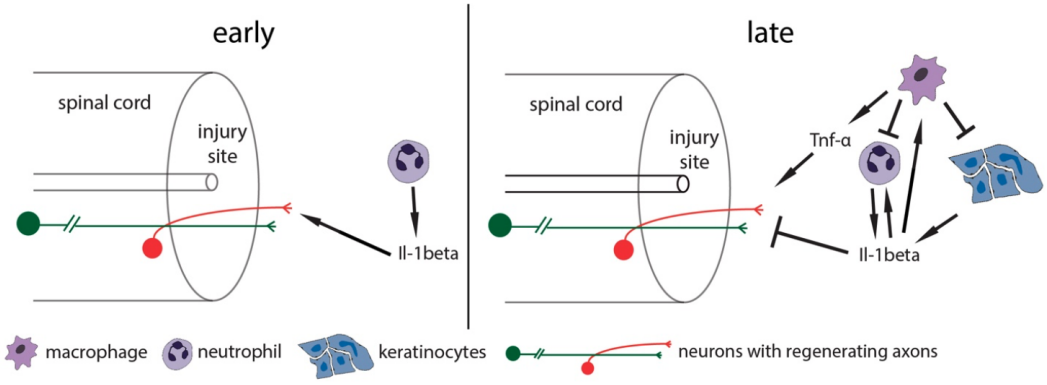

Figure 2.1: Working model of the influence of the innate immune system on axonal regrowth, from Tsarouchas et al. [1]. Lines ended with arrows represent promoting effect, lines ended with T-connectors represent inhibition

cytokines. Our interest lies in the modelling of the regeneration period of the dynamic system, and we wish to find that if we can quantitatively study the interactions inside the system, based on the proposed hypothesis [1], furthermore based our hypothesis which may include different interactions.

The biological background of works in this dissertation is mostly related to the experimental evidence, quantitative data and proposed hypothesis in [1]. The published data on the quantitative measurement of cells and cytokines make the modelling and further analysis possible. Some other interactions proposed in our modelling stages are enlightened from Anderson et al. [4], where intracellular cytokine signalling networks were established and computationally modelled.

2.2 Mathematical modelling

Mathematical models can express real-life problems in mathematical forms, and provide qualitative or quantitative inside-look of the question through various mathematical tools and techniques. They can be used for prediction, classification, testing our understanding or hypothesis and many other tasks; thus, they are widely used to solve practical problems. However, for a model that describes the zebrafish spinal cord regeneration process, we were less interested in the model's predictive functionality; the main focus is on the exploration of uncertainties in parameter estimates and comparison with alternative models. In this case, we wish to use modelling techniques and implement inference methods, test our hypothesis and hopefully find and compare well-fitted models.

According to our understanding of the biological mechanism in the regeneration, hypotheses should be proposed in a way that can represent real mechanism as much as possible. Usually, a real-life problem, e.g. the spinal cord regeneration, consists of

complex dynamics and interactions with many other environmental factors to be considered. An elaborate description of the real cases using an overly complicated mathematical model sometimes can be impossible; assumptions and hypotheses are often proposed to make necessary abstract and simplification for a final reasonably detailed model. Some qualitative models are illustrative but straightforward for the observed patterns, e.g. Lotka-Volterra (L-V) equation for predator and prey models. They are usually analytically solvable, and their properties are used to make predictions (e.g. steady states in the ecosystem). If a more precise model is desired, usually more influences or interactions are considered and added as terms of equations to a re-constructed model, e.g. considering environmental factors, a changing environmental capacity and predation from other species in L-V model.

Models like the L-V equation are mostly analytically solvable when analysing properties of the model. However, analytical solutions for many other general models are not feasible because of more complicated dynamics, or the model itself is not a deterministic model. In the areas where solutions are not analytically feasible, computational techniques are usually applied for approximate solutions. Solvers for different models had been proposed to simulate the system or find roots for some special points, such as Explicit Runge-Kutta [5] and LSODA.

For the regeneration process that we wanted to model, differential equations are our preference. Non-linear ordinary differential equations are used in this dissertation, where the only independent variable is time t, denoting the post-lesion time (hours). Further models can also consider other models in other forms, e.g. stochastic models and partial differential equations. For ordinary differential equations (ODE) system, there are several available computational solvers using different algorithms, implemented in MATLAB[1], Python[2] and many other languages. These solvers help to simulate the trajectories of the dependent variables, given initial values and a set of fixed parameters. The decency of fit can be measured by compared the simulated data and observed real data.

Often further researches on the modelled problems require the model to be deterministic on its parameters. Parameters in dynamic system models usually have their practical significance, e.g. describing reaction speed, environmental capacity or the birth rate. There are many approaches to derive these parameters from data or experience. For a popular topic and widely used models, parameters' values can be read from previous literature. However, in many other cases, we need to find values explicitly for our model, where numerical values for parameters are mostly preferred as they are suitable for practical analysis and comparison.

To find the best parameters' values given observed data, either optimisation-based approaches (e.g. least-squares fit (LS), maximal likelihood estimation (MLE)) or inference-based approaches can be applied. Optimisation-based approaches may face two major

[1] `https://uk.mathworks.com/help/matlab/math/choose-an-ode-solver.html`

[2] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.solve_ivp.html`

drawbacks: over-fitting and local optima [2]. Bayesian inference approaches gain popularity in recent years, as it can relieve the concerns in over-fitting and provide a measurement for the uncertainty in the inferred values. Local optima can still be a concern, and many efforts have been proposed for a more thorough exploration of the parameter space to avoid local modes.

2.3 Bayesian inference

Bayesian inference approaches for parameter estimation put Bayes Rule in the centre:

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \tag{2.1}$$

where p is the probability density function, θ denotes the parameter set, D is the given observed data.

Alternatively, as $p(D)$ is fixed once the observed data is provided, the rule can be rewritten as

$$p(\theta|D) \propto l(D|\theta)p(\theta) \tag{2.2}$$

where l represents the likelihood function. In other words, Eqn. 2.2 can be described as 'posterior is proportional to the product of likelihood and prior', and here posterior of parameters is our target in the parameter estimation task. Prior $p(\theta)$ should express our initial believes or estimates on the parameter values before observing the data and conducting experiments; likelihood function $l(D|\theta)$ is used to describe how well the parameter set θ explains the data D, in probability values. Once the data is given, θ is the only parameter in the likelihood function. Some optimisation-based methods attempt to find estimates of parameter values θ s.t. maximise the likelihood. However, such methods will struggle when facing a problem where it is hard to specify an analytical expression for likelihood function, let alone calculating and comparing values of it.

Based on Eqn. 2.2, to find the posterior of the parameter set, prior can be set according to the problem context and our understanding and cover a reasonable range. Computational methods can be used to approximate likelihood function, e.g. Markov Chain Monte Carlo (MCMC) [6]. As an alternative solution, Approximate Bayesian computation (ABC) approaches do not focus on the evaluation of likelihood function but focus on simulating the data from the given model and observed data, then use the simulated data to approximate the true posterior. ABC-rejection [7] is a simple example of the ABC process, and further algorithms e.g. sequential Monte-Carlo (ABC SMC)[8] were proposed to improve the efficiency.

General steps for ABC algorithm includes the following (ABC-rejection):

1. Draw a sample θ^* from parameters' prior distribution. θ^* is a vector that consists of values for each parameter

2. Plug θ^* into the model, and use the model to generate simulated data D^*, where D^* is of the same form as the observed data D, e.g. includes same summary statistics

3. The discrepancy between the simulated data and observed data is measured by a distance function $d(D^*, D)$. For a pre-fixed threshold value ϵ, if $d(D^*, D) < \epsilon$ then the sample θ^* is accepted

4. Repeat 1, 2 and 3 until enough numbers of samples are accepted. The posterior distribution then can be approximated by these samples

In our study of the regeneration model, full trajectories of the dependent variables are used as the data to be compared, as our goal is to find a model with a parameter set that can intuitively recover the real process as much as possible. The mean values at each time point for each dependent variable are calculated to form D. For each sample, the ODE is accumulatively solved at the same time points and generate the simulated trajectories D^*. Using other statistics to summarise the raw data are also possible, but care must be taken as the choice of summary statistics can largely affect the resultant posterior [9, 10].

Instead of fixing the threshold value, the algorithm can be more adaptive by using a sequence of decreasing threshold values ϵ_t, and derive new populations (called generation) of samples from the accepted samples (called particles) in the previous population. An example is ABC SMC [8], which can significantly reduce the required samples compared to ABC-rejection [11]:

1. Set a threshold schedule $\{\epsilon_1, \epsilon_2, ..., \epsilon_T\}$ where T is the total number of populations and $\{\epsilon_1, \epsilon_2, ..., \epsilon_T\}$ is a descending sequence; set the number of particles in each population as N, set the population index $t = 1$

2. if $t = 1$, draw samples from prior distribution until N particles are accepted under threshold ϵ_1, i.e. each of N particles generates data D^* s.t. $d(D, D^*) < \epsilon_1$

   If $t > 1$, draw samples from previous population $\{\theta^*_{t-1}\}$ with weights $\{w_{t-1}\}$, perturb the particles using a perturbation kernel $K(\theta^*|\theta)$ and obtain $\theta^{**} \sim K(\theta^*|\theta)$

3. Repeat step 2 until N particles are accept under threshold ϵ_t

4. Calculate the weights of the accepted particles. For particle i, the weight is

$$w_t^i = \begin{cases} 1 & \text{if t=1} \\ \frac{p(\theta_t^i)}{\sum_{j=1}^N w_{t-1}^j K(\theta_t^j|\theta_{t-1}^j)} & \text{if t} \neq 1 \end{cases} \tag{2.3}$$

5. Normalise $\{w_t^i\}$, increase population index t by 1

6. Repeat step 2, 3 and 4 until $t > N$

7. Output the accepted particles $\{\theta_t^*\}$ and their weights $\{w_t^i\}$ in each generation t

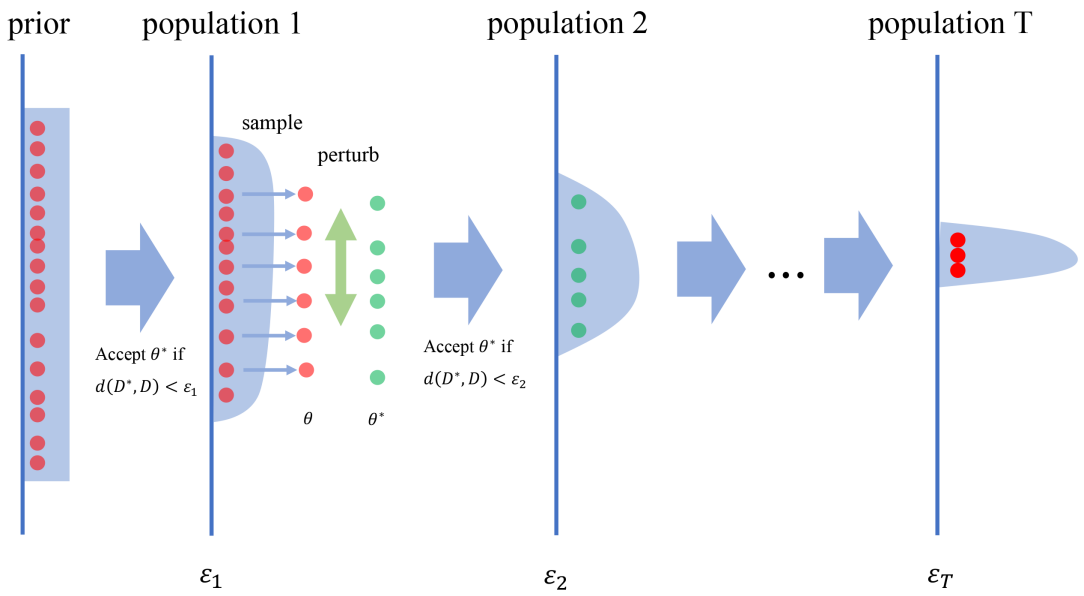

Figure 2.2: ABC SMC sampling process. θ^* denotes a sample drawn from previous population, which is used to generate simulated data D^*. d is a distance measurement, $d(D^*, D)$ represents the discrepancy between observed data and simulated data. for each population t, ϵ_t is a threshold criterion to determine whether to accept that drawn sample

An illustrative graph of the ABC SMC procedures can be found in Figure 2.2. In implementations, many factors can affect the efficiency and goodness of fit; some of them are discussed in Chapter 4. Rather than a pre-fixed threshold schedule, median or quantile based epsilon schedule can save the total sampling size and thus improve acceptance rates of particles [12]. Some other improvements include using adaptive population size for each generation [13], using adaptive distance function [14] and switching kernels [15], etc. In our implementations, these options were tried and compared when applicable.

In addition, ABC SMC is also capable of model selection tasks [16]. Instead of draw samples from prior of a single model, we can draw samples from multiple models m_i and approximate $p(m_i, \theta|D)$, which is the joint probability distribution of model m_i and its accepted particles θ. In each generation, the marginal distribution $p(m_i|D)$ is calculated as the accepted θ is known; thus models can be ranked according to their probabilities.

The feature that ABC algorithm can perform likelihood-free inference makes it widely used in many dynamic modelling problems, especially in the area of biology [2, 11, 17]. However, ABC is not restricted to these areas, as it can be generally used in inference tasks to give parameter estimates or model rankings. E.g., it has been successfully applied in cosmology studies [18].

2.4 Software tools

As we decided to apply ABC SMC on our regeneration models for its high-dimensional parameter estimation and comparison of different models, a code development and experiments workflow was expected to be built. The ABC SMC algorithm needs a whole set of mathematical environments (e.g. class of distributions, models and particles) and closely linked sampling and fitting algorithms (e.g. KDE fitting and Gaussian sampling), thus suitable platforms with related software packages are preferred. Regarding this, several packages in `R` and `Python` were studied.

Software packages that integrally implement ABC SMC are preferred, and more options in the settings of the algorithm, more customisable features make it better. Moreover, as the ABC SMC itself is a computationally intensive task that contains massive parallel potentials, an ideal software package is expected to support as least one kind of parallel options, e.g. multi-core, GPU accelerating and clusters. Otherwise, a modification or rewrite of the software package is expected to produce a reliable input-output framework for our parameter estimation and model selection experiments, where much more work would be introduced.

Preliminarily `pyABC` packages [19] in `Python` was chosen for implementations. `pyABC` supports multiple types of parallelisation, e.g., multi-core processing and distributed cluster, and supports resume stored runs, which made our experiments easier. It is well documented and flexible in customisations of the algorithm.

Some other ABC SMC related packages were also examined and considered as backups, include ELFI [1], ABC-SysBio [2], EasyABC[2], GpABC [3], etc. These packages may have their advantages (e.g. ABC-SysBio supports CUDA acceleration) and considered as alternative implementation methods, or can be used to compare performance difference in future works.

Additionally, some other software tools were also needed in ODE solving, result analysis and visualisations. The software and hardware environments used in this dissertation are listed in Appendix A.

[1] `https://elfi.readthedocs.io/en/latest/`
[2] `https://cran.r-project.org/web/packages/EasyABC/index.html`
[3] `https://github.com/tanhevg/GpABC.jl`

Chapter 3

Mathematical modelling

Mathematical models can describe different kinds of dynamic systems, and thus can be used in prediction, analysis and other tasks. An ideal model in our case, can represent reasonable interactions or effects between immune cells and cytokines and reproduce features and characteristics that were observed in experimental data.

Proposed models for the regeneration process were in the form of non-linear ordinary differential equations (ODE). The only independent variable is post-lesion time t. The equations consist of four derivatives of dependent variables (number of cells, mRNA expression of cytokines) with respect to t. The equations were mostly an interpretation of the proposed interaction maps, with terms correspond to interaction paths.

3.1 Observed data

Our models were built based on the existing experimental findings from Tsarouchas et al. [1]. The available measurement data included the numbers of three kinds of cells (neutrophil N, macrophage Φ and microglia) and the relative mRNA expression of four kinds of cytokines (il-1β, tnf-α, tgf-β1a and tgf-β3). As proposed by Tsarouchas et al., neutrophil and macrophage play essential roles in controlling the il-1β and tnf-α mediated inflammation. Our initial models focused on the changes of four dependent variables N, Φ, β (for il-1β) and α (for tnf-α) that are most important and determinative in the regeneration. N and Φ is of the unit 'number of cells', β and α is recorded as 'relative mRNA expression' (compared to the value at the initial time point, i.e. 0 hpl) and considered unitless.

It was noted that the variance of the measured data is relatively high. The summary statistic was the mean of measurement data at each time point, assuming that measured data was Gaussian-like distributed. To validate this, the distribution of the measured data points was plotted. The result was that at most time points the measurement values are Gaussian-distributed, although some distributions were skewed. One abnormal distribution was observed at time point 120 hpl for macrophage where there are two

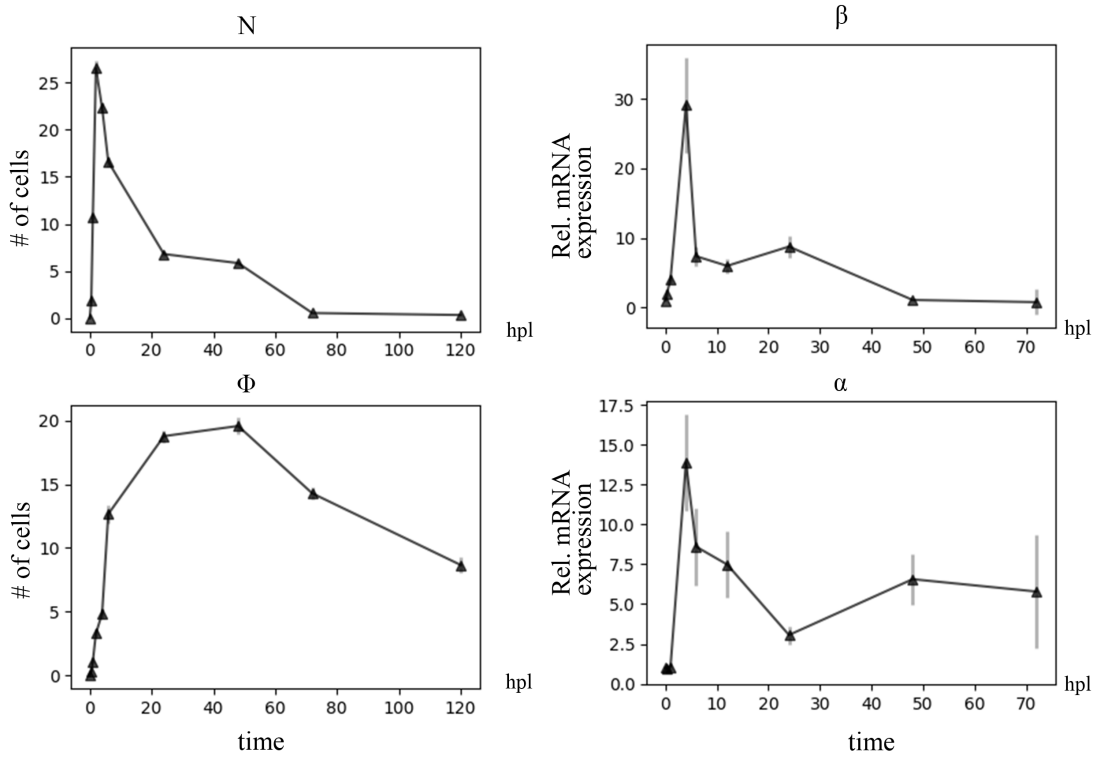

Figure 3.1: Mean of the observed data for neutrophil (N), macrophage (Φ), il-1β and tnf-α, from experiment results of [1]. Error bars indicate standard error of mean (SEM). For N and Φ the SEM is relatively small

concentrations. Despite this, the mean value could summarise most data and thus was still used as the target summary data. The mean value at each time point is shown in Figure 3.1, which is our target trajectories for the models to fit.

3.2 Hypotheses and models

Five models in total are proposed according to different hypotheses. At first, our tests and implementations of ABC SMC for parameter estimations used only the basic model for developing propose. After the parameter inference framework was built and tested, more models were proposed in order to calibrate and improve the basic model such that the observed regeneration process can be better represented.

All these models assumed that the involved interactions are within two kinds of cells (neutrophil and macrophage) and two kinds of cytokines (il-1β and tnf-α), and used the data presented in Figure 3.1 for the inference task. Interactions or influences from other cells or cytokines are not considered as there might not be corresponding data available. If such data or experimental findings are available in the future, we can consider them by then as long as these interactions are indeed necessary for our model.

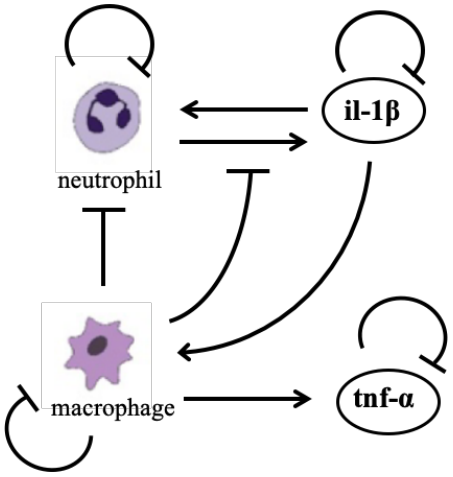

Figure 3.2: Interactions modelled in the basic model (model 1) based on Tsarouchas et al.[1]. Lines ended with arrow represent promoting effect, lines ended with T-connectors represent inhibition

3.2.1 Basic model

A preliminary model (Eqn. 3.1) was proposed according to [1] and then mainly used to build and test the code of inference framework. An interaction map of the model is shown in Figure 3.2. This model is a simplification of the process described in [1] (Figure 2.1) with some minor interactions ignored. This model included 12 parameters, all of which should be positive real numbers. A description of these parameters can be found in Table 3.1.

$$\begin{aligned}
\frac{\mathrm{d}N}{\mathrm{d}t} &= \lambda_N + \kappa_{N\beta}\beta - \mu_N N - \nu_{N\Phi} N\Phi \\
\frac{\mathrm{d}\Phi}{\mathrm{d}t} &= \lambda_\Phi + \kappa_{\Phi\beta}\beta - \mu_\Phi \Phi \\
\frac{\mathrm{d}\beta}{\mathrm{d}t} &= \frac{s_{\beta N} N}{1 + i_{\beta\Phi}\Phi} - \mu_\beta \beta \\
\frac{\mathrm{d}\alpha}{\mathrm{d}t} &= s_{\alpha\Phi}\Phi - \mu_\alpha \alpha
\end{aligned} \tag{3.1}$$

It was assumed that there are negative feedbacks the cells and cytokines, i.e. overcrowding inhibits the increasing rates. As sources, macrophage produces tnf-α and neutrophil produces il-1β; however, macrophage was assumed to have a negative effect on the il-1β, term $1+i_{\beta\Phi}\Phi$ in Eqn. 3.1 was used to represent this. Despite the self-driven increase of immune cells, il-1β was assumed to promote the increase of both cells. Neutrophil was also assumed to be inhibited by the total immune cells presented in the lesion site, the term $\nu_{N\Phi} N\Phi$ was used to denote this.

Parameter	Definition	Units
λ_N	Self-increase rate of neutrophil	$cell/h$
$\kappa_{N\beta}$	Promoting effect coefficient by il-1β	$cell/h$
μ_N	Coefficient of negative feedback of N	h^{-1}
$\nu_{N\Phi}$	Coefficient of inhibition of both N and Φ	$cell^{-1}\cdot h^{-1}$
λ_Φ	Self-increase rate of macrophage	$cell/h$
$\kappa_{\Phi\beta}$	Promoting effect coefficient by il-1β	$cell/h$
μ_Φ	Coefficient of negative feedback of Φ	h^{-1}
$s_{\beta N}$	Production rate from N	$cell^{-1}\cdot h^{-1}$
$i_{\beta\Phi}$	Coefficient of inhibition to the production	$cell^{-1}$
μ_β	Coefficient of negative feedback of β	h^{-1}
$s_{\alpha\Phi}$	Production rate from Φ	$cell^{-1}\cdot h^{-1}$
μ_α	Coefficient of negative feedback of α	h^{-1}

Table 3.1: Parameters introduced in the basic model (model 1) [REMOVE unit]

3.2.2 Alternative models

Model 2 and model 3 As the observed data indicated, this dynamic system has a steady-state where after a long time ($\geq$ 120 hpl), the inflammation is resolved, and immune cells should not be present at the injury site any more. Regarding this, the parameter for self-increase rate for the macrophage and neutrophil, i.e. λ_Φ and λ_N cannot be constant. An exponentially decaying λ term is introduced in the new model (model 2, Eqn. 3.2). Also, the inhibition of il-1β production, i.e. the term $i_{\beta\Phi}\Phi$ is considered to be ignored for a more simple model, in which case the relative expression of il-1β is only affected by the number of neutrophils and the negative feedback from itself. This case corresponds to model 3, written as Eqn. 3.3.

Model 2 and model 3 introduced one extra parameter a and removed λ_Φ, which is a coefficient in the exponentially decaying λ_N, determining the decay speed, with the unit h^{-1}.

$$\begin{aligned}
\frac{\mathrm{d}N}{\mathrm{d}t} &= \lambda_N e^{-at} + \kappa_{N\beta}\beta - \mu_N N - \nu_{N\Phi} N\Phi \\
\frac{\mathrm{d}\Phi}{\mathrm{d}t} &= \kappa_{\Phi\beta}\beta - \mu_\Phi \Phi \\
\frac{\mathrm{d}\beta}{\mathrm{d}t} &= \frac{s_{\beta N} N}{1 + i_{\beta\Phi}\Phi} - \mu_\beta \beta \\
\frac{\mathrm{d}\alpha}{\mathrm{d}t} &= s_{\alpha\Phi}\Phi - \mu_\alpha \alpha
\end{aligned} \tag{3.2}$$

$$
\begin{aligned}
\frac{\mathrm{d}N}{\mathrm{d}t} &= \lambda_N e^{-at} + \kappa_{N\beta}\beta - \mu_N N - \nu_{N\Phi} N\Phi \\
\frac{\mathrm{d}\Phi}{\mathrm{d}t} &= \kappa_{\Phi\beta}\beta - \mu_\Phi \Phi \\
\frac{\mathrm{d}\beta}{\mathrm{d}t} &= s_{\beta N} N - \mu_\beta \beta \\
\frac{\mathrm{d}\alpha}{\mathrm{d}t} &= s_{\alpha\Phi}\Phi - \mu_\alpha \alpha
\end{aligned}
\tag{3.3}
$$

Model 3 can be regarded as a simplification of model 2, as it can be treated as model 2 with parameter $i_{\beta\Phi} = 0$. Among the proposed three models, model 1 is a naive one that was proposed at the very first time and used as an ODE 'template' to build and test our parameter inference framework. As the implementation was successful, model 2 and 3 was proposed, as we were trying to calibrate some terms and find better models. After fitting, model 2 is supposed to be better than model 1 as it corrects the problem that appears at the final time points (which were related to the steady states of immune cells). Model 3 made a small simplification based on model 2, and thus it was theoretically less 'general' than model 2.

Model 4 and model 5 After the first model selection experiment among the exiting three models, it was found that any of the models did not recover some significant features presented in the observed data. To resolve this, attempts were tried to introduce additional reasonable interactions within the dynamic system, considering the biological and mathematical context. Extra promoting effect to the expression of tnf-α was considered, by either adding a phenomenological term $d_{\beta\alpha}\beta$ (which means the same effect as directly promoting but the underlying mechanism is unclear), or adding a term $f_{\beta\alpha}$ that represents a promoting effect to the production process of tnf-α, namely model 4 (Eqn. 3.4) and model 5 (Eqn. 3.5).

$$
\begin{aligned}
\frac{\mathrm{d}N}{\mathrm{d}t} &= \lambda_N e^{-at} + \kappa_{N\beta}\beta - \mu_N N - \nu_{N\Phi} N\Phi \\
\frac{\mathrm{d}\Phi}{\mathrm{d}t} &= \kappa_{\Phi\beta}\beta - \mu_\Phi \Phi \\
\frac{\mathrm{d}\beta}{\mathrm{d}t} &= s_{\beta N} N - \mu_\beta \beta \\
\frac{\mathrm{d}\alpha}{\mathrm{d}t} &= s_{\alpha\Phi}\Phi - \mu_\alpha \alpha + d_{\beta\alpha}\beta
\end{aligned}
\tag{3.4}
$$

$$
\begin{aligned}
\frac{\mathrm{d}N}{\mathrm{d}t} &= \lambda_N e^{-at} + \kappa_{N\beta}\beta - \mu_N N - \nu_{N\Phi} N\Phi \\
\frac{\mathrm{d}\Phi}{\mathrm{d}t} &= \kappa_{\Phi\beta}\beta - \mu_\Phi \Phi \\
\frac{\mathrm{d}\beta}{\mathrm{d}t} &= s_{\beta N} N - \mu_\beta \beta \\
\frac{\mathrm{d}\alpha}{\mathrm{d}t} &= (s_{\alpha\Phi} + f_{\beta\alpha}\beta)\Phi - \mu_\alpha \alpha
\end{aligned}
\tag{3.5}
$$

Parameter	Definition	Units
$d_{\beta\alpha}$	Coefficient of promoting effect from β	h^{-1}
$f_{\beta\alpha}$	Coefficient of promoting the production of α by β	$cell^{-1} \cdot h^{-1}$

Table 3.2: Parameters introduced in model 4 and 5

3.2.3 Limitations

Some limitations were included in Section 3.1, where the observed data have a relatively high standard deviation at certain measurement points. More measurement repeats or more measurement points could help to represent more accurate trajectories of the modelled variables, and thus more possible models can be proposed. On the other hand, more accurate observed data are always beneficial to mathematical modelling, especially to complex dynamic systems.

As stated, our models are largely based on the experimental evidence from Tsarouchas et al.[1]. Some hypothesised interactions may differ from a real case, and there may exist undiscovered mechanisms. These could lead to model misspecification, where the model could fit the data but with wrong interaction paths, or the model can hardly fit the data and recover some local features observed in the data.

Chapter 4

Parameter estimation and model comparison

For the proposed models, exploration of the uncertainties in the parameter estimations is our main task in this chapter, which involved the application of ABC SMC and several experiments. After that, the parameter inference framework was also used for model selection.

The build and test of the code were under `Python` environment with `pyABC`[19]. Some other code, e.g. shell scripts and `R` code were also used to perform the experiments and result analysis. Software and system environment used in our implementation can be found in Appendix A. The build and test were performed on local computers and HPC facilities available within EPCC.

4.1 Code and implementations

According to our preliminary studies, an ABC SMC framework implemented using `pyABC` in `Python` was adopted in this project. `pyABC` is a popular SMC software packages[19] which has been used form many ABC SMC inference applications[13]. `pyABC` provides an open-source framework for likelihood-free inference, which is a `NumPy` implementation of ABC SMC algorithm and a useful tool-box for our inference and analysis tasks. Besides, it supports multi-core execution implemented using `multiprocessing` built-in package, and it made our further studies in the scaling-up performance possible.

The inference framework code for the project was developed and tested in a local environment (macOS 10.15.6, see Table A.1) at first using a small input, then the functional version was deployed on compute node of ARCHER and Cirrus for parameter inference experiments and model selection tasks. The scaling-up experiment was performed on Cirrus. The profiling and its analysis were performed on a local machine. Some nec-

essary development and debug were performed on the remote machines to ensure that experiments run correctly on the compute node.

The ODE model-related code and utilities were first developed to enable reusable functions, variables and data calls. It included the code for the ODE equations, ODE solver, data structures and data format transform functions, etc. By doing this in separate code files, we could change implementation options and activate ABC SMC easily in another 'trigger' file, without creating definition and duplicated code for the models, observed data and parameter priors.

Some missing features were found when using `pyABC`; therefore some modifications were made to the source code of `pyABC`.

For example, the laboratory-measured data were not considered 'complete' or 'tidy' for modelling and analysis purpose: the available time points for cells and cytokines were different, e.g. cell number was not measured at 0.25 hpl and cytokines' expression was not measured at time point 120 hpl. These missing values were treated as 'ignored' when calculating the distance between observed data and simulated data, and represented by `NaN`s. However, in some distance calculations, `NaN` was not acceptable. To cope with this, the built-in distance function was modified. Other modified code included some visualisation and result representation enhancements.

Analysis of the program output was also wrapped up into functions which can be easily called after an ABC SMC run is finished, which includes multiple visualisations and summary statistic of the results. For different tasks, there are also separated analysis code file. Other analysis tools used in the project include built-in database visualisation tool, R code and Microsoft Excel.

The project code was managed via `git` version-control and available as a GitHub repository[1]; it also made the cross-hardware synchronisation of code versions convenient.

After several development iterations, code files for ABC SMC implementation now mainly had two types: code files for infer-back experiments (using synthetic data), and code files for parameter inference (using experimental data). Model selection can be easily performed by adding more model to the implementation code file.

4.2 Hyperparameters and implementation options experiments

As the key focus of this project was on the parameter estimations of a dynamic system, the ability of inference on the proposed models was firstly examined before actual inference using the experimental data.

[1] `https://github.com/chaolinhan/MSc_Project`

In the first part, synthetic data (Table B.1) was used with known true parameter values. The algorithm implementation was evaluated by the efficiency and goodness of the resultant model under different implementation options and ABC SMC hyperparameters.

Then according to findings in the first part, several options with considerable performance were tried in the actual inference with observed data (Figure 3.1). To obtain more accurate and general results, these experiments were repeated for three or more times.

The ABC SMC inference can have widely different performance when using different hyperparameters and implementation options. To firstly test the ability of inference and observe the results of different options, synthetic trajectories with known true parameter values were used as target data. By doing this, we could directly compare the inferred parameters to the true values and compare the simulated trajectories to the true trajectories, then observe the result to see whether the inference is successful and how well the model fit the data. Also, by setting the same final threshold value, the efficiency under different options can be compared and help us to choose proper options when applying ABC SMC.

The true values (Table B.1) were obtained by fitting model 1 onto the experimental data (Figure 4.1) using least-square optimisation method. Available least-square fit tools in `SciPy` were used. The fitting algorithm was Trust Region Reflective[2]. Target data was then generated using these parameter values via ODE integrate solver[3].

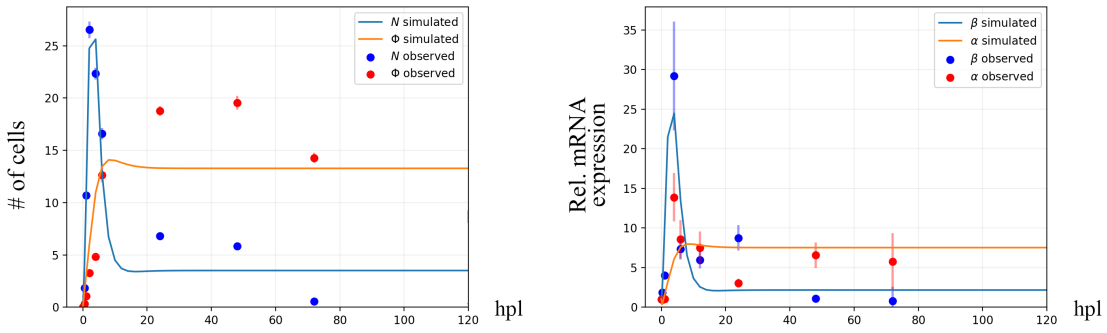

Figure 4.1: Synthetic data generated with known parameter values (line plot), compared to experimental data (scatter plot). Known parameter values are obtained from a least square fitting of the observed data

The following topics were studied using the synthetic data and the developed ABC SMC inference framework.

[2] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.least_squares.html`

[3] `https://docs.scipy.org/doc/scipy/reference/generated/scipy.integrate.odeint.html`

4.2.1 Perturbation kernels

Perturbation kernels work in the sampling process. In each generation t, samples are firstly taken from the previous population $\{\theta_{t-1}\}$ with weights $\{w_{t-1}\}$, then perturbed using the perturbation kernel $K(\theta|\theta^*)$. We keep sampling until N particles are accepted, where N is the pre-set population size. After that, the new weights are calculated and normalised. Figure 2.2 illustrates the process.

The perturbation kernel is called in the sampling of every particle and determines how the new perturbed particle is chosen; thus it is influential to both computation complexity and the resulting posterior distribution. Generally, a local perturbation kernel may face the risk of being stuck in local modes (e,g., local optimal), but it may need less computational operations or could generate a population with a higher acceptance rate if the successive epsilon values are close. A kernel with a wide variance, or spreading out in a large space could help in resolving the local optima problem by a more thorough exploration of the parameter space. However, it can be more computation-intensive and result in a lower acceptance rate. A desired optimal kernel should balance the trade-offs; their properties and criteria were discussed in [15].

There are several common choices of perturbation kernels. Among them, the multivariate normal kernel and local M-nearest neighbour model is preferred to be applied to our models. A covariance matrix Σ_t of accepted particles is calculated from the previous generation and used in multivariate normal kernel: $K(\theta|\theta_{t-1}) \sim \mathcal{N}(\theta_{t-1}, \Sigma_t)$. It is illustrated to be more efficient than uniform kernel and component-wise normal kernel in reflecting the true posterior structure [15]. It has been proved to perform well in several dynamic system models [2, 17, 11] for the parameter estimation and model selection tasks. The $scaling \in (0, 1]$ parameter in `pyABC` will be multiplied to the covariance to produce a 'thinner-distributed' perturbation result.

Local M-NN kernel provided by `pyABC` was also tried to provide a comparison. A local kernel density estimation (KDE) fit is used with M nearest neighbours considered.

The kernel experiment is designed to explore the efficiency of SMC on our dynamical systems. Given the same fixed threshold schedule, kernels that need less total samples and have higher acceptance rates would be our preference. The experiments compared multivariate normal kernel with local MNN kernels with different parameters, using synthetic data to infer back the parameters. The acceptance rate among each time point and total required samples are compared after the experiment to give suggestions on the kernel selection in the real data inference. As the threshold schedule was fixed, the final population should have similar discrepancy to the target data thus here the goodness of fit, i.e. recover of target data trends/features and errors of the inferred parameters compared to true values were not discussed.

Kernel experiment results

To compare kernels, a fixed schedule was used with the final epsilon $\epsilon_t = 10.0$. From the total required samples plot (Figure 4.2) the efficiency can be compared. Among the tested kernels, the local M-NN with M=50 has the best performance: it requires the least number of samples and has higher acceptance rates among almost all generations. Local M-NN with M=750 is the slowest kernel.

For local M-NN kernels, generally, a higher M leads to lower acceptance rates and more required particles in each generation. As shown in Figure 4.3, local M-NN with M=750 is the lowest curve. Consequently, if greater M was tested, then it would have even lower acceptance rates; the maximum M=2000 (the whole population is considered) would have the lowest acceptance rates.

Multivariate normal kernel has a performance between local M-NN M=250 and M=100. It proves that rather than a trivial normal kernel, multivariate normal kernels are more efficient facing the concentrations of joint distributions among multiple parameters [15]. 'Scaling' option can narrow the distribution calculated from the last population, thus making the kernel more 'local' by sampling under a smaller variance. Similar to small M in local M-NN, small 'scaling' is also more efficient in our experiment of model 1. Besides the fixed threshold schedule, a median threshold schedule with the same final threshold $\epsilon_{20} = 10.0$ was also tested and similar results are observed (Figure B.1 and B.2, Appendix B2).

A more 'local' kernel is more likely to give better performance by efficiently sampling around concentrations in parameters' distribution and approximate the posterior distribution. This holds in most cases under a given target final threshold value ϵ_t. However, it may not produce the proper result want: a more local kernel, e.g. local M-NN with small M and multivariate normal with 'scaling' is more likely to be stuck in local optimality, as the kernel is less likely to sample particles that are far from local concentrations, although these local optimal are still accepted before even smaller threshold values is demanded (see Section 5.2 for more discussions). One obvious example is presented in Figure 4.2, where local M-NN with M=50 suffered from a local optimal: the last generation takes much more samples to meet the required threshold.

To conclude, Local kernels can be fast but unstable in some case. For a general parameter inference of our models, multivariate kernels were preferred, as a good fit is our prior target. Some local kernels are also worth trying after the multivariate normal kernel, e.g. local M-NN with M≤ 250 (for a population size of 2000, i.e. 12.5% of the population) and multivariate normal with scaling, but extra cares must be taken: although they are fast, moving in and moving out of local optimum make them more unstable in efficiency.

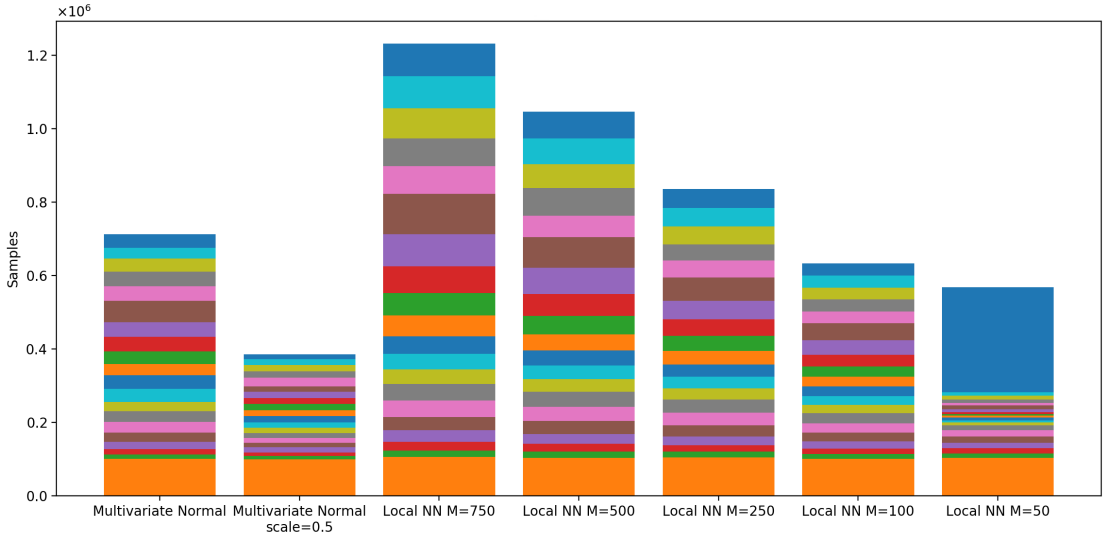

Figure 4.2: Total required samples of different kernels after 20 populations (2000 particles in each population). Different color represents different generations (bottom to top: population 1 to population 20)

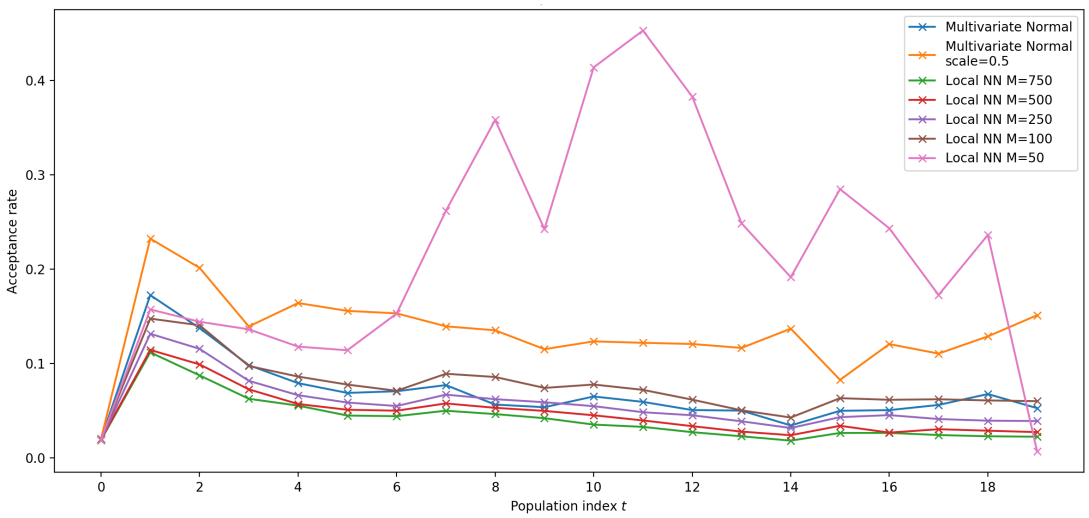

Figure 4.3: Acceptance rates of different kernels in 20 populations. Each population has 2000 particles

4.2.2 Adaptive functions and factors

SMC can be more adaptive by introducing adaptive distance function and adaptive population; besides, factors can be applied to manually 'normalise' the data.

The trivial distance function is set to Euclidean distance (2-norm distance)

$$D = \sqrt{\sum_i \Delta x_i^2} \tag{4.1}$$

where i is the index of data points and Δx_i is the discrepancy between observed data and simulated data at data point i, i.e. $\Delta x_i = x_{i,simulated} - x_{i,observed}$. In this case, data points are 12 time points of four variables, i.e. 48 data points in total.

A weighted 2-norm distance can be written as

$$D = \sqrt{\sum_i w_i \Delta x_i^2} \tag{4.2}$$

where a weight w_i is assigned to every data point. If data point i has a higher weight, then the data value is regeared to be more informative, i.e. giving more help in inferring the true posteriors. Weights can be either pre-set according to prior knowledge of the problem or updated using an adaptive method. This method is known as adaptive distance.

Adaptive distance is changing the weights of all data points after each generation iteration, trying to assign informative data points higher weights. Additionally, implementation of adaptive in `pyABC` also introduces additional factors f_i multiplied to weights [14]

$$D = \sqrt{\sum_i f_i w_i \Delta x_i^2} \tag{4.3}$$

Factor is helpful as an option for efficiency. When some data points are equally informative, the manually pre-defined can make the distance more focused on certain data points, or behave like a normalisation that can balance the scales of different summarised statistics (here are the 48 mean values). There are two appliance case studied: (1) factors used as a normalisation option and (2) factors used to give more focus on some features of the observed data. For (2), as we noticed that the main features (e.g. rapid increase, peaks, fluctuations) are mostly observed in the first half of the time points, i.e. 0 - 30 hpl, factors were tried in later experiments. Such factor scheme

assigns data in the first half of the time point with higher factors (75) and data in the second half with lower factors (25).

Adaptive population [13] can adapt the population size of each generation according to the mean correlation of variation error of the previous population. However, in our early tests, the maximal allowed population size is set to 10,000, and the adaptive strategy always selected the upper bound (10,000) for new populations, as a result of wide parameter space or the wide posterior approximation shape. Additionally, some internal bugs were observed when applying it. Hence, the adaptive population was not tried in this project; however, it is worth trying in future works as an efficiency improvement.

Experiment results

Two types of factor and adaptive distance function were tested by running the same number of generations (20 generations, 2000 particles per generation). The first type of factor is 75:25 factor, where the first half of the trajectory data were assigned higher factor (75), and the second half of the data was assigned lower factor (25); this could make the first half of the data more 'important' in the inference. The second factor was 'range' factor, which tried to normalise the trajectory of each dependent variable according to its data range.

From Figure 4.4, different types of factor gave different performance. 75:25 factor required much more samples in the first generation (orange bar in the left of Figure 4.4) and also did not improve much in later generations; thus, it is the slowest one. The first half of observed data contained more features and information, assigning it with higher factor would make the algorithm think that it is more important to fit the first half and thus focus on the hard part, and consequently more particles were needed in the inference process. The acceptance rates of 75:25 factor among different generations were also generally slightly lower than the standard. By contrast, range factor required less total samples and gave generally higher acceptance rates than standard, which made it considerably faster. The adaptive distance gave the best efficiency, as it significantly improved the acceptance rates and resulted in even less required samples (right, Figure 4.4).

However, the result of factors adaptive distance could not be directly compared to standard run, because it changed the distance measurement by adapting weights or manually setting factors. Factors were normalised to make it closer to the standard distance measurement. As a result, the same final threshold value did not guarantee that adaptive distance can reach the same level of fit as others. After 20 populations, adaptive distance had a poor fit (Figure 4.5), while the standard run and two types of factors had similar tight fits of the data, which all converge to the true posterior. Additional in our implementations, variance factor (which tries to normalise the data according to the dependent variables' variance) was also tried and gave a very similar performance as the range factor.

To conclude, because the factors and weights are directly multiplied to Δx, and thus the

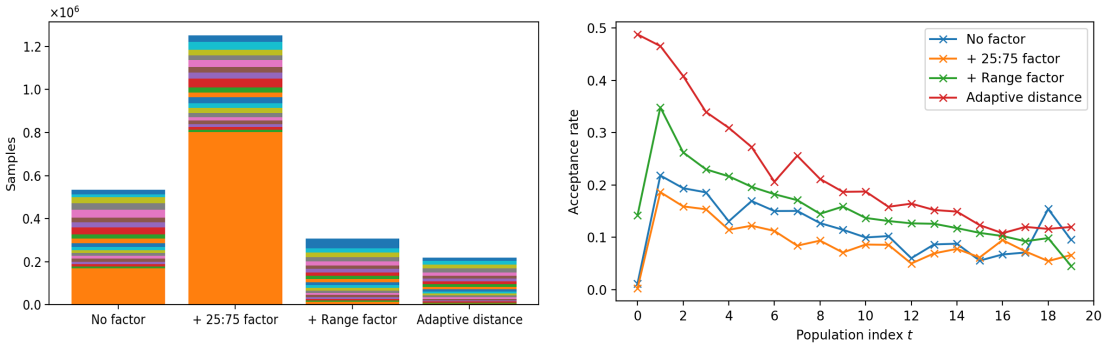

Figure 4.4: Factors and adaptive distance experiments. Left: total required number of samples. Different color represents different generations (bottom to top: population 1 to population 20). Right: acceptance rates in each generation.

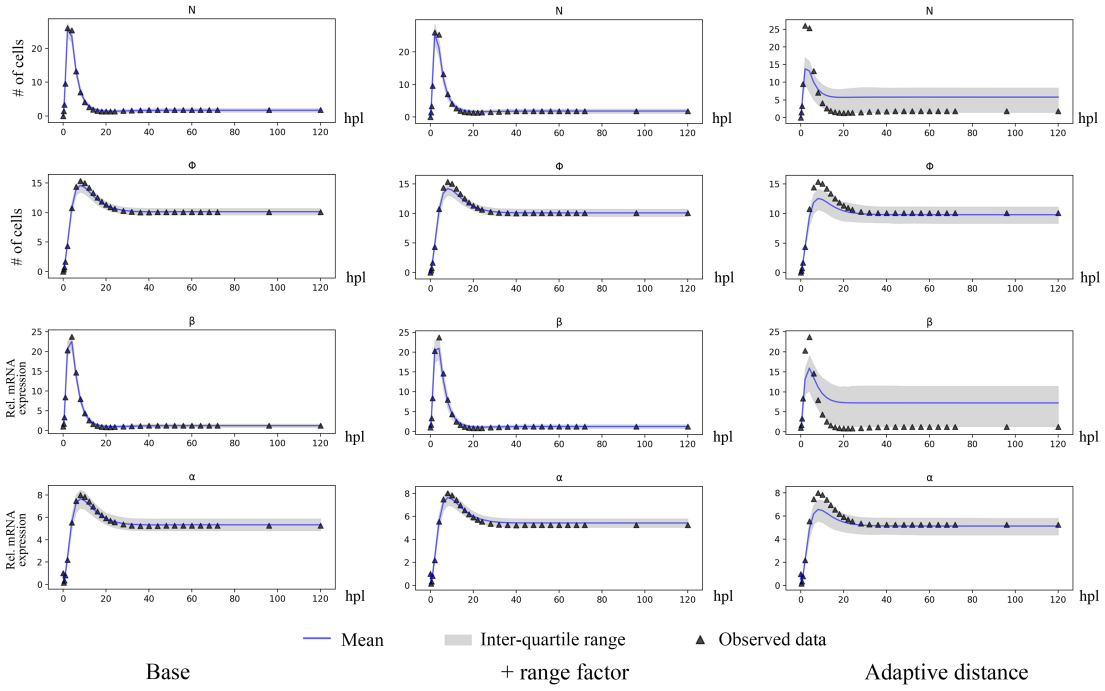

Figure 4.5: Simulated trajectories from the last population in the factor and adaptive distance experiment. 500 randomly chosen particles from last population are used to generate a sequence of trajectories where mean and inter-quartile range are calculated from. 75:25 factor showed very similar curve as the range factor.

metrics for distance are changed, the direct comparison of the efficiency under the same threshold schedule was unfeasible; we could only compare the results after the same number of generations. Factors gave nearly the same inferred results but required a different number of samples, hence cares must be taken when choosing factor schemes. Adaptive distance did not provide an accurate result as others. As a result, our further implementations used standard distance function, and factors were also tried for

possible improvements.

4.2.3 Data size, prior distribution and population

Some other hyperparameters, e.g. feed-in data size, prior distribution, number of populations and number of particles in each population were also of our interest. Experiments were designed to explore how these options affect the goodness of fit and efficiency of the implementation.

Regarding the existing dynamic systems model, SMC was applied with two data size options and two ranges of the prior interval. The population options, i.e. population size and the number of populations were also tried with different values. These experiments intended to give suggestions on the later implementation on experimental data.

Experiments results

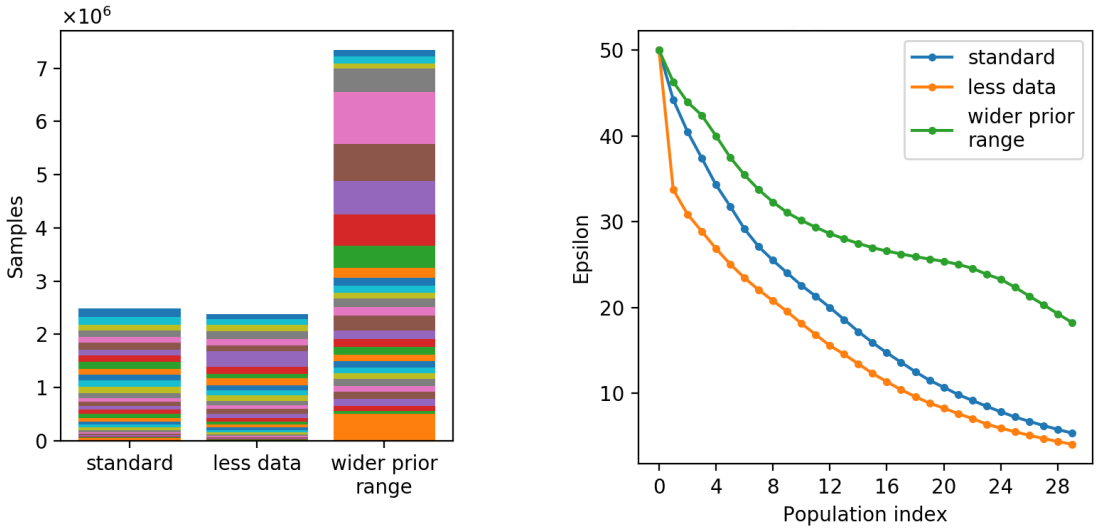

Figure 4.6: Data size and prior distribution range experiment. Left: total required number of samples. Right: epsilon values under median epsilon schedule

The results from data size and prior distribution range are shown in Figure 4.6. The standard implementation is fed with 120 data points (30 data points for each of the four variables), the 'less data' is fed with 48 data points, which is the case of the real experimental measurement. Although much less (60%) data was used, the total required sampling numbers was only 4.47% less. This indicated that the data size does not affect the sampling process much and the inferred model are all considered acceptable when compared to the synthetic data, as they finally reached a similar epsilon threshold (Figure 4.6), but cares should still be taken when using less data points, or using other

more summative statistics (e.g. mean, standard deviation, peak values, starting and ending values etc.): as shown in [11], fewer data can lead to a model with more variance, under-fitting or missing some local features such as local peaks.

The prior range and distribution can largely affect the inference progress (Figure 4.6. (a)). Samples were drawn from a high dimensional parameter space in each sampling process. A wider parameter prior interval in only one dimension can cause the parameter space to be much bigger. Consequently, the exploration and sampling of parameter vectors would take much more efforts. Right plot in Figure 4.6 also indicates that the convergency is slower for wider prior ranges. As a result, a proper prior interval based on our experience and belief with seeing the data should be set as narrow as possible, as the inference is largely affected by the prior interval and its length. Improper prior could even make the algorithm miss the true posterior [2].

How to specify prior can be tricky from problem to problem. For our task in the parameter inference of regeneration model, biologically motivated priors were used, which were a general reflection of biological constraints. Also, the units of the parameters helped us in identifying the possible scale of the parameters.

Choise of options

This section aims to explore options in ABC SMC implementations preliminarily. From the above results, we adopted multivariate normal kernel as the kernel for our real inference on observed data; Local M-NN kernels were also planned to be tested. In the real inference, 75:25 factor were also tried to see if it could help in recovering more features and resulting in a better fit. Prior intervals were explored explicitly and gradually shrunk for our models.

Some other hyperparameters, e.g. the number of generations, population size (i.e. the number of particles per generation) were not addressed in this subsection. In the real inference, population size was set to be greater than 2000 to avoid insufficient exploration of the previous population; in some case than high variance or uncertainties in the results were observed, population size as large as 20,000 were used. If the epsilon did not show a convergency trend, the run could be extended to more generations until a stable result was observed.

4.3 Parameter estimation and model comparison

4.3.1 Model 1, 2 and 3

Using the suggested options in the previous section, the target data (Figure 3.1) was prepared as input for the SMC inference framework. Regarding the result, the population

size and number of generations were adjusted several times to obtain a general informative posterior which should be stable across repeated runs and avoid local optima as much as possible.

Two prior intervals ([0, 25] and [0, 75]) and two distribution shape (uniform and log-uniform) were tried in the parameter estimation of the first three models. All these ABC SMC runs used 2,000 particles per population and 30 generations. The population size should be set at a high level (e.g. ≥ 1000 in our case) when having models with many parameters to ensure a statistically adequate exploration. The convergency were usually observed around generation 20 in our previous experiments; the length of generation is set to 30 to ensure that most runs converge in the end.

In practice, we found that log-uniform-shaped prior gave a better fit of the model than uniform-shaped prior, and wider prior range [0, 75] does not give a more accurate result than [0, 25]. Then prior interval [0, 50] was tried, in case some parameters have a true posterior laying between [25, 50].

The results of prior distribution log-uniform [0, 50] is shown in Figure 4.7, which is considered to give a better fit than other prior options. For these models, the acceptance rates and epsilon path are presented in Figure 4.8, and the estimated parameters' posterior of model 3 is shown in Figure 4.9. Figure 4.7 plots the simulated trajectories of the three models. One thousand particles (each particle is a parameter set) were sampled, and the mean (blue) and 25th to 75th percentile interval (grey) were calculated from the data simulated by these 1000 particles.

Before applying model comparison methods, some features of the three model can still be compared. From simulated curves, It can be seen that all the inter-quartile ranges are tight around the mean value curve, which indicates the approximated posterior are concentrated; the bell-shaped posterior distributions with a single peak (Figure 4.9) agrees with this. All epsilon values converge to a steady level, and model 3 has a lower final epsilon value, and consequently, its simulated trajectory is more close to the observed data (model 3, Figure 4.7).

The acceptance rates are fluctuating and have a gradually increasing trend for all the three models, as the acceptance rate becomes higher when the true posteriors are gradually approximated. Compared to model 1, model 2 gives a better fit for the decreasing trend in the second half of the observed data. This proves that using an exponentially decaying self-increase rate λ_N instead of a constant term indeed help to improve the model.

The resultant simulated data from model 2 and 3 have similar trends, while the most obvious difference is that for model 3, the simulated data of Φ and α are more 'flat'. We expected model 3 to be the best as it reached a lower threshold value, and intuitively gives a better fit. However, the fit of all these models could not be considered as a generally good representation of the biological process that we want to approximate: for tnf-α, the simulated data and the observed data show largely different trends. A sharp peak at 4 hpl was observed in the experimental data, but none of the three models represents similar features (Figure 4.7).

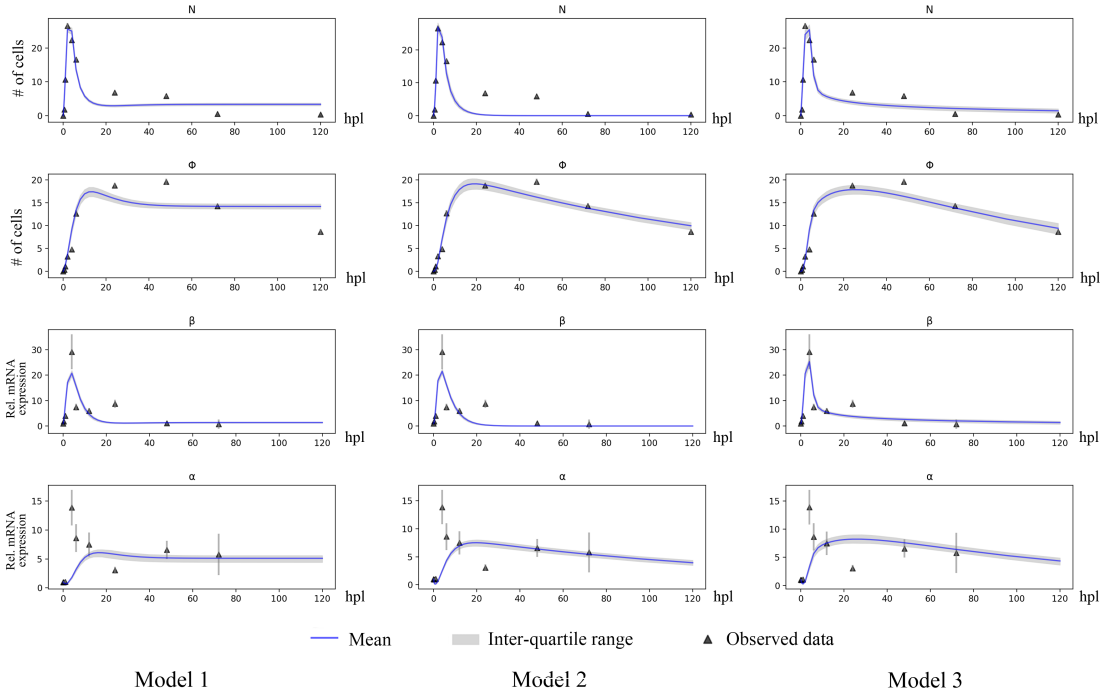

Figure 4.7: Simulated trajectory from the last population of model 1, 2 and 3. Error bar indicates SEM. 500 randomly chosen particles from last population are used to generate a sequence of trajectories where mean and inter-quartile range are calculated from

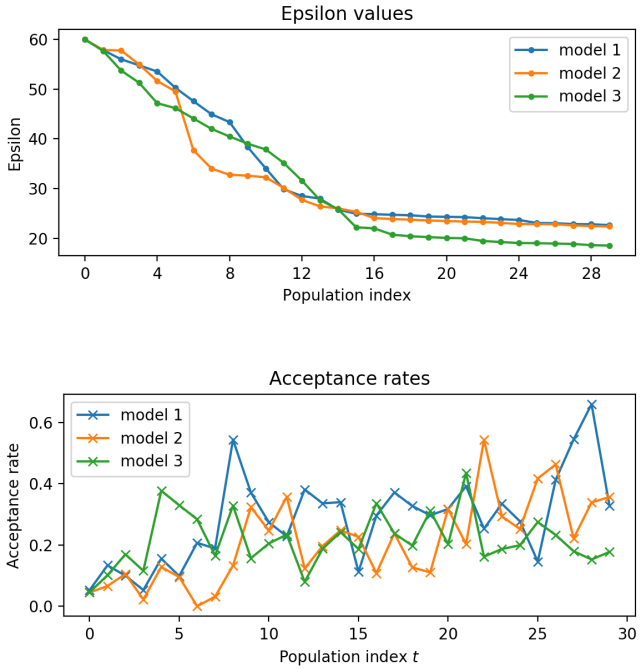

Figure 4.8: Epsilon trends and acceptance rates of model 1, 2 and 3

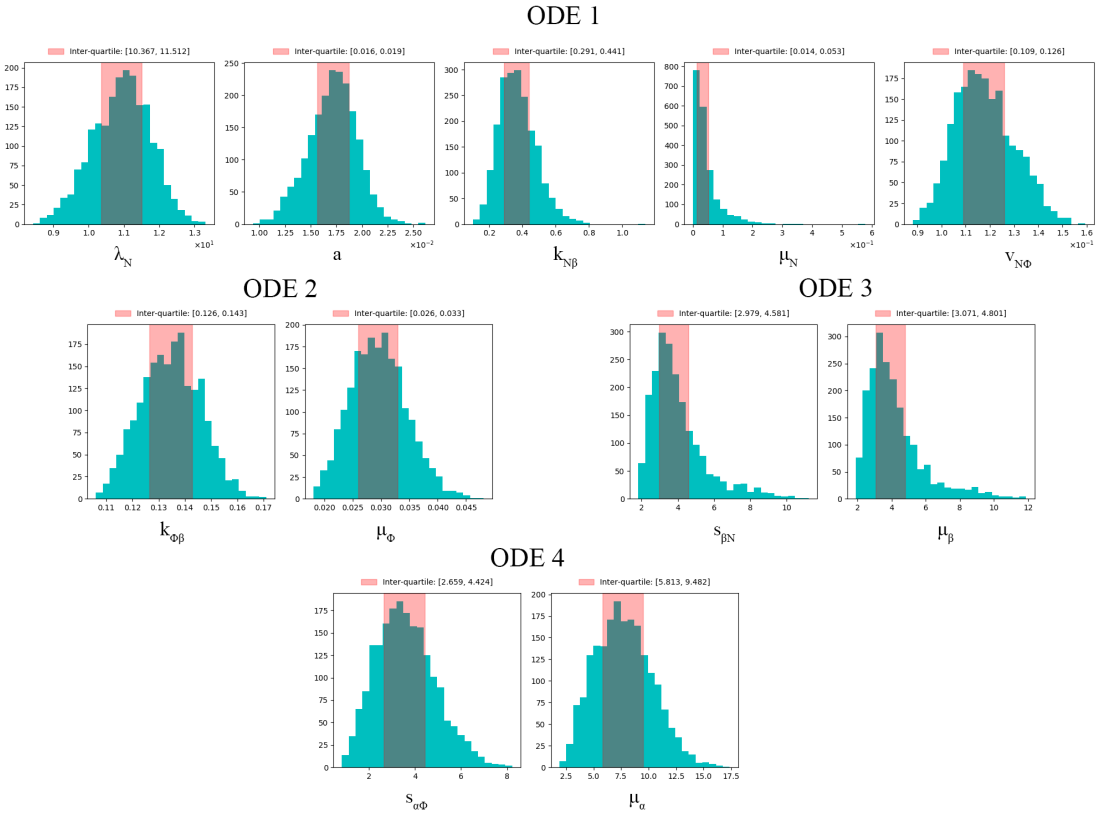

Figure 4.9: Estimated posterior distribution of parameters in model 3. Shaded range indicates inter-quartile interval

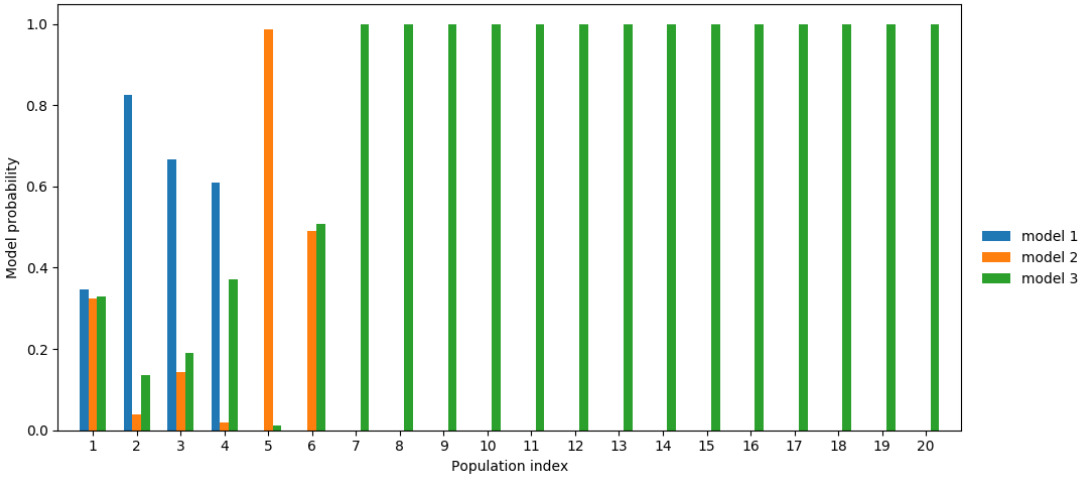

Figure 4.10: ABC SMC based model comparison of model 1, 2 and 3. From population 8 to population 30, model3 always wins

Parameter	Estimated mean value (uniform)	Estimated mean value (log-uniform)
λ_N	14.1	13.3
a	9.67	0.0170
$\kappa_{N\beta}$	18.0	0.259
μ_N	13.2	0.0180
$\nu_{N\Phi}$	0.742	0.131
$\kappa_{\Phi\beta}$	0.239	0.156
μ_Φ	0.154	0.0350
$s_{\beta N}$	6.75	1.21
μ_β	6.38	1.32
$s_{\alpha\Phi}$	17.0	5.83
μ_α	11.9	2.43

Table 4.1: Estimated parameter values of model 3, using uniform prior distribution and log-uniform prior distribution respectively

Next, a model selection experiment with different prior was conducted, using the same ABC-SMC framework. As in the parameter inference, two distributions (uniform and log-uniform) are tested with three different interval ([0, 25], [0, 50] and [0, 75]). The results show that model 3 wins with 100% model probability after 30 generations for most runs, except log-uniform [0, 75]. All the model probability results are of great discrepancy (e.g. 100% model 3 and 0% for model 1 and 2), so Bayesian factor is not calculated.

Figure 4.10 shows the model selection process, under log-uniform [0, 50] prior. Model 3 gains total advantage over model 1 and 2 after the seventh population. Together with the conclusions above, model 3 is considered to be the best model to represent the target data so far.

Compared to uniform-distributed prior, we found that log-uniform distributed prior is more suitable in our parameter inference. Log-uniform resulted in an obviously better fit and narrower inter-quantile interval (Figure 4.7 and 4.11), and the final epsilon value was also smaller. Log-uniform tends to assign values that are close to the left boundary (i.e. zero in our case) with higher density. As a result, the parameter estimates from log-uniform prior are highly likely to have smaller mean values than that from uniform prior, and the inferred parameter values proved this (Table 4.1). This was also observed in other models and prior interval experiments. It indicated that some of the parameters tend to have small true values that are close to zero, and using log-uniform prior could result in a faster and more satisfying inference. Regarding this, further experiments all used log-uniform prior only.

4.3.2 Model 4 and 5

We observed that the existing models could not well represent the trajectory of relative expression of tnf-α. From experimental data, a rapid increase of tnf-α expression was observed in 0–4 hpl, followed by a rapid drop. All the models cannot well recover this feature, and most results saw a less-rapid increase followed by a gradual decrease, while the observed peak value was not reached (Figure 4.7).

Also, in some other experiment results, e.g. uniform runs (Figure 4.11), tnf-α trajectories was well fitted; however, at the cost of under-fitting Φ curve. Hence efforts had been spent to propose more alternative models that can potentially solve this problem.

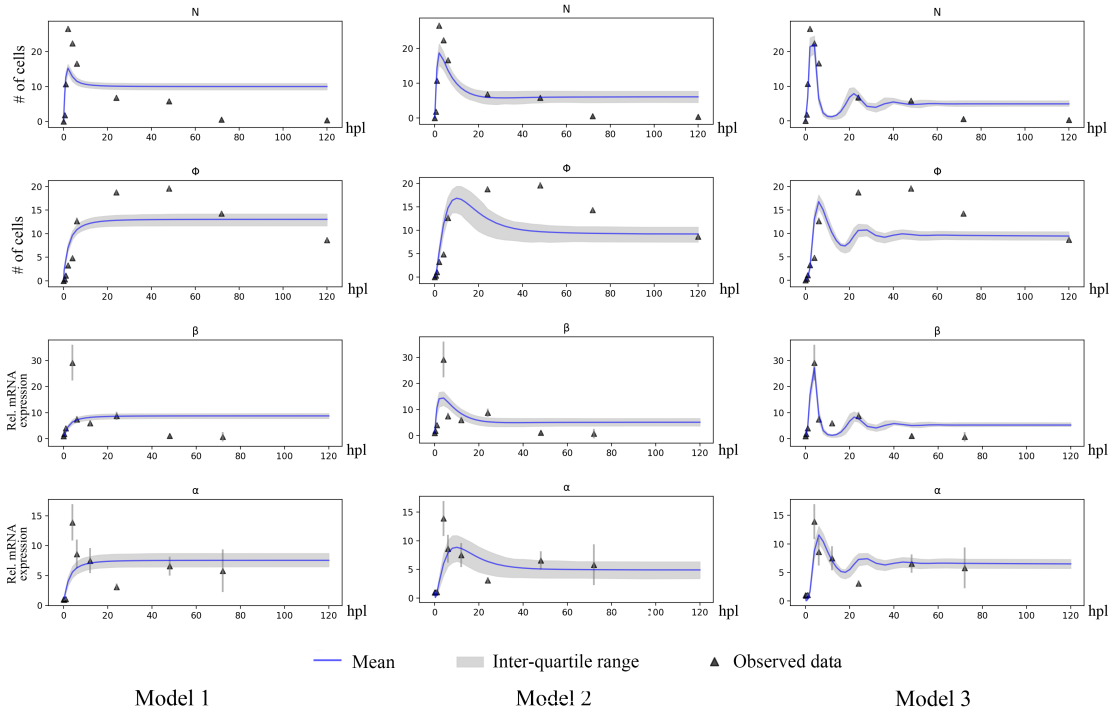

Figure 4.11: Simulated data from the last population of model 1, 2 and 3, using uniform prior. Error bar indicates SEM (for N and Φ the SEM is relatively small)

The two proposed alternative models are based on model 3 and try to add extra term in the tnf-α equation $\mathrm{d}\alpha/\mathrm{d}t$. As macrophage is the main source of tnf-α production, and a similar sharp increase-and-drop trend is observed in the il-1β trajectory, two hypotheses and corresponding terms are propose as follows.

Model 4 It is assumed that the expression of il-1β would have a promoting effect, representing by a phenomenological term $d_{\beta\alpha}\beta$ and $d_{\beta\alpha}$ is a new model parameter (positive constant). The promoting effect here is regarded to have the equivalent effect as directly

promoting by il-1β, but the underlying mechanism is unclear so far. In mathematical view, this additional term can accelerate the expression of tnf-α in the first few time points and help to recover the peak-shaped trajectory. The proposed model is written as Equation 3.4.

Model 5 Alternatively, we considered promotions to the tnf-α production, i.e. $s_{\alpha\Phi}\Phi$. It is assumed that il-β could accelerate the production process. An additional term $f_{\beta\alpha}\beta$ is introduced to production rate, with $f_{\beta\alpha}$ being a new model parameter (positive constant). This model meets the biological context, considering the source of tnf-α. The proposed model is written as Equation 3.5.

Model 4 and 5 are based on model 3, so in this phase experiments of these three models were conducted for separately parameter inferences and overall model comparison. Based on our experience in conducted experiments, we used the following setting for parameter inference:

- population size: 2000 and 5000
- number of populations: 30 generations
- prior: [0, 50] log-uniform
- perturbation kernel: multivariate normal kernel

Factors (Equation 4.3) were also tried to find if we can enforce the inference framework to give more importance to the first few data points. An optional factor scheme was proposed: for each trajectory, the first 8 data points were assigned higher factors (75), and the rest 4 data points are assigned with lower factors (25).

All these experiments were conducted on remote machines, and the output database files were retrieved for analysis.

Figure 4.12 shows the simulated data from the inferred models, with population size 2000. It can be seen that the addressed problem in the trajectory of tnf-α is partially relieved in model 4 and 5: compared to model 3, a peak appears around 4 hpl; it can represent more features in the observed data and consequently the final reached epsilon (after 30 populations) is smaller than that of model 3, which proves that our proposed modifications in model 4 and 5 are helpful in recovering more features in observed data.

By intuition model 4 and 5 would be the best models so far that can recover most of the observed data features. Model 1, 2 and 3 could also converge to a low final epsilon value (20), but a key feature observed in target data – a peak at around 4 hpl – was not fitted. Some features, e.g. fluctuations in il-1β and tnf-$alpha$ trajectories at around 20 hpl were still not ideally fitted. All existing models gave steady or gradually decreasing trends for all the four variables after 20 hpl, where fluctuations in the observed data are fitted by smooth and steady curves.

After applying factors, the results were not largely different. As expected, applying factors made the data points at first half (where exists most of the fluctuations) to be

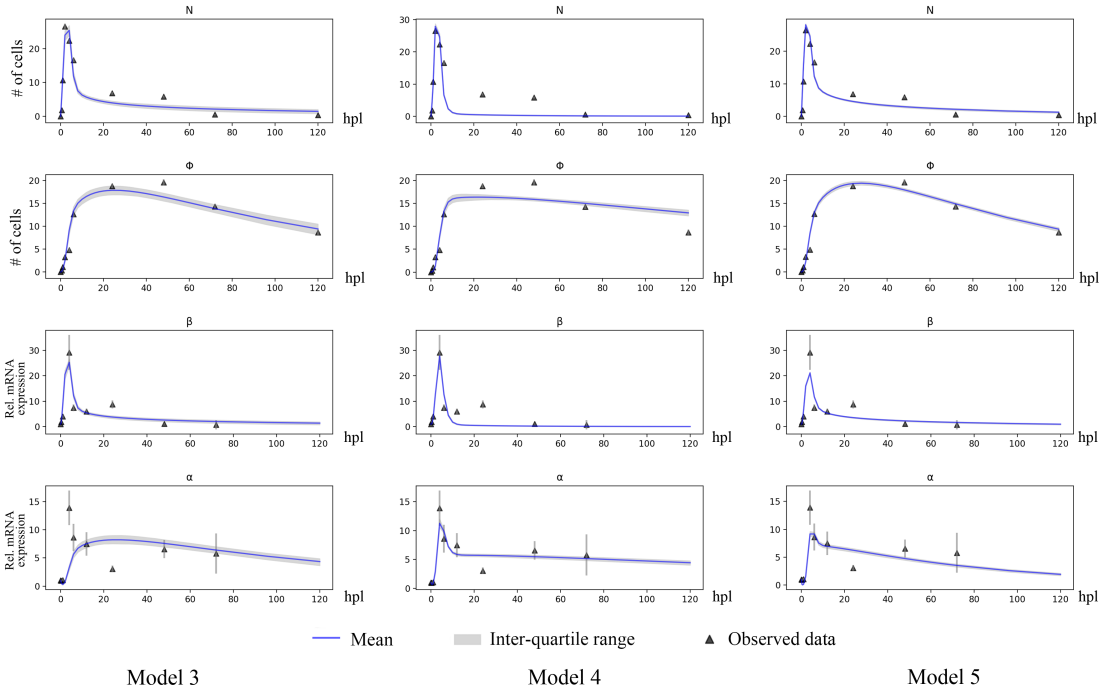

Figure 4.12: Simulated data from the last population of model 3, 4 and 5. Error bar indicates SEM (for N and Φ the SEM is relatively small)

more 'focused'. However, the inferred models were not largely different from ones without factors, which indicated that for model 4 and 5, applying factors will not largely affect the final convergency and the resultant posterior.

After we tried more generations with factors, an 'over-fitting' behaviour was observed at the trajectory of macrophage (Figure 4.13). In this case, the discrepancy of simulated data and observed data mainly comes from the last 2 data point (72 hpl and 120 hpl) of macrophage. As additional importance was assigned to the first few data points, the inference algorithm tended to fit first data points preferentially, and thus relatively large discrepancy at the tails were acceptable for it.

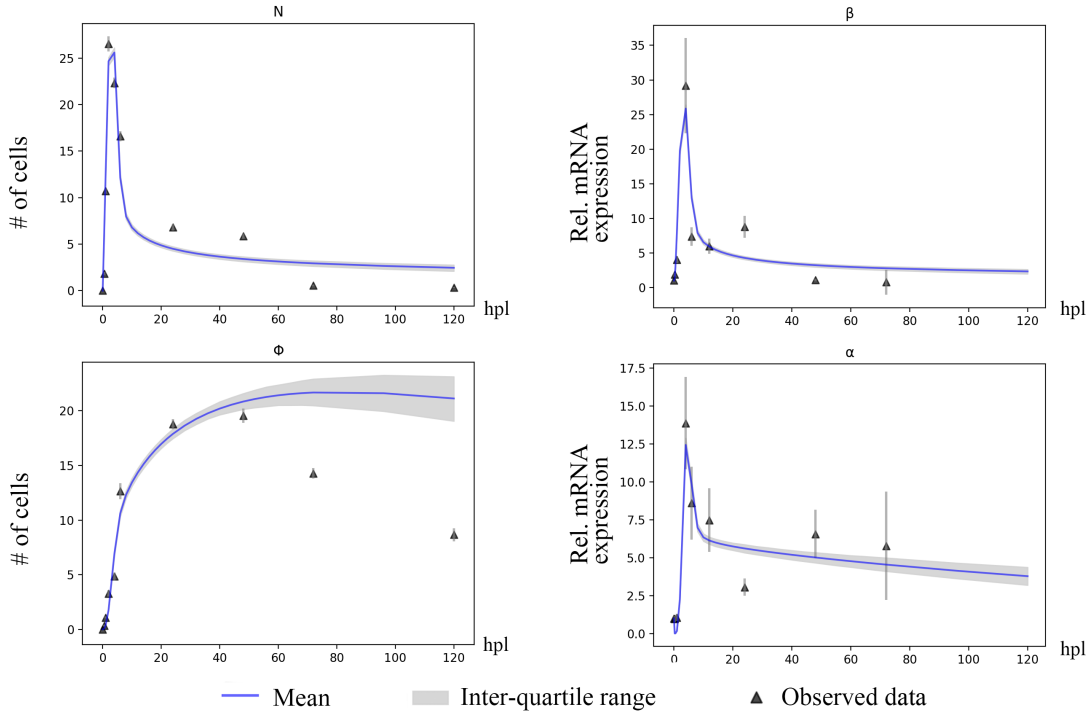

Figure 4.13: Simulated trajectory from the 30th population of model 5, with 75:25 factor applied. Error bar indicates SEM (for N and Φ the SEM is relatively small).

Next, we performed a model comparison on model 3, 4 and 5, using the same ABC SMC inference framework. Figure 4.14 shows the model probabilities at different generations.

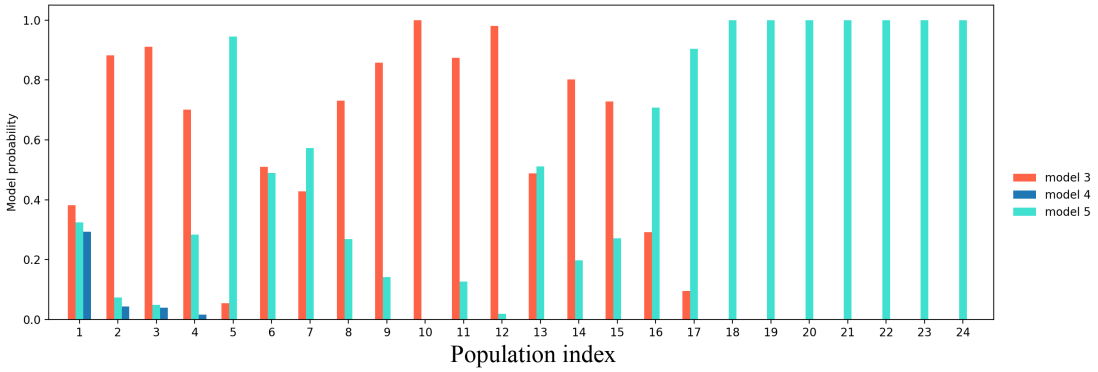

Figure 4.14: Model probabilities of model 3, 4 and 5 in ABC SMC based model comparison. Model 5 also wins in further populations (population 25 to 30)

From the model probability plot (Figure 4.14), we found that model 4 did not overweight other models across all the generations. Model 3 was comitative and gained

advantages at most early generation. From generation 14 and onwards, model 5 saw a gradually increasing probability and gained 100% model probability after generation 18 ($\epsilon_{18} = 16.98$) and won in further generations. Also, repeated runs showed similar trends, although uncertainties (e.g. drastically model probability changes) were present before generation 20, model 5 won with nearly 100% probability in later generations where epsilon values were considerably low, i.e. $\epsilon_t < 16$.

Based on the model comparison results, we could conclude that model 5 was the best model among the tested models. Uncertainties were observed in early generations, where similar patterns were also observed in the model comparison of model 1, 2 and 3. We considered the drastically changing model probability in early generations is related to the inference ability at high epsilon threshold values (epsilon) and possibly the insufficient exploration of the parameter space. However as we only focused on the best model with small final threshold ϵ_T, the easiest way is to extend the number of generation until epsilon converges, or increase the population size for a more statistically confident probability.

For the best model (model 5), the inferred parameter posterior distribution of model 5 is shown in Figure 4.15, the estimated mean values is shown in Table 4.2. The pair-wise joint density distribution is shown in Figure 4.16. Most of the inferred posteriors show a bell-shaped density distribution with a single peak.. A thin and concentrated shape indicates a well-inferred parameter, e.g. $\mu\Phi$ and $\kappa_{\Phi\beta}$. Some distributions, however, are skewed to zero, e.g. μ_N and $s_{\alpha\Phi}$. Concentrations in the posteriors were presented, and using inter-quartiles (shaded interval in Figure 4.15) or mean values to summarise the posteriors is feasible in practice uses, e.g. simulating data (Figure 4.13).

By looking at the joint density plot (Figure 4.16), some correlated patterns are observed. For example, an obvious linear relationship is presented in the joint distribution of μ_β and $s_{\beta\Phi}$. This leads us to consider that we could possibly reduce the number of parameters, e.g. replace $s_{\beta\Phi}$ by $A\mu_\beta + B$ where A and B can be calculated from the joint distribution by fitting a linear regression. The reason for the highly correlated pattern, or 'fat' distribution shapes could be that the provided data are lees informative about the parameters.

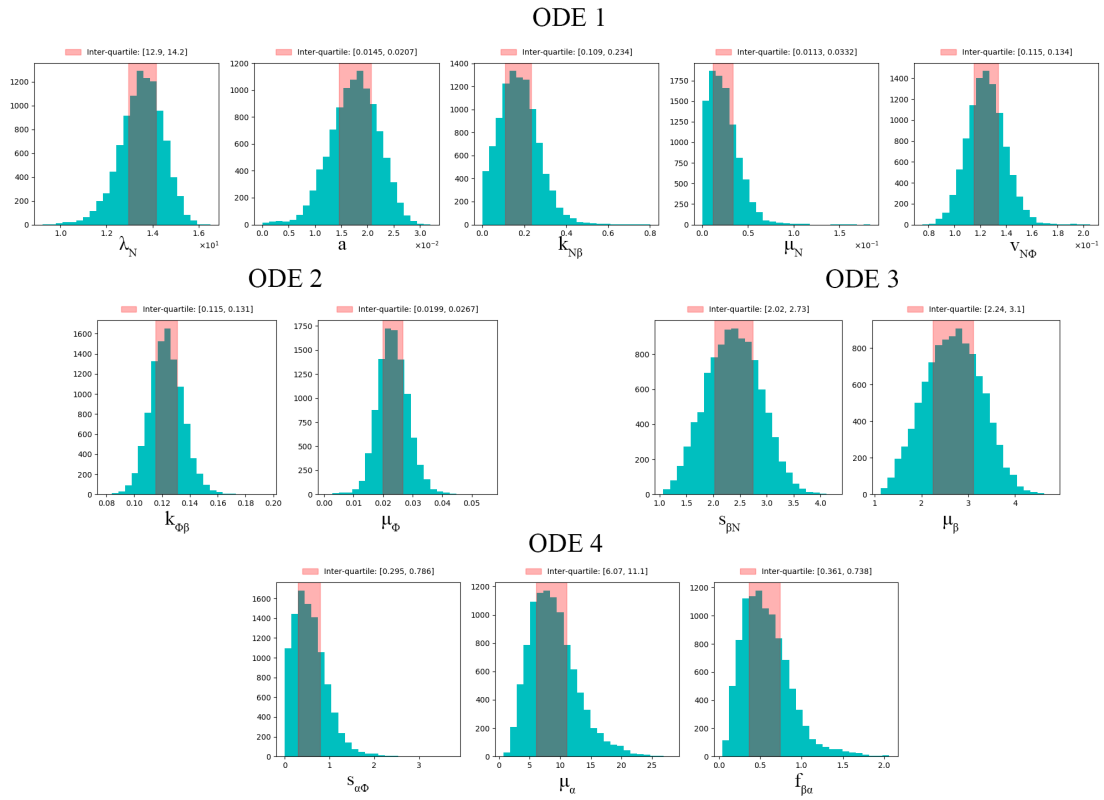

Figure 4.15: Inferred posterior distribution of parameters in model 5. Shaded range indicates inter-quartile interval of the population

Parameter	Mean value	Inter-quartile interval
λ_N	13.5	[12.9, 14.2]
a	0.0175	[0.0145, 0.0207]
$\kappa_{N\beta}$	0.176	[0.109, 0.234]
μ_N	0.0239	[0.0113, 0.0332]
$\nu_{N\Phi}$	0.125	[0.115, 0.134]
$\kappa_{\Phi\beta}$	0.123	[0.115, 0.131]
μ_Φ	0.0234	[0.0199, 0.0267]
$s_{\beta N}$	2.38	[2.02, 2.73]
μ_β	2.67	[2.24, 3.10]
$s_{\alpha\Phi}$	0.574	[0.295, 0.786]
μ_α	8.91	[6.07, 11.1]
$f_{\beta\alpha}$	0.577	[0.361, 0.738]

Table 4.2: Inferred parameter values of model 5

4.3.3 Sensitivity of parameters

A quick sensitivity analysis of the dynamical systems models can be quantified using principal component analysis (PCA) [8], where the first PCs corresponds to sloppy parameters while the last PCs corresponds to stiff parameters [20]. PCA can only provide rough approximated sensitivity behaviour. By identifying the component loadings, the PCs can be visualised in pie charts and used to indicate the sensitivity.

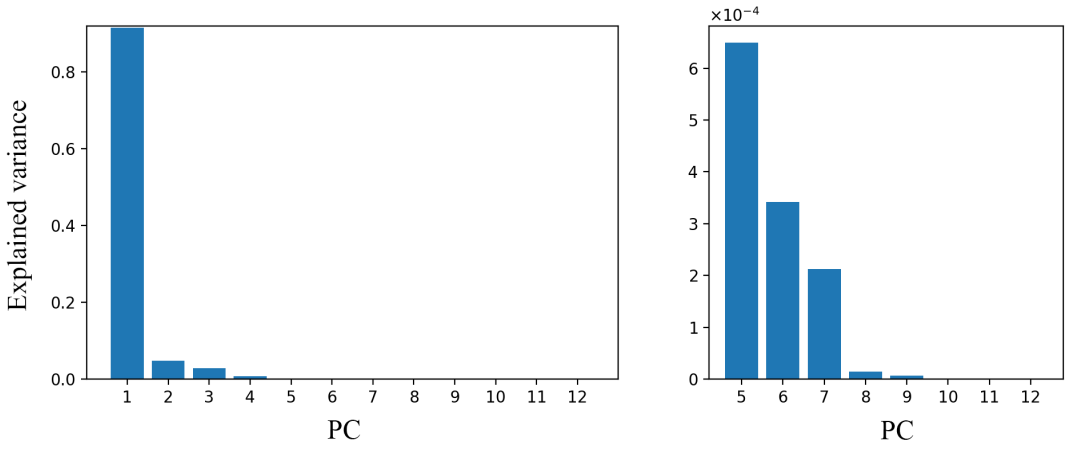

Figure 4.17: Explained total variance in original parameters by each PC. From PC 5 on. little variance is explained by each PC, which is enlarged in Right

As models 5 won in the model selection, further interest was to explore the parameter-to-model sensitivity of the resultant model. PCA was performed on the last generation of inferred model 5, Figure 4.17 shows the total explained variance by each PC. The first PC explained a major part of the variance (91.5%), and the rest PCs explained less than 5% of the variance each. PC 5 to PC 12 explained much less variance (less than 0.1% of the total variance) than the first four PCs. Thus, they were considered to extend much narrower regions on the posterior distributions and could be used to identify the parameters that the model is sensitive to.

Figure 4.18 shows the fractions of the PCs explained by individual parameters, and a sum of the fractions in the last 8 PCs, i.e. PC 5 to PC 12. The last PC (PC12) is mostly the linear combination of a (from the equation of neutrophil) and μ_Φ (from the equation of macrophage), to which the model is most sensitive. If we conclude the last 8 PCs, it can be found that the biggest portions ($v_{N\Phi}$, a, $\kappa_{N\beta}$, $\kappa_{\Phi\beta}$, μ_N, μ_Φ, $\nu_{N\Phi}$) mostly come from the equation of neutrophil and macrophage i.e. equations of the cells (the unexploded portion in Figure 4.18 a); if we look at the last 8 PCs individually, the same pattern can be observed. This might suggest that the model is more sensitive to cells' dynamic rather than cytokines' dynamics.

The credible intervals (Figure B.3) of each parameter across generations also agreed with the above conclusion from PCA. The stiff parameters, e.g. a and μ_N, extend

narrow credible intervals, while sloppy parameters, e.g. μ_α and $s_{\alpha\Phi}$, have wider credible intervals across the last few generations.

To conclude, model 5 is most sensitive to the parameters in the equations for cells' dynamics (i.e. first two equations in Eqn.3.5), especially to a and μ_Φ.

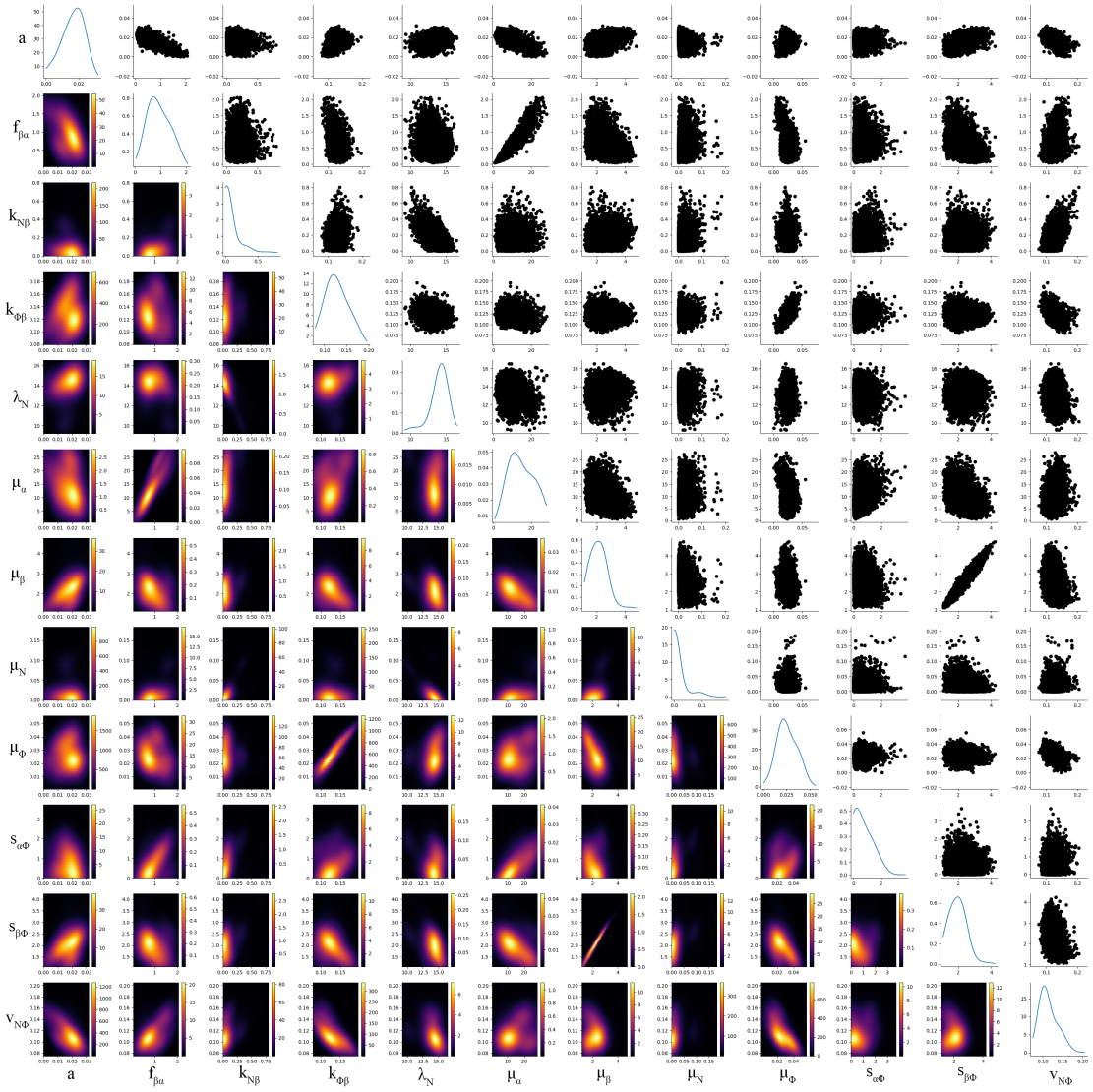

Figure 4.16: Joint distribution of each parameter pair. The diagonal is KDE fit of each individual parameter, the upper triangle contains scatter plot of the particles and the lower triangle is the joint density distribution for each parameter pair

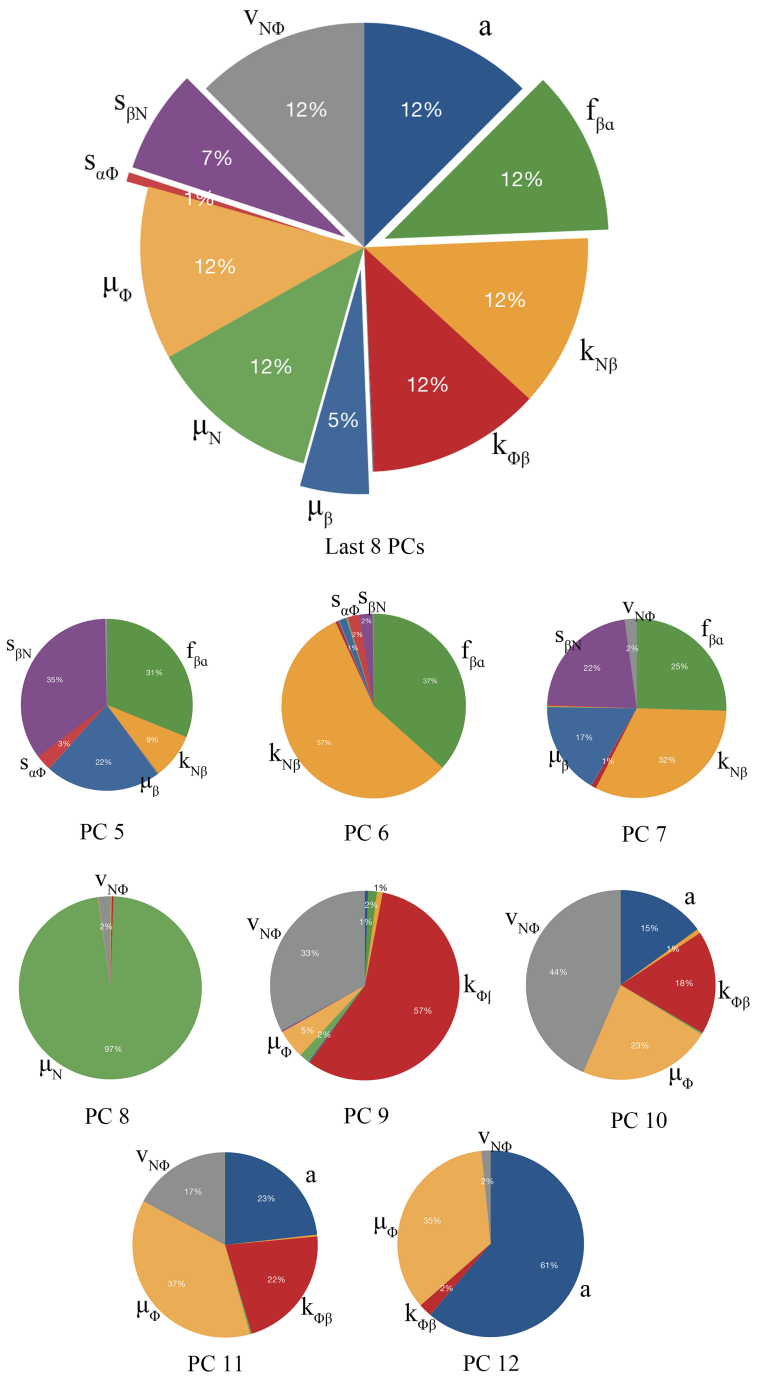

Figure 4.18: Fraction of the length of PCs explained by parameters. Portions less than 2% are not labelled

Chapter 5

Performance experiments

The performance experiments were designed to explore the parallel performance of ABC SMC inference framework that was used in our parameter estimation and model selection tasks. Generally, ABC SMC is a time-consuming and computation-intensive task and ideally executed on large clusters. The scheduling strategy, implementation details, randomness in the algorithm and many other factors can affect the parallel efficiency. Thus, we designed a scaling-up experiment and studied a case where abnormal performance was observed.

5.1 Scaling-up

First experiments are designed to illustrate the scaling-up performance. The program used here is an implementation of ABC SMC on model 5. The details of the ABC SMC settings are listed below

- Prior distribution: default to log-uniform distribution $[1 \times 10^{-6}, 50]$ for all the 12 parameters
- threshold schedule: median epsilon
- No factors, no adaptive distance or adaptive population applied
- Population size is 2000, with 20 generations

For HPC systems like Cirrus, `pyabc` uses `multiprocessing` for multi-core parallel sampling. By default, if the number of cores is not specified, it will automatically read the number of available cores and use them all. Cirrus has a 36-core CPU which support hyperthreading, such that the maximal number of cores available to `multiprocessing` is 72.

The experiment was executed on Cirrus, using 8, 16, 24, 36, 54 and 72 cores respectively. Each run was repeated ten times and the average execution time, required sampling numbers are recorded. Hyperthreading was enabled when using 54 and 72 cores.

The access to the node that contains computation cores is exclusive, such that the execution would not be affected by other programs of operations.

The implementation of ABC SMC in `pyabc` enables the parallelisation of sampling, which is the most time-consuming part. The rest part of the program is mostly not parallelised, e.g. database I/O and reductions operations. The sampling process involves sampling, perturbation and test of the acceptance criteria, which are computation-intensive.

In practice, using ABC SMC to estimate the parameters of a given model could cost up to several hundred of hours if the computational resources are limited [17]. The performance experiment result could provide a reference that illustrates how the efficiency changes when scaling-up or the trade-offs in computational resources' cost and their benefit.

Results

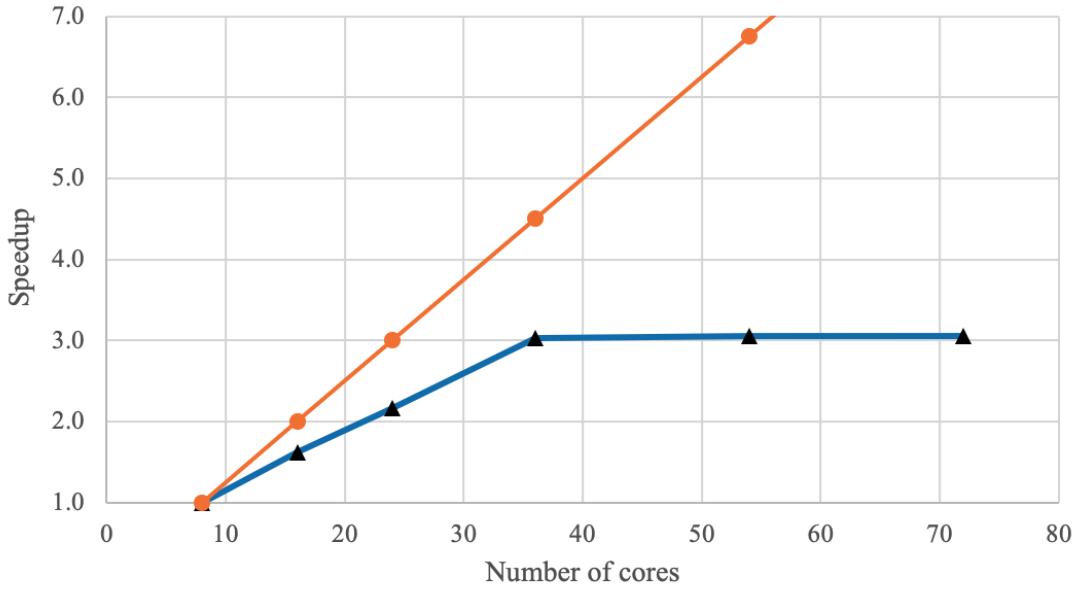

Figure 5.1: Speedup of the program. Line in orange indicates the ideal linear speedup

The recorded execution time under different numbers of cores was used to calculate the speedup of the program (Figure 5.1). Here 8-core run is regarded as the baseline for speedup and efficiency, as the serial version (using only one core) can take quite a long time to finish 10 repeats. When increasing number of cores, the execution time drops fast at first, but the decreasing trend became slower for large numbers of cores; 36, 54 and 72 cores gives nearly the same average execution time and consequently gains very close speedup values (Table 5.1).

Number of cores	Avg. time (second)	Avg. sample size	Speedup	Efficiency
8	6617	445406.8	1	1
16	3926	590221.0	1.61	0.807
24	2921	573339.8	2.17	0.723
36	2096	446948.7	3.02	0.672
54	2076	640404.9	3.05	0.452
72	2071	768096.0	3.06	0.340

Table 5.1: Performance data

Denote the number of cores used in the ABC SMC runs as variable p. The speedup curve shows a linear trend when $p \leq 36$; when $p \geq 36$ the speedup stays nearly unchanged. $p = 36$ is an inflexion point, where 36 is the maximum numbers available physical cores in a compute node of Cirrus. A constant drop of efficiency is observed when increasing p; smaller p ($p < 36$) gives an efficiency higher than 65% but greater p e.g. $p = 72$ results in a low efficiency (34%).

5.2 Discussions

A high variance of total required samples was observed in the scaling-up experiments: some single runs required much more samples to finish 20 populations, where they were supposed to have relatively similar average required samples such that we can compare the performance directly via speedup or efficiency.

Due to the variant total required samples drawn our attention to the per-second performance. Although the execution time was not improved by using 54 or 72 cores, the required number of samples were higher than that of $p = 36$, which indicates slight improvements. However, the high variance made us hard to gave statistically significant conclusions on the per-second performance.

The variant total required samples for different runs under the same setting was most likely the result of different threshold evolve path. Median epsilon schedule and randomness in the sampling process resulted in different sequences of thresholds for different runs, which means different repeated runs may have different convergency path [12]. For example, Some runs temporarily converge to a local optimum, while the target epsilon in the next generation cannot possibly be reached if the new particles are still concentrated in the local optimum. In this case, a huge amount of sample will be drawn to find enough accepted particles, and the new generation will be able to jump out of the local optimum and have different concentrations.

To illustrate this, Two independent run, namely Run A and Run B, were chosen from the experiment of $p = 24$ and visualised in Figure 5.2 and Figure 5.3. These two runs used identical options of the algorithm. Figure 5.2 (a) shows the comparison of the required samples, where Run A converged to a local mode, and in population 11 took much

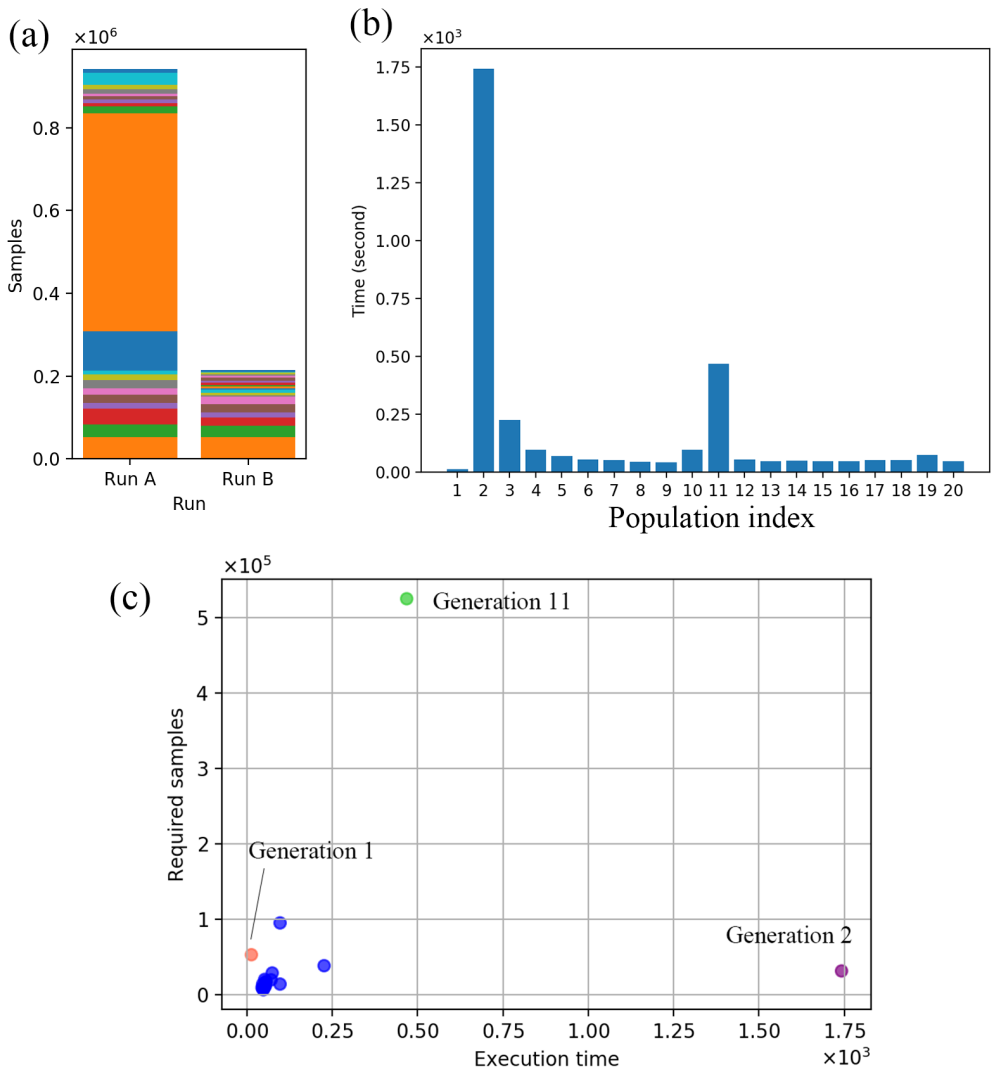

Figure 5.2: (a) Required number of samples for two example runs (A and B). Different colour represents different generation (from bottom to top: generation 1 to generation 20). Run A took huge amount of time in population 11. (b) Execution time for each generation of Run A. (c) Scatter plot of required number of samples against execution time for each generation. Generation 1 and two outliers, i.e. generation 11 and generation 2, are marked in different colours from other generations

more sample tires to move away from the local optimum (Figure 5.2 (b)). Figure 5.3 compares the posterior density distribution before (t=9, cyan) and after (t=13, red) the jumping away: significant movement of concentrations in posterior density distributions are observed, especially in parameters from ODE 1, 2 and 4. An interning movement is that there are two concentrations in $\kappa_{\Phi\beta}$ and μ_{Φ} when t=9, and the majority of particles are concentrated in the second concentration with high parameter values; when t=13, the second concentration distribution disappears and all particles are concentrated near zero.

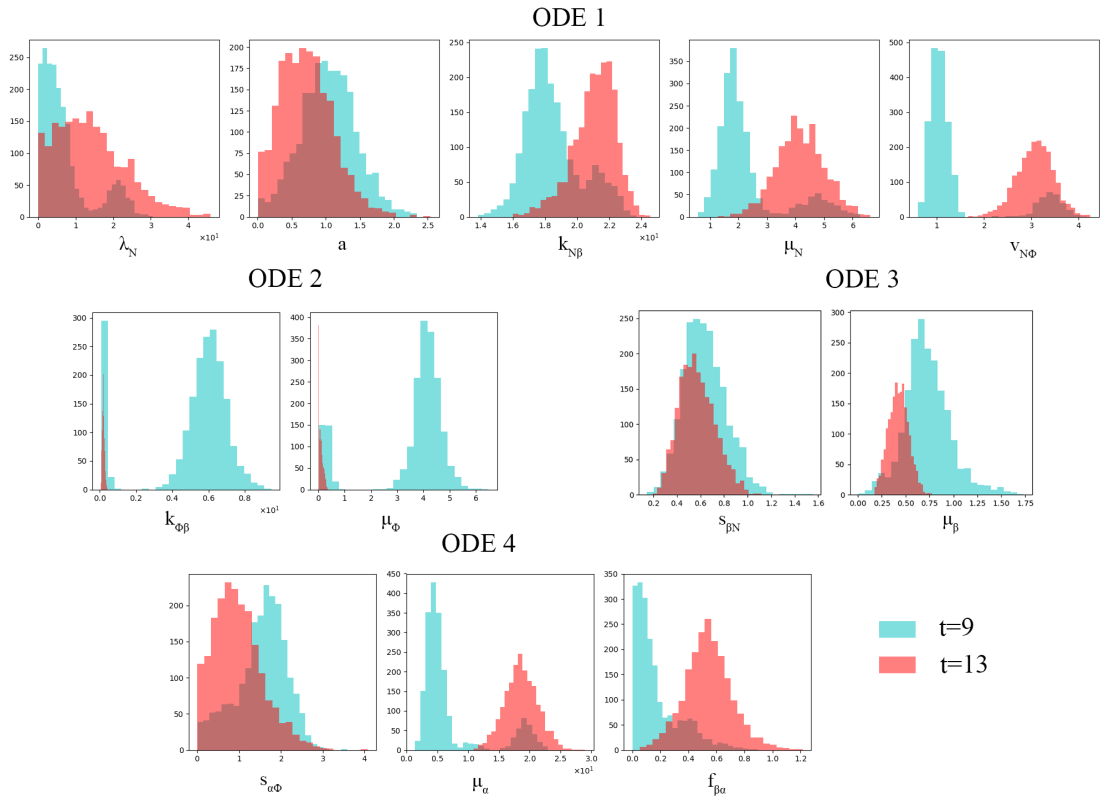

Figure 5.3: Parameter density distribution before (generation 9, cyan) and after (generation 13, red) the concentration movement in generation 11

Moreover, we found that the execution time was not ideally proportional to the sampling tires in that period of time, for both different repeated runs and different generations in a single run. From the same example, the sample size for each generation of Run A (Figure 5.2 (a)) is not proportional to its execution time (Figure 5.2 (b)), especially for the first two generations and generation 11. Figure 5.2 (c) shows a scatter plot of the required samples and execution time for each generation, where two obvious outliers (generation 1 and generation 11) are observed.

Generation 1 (red pont in Figure 5.2 (c)) drew samples from prior, but it was done with high efficiency as only a small amount of time are taken; generation 2 (purple pont in Figure 5.2 (c)) took an unusually large amount of the while it did not require much more samples. The abnormal execution time of generation 1 and 2 were observed in all other runs, where for $p = 24$ generation 1 usually takes c. 16s and generation 2 takes c. 1600s. This is believed to be related to the initialisation of the algorithm. For local modes, e.g. generation 11 (green pont in Figure 5.2 (c)), high efficiency were achieved.

By looking into the code, some reasons can be found. There exist some 'cheap' particles. If at the first time points, a variable reached unacceptable high value (regarded as `Inf` in `Python`), the ODE solver cannot perform further integration but result in a data full of `NaNs`. Such particles cost less time than an usual particle (where values at

120 data points are explicitly calculated and then compared with observed data). Local mode, e.g. generation 11, might take large amount of these 'cheap' particles. Also, generation 1 draws samples directly from prior without fitting and using kernels, so it is much faster. The huge amount of time observed in generation 2 is believed to be a result of some initialisation steps in the ABC SMC algorithm.

To conclude, the performance experiments revealed reasonable scaling-up behaviour when the `Python` processes did not exceed number of physical cores. Local modes lead to high variance in total required number of samples, and the relationship between required sample numbers and required execution time are not ideally linear; they all made further detailed analysis difficult.

Additionally, by observing CPU frequency and load curve in-time while running the program, we found that the parallelised part is limited to the sampling, calculation and criterion test process. Some reduction calculations and preparation work for the next generation (e.g. calculation of the new epsilon values, normalisation of data, fit of multivariate kernel) are serial. The database I/O is also serial. It is noticed that the program took no advantage of graphics card, as `multiprocessing` along can only exploit the resources inside CPU.

Chapter 6

Conclusion and future works

To conclude, ABC SMC was successfully applied to the models of zebrafish spinal cord regeneration. These models were all ODE equations that described the interactions and effects between cells and proteins in the lesion site. These equations contain four dependent variable and more than 10 model parameters. Parameter estimation for high-dimensional parameter space is a challenging work for many dynamic system models, as insufficient explore of parameter space can easily lead to local optimal. Using a large population size in ABC SMC in our models could partially relief the concerns in local modes, and give considerable parameter estimates. We found that the first three models could not recover some local features observed in the early times, so further models – model 4 and 5 – were proposed. Additional interactions were introduced in the new model, and the results proved that they were helpful in resolving the under-fit.

Also, ABC SMC successfully helped in the model comparison and thus can be used for suggestions of the best model, although there were uncertainties remained in the process and a large number of population size needed. In our practice, model 5 won with absolute advantages when the threshold is converged to a small value. The approximated posteriors showed bell-shaped density distribution for most of the parameters (Figure 4.15 and Figure 4.9), indicating that most parameters were well-inferred in our implementations.

The inference program was mostly run on multi-core machines, and additional scaling-up performance test illustrated the scaling behaviour.

When the number of running `Python` processes is less than the number of physical cores, a considerable scaling performance was observed. Studies on the variant required sample numbers revealed that some threshold paths could lead to local optimum and significantly affect the efficiency. A non-linear relationship between the required sample numbers and required execution time was also observed and made the detailed analysis complicated.

From this application, some suggestions on ABC SMC implementation could be drawn.The prior distribution and interval of each parameter can affect the inference results signif-

icantly, and improper prior cold lead to poor fit. In our cases, log-uniform prior gave better fits than uniform prior. The threshold path and plot of required samples in each generation can be used to identify possible local modes; if stuck, it takes much more samples than usual to move out. To avoid this, large population size and a proper epsilon schedule may help. For the uncertainties observed in model selection and some duplicated runs with high variance, a longer run with more generation could help to identify whether the convergency is consistent such that a reproducible inferred results can be obtained.

In terms of the project progress regarding original project proposal, all the proposed subjects were explicitly studied as planned. We found that code development cost less time than expected, however, implementations and debugging cost more efforts than expected because of both uncertainties in both the inference ability of the algorithm for our problem, and the implementation options and algorithm hyperparameters. Further on our results, we studied some in-depth topics and derived more conclusions. Some interesting further topics were found in the studies, e.g. comparison with other algorithms and implementation optimisations, however, they were not conducted due the insufficient time.

Also, we found that risk analysis and management throughout the project were helpful. Thanks to the planned mitigations and backup strategies, we were able to proceed with our project as planned when interruptions happened. We experienced local laptop broken and unavailability of remote machines, which consequently made our development and implementations harder. We were able to proceed with the project by using backup plans for local development and using Cirrus or ARCHER interchangeably during their downtime.

In terms of the future works, some more comparison with other algorisms could be of our interest. Least square fitting using MCMC [6] and some other exact inference approaches could be performed on the same model and observed data, as the standard deviation at each measurement point is known. By comparing to these methods, differences in the inference results, algorithm robustness and required computational resources could be revealed and possibly help us in choosing an inference algorism that can balance the trade-off between efficiency and a reasonable good 'fit'. Also there other implementation options of ABC SMC worth trying, e.g. CUDA acceleration and `Julia` implementations[1] [2].

In the aspect of models, some further models can be studied. If there were some more experimental evidence on the proposed interaction map, we could propose more precise models, or calibrate some terms in our existing models. Also, models of other forms, e.g. PDE and stochastic models, could also be studied if a more complex model is needed for the dynamic system. Simplification is also possible of existing models. Some highly-correlated parameters could be reduced, as there exist some linear relationships between parameters.

[1] `https://github.com/tanhevg/GpABC.jl`
[2] `https://github.com/marcjwilliams1/ApproxBayes.jl`

Appendix A

System and software environment

A.1 ABC SMC implementation

A.1.1 Local machine

Local development machine is a Mac laptop, running on macOS 10.15.6. The environment of the development is listed in Table A.1.

Environment	Version
Operating system	macOS Catalina 10.15.6
PyCharm (IDE)	Professional 2020.2
Python interpretor	3.7.0
IPython	7.12.0
Clang	6.0 (clang-600.0.57)

Table A.1: Environment on local machine

Version requirements for some critical site packages (`Python`) used in the code are listed in Table A.2. When replicating the experiments, higher version of these packages are acceptable, except `bokeh`.

Local development and some runs of small size were done under the hardware listed in Table A.3. When running ABC SMC, the program can make used of 8 python process (Hyper-Threading enabled) and run under 3.8GHz (maximal) for a long time under proper cooling, proved that the program could efficiently use the computation resources for a local personal computer. The execution time and other performance data presented in Chapter 5 were obtained using remote machines but not local machine.

Environment	Version
pyABC	0.10.3
NumPy	1.18.4
SciPy	1.4.1
pandas	1.1.0
matplotlib	3.0.1
bokeh	1.4.0

Table A.2: Site packages on local machine

Hardware	Detail
CPU	Quad-Core Intel Core i5 8259U 2.3 GHz
Architecture	64 Bit
Hyper-Threading	Supported
Turbo Boost	Max 3.8 GHz
Memory	16 GB
Graphics card	Iris Plus Graphics 655

Table A.3: Hardwares on local machine

A.1.2 Remote machine

Most experiments were performaed on Cirrus, its hardware can be found in Table A.4. The program can run in sigle compute-node which contains two processors[1]. The software environment used in the development and experiments are listed in Table A.5. Requirement of site packages is the same as that on local macine (Table A.2)

Environment	Version
Operating system	Red Hat Enterprise Linux 8.1
miniconda environment	4.8.3
Python interpreter	3.7.7
gcc	6.3.0

Table A.5: Environment on local machine

[1] `https://www.cirrus.ac.uk/about/hardware.html`

Hardware	Detail
Processors	2.1 GHz, 18-core Intel Xeon E5-2695 (Broadwell)
Architecture	x86-64
Hyper-Threading	Supported
Memory	256 GB NUMA, shared between two processors

Table A.4: Cirrus hardware (compute node)

Appendix B

Supplementary figures and data

Parameter	Value
λ_N	2.20
$\kappa_{N\beta}$	3.96
μ_N	1.72
$\nu_{N\Phi}$	0.219
λ_Φ	1.31
$\kappa_{\Phi\beta}$	0.124
μ_Φ	0.145
$s_{\beta N}$	6.55
$i_{\beta\Phi}$	1.71
μ_β	0.521
$s_{\alpha\Phi}$	10.2
μ_α	19.7

Table B.1: Parameter values used to generate synthetic data for model 1. They were obtained from a preliminarily least-square fitting

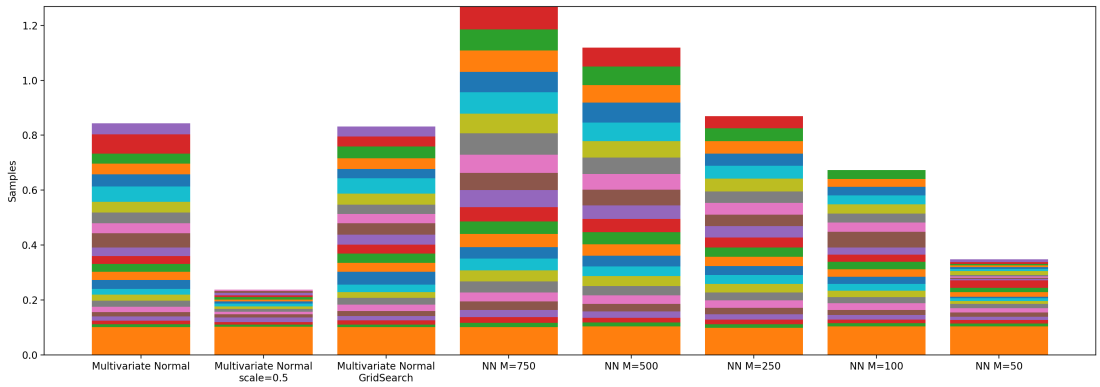

Figure B.1: Total sampling size of different kernels, using median epsilon strategy. Different color represents different generations (bottom to top: population 1 to population 20)

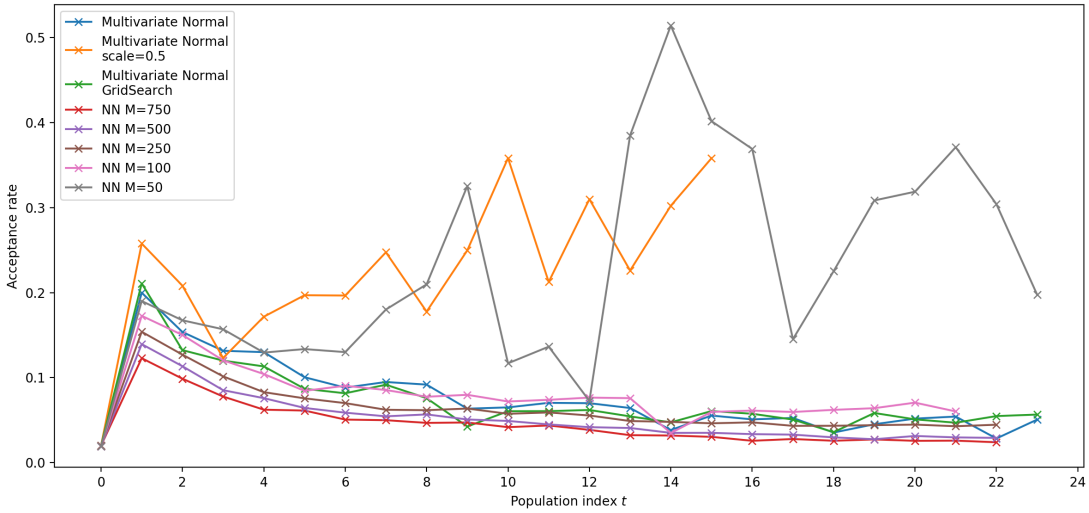

Figure B.2: Acceptance rates of different kernels, using median epsilon strategy. Each population has 2000 particles. The total length of populations is different because some runs reached the final ϵ_T earlier than others

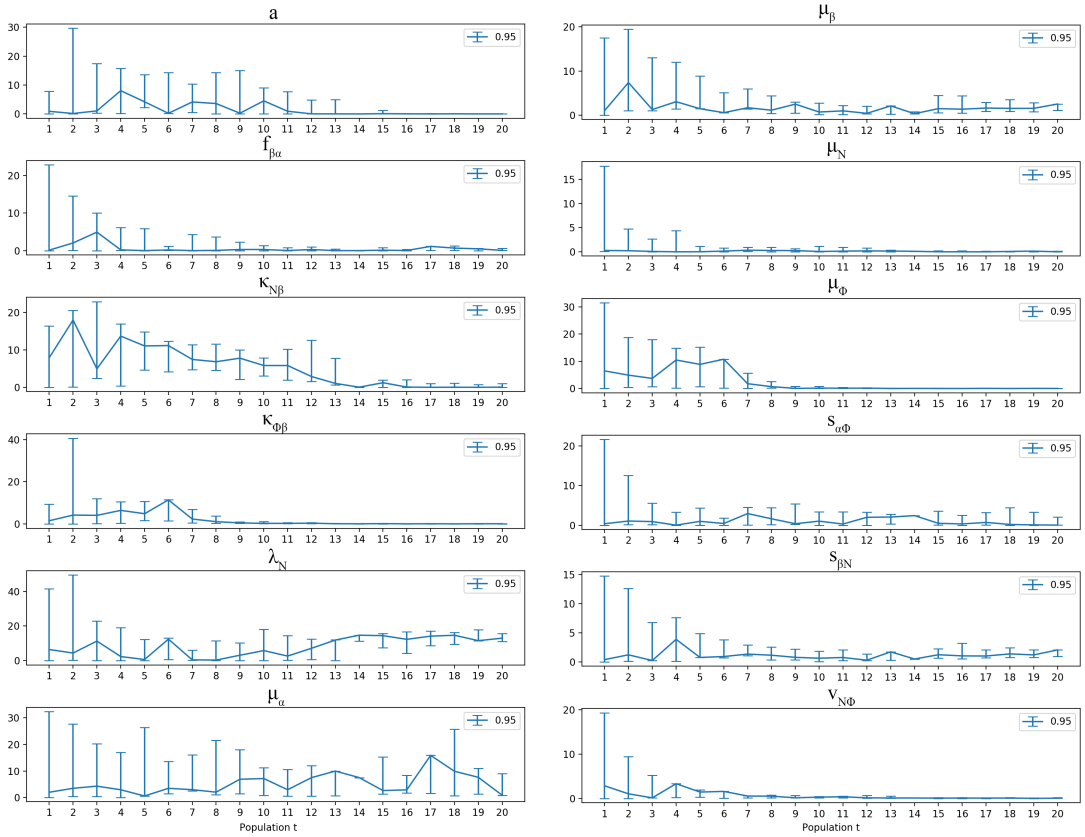

Figure B.3: Credible intervals of parameters in model 5 across all generations

Bibliography

[1] T. M. Tsarouchas, D. Wehner, L. Cavone, T. Munir, M. Keatinge, M. Lambertus, A. Underhill, T. Barrett, E. Kassapis, N. Ogryzko, *et al.*, "Dynamic control of proinflammatory cytokines il-1β and tnf-α by macrophages in zebrafish spinal cord regeneration," *Nature communications*, vol. 9, no. 1, pp. 1–17, 2018.

[2] J. Liepe, P. Kirk, S. Filippi, T. Toni, C. P. Barnes, and M. P. Stumpf, "A framework for parameter estimation and model selection from experimental data in systems biology using approximate bayesian computation," *Nature protocols*, vol. 9, no. 2, pp. 439–456, 2014.

[3] T. Becker, M. F. Wullimann, C. G. Becker, R. R. Bernhardt, and M. Schachner, "Axonal regrowth after spinal cord transection in adult zebrafish," *Journal of Comparative Neurology*, vol. 377, no. 4, pp. 577–595, 1997.

[4] W. D. Anderson, H. K. Makadia, A. D. Greenhalgh, J. S. Schwaber, S. David, and R. Vadigepalli, "Computational modeling of cytokine signaling in microglia," *Molecular BioSystems*, vol. 11, no. 12, pp. 3332–3346, 2015.

[5] J. R. Dormand and P. J. Prince, "A family of embedded runge-kutta formulae," *Journal of computational and applied mathematics*, vol. 6, no. 1, pp. 19–26, 1980.

[6] W. Gilks, S. Richardson, and D. Spiegelhalter, "Markov chain monte carlo in practice chapman & hall: London," *Markov Chain Monte Carlo in practice. Chapman and Hall, London.*, 1996.

[7] S. Tavaré, D. J. Balding, R. C. Griffiths, and P. Donnelly, "Inferring coalescence times from dna sequence data," *Genetics*, vol. 145, no. 2, pp. 505–518, 1997.

[8] T. Toni, D. Welch, N. Strelkowa, A. Ipsen, and M. P. Stumpf, "Approximate bayesian computation scheme for parameter inference and model selection in dynamical systems," *Journal of the Royal Society Interface*, vol. 6, no. 31, pp. 187–202, 2009.

[9] P. Joyce and P. Marjoram, "Approximately sufficient statistics and bayesian computation," *Statistical applications in genetics and molecular biology*, vol. 7, no. 1, 2008.

[10] M. A. Nunes and D. J. Balding, "On optimal selection of summary statistics for approximate bayesian computation," *Statistical applications in genetics and molecular biology*, vol. 9, no. 1, 2010.

[11] A. Minter and R. Retkute, "Approximate bayesian computation for infectious disease modelling," *Epidemics*, vol. 29, p. 100368, 2019.

[12] D. Silk, S. Filippi, and M. P. Stumpf, "Optimizing threshold-schedules for approximate bayesian computation sequential monte carlo samplers: applications to molecular systems," *arXiv preprint arXiv:1210.3296*, 2012.

[13] E. Klinger and J. Hasenauer, "A scheme for adaptive selection of population sizes in approximate bayesian computation-sequential monte carlo," in *International Conference on Computational Methods in Systems Biology*, pp. 128–144, Springer, 2017.

[14] D. Prangle *et al.*, "Adapting the abc distance function," *Bayesian Analysis*, vol. 12, no. 1, pp. 289–309, 2017.

[15] S. Filippi, C. P. Barnes, J. Cornebise, and M. P. Stumpf, "On optimality of kernels for approximate bayesian computation using sequential monte carlo," *Statistical applications in genetics and molecular biology*, vol. 12, no. 1, pp. 87–107, 2013.

[16] T. Toni and M. P. Stumpf, "Simulation-based model selection for dynamical systems in systems and population biology," *Bioinformatics*, vol. 26, no. 1, pp. 104–110, 2010.

[17] A. C. Daly, J. Cooper, D. J. Gavaghan, and C. Holmes, "Comparing two sequential monte carlo samplers for exact and approximate bayesian inference on biological models," *Journal of The Royal Society Interface*, vol. 14, no. 134, p. 20170340, 2017.

[18] A. Weyant, C. Schafer, and W. M. Wood-Vasey, "Likelihood-free cosmological inference with type ia supernovae: approximate bayesian computation for a complete treatment of uncertainty," *The Astrophysical Journal*, vol. 764, no. 2, p. 116, 2013.

[19] E. Klinger, D. Rickert, and J. Hasenauer, "pyabc: distributed, likelihood-free inference," *Bioinformatics*, vol. 34, no. 20, pp. 3591–3593, 2018.

[20] R. N. Gutenkunst, J. J. Waterfall, F. P. Casey, K. S. Brown, C. R. Myers, and J. P. Sethna, "Universally sloppy parameter sensitivities in systems biology models," *PLoS Comput Biol*, vol. 3, no. 10, p. e189, 2007.