Academic year 2007

Computing project
Group report

SBML-ABC, a package for data simulation, parameter inference and model selection

Imperial College- Division of Molecular Bioscience

MSc. Bioinformatics

Nathan Harmston, David Knowles, Sarah Langley, Hang Phan

Supervisor:

Professor Michael Stumpf

June 13, 2008

Contents

1 Introduction **1**
1.1 Background . . . 1

2 Features and dependencies **2**
2.1 Project outline . . . 2
2.2 Key features . . . 2
2.2.1 SBML . . . 2
2.2.2 Stochastic simulation . . . 4
2.2.3 Deterministic simulation . . . 4
2.2.4 ABC inference . . . 5
2.3 Interfaces . . . 5
2.3.1 C API . . . 5
2.3.2 Command line interface . . . 5
2.3.3 Python . . . 5
2.3.4 R . . . 6
2.4 Dependencies . . . 6

3 Methods **8**
3.1 SBML Adaptor . . . 8
3.2 Stochastic simulation . . . 8
3.2.1 Random number generator . . . 9
3.2.2 Multicompartmental Gillespie algorithm . . . 9
3.2.3 Tau leaping . . . 10
3.2.4 Chemical Langevin Equation . . . 11
3.3 Deterministic algorithms . . . 11
3.3.1 ODE solvers . . . 11
3.3.2 Time Delay Differential Equation solver . . . 11
3.4 Parameter inference . . . 12
3.4.1 ABC rejection . . . 12
3.4.2 ABC MCMC . . . 12
3.4.3 ABC SMC parameter inference . . . 12
3.5 ABC SMC model selection . . . 13
3.6 Interfaces . . . 14
3.6.1 Command line interface . . . 14
3.6.2 C API . . . 14

3.6.3 Python - C Binding . . . 15
3.6.4 R interface . . . 16

4 Examples and results **19**
4.1 Data simulation . . . 19
4.1.1 Lotka-Volterra model . . . 19
4.1.2 Repressilator . . . 20
4.1.3 Brusselator model . . . 20
4.2 Parameter inference . . . 21
4.2.1 ABC rejection . . . 21
4.2.2 ABC MCMC . . . 21
4.2.3 ABC SMC . . . 25
4.3 Model selection . . . 25

5 Future work **28**

List of Figures

2.1 Flowchart of SBML-ABC package 3

3.1 Screen shot of instruction on how to use the the program $sbmlabc$ 18

4.1 Lotka-Volterra simulated using Gillespie's algorithm. 19
4.2 Repressilator simulated using ODE solver. 20
4.3 Feedback loops present in repressilator 20
4.4 Simulations of the Brusselator model using different algorithms. 22
4.5 Results of ABC rejection on Brusselator model using ODE solver. 23
4.6 Observed data (blue) and accepted simulated data (gray lines). 23
4.7 Results of ABC MCMC on Brusselator model using ODE solver. 24
4.8 Histograms of log(distance) for all simulated datasets. 24
4.9 Parameter 1 vs. 4 from ABC SMC with 8 populations, 1000 particles on Brusselator using ODE solver. Populations are shown added two per plot. 26
4.10 Acceptance ratio against ϵ for each population of ABC SMC. 27

Chapter 1

Introduction

We present a computational package for the simulation of dynamical systems and the inference of parameters in these models based on experimental time course data. Models are imported using the Systems Biology Markup Language (SBML) [1] and simulated using a range of deterministic and stochastic algorithms. The main contribution of this project however is to provide an accessible package to infer feasible model parameters using Approximate Bayesian Computation methods [2], including ABC rejection, Markov Chain Monte Carlo, and Sequential Monte Carlo.

1.1 Background

Several packages exist to search the parameter space of a dynamical model to find the "optimum" values. The complex pathway simulator Copasi [3], has a range of stochastic optimisation methods which use steady state or time course experimental data and deterministic or stochastic simulations. COPASI supports import of Systems Biology Markup Language (SBML) [1] models and provides the choice of a command line, graphical user interface or C Application Programming Interface (API). SBML-PET [4] is an alternative package which performs stochastic ranking evolutionary search (SRES) [5] based on the ODEPACK [6] solver LSODAR. A key advantage of this package is its ability to handle SBML events.

The key flaw with existing packages is the lack of any consideration of the range of feasible parameters. In a Bayesian framework we would ideally like to do this by calculating the posterior over the parameters given the observed data. Since we cannot directly evaluate the likelihood function in complex biological models, we employ the Approximate Bayesian Computation framework [2] where we compare data simulated using various parameter values to observed data. This framework provides a theoretical foundation which is lacking from existing methods, where the meaning of "best parameters" is ill-defined. The samples found can provide insight into the model behaviour and give confidence intervals for each parameter.

Chapter 2

Features and dependencies

In this chapter, we introduce the project, in terms of the key features, the four interfaces, and dependencies on specific packages and platforms.

2.1 Project outline

In order to provide a complete package for model inference, we implemented a three-faceted software package, consisting of modules for SBML parsing, stochastic and deterministic data simulation, and ABC inference. A schematic of the components is shown in Figure 2.1. Four distinct interfaces are available: a command line executable, a C API, and R and Python interfaces, providing options for users of any level of computing knowledge.

2.2 Key features

2.2.1 SBML

SBML is an xml-based markup language specifically designed to aid the "exchange and re-use of quantitative models" [1]. It is designed to be both computer and human readable and a number of packages exist which are able to take a model defined in an SBML file and perform simulation or other algorithms on the model. *LibSBML* [7] is a library designed specifically for reading, writing and modifying SBML models. This library is written in C and C++, although it has bindings available for a number of languages including, Python, Java, Perl and Ruby. It is designed to be portable and it has been ported to many different platforms including, Windows, Linux and Mac OS X. Many researchers have made the models they have produced freely available at `www.biomodels.org`, leading to a growing corpus of models for examination and testing of biological simulators. The models encoded within SBML can be extremely complex and SBML supports encoding of both stochastic and deterministic models. It is possible to encode not only cellular models within SBML but also population genetics models, although not many of this type are currently in use. There are many competitors to the SBML format including BioPAX and CellML, although BioPAX is primarily for the encoding of biological pathways rather than biological models.

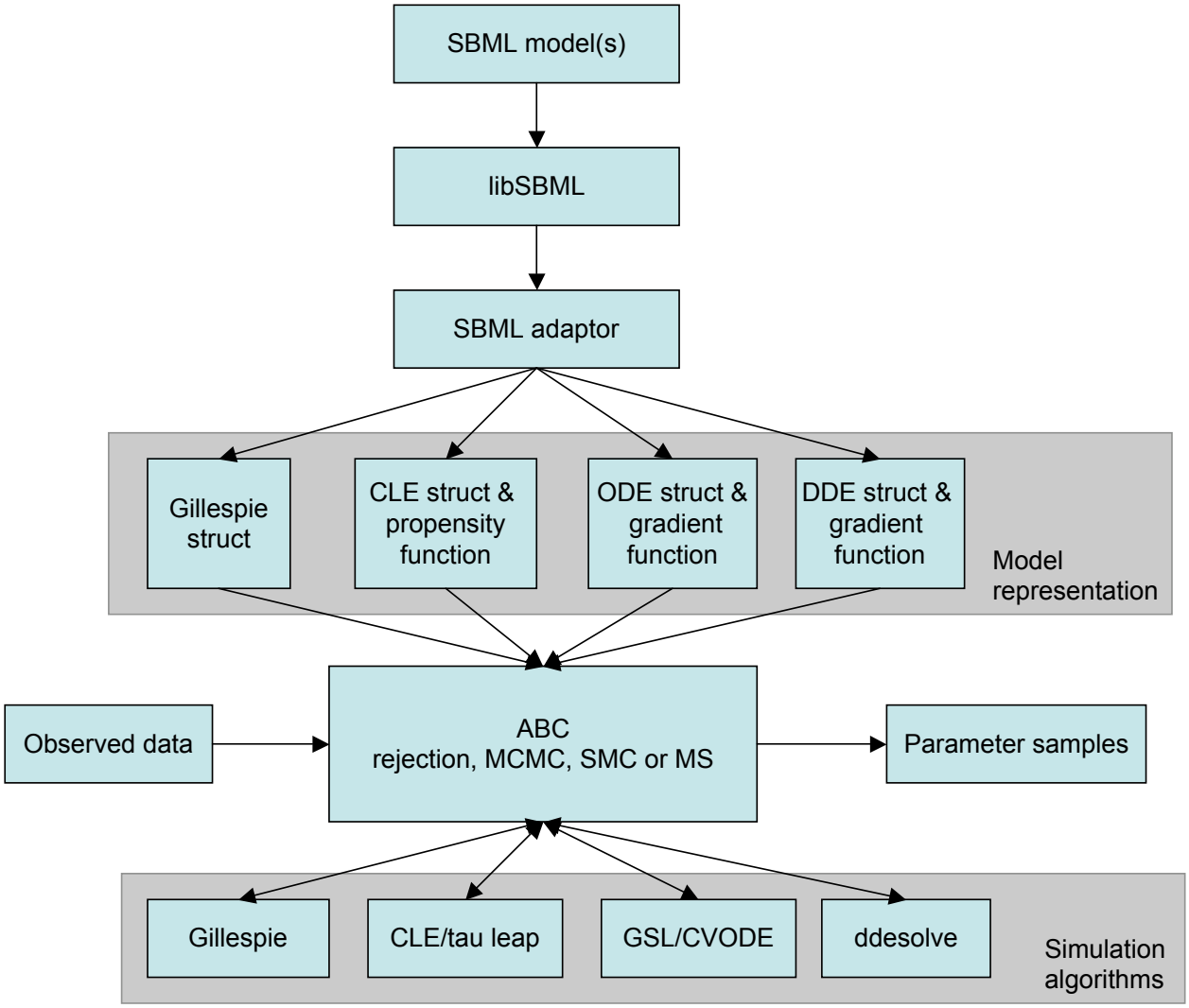


Figure 2.1: Flowchart of SBML-ABC package

Our software package supports the importing of biological models written in SBML and the ability to simulate these models either stochastically or deterministically and perform parameter inference on these models. The SBML-ABC package supports a subset of the current SBML specification. The main components of the specification are supported including reactions, compartments, species, parameters, kinetic laws and rate laws (differential equations). Currently the package only supports inference on global parameters and all parameters declared locally to a reaction will be used with the value as set within the list of parameters local to that kinetic law. Currently the simulation algorithms for CLE, DDE and ODE all can make use of generated shared libraries to enable faster running times of these algorithms. The generated libraries contain either a function containing the differential equations corresponding to each species or a representation of the kinetic law. The differential equations can either be generated using the kinetic laws which are set for each reaction or have a rate rule set for the species. In the case where kinetic laws are not defined for a model, the SBML adaptor will automatically generate a file containing the standard

mass action kinetic laws for the model. This process is not seen by the user. This auto-generation of the gradient function is only supported for ODE simulation algorithms. In the case of a user wanting to run a Gillespie simulation on a SBML model, the SBML file must be written with each of the global parameters corresponding to the reaction rate of a reaction, for example the rate of the first reaction in the SBML file should be encoded as the first global parameter.

A significant subset of MathML in the SBML specification is supported: all of the main arithmetic operations and trigonometric functions are implemented have been tested. However, logical and relational operations and custom function definitions are not supported.

2.2.2 Stochastic simulation

There are two analytical approaches to modeling biochemical systems: deterministic and stochastic. Deterministic methods are used when a system can be defined while neglecting random effects and species can viewed as continuous without drastically distorting the behavior of the system. Systems of ordinary and time-delay differential equations can then model the system to an acceptable accuracy. Stochastic methods describe systems where species are defined in discrete terms and failing to consider random effects can cause simulations to differ from observed results.

Stochastic simulation plays an important part in the understanding of biochemical systems of reactions including gene regulation networks, metabolic networks, and reaction systems where the deterministic approach is not suitable because very low particle numbers make stochastic effects significant.

The foundation of the stochastic simulation of biochemical system is the Gillespie Stochastic Simulation Algorithm (SSA) [8, 9] which is “exact” given some basic physical assumptions, such as having a ”well-mixed” system. Stochastic algorithms have seen significant improvements in speed during the past three decades allowing the simulation of more complex systems. Notable algorithms are the Gibson & Bruck’s next reaction method [10], chemical Langevin equation (CLE) [11], and tau leaping methods [12, 13, 14]. As the biological systems are complex, and it is vital to understand it closer to its natural form, a new computing system has been developed, the P-system [15] in 2000 by Paun Gh., abstracting the way the alive cells manipulate chemical substances in multi compartmental situations. The original Gillespie’s algorithm has been extended to simulate the behaviour of transmembrane P-systems, introduced as the multicompartmental Gillespie algorithm [16] where the movement of crossing the membrane from one compartment to the other is taken into account.

Three simulation algorithms are implemented in our project: tau leaping, the CLE method and the multicompartmental Gillespie algorithm (which is the SSA when the number of compartments is 1). The SSA, tau leaping and CLE method are widely available in many packages but the multicompartmental algorithm is not yet commonly used. The SSA is not implemented separately as the multicompartmental Gillespie algorithm reduces to the SSA when it is a single compartmental system.

2.2.3 Deterministic simulation

The Chemical Langevin Equation is equivalent to the Euler-Maruyama scheme for the numerical integration of stochastic differential equations (SDE), when applied to the chemical master equation. The system is therefore implicitly being represented as an SDE, which motivates the

approximation of such a system by a system of deterministic equations, i.e. ordinary differential equations (ODEs). Deterministic simulations have the advantage of being much faster than their stochastic counterparts, and the literature on the numerical integration of ODEs is very well developed. Runge-Kutta schemes are the *de facto* solution, and coupled with adaptive step size control, which ensures the local error remains below a user specified tolerance, they provide a powerful tool for rapid data simulation in situations where particle numbers are sufficiently large to allow us to ignore stochastic effects. Runge-Kutta-Fehlberg 45 is the default solver in our package, but more advanced solvers are availablse through the GSL and CVODE which can handle stiff sets of equations where differential components of the system operate on very diverse time scales.

2.2.4 ABC inference

This project is complementary to work by Toni et al. developing the ABC Sequential Monte Carlo for parameter inference and model selection [17]. The basic concept in Approximate Bayesian Computation is to repeatedly draw parameter samples from their prior and simulate data under the model of interest using these parameter samples. If the data simulated with a particular parameter vector is sufficiently similar to observed data on some choice of distance measure, then we accept the sample as being from the approximate posterior $P(\theta|\rho(x, x^*) \leq \epsilon)$, where x is observed data, x^* is the simulated data, ρ is the distance function and ϵ is the threshold for acceptance. The choice of ϵ is crucial. Too small, and the acceptance ratio will be unacceptable small, too large and the approximate will be very poor.

For computationally intensive algorithms, such as ABC SMC parameter inference and model selection, it is of great value to be able to recover and resume a simulation if it is stopped. We have designed the ABC SMC methods to retrive the previous population of samples particles and continue the algorithm from that population. This is also a useful feature if one wants to alter the parameter settings for future populations.

2.3 Interfaces

2.3.1 C API

The C API provides the most flexible interface to the package. It is the only interface through which arbitrary parameter assignment functions can be used, allowing specific subsets of parameters or parameters such as the initial conditions to be inferred. It also allows straightforward integration of new simulation algorithms.

2.3.2 Command line interface

The command line interface is very user friendly, using flags and defined input files to control the operation. The full range of simulation and inference algorithms are available.

2.3.3 Python

Python is a multi-paradigm scripting language which is widely used in both academia and industry. It is a dynamic, strongly typed object-oriented programming language. Python has features

which support both aspect-oriented and functional programming as well as imperative and object-oriented programming styles. Most programmers regard Python as an very productive language to program in due to its concise syntax and uses indentation to define code blocks. This syntax forces programmers to write code that is easier to read and makes python code extremely maintainable and easily extendable. Python has automatic garbage collection based on reference counting and cycle detection and as such it reduces the programming time associated with dealing with memory related issues. Python also comes with an extensive standard library which as of Python 2.5 includes ctypes and a BioPython package [18] is also freely available which contains code useful to the bioinformatics community. Python is freely available and released under an OSI approved open source license. Python is very popular with researchers from a Physics, maths or computer science background and is gaining popularity in the bioinformatics and systems biology community. There are several implementations of the Python specification available including Jython (based on Java), IronPython (based on .NET) and CPython (based on C) which is the most commonly used implementation. Although there should be no major problems with portability between different Python implementations, this package has only been tested using CPython.

A number of different methods exist for integrating Python with C code such as Weave, SWIG and ctypes. Ctypes is a foreign function interface which supports accessing methods and variables located in a shared library and allows a Python programmer to access these almost transparently without the need to write any bespoke C code. This reduces the development time in interfacing C and Python code.

2.3.4 R

The freely available statistics programming environment R [19] is used greatly within the biological community, and so creating an interface between R and the above Approximate Baysian Computation (ABC) methods, differential equation solvers and SBML parser was an obvious. A limited SBML parser and simulation package exists [20] and is available in the BioConductor download `www.bioconductor.org`. However, this package is limited in the variety of SBML files it is able to parse and is only able to call the existing odesolve() function in R to simulate the model.

R has a vast amount of external packages pertaining to the biological sciences as well as protocol for writing new packages. It has the ability to seamlessly integreate several packages and data files into one workspace and provides a publication-worthy base graphics package. Data manipulation in R depends on vector, matrix and list structures, which makes it ideal for representing the data structures needed and produced by our package. The R implementation is currently implemented and supported on Linux and MacOS, and will become available on Windows in the future.

2.4 Dependencies

GNU Scientific Library. SBML-ABC makes extensive use of the GNU Scientific Library (GSL). We use the implementation of the Mersenne twister random number generator from GSL to generate the random numbers used in the simulations and uses the `gsl_matrix` and `gsl_vector` data structures for large portions of handling options and data handling. We provide support for

using the ode solving algorithms found in GSL. The package has been tested with version 1.11 of the GSL.

libSBML. The package is dependent on the libSBML library, which is available from http://www.sbml.org/Software/libSBML and provides the functionality to read, write and manipulate SBML files. LibSBML is used to extracts the model encoded in an SBML file and then builds an internal representation of the model, which is a more efficient representation and reduces the time and space requirements for the simulation algorithms.

gcc. The GNU Compiler Collection (gcc) is used to compile the gradient and propensity functions derived from the SBML files. The package has been tested with gcc4.3.1.

Python. The Python modules included in this package require Python 2.5 or later due to the need for ctypes and the use of language features only available in Python 2.5.

R. The R package derived from the C code was built and tested under R 2.5.

Platform. The package has been tested on Linux and Mac OSX and has not been tested on a Windows platform and there will be difficulties in compiling and loading the generated shared libraries used for the gradient and propensity functions.

Chapter 3

Methods

In this chapter, we describe in detail the method provided in the package. The core algorithms are implemented in C, including the SBML adaptors, the deterministic and stochastic simulators, ABC parameter inference and model selection algorithms. Finally the interfaces and their usage will be described.

3.1 SBML Adaptor

Although the SBML format is a good way of encoding models it is an extremely inefficient representation for use by simulation algorithms. A C module was created which made it possible to import an SBML model and build a representation of the model which we designed for use in the simulation algorithms. The module was designed to be as modular as possible in order to minimise code redundancy and decrease development time. The conversion code makes extensive use of the libSBML library which is designed to allow the manipulation of SBML models. The SBML adaptor code supports the generation of shared libraries containing gradient and propensity functions for use in the ODE, DDE and CLE simulation algorithms. This was done to reduce the execution time of the simulation algorithms and increase the flexibility of the algorithms.

3.2 Stochastic simulation

A biochemical system in stochastic simulation scheme is defined as a well stirred system of N species $\{S_1, S_2, ..., S_N\}$ and M reactions $\{R_1, R_2, ..., R_M\}$ with relative rates $c_1, c_2, ...c_M$. $X_i(t)$ the number of species S_i in the system at time t. The stochastic simulation problem is to estimate the state vector $\mathbf{X}(t) \equiv (X_1(t), X_2(t), ...X_N(t))$ given the initial state vector $\mathbf{X}(t_0) = \mathbf{x}_0$ at initial time t_0.

The propensity function $h_i(t)$ of a reaction, e.g. the probability a reaction takes place in the system, depends on the availability of the species required for the reaction and the reaction rate. It is a function proportional to the combinatory of number of species available and the reaction rate. For example, a first degree reaction $R_1(c_1) : X \rightarrow Y$ has a propensity $h_1(t) = c_1 \times [X](t)$, and a second degree reaction $R_2(c_2) : X + Y \rightarrow 2X$ has the propensity $h_2(t) = c_2 \times [X](t)[Y](t)$.

3.2.1 Random number generator

An important issue in stochastic simulation problem is the random number generation(RNG) to obtain the random factor in the simulation process, e.g. computing the time until the next reaction and choosing the next reaction. It is also useful in other parts of the project, i.e., the ABC MCMC, ABC SMC algorithms where the Monte Carlo process requires the RNG. Random numbers generated by any computational algorithms are only pseudo-random number. A good random number generator is the one that satisfies both theoretical and statistical properties, which are hard to obtain.

In our project, we use the random number generators provided by GSL (GNU Scientific Library, `http://www.gnu.org/software/gsl/manual/`), the `gsl_rng`. `gsl_rng` is a class of generators that generate random number from different distributions, such as uniform, normal, and exponential, and are convenient to use. The core of the generators we use is the default generator `mt19937`, the portable Mersenne Twister random number generator. In the stochastic simulation scheme, only sampling from a uniform distribution is required. However, in the ABC scheme, Gaussian distribution sampling and others are necessary to extend the variety of the algorithms.

3.2.2 Multicompartmental Gillespie algorithm

Gillespie's algorithm ([8], [9]), or stochastic simulation algorithm (SSA) is an exact method for the stochastic simulation of biochemical system of reactions described above. It stochastically simulates the system depending mainly on the propensity of reactions in the system by taking into account the random factor to the next reaction to occur and the time until the next reaction occurs. There are two Gillespie's algorithms, the direct method and the first reaction method, which are equivalent in terms of statistic. However, the direct method is faster and simpler to implement than the other, thus it is more commonly used.

Since the development of the SSA, several improvements have been made to improve or extend it. Gibson & Bruck's next reaction method [10], tau-leaping [12, 13, 14] and chemical Langevin equation (CLE) method [11] speed up the stochastic simulation by adapting a certain time interval for the system to proceed. Another direction in further improvement of the SSA has been made by extending it to the P-system [15], a computing model abstracting from the compartmental structure of alive cells in processing chemicals. The P-system is capable of describing the behaviour of a biologically meaningful system. A stochastic simulation method has been introduced for this system to simulate the kinetics of species in the multicompartment system based on the SSA, which we will refer to as the multicompartmental Gillespie algorithm [16]. The key idea of the multicompartmental Gillespie algorithm is to decide the compartment where the next reaction occurs, by computing the τ and μ value for each compartment, and then choosing the one with the smallest τ.

In our project, we implement the multicompartmental Gillespie algorithm. However, as the simulation framework is designed to fit the main overall targets of the project which are the ABC scheme for parameter approximation, it is required that the system is simple enough to compare models. Thus, the system in our implemented version (see Algorithm 1) is reduced to a simpler system compared with the P-system. The same type of species in different compartments are defined in the SBML file with different species name and thus species ID, thus when it moves from

one compartment to another, it is not considered a diffusion of chemical substances as described in the paper [16]. Furthermore, this simplified version ignores the "environment" element and it does not use a heap sort for the triple (τ_i, μ_i, i) where i is the compartment index, but performs a minimum finding every loop. This version of the multicompartmental Gillespie simulation becomes the original Gillespie's algorithm when the number of compartments is 1.

Algorithm 1 Multicompartmental Gillespie's algorithm

1: INPUT: Time series $(t_0, t_1, ..., t_N)$of simulated data, model of simulation
2: OUTPUT: System states at desired time points
3: Initialize the time $t = t_0$, system's state $\mathbf{x} = \mathbf{x}_0$
4: Calculate reaction propensity functions $a_j(\mathbf{x})$ and $a_0^i(\mathbf{x} = \sum_j^{r_j \in C_i} a_j(\mathbf{x}))$
5: Calculate τ_i = $\frac{1}{a_0^i} \ln(rand(0,1))$
6: Calculate μ_i: $\sum_{j=0}^{\mu_i - 1, r_j \in C_i} a_j < rand(0,1) a_0^i < \sum_{j=0}^{\mu_i, r_j in C_i} a_j$ (j is the index of reactions in the compartment only)
7: Set $i = 0$
8: **repeat**
9: Get C_i where τ_i is smallest
10: Execute reaction μ_i of compartment C_i, update the system's state: $t \rightarrow t + \tau$, $\mathbf{x} \rightarrow \mathbf{x} + \nu_\mu$.
11: Recalculate reaction propensity functions $a_j(\mathbf{x})$ and $a_0^i(\mathbf{x} = \sum_j a_j(\mathbf{x}))$ of reactions affected by r_{μ_j}
12: Recalculate τ_i = $\frac{1}{a_0} \ln(rand(0,1))$ of compartments affected by r_{μ_j}
13: Recalculate μ_i: $\sum_{j=0}^{\mu_i - 1, r_j \in C_i} a_j < rand(0,1) a_0^i < \sum_{j=0}^{\mu_i, r_j inC_i} a_j$ of comparments affected by r_{μ_i} (j is the index of reactions in the compartment only)
14: **if** $t = t_i$ **then**
15: record $\mathbf{x}(t)$
16: increment i
17: **end if**
18: **until** End condition

The multicompartmental Gillespie algorithm use a data structure close to the model description in SBML, the `StochasticModel` comprising of Compartment information, Compartment (with a pointer to the Compartment information), Species information, Species (with a pointer to the Species information), Reaction (containing Species type Reactants, Modifers and Products and reaction rate). This data structure enables the algorithm to work in an efficient interlinked manner that helps avoiding random errors. The structure include a pointer to a random number generator to be activated everytime the model is parse from the SBML file. It also has a block of structures to increase the speed of updating the system status by storing the reactions being affected by the execution of a reaction (similar to the idea of dependency graph notion introduced in the Next reaction method [10]).

3.2.3 Tau leaping

Tau leaping is an approximation to the exact Gillespie algorithm. Consider all the reactions in a time interval $[t, t + \tau]$. Denote the number of times reaction j occurrs as P_j. If the time interval

is short enough the propensity functions $a_j(\cdot)$ will not change much over this interval. Indeed, if we assume $a_j(\cdot)$ has the value at the beginning of the interval over the whole interval, then each P_j will have Poisson distribution with expected value $a_j(\mathbf{X}(t))\tau$. Once we have the P_j's, we apply the equation:

$$\mathbf{X}(t+\tau) = \mathbf{X}(t) + \sum_{j=1}^{M} \boldsymbol{\nu}_j P_j(a_j(\mathbf{X}(t)), \tau) \tag{3.1}$$

where ν_j is the stoichiometry of the jth reaction. Tau leaping is analogous to the simple forward Euler method for simulating ODEs, and suffers from the same numerical instability. More advanced alternatives such as implicit tau leap are outside the scope of this project, but would be an interesting avenue for further investigation. Tau leap uses the `CLEModel` settings structure.

3.2.4 Chemical Langevin Equation

The tau leap algorithm can also be seen as a stepping stone to the Chemical Langevin Equation (CLE), a Stochastic Differential Equation (SDE) that approximates the solution further. Recall that the amount each reaction occurs in the interval $[t, t+\tau]$, denoted P_j, is Poisson distributed with mean $\lambda = a_j(\mathbf{X}(t))\tau$. It is a basic result from probability theory that for large counts, a Poisson random variable is well approximated by a normal variable with mean and variance both equal to λ. In applying this approximation our state variables become continuous rather than discrete, and we arrive at the CLE:

$$\mathbf{X}(t+\tau) = \mathbf{X}(t) + \tau \sum_{j=1}^{M} \boldsymbol{\nu}_j a_j(\mathbf{X}(t)) + \sqrt{\tau} \sum_{j=1}^{M} \boldsymbol{\nu}_j \sqrt{a_j(\mathbf{X}(t))} Z_j \tag{3.2}$$

where Z_j are unit normal random variables.

3.3 Deterministic algorithms

3.3.1 ODE solvers

For simulating ODEs both the full range of solvers in the GSL and a subset of the CVODE solvers are available. GSL solvers are used via `odesolve` whereas CVODE solvers are accessed via `cvodesolve`. The default GSL solver is Runge Kutta Fehlberg 4-5, where the gradient evaluations for a 5th order Runge Kutta (RK) step are used to take a 4th order step "for free". The comparison between these two steps allows an estimate of the error, and hence facilitates error control. For stiff problems the implicit GEAR 1 and 2 methods are available. If a Jacobian is available the Burlish-Stoer implicit method should be used. The default CVODE solver is the Backward Differentiation Formula, with Newton iterations to solve the resulting fixed point problem.

3.3.2 Time Delay Differential Equation solver

The time delay differential equation (DDE) solver code is based on the R package `ddesolve` which is turned based on `solv95`. The core is an embedded RK 23 step and cubic Hermitian

interpolation to calculate lagged variables between simulated time points. Various modifications and simplifications have been made, including moving to an object orientated framework to make dynamically loading a compiled gradient function possible. Although switch functions are not used their associated code has not been removed since they would be useful if SBML events were incorporated in the future.

3.4 Parameter inference

3.4.1 ABC rejection

Approximate Bayesian Computation(ABC) allows us to infer the parameters of a system in problems where we cannot evaluate the likelihood function $f(x|\theta)$ but can draw samples from it by simulating the system. The simplest algorithm is ABC rejection [21], as described in Algorithm 2. ABC rejection is implemented by constructing an `ABCRejectionSettings` structure and calling `abcRejection`. For more details please refer to the C API User Guide in the Appendix.

Algorithm 2 ABC rejection. x_0 denotes observed data. $\rho(\cdot)$ denotes a distance measure. ϵ is a threshold set by the user.

repeat
  Sample θ^* from the prior $\pi(\theta)$
  Simulate x^* as a sample from $f(x|\theta^*)$
  Accept θ^* if $\rho(x^*, x_0) \leq \epsilon$
until enough samples found

3.4.2 ABC MCMC

ABC rejection is inefficient because it will continue to draw samples in regions of parameter space that are clearly not useful. One approach to this problem is to derive a ABC Markov Chain Monte Carlo (MCMC) algorithm [2], as specified in Algorithm 3. The hope is that the Markov chain (MC) will spend more time in interesting regions of high probability compared to ABC rejection. However, strongly correlated samples and low acceptance ratios means that ABC MCMC can in fact be highly inefficient, especially if the MC gets stuck in a region of low probability for a long time, requiring a significant burn in period.

3.4.3 ABC SMC parameter inference

ABC MCMC has a potential disadvantage being inefficient, being stuck in a regions of low acceptance probability for a long time due to the correlated nature of samples and thus more computationally intensive. This problem can be tackled by using another approach, the ABC Sequential Monte Carlo (SMC) methods [22], which introduces a sequence of intermediate distributions. This approach has been adapted with the sequential importance sampling algorithm [23, 24] and introduced in a paper to be published by Tina Toni [17]. In our project, we implement the algorithms described in this paper, the weighted ABC SMC algorithm 4 where the samples are derived from

Algorithm 3 ABC MCMC.

initialise θ_1
for $i = 1 \cdots N$ **do**
  Sample θ^* from kernel $q(\theta^*|\theta_i)$
  Simulate x^* as a sample from $f(x|\theta^*)$
  if $\rho(x^*, x_0) \leq \epsilon$ **then**
    if $U(0,1) < \min\left(1, \frac{\pi(\theta^*)q(\theta_i|\theta^*)}{\pi(\theta_i)q(\theta^*|\theta_i)}\right)$ **then**
      $\theta_{i+1} = \theta^*$
    else
      $\theta_{i+1} = \theta_i$
    end if
  else
    $\theta_{i+1} = \theta_i$
  end if
end for

the previous distribution with perturbation and the tolerances are chosen so that the samples evolve closer to the posterior. The ABC SMC algorithm becomes the ABC rejection algorithm when the number of populations is reduced to 1.

There are cases where the tolerances are set too low for the algorithm that it is overly time consuming to continue go to the next population. Thus it is useful to have a way to change the settings and resume the work where it stops. To meet this need, we have designed the main algorithm function so that it can retrieve information from the last completed population to continue the parameter inference in the next population without having to rerun the program from the start. This feature allow the users to adjust the parameters such as the perturbation kernels (i.e. make the perturbation size smaller or larger to allow for a more rigid or flexible evolution) or the tolerance. The same feature was added to the ABC SMC model selection method described in the next part.

3.5 ABC SMC model selection

The model selection framework introduced in the paper to be published by Tina Toni [17] performs Bayesian model selection allowing the estimation of Bayes factors [25]. The algorithm (see Algorithm 5) is similar to the ABC SMC simulation, with the differences of sampling a model from a prior distribution before sampling the parameters for the simulation of that model. With a number of input models, it is possible that a model can die out in the evolution of the populations through the iteration of population. The same feature of running the algorithm from an available population rather than from the start as with the ABC SMC algorithm is implemented for the model selection process.

A more detail review and description of the ABC scheme parameter inference and model selection can be viewed in the paper that this project is based on by Tina Toni [17]

Algorithm 4 ABC SMC algorithm

INPUT: model M,observed data $ObsData$, number of populations $NumPop$, size of each population $SizePop$, tolerance vector ϵ, prior distribution $\Pi(\theta)$, perturbation kernels $K_i(\theta|\theta*)$

OUTPUT: The last population of sampled particles that satisfied the condition being smaller than the tolerance, log file for each population
Set population ID $idPop$ = 0
Set weights of particles $\mathbf{w}^{(0)} = \{w_i = \frac{1}{SizePop}\}$
for $idPop = 0$ to $NumPop$ **do**
    Set $i = 0$
    repeat
        if $idPop = 0$ **then**
            Sample parameter $\theta**$ from prior distribution $\Pi(\theta)$
        else
            Sample parameter $\theta*$ from previous population with weight $\mathbf{w}^{(idPop)}$ and perturb to obtain $\theta* \sim K_{idPop}(\theta|\theta**)$
        end if
        if $\Pi(\theta**) = 0$ **then**
            Back to the beginning of repeat loop
        end if
        Simulate B data sets $x* \sim f(x|\theta**)$
        Set $b = \sum_{x*} \mathbf{1}(d(x*, ObsData) < \epsilon_{idPop})$ and calculate weight $w_i = b(idPop = 0)$ or $w_i = b * \Pi x**/\sum_{x_{idPop-1}} K_{idPop}(x_{t-1}, x**)$
    until $i = SizePop$
    Normalize $\mathbf{w}^{(idPop)}$
end for

3.6 Interfaces

3.6.1 Command line interface

We have designed the command line front end as a single program capable of multiple tasks. It is a user friendly interface in a command line with flags that allow users to indicate the desired task as shown in Figure 3.1. All the parameter settings such as prior distribution of the parameters, the perturbation kernels, observed data for parameter inference, or the time series for simulation are stored in simple matrix format in text files which are easy to understand and manipulate. We also provides the log files for populations after each population iteration and the log files for the distance between the simulated data and observed data which might be helpful for the users to determine the tolerance for each population.

3.6.2 C API

The C API is distributed as source code that can be easily compiled to a shared library using the provided makefile. The required header files are provided in a separate folder. A user guide is provided (see Appendix) which describes how to run simulations and inference algorithms, and

Algorithm 5 ABC SMC Model selection algorithm

INPUT: models M_m, observed data $ObsData$, number of populations $NumPop$, size of each population $SizePop$, tolerance vector $\epsilon^{(m)}$, prior distribution $\Pi^{(m)}(\theta)$, perturbation kernels $K_i^{(m)}(\theta|\theta*)$ for each model
OUTPUT: The last population of sampled particles that satisfied the condition being smaller than the tolerance, log file for each population, particles can be from different models
Set population ID $idPop$ = 0
Set weights of particles $\mathbf{w}_0^{(m)} = \{w_i = 1\}$
for $idPop = 0$ to $NumPop$ **do**
    Set $numParticle = 0$
    repeat
        Sample a model to simulate data M_m
        if $idPop = 0$ **then**
            Sample parameter $\theta**$ from prior distribution $\Pi_m(\theta)$
        else
            Sample parameter $\theta*$ from previous population of M_m with weight $\mathbf{w}_{(idPop)}^{(m)}$ and perturb to obtain $\theta* \sim K_{idPop}^{(m)}(\theta|\theta**)$
        end if
        if $\Pi_m(\theta**) = 0$ **then**
            Back to the beginning of repeat loop
        end if
        Simulate B data sets $x* \sim f^{(m)}(x|\theta**)$
        Set $b = \sum_{x*} \mathbf{1}(d(x*, ObsData) < \epsilon_{idPop}^{(m)})$ and calculate weight $w_i = b(idPop = 0)$ or $w_i = b * \Pi^{(m)} x**/\sum_{x_{idPop-1}} K_{idPop}^{(m)}(x_{t-1}, x**)$
    until $numParicle = SizePop$
    Normalize $\mathbf{w}_{idPop}^{(m)}$
end for

complements the *doxygen* documentation of the individual functions.

3.6.3 Python - C Binding

A number of different methods exist for integrating Python and C code, such as Weave, SWIG and ctypes. Ctypes was added to the python standard library in version 2.5 and allows a programmer to access functions and member variables in shared libraries or DLLs without having to write an C code at all. The Python library has all of the functionality available that is available in C, except for model selection. This functionality is easily accessible to the user through the supplied Python modules. The API has been designed and implemented to try to hide as much of the underlying complexity of the C interface from the end-user and make it easy to set up the system for simulation and running the ABC algorithms. The Python API currently supports the results matrix from the functions to files, making it easy to generate output. Although it has not been implemented it would be possible to produce graphs of these results using a python graphing library such as matplotlib.

3.6.4 R interface

The R interface includes seven R functions, each of which include the SBML parser. There are four solvers (Ordinary Differential Equations, Delay-Differential Equations, Chemical Langevin Equations and Multi-Compartment Gillespie) and three ABC methods (Rejection, MCMC and SMC). Due to the nature of R's internal structures and its C interfacing functions, the R version of our package is simple to use but it does not have the same user manipulability as the C and Python versions. Brief descriptions of the functions and the differences between the versions are listed below.

The package consists of R wrapper functions which then call the appropriate C wrapper functions which then call the appropriate C functions. Because of the internal structures of R and the desire to have our package also be a stand alone C package, we needed both R and C wrapper functions. Unfortunately for the users, this means the source code will not be as readily avaible to search through and modify.

The complex nature of our package required us to make use of the R function .Call, which is newer and less documented than the standard .C function. The package makes use of the R macros found in the R header files (R.h, Rinternals.h and Rdefines.h). The header files are included in the standard installation of R.

The package is currently only suppored on UNIX type systems and MAC OS X; with the package submission to CRAN, it should be made available on Windows as well.

SBMLodeSolve()

SBMLodeSolve() takes an SBML file and a vector of times to sample; it returns a matrix consisting of the sample times and the populations for each of the species in the reactions. There is a choice between 10 differential equation solvers, two of which are stiff differenetial equation solvers. In the C version, the user can define their own gradiant functions;this feature is not available in the R version.

SBMLddeSolve

SBMLodeSolve() takes an SBML file and a vector of times to sample; it returns a matrix consisting of the sample times and the populations for each of the species in the reactions. It is a dely-differential equation solver. In the C version, the user can define their own gradiant functions; this feature is not avalible in the R version.

SBMLcleSolve

SBMLcleSolve() takes an SBML file and a vector of times to sample; it returns a matrix consisting of the sample times and the populations for each of the species in the reactions. It is a stochastic solver, with the option to run Tau Leaping or Chemical Langevin Equation. In the C version, the user can define their own propensity functions; this feature is not avaiable is not availible in the R version.

SBMLgillSolve

SBMLgillSolve() takes an SBML file and a vector of times to sample; it returns a matrix consisting of the sample times and the populations for eacg of the species in the reactions. It is a stochastic solver using a multi-compartment Gillespie algorithm and there is no loss of functionality between the C and R versions.

SBMLabcRej

SBMLabcRej() takes an SBML file, an observed data file, a vector of prior distribution types, two vectors describing the prior disribution settings, the number of samples, the solver type, the threshold on the distance measure, the maximum number of iterations and the output file name. It returns a matrix of the accepted values of the last population. In the C version, the user can pass their own function to set the parameters, they can choose their own distance metric and they can choose whether or not missing data is included in the algorithm. In the R version, the set parameters function is default, the distance metric is sum of squares, and the algorithm does not include missing data.

SBMLabcMCMC

SBMLabcMCMC() takes an SBML file, an observed data file, a vector of prior distribution types, a vector of kernel distribution types, two vectors describing the prior disribution settings, two vectors describing the kernel distribution settings, the number of samples, the solver type,a vector of inital parameter settings, the threshold on the distance measure, and the output file name. It returns a matrix of the accepted values of the last population. In the C version, the user can specify which distance metric to use and how to deal with missing data. In the R version, the distance is set to sum of squares and the missing data is included.

SBMLabcSMC

SBMLabcSMC takes an SBML file, an observed data file, a vector of prior distribution types, two vectors of prior distribution settings, a vector of Gaussian standard deviation for the perterbation kernel, a vector of distance measure thresholds for each population, the number of populations, the size of inital population, the number of stochastic simulations for each parameter and a output file name. In the C version, the user can define the distance metric, the handling of missing data, set the perterbation kernel for each population and the ability to recover the previous population of samples particles if the simulation is halted. In the R version, the distance metric is sum of squares, missing data is included, the perterbation kernel is set for all populations and there is no recovery mechanism.

```
[hang@kolmogorov1 ABCSMC1.0]$ ./sbmlabc -h
Usage: sbmlabc -Mmiorsvh
        -M       specify mode of running
                (S)imulation, ABC (R)ejection, ABC MCM(C), (A)BC SMC, ABC (M)odel Selection
        -m       specify method of Simulation
                (G)illespie, (T)au leaping, (C)LE, (O)DE, (D)DE
        -i       input filename for each mode
                   -M S: sbml file for model
                   -M (R, MCMC, SMC, M): run file for model
                     Example of run file for SMC, "brusselator.run":
                       GILLESPIE_MODEL SUM_OF_SQUARE 5 100 5
                       brusselator.sbml brusselator.data brusselator.prior brusselator.sigma
                       brusselator.eps  brusselator.mark brusselator.out
                   The first line describe the model type, distance type, number of population
                   The remainings are the file names of information such as model description(
                     Example of run file for SMC model selection, "brusselator.runMS":
                       2 100 5
                       GILLESPIE_MODEL SUM_OF_SQUARE 5 100 5
                       brusselator.sbml brusselator.data brusselator.prior brusselator.sigma
                       brusselator.eps  brusselator.mark brusselator.out
                       ODE_MODEL SUM_OF_SQUARE 5 100 5
                       brusselator.sbml brusselator.data brusselator.prior brusselator.sigma
                       brusselator.eps  brusselator.mark brusselator.out
                   The first line describe the number of model, the number of population. The
                   Please make sure that the observed data, epsilon value and mark files are t
                   Model type: GILLESPIE_MODEL, ODE_MODEL, CLE_MODEL, DDE_MODEL
                   Distance type: SUM_OF_SQUARE, CITY_VECTOR, COSINE
        -o       output filename for Simulation
        -t       time serries filename for Simulation
        -s       solver for ode Simulation
                   (0) rk2, (embedded Runge-Kutta (2, 3) method)
                   (1) rk4, (4th order classical Runge-Kutta)
                   (2) rkf45, (embedded Runge-Kutta-Fehlberg (4, 5) method)
                   (3) rkck, (embedded Runge-Kutta Cash-Karp (4, 5) method)
                   (4) rk8pd, (gsl_odeiv_step_rk8pd)
                   (5) rk2imp, (gsl_odeiv_step_rk2imp)
                   (6) rk4imp, (gsl_odeiv_step_rk4imp)
                   (7) bsimp, (implicit bulirsch-stoer method, uses rkf45 instead)
                   (8) gear1, (implicit gear method)
                   (9) gear2 (implicit gear method)
        -r       retrieve model selection from previous population
                    Put number for next population index(index start from 0)
        -v       verbose: 0 (no verbose) 1 (verbose) 2(debug) (This function is currently disa
        -h       help

        Examples:
        Run simulation data from brusselator.sbml file, with Gillespie model:
                 >./sbmlabc -M S -m G -i brusselator.sbml -t time.txt -o sim.txt

        Run simulation data from brusselator.sbml file, with ODE model using rk4 solver:
                 >./sbmlabc -M S -m O -s 1 -i brusselator.sbml -t time.txt -o sim.txt

        Run ABCSMC from brusselator.run and the next line to retrieve from previous population
                 >./sbmlabc -M A -i brusselator.run
                 >./sbmlabc -M A -i brusselator.run -r -2
        Run ABC SMC Model selection from brusselator.runMS
                 >./sbmlabc -M M -i brusselator.runMS
```

Figure 3.1: Screen shot of instruction on how to use the the program $sbmlabc$

Chapter 4

Examples and results

4.1 Data simulation

4.1.1 Lotka-Volterra model

The Lotka-Volterra (LV) model is a classic model describing the relation between preys and predators in an ecological system. The model is described by the reactions:

$$X \rightarrow 2X \quad \text{with rate } c_1 \tag{4.1}$$

$$X + Y \rightarrow 2Y \quad \text{with rate } c_2 \tag{4.2}$$

$$Y \rightarrow \emptyset \quad \text{with rate } c_3 \tag{4.3}$$

where X denotes the prey and Y denote the predator.

The stochastic simulation of this model with Gillespie's algorithm with initial condition $(X, Y) = (1000, 1000)$ with the set of parameters $(c_1, c_2, c_3) = (10, 0.01, 10)$ is shown in Figure 4.1, showing the oscillatory characteristic of the model.

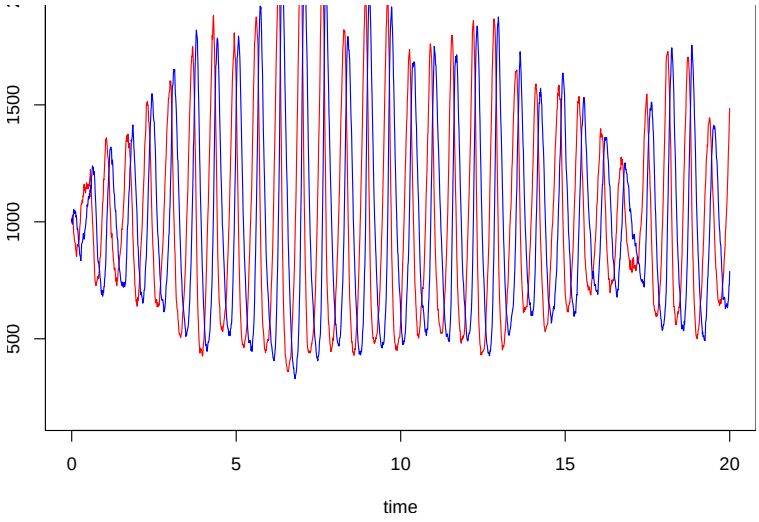


Figure 4.1: Lotka-Volterra simulated using Gillespie's algorithm.

4.1.2 Repressilator

This model involves three genes and their transcripts which suppress each other's expression. Figure 4.2 shows the model simulated using ODE solver. It is model which exhibits a stable oscillation with fixed time periods. The repressilator consists of three genes connected together in a feedback loop, as shown in Figure 4.3.

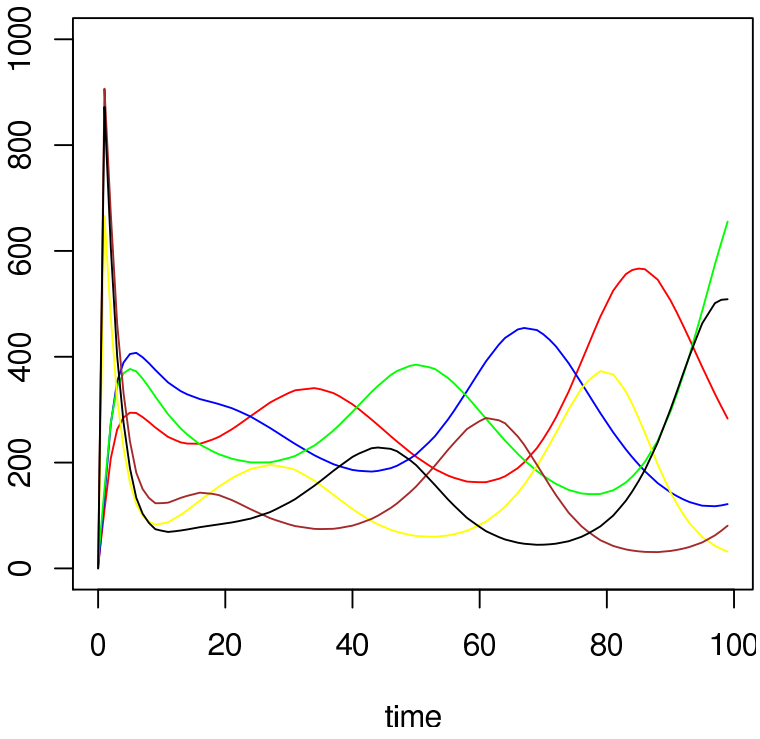


Figure 4.2: Repressilator simulated using ODE solver.

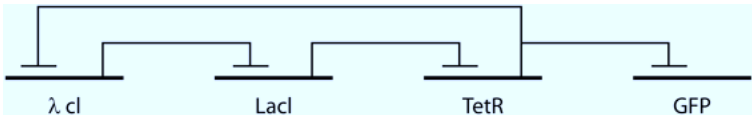


Figure 4.3: Feedback loops present in repressilator

4.1.3 Brusselator model

This is a classic autocatalytic, oscillating system of chemical reactions, which exhibits a stable limit cycle for a certain range of parameter values. We will use this model for testing the inference algorithms. The component reactions are:

$$0 \rightarrow X \tag{4.4}$$
$$2X + Y \rightarrow 3X \tag{4.5}$$
$$X \rightarrow Y \tag{4.6}$$
$$X \rightarrow 0 \tag{4.7}$$

The ODEs are just in terms of the rate constants:

$$\frac{d[X]}{dt} = p_1 + p_2[X]^2[Y] - p_3[X] - p_4[X] \tag{4.8}$$

$$\frac{d[Y]}{dt} = -p_2[X]^2[Y] + p_3[X] \tag{4.9}$$

The Jacobian of system of ODEs:

$$\frac{dx_i}{dt} = F_i(x_1, \ldots, x_n) \tag{4.10}$$

is given by

$$J_{ij} = \frac{\partial F_i}{\partial x_j} \tag{4.11}$$

So for this model,

$$J = \begin{bmatrix} 2p_2[X][Y] - p_3 - p_4 & p_2[X]^2 \\ -2p_2[X][Y] + p_3 & -p_2[X]^2 \end{bmatrix} \tag{4.12}$$

Figure 4.4 shows simulations of this model using four different simulation algorithms.

4.2 Parameter inference

We initially tested the inference algorithms on the Brusselator model and artificial observed data with 100 time points.

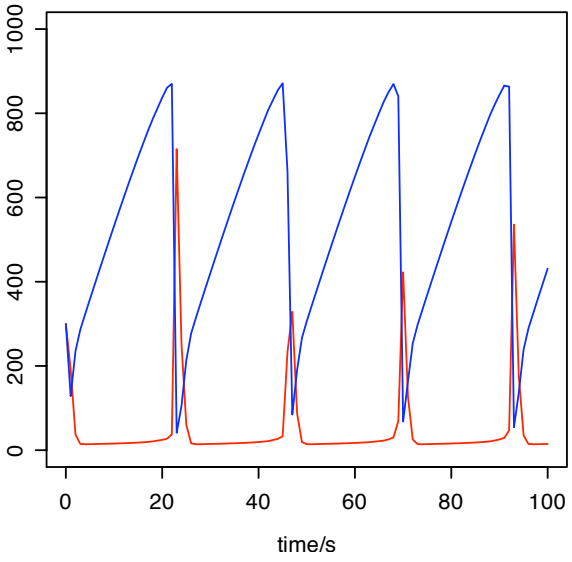
4.2.1 ABC rejection

ABC rejection was run requiring 1000 accepted samples, sum of squares distance function, and a tolerance of $\epsilon = 5 \times 10^6$. This level of ϵ across 100 time points and 2 species corresponds to a root mean square error (RMSE) of 158, where the species concentration peaks are around 1000. The run required a total of 319967 samples, giving an acceptance ratio of 0.31%. Figure 4.5(a) shows a histogram for parameter 1 (the rate of reaction 1) over the 1000 samples. Figure 4.5(b) shows the accepted samples plotted as parameter 1 (the rate of reaction 1) against parameter 2 (the rate of reaction 2). It is more informative to investigate plots of one parameter against another than simple histograms of one parameter as this highlights correlations between parameters.

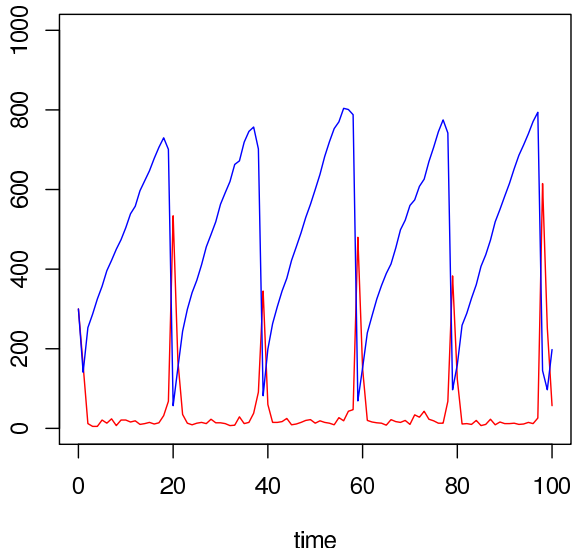
To put the $\epsilon = 5 \times 10^6$ threshold into context it is interesting to compare simulated data that was accepted to the observed data, as shown in Figures 4.6(a) and 4.6(b).

4.2.2 ABC MCMC

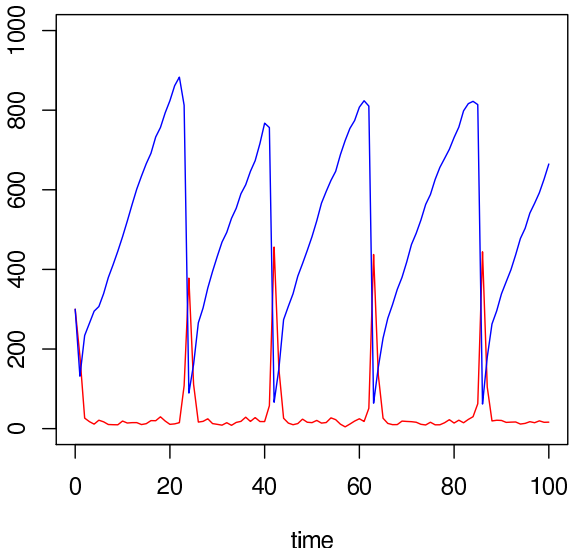
ABC MCMC was run with the same epsilon threshold of 5×10^6. For 1000 accepted samples 90204 time courses were generated, giving an acceptance ratio of 1.1%. As expected this is slightly higher than for rejection, but it is at the cost of correlated samples. Figure 4.7(b) shows parameter 1



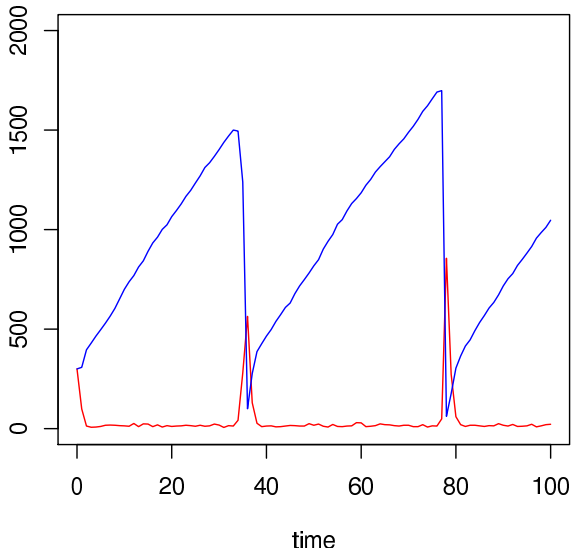
(a) Ordinary Differential Equation solver



(b) Tau leap

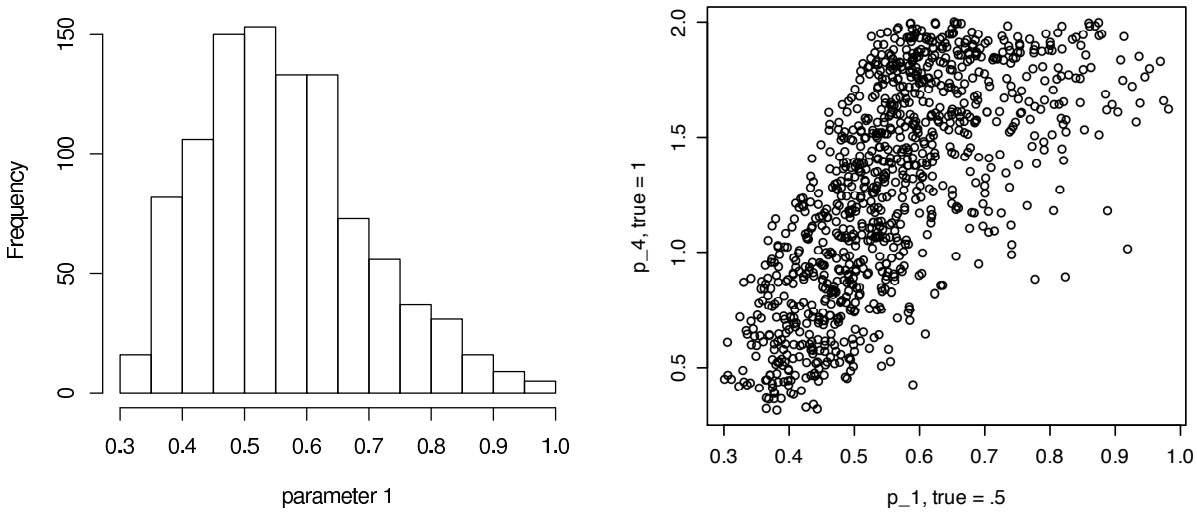


(c) Chemical Langevin Equation solver



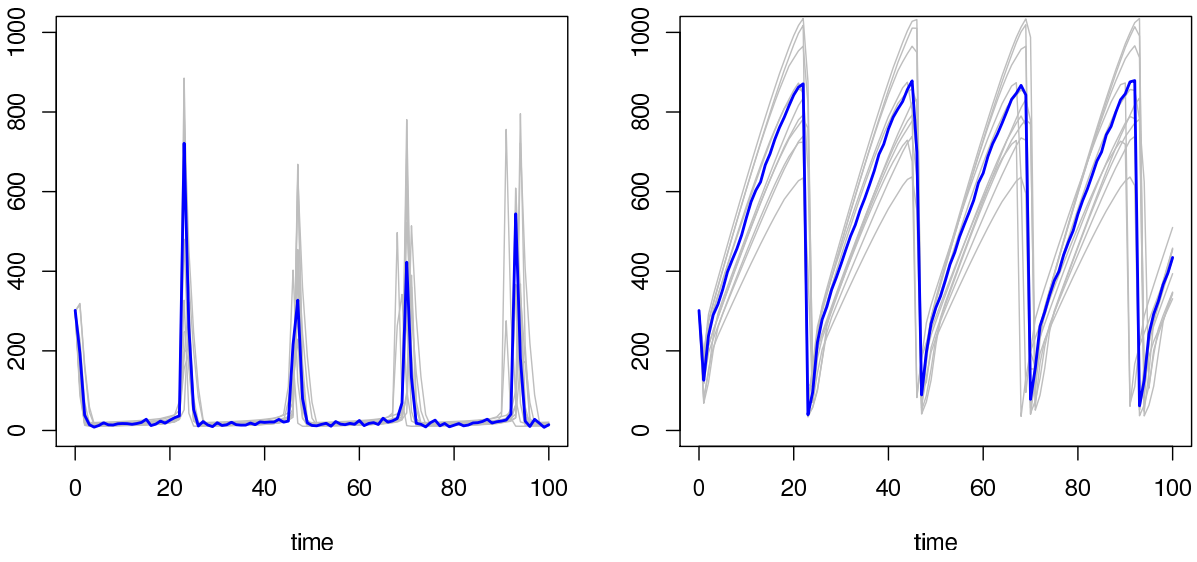
(d) Gillespie's algorithm

Figure 4.4: Simulations of the Brusselator model using different algorithms.



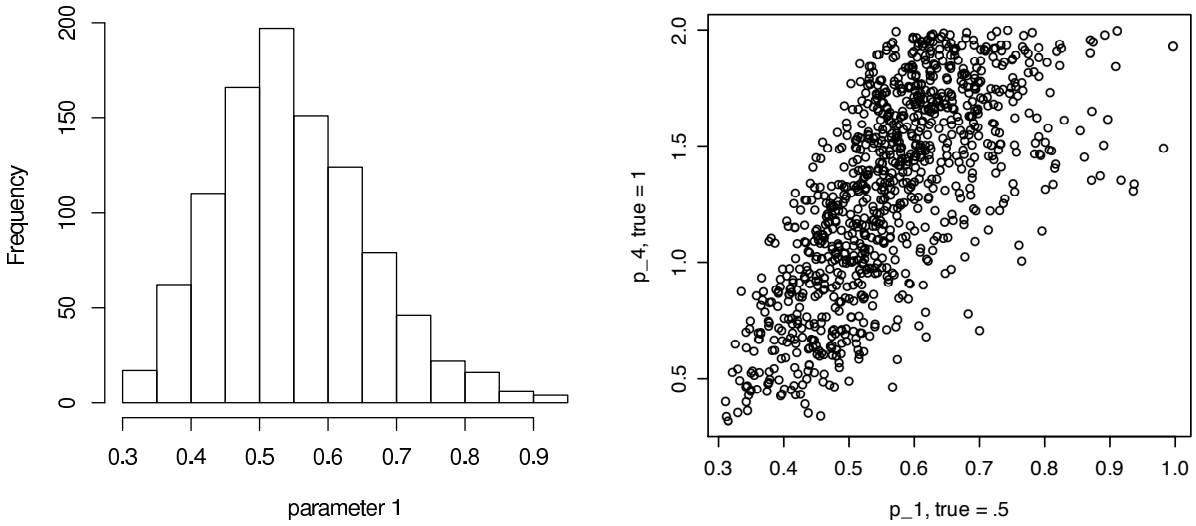
(a) Histogram of parameter 1 (b) Parameter 1 vs. 4 samples

Figure 4.5: Results of ABC rejection on Brusselator model using ODE solver.



(a) Species 1 (b) Species 2

Figure 4.6: Observed data (blue) and accepted simulated data (gray lines).



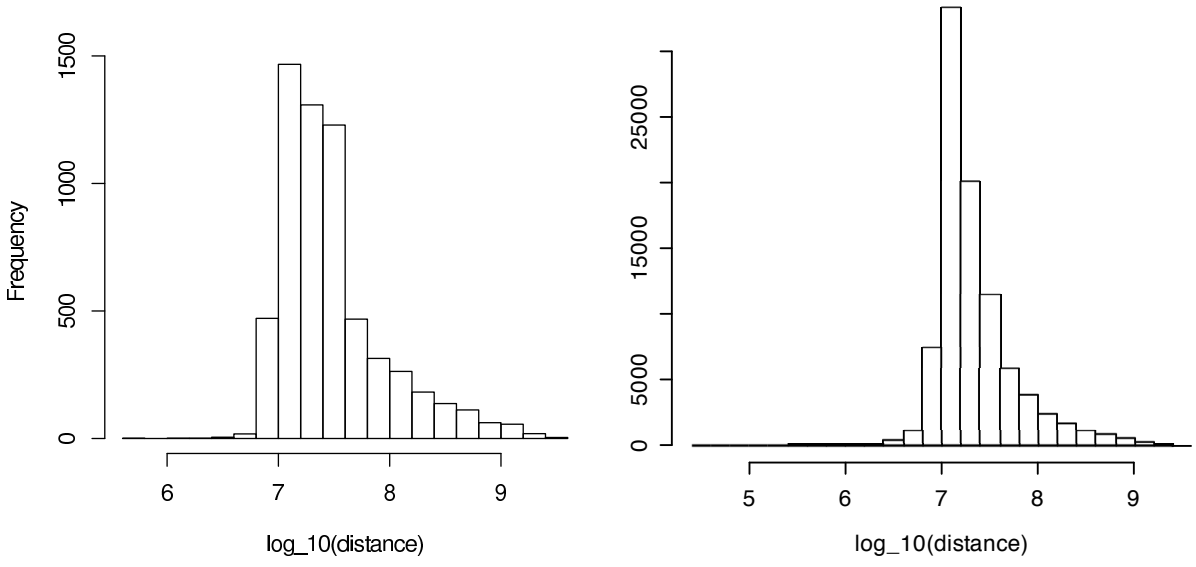
(a) Histogram of parameter 1

(b) Parameter 1 vs. 4 samples

Figure 4.7: Results of ABC MCMC on Brusselator model using ODE solver.

against 4 for accepted samples, and the agreement with the range of feasible values from rejection is very good.

Figure 4.8(a) shows a histogram of all the distances calculated during the run. The threshold we use corresponds to $\log_{10}(5 \times 10^6) = 6.7$, so we see that just the tail is accepted. Figure 4.8(b) shows the distribution of the distances calculated. Significantly less very large distances are produced, which corresponds to the the fact that unlike rejection, MCMC does not waste many samples in regions of low probability.



(a) Rejection

(b) MCMC

Figure 4.8: Histograms of log(distance) for all simulated datasets.

4.2.3 ABC SMC

ABC SMC was run for Brusselator using RKF45 ODE solver. 1000 particles were used with 8 populations, with ϵ decreasing linearly from 10^7 to 3×10^6. The number of samples required for each population, and the corresponding acceptance ratio are shown in Table 4.1. The acceptance ratio decreases as the threshold decreases, as expected. This can be clearly seen in Figure 4.10, which shows the roughly exponential relationship between the acceptance ratio and ϵ. It would be interesting to try reducing the width of the perturbation kernel through the populations, as this might help the samples stay in regions of higher probability. It would also be interesting to see if further reduction of ϵ was possible, as it appears that the reduction from 5×10^6 to 3×10^6 actually has little effect on the acceptance ratio. Figure 4.9 shows plots of parameter 1 vs. 4 for the successive populations. The samples start out relatively diffuse, and concentrate on regions of higher posterior probability as the threshold is decreased. It is interesting that for large values of ϵ a second mode with parameter 1 close to zero is possible, but which is eliminated as ϵ is reduced.

Population	$\epsilon/10^6$	# simulations	Acceptance ratio
1	10	6772	0.147667
2	9	17977	0.055627
3	8	23389	0.042755
4	7	34253	0.029195
5	6	59936	0.016684
6	5	290406	0.003443
7	4	224550	0.004453
8	3	299915	0.003334

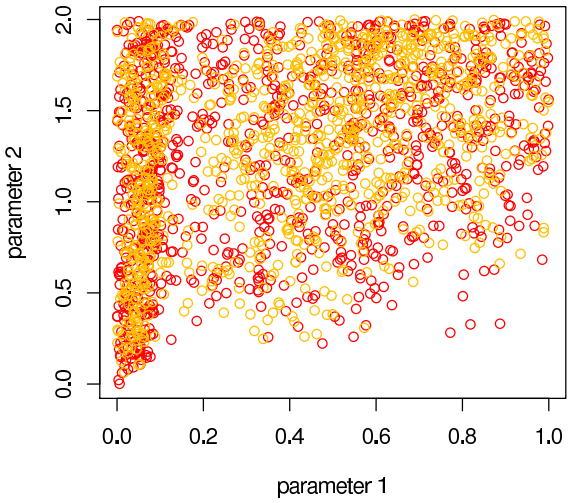
Table 4.1: Statistics for each population of ABC SMC.

4.3 Model selection

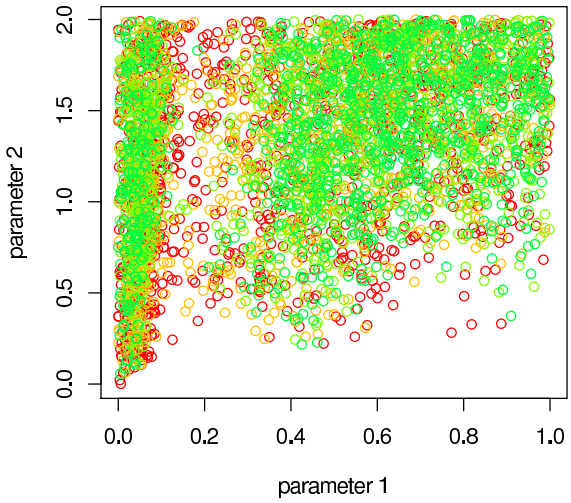
We did an experiment with observed data generated from the Brusselator model with ODE solver to observe the performance of the model selection algorithm. We assumed that the data is from an unknown source and the objective is to try different models to see which one best describes the behaviour of the model.

We did model selection with two models: the Brusselator model itself and the Lotka-Volterra model, both simulated using Gillespie's algorithm. The settings for parameters for each model (perturbation kernel and prior distributions) are set separately, however the tolerances, the mark vector and so on are shared between the two. The algorithm was run for 5 populations to generate 100 parameter vectors. The results show that from the first population of parameters, only Brusselator model is present. Consequently, in the following populations, only the Brusselator model remains.

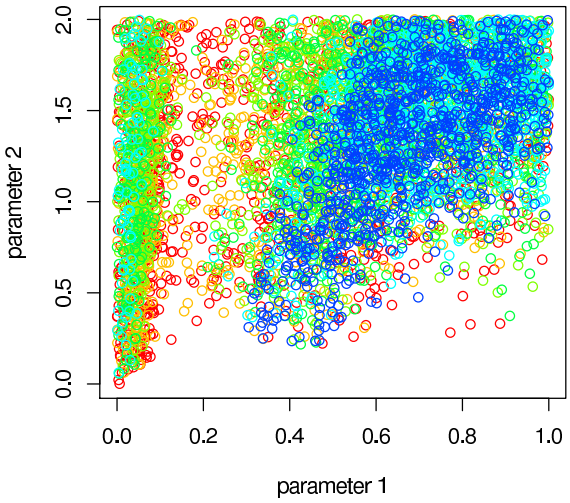
This result shows that the model selection was able to correctly choose the model which best predicts the unknown observed data.



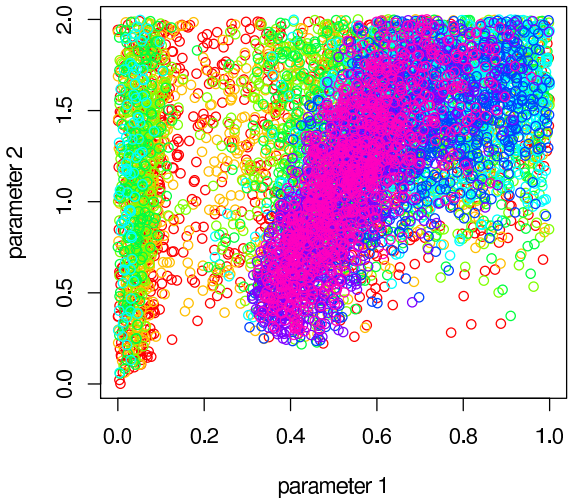
(a) Populations 1 (red) and 2 (orange)



(b) ...plus populations 3 (light green) and 4 (green)



(c) ...plus populations 5 (light blue) and 6 (blue)



(d) ...plus populations 7 (purple) and 8 (magenta)

Figure 4.9: Parameter 1 vs. 4 from ABC SMC with 8 populations, 1000 particles on Brusselator using ODE solver. Populations are shown added two per plot.

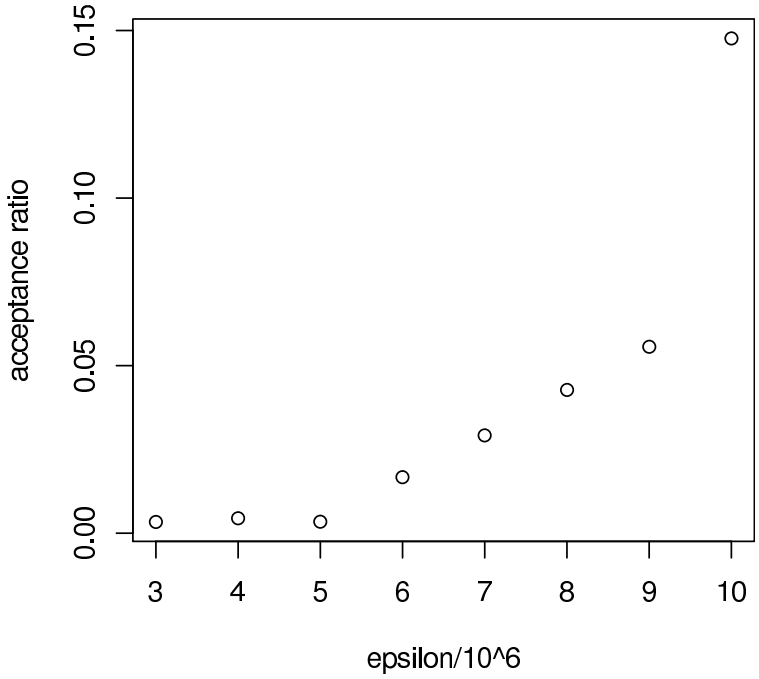


Figure 4.10: Acceptance ratio against ϵ for each population of ABC SMC.

Chapter 5

Future work

Additional simulation algorithms. Various other simulation algorithms could be valuable additions to the package. Additional stochastic algorithms include the Next Reaction method (Gibson & Bruck), more advanced tau leaping algorithms (e.g. implicit tau leap), and higher order Stochastic Differential Equations than the CLE. Hybrid algorithms involving both stochastic and deterministic components are available which choose the most appropriate for each reaction and species.

Advanced implementations. Monte Carlo algorithms are good candidates for parallelisation. Technologies such as CUDA might offer a good way of improving simulation throughput, although this would mean that the package would be tied to a specific vendor, which may not be ideal. It may be possible to create an MPI or OpenMP enabled version of the package which could mean that the algorithms could be run on high performance computing resources. Difficulties do exist however with implementing efficient and robust random number generation across a distributed environment.

Random number generator. It could be valuable to investigate the use of a different random number generator. Other random number generators are available that have longer periods, are just as random and are less computationally intensive to compute [26]. Since a lot of random numbers are generated by our system this could lead to a reduction in compute time whilst not impacting on the soundness of the algorithm.

Distance measures. The existing distance measures calculate only the error in the dependent variable, i.e. the species. This means that very similar observed and simulated data which is slightly misaligned in time will have an undesirably large distance. As a result samples may be rejected which actually represent quite good parameters. Some kind of time shift invariant distance function, or inference of an initial lag in the model, could be ways of addressing this problem.

Automatic conversion between stochastic and deterministic models Currently it is not possible to take a model encoded with stochastic parameters and simulate this deterministically and get results which make sense.

Lyapunov exponents. How close the samples ABC produced are to being samples from the true posterior will clearly depend on to what extent similar data corresponds to similar parameters. For inferring initial conditions especially, this is closely related to Lyapunov exponents as they control the rate of divergence from different starting conditions. Functionality to estimate Lyapunov exponents could therefore be a useful addition.

Improved data structure for MCMC. The MCMC algorithm stores duplicates of the current vector of parameters every time a sample is rejected. As a result a single vector is stored many times in the output matrix. A more efficient structure would simply store unique vectors and the number of repeats of each.

Autocorrelation times for MCMC. For evaluating how long the burn in period of an MCMC algorithm is it is useful to be able to estimate the integrated autocorrelation time of the samples: the burn in period should be at least this long. Built in functionality to achieve this would be very useful.

Acknowledgements

We would like to thank Tina, Kamil, Paul and Michael for their support and advice.

Nathan would like to thank nature for evolving *Camellia Sinensis*.

Bibliography

[1] M. Hucka, A. Finney, H. M. Sauro, H. Bolouri, J. C. Doyle, and H. Kitano. The systems biology markup language (sbml): a medium for representation and exchange of biochemical network models. *Bioinformatics*, 19(4):524–531, 2003.

[2] P. Marjoram, J. Molitor, V. Plagnol, and S. Tavare. Markov chain Monte Carlo without likelihoods. *Proc Natl Acad Sci U S A*, 100(26):15324–15328, 2003.

[3] Stefan Hoops, Sven Sahle, Ralph Gauges, Christine Lee, Jrgen Pahle, Natalia Simus, Mudita Singhal, Liang Xu, Pedro Mendes, and Ursula Kummer. Copasi–a complex pathway simulator. *Bioinformatics*, 22(24):3067–3074, Dec 2006.

[4] Zhike Zi and Edda Klipp. Sbml-pet: a systems biology markup language-based parameter estimation tool. *Bioinformatics*, 22(21):2704–2705, Nov 2006.

[5] Xinglai Ji and Ying Xu. libsres: a c library for stochastic ranking evolution strategy for parameter estimation. *Bioinformatics*, 22(1):124–126, Jan 2006.

[6] A.C. Hindmarsh. Odepack, a systematized collection of ode solvers. In R.S. et al. Stepleman, editor, *Scientific Computing: Applications of Mathematics and Computing to the Physical Sciences*, volume 1 of *IMACS Transactions on Scientific Computing*, pages 55–64, Amsterdam, Netherlands; New York, U.S.A., 1983. North-Holland.

[7] Benjamin J Bornstein, Sarah M Keating, Akiya Jouraku, and Michael Hucka. Libsbml: an api library for sbml. *Bioinformatics*, 24(6):880–881, Mar 2008.

[8] Gillespie D.T. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics*, 22:403–34, 1976.

[9] Gillespie D.T. Exact stochastic simulation of coupled chemical reactions. *Journal of Computational Physics*, 81:2340–61, 1977.

[10] Gibson M.A. and Bruck J. Exact stochastic simulation of chemical system with many species and many channels. *Journal of Physical Chemistry*, 105:1876–89, 2000.

[11] Gillespie D.T. The chemical langevin equation. *Journal of Chemical Physics*, 113:297306, 2000.

[12] Gillespie D.T. Approximate accelerated stochastic simulation of chemically reacting systems. *Journal of Chemical Physics*, 115:171633, 2001.

[13] Gillespie D.T. and Petzold L.R. Improved leap-size selection for accelerated stochastic simulation. *Journal of Chemical Physics*, 119:822934, 2003.

[14] Cao Y.and Petzold L.R. Improved leap-size selection for accelerated stochastic simulation. *Journal of Chemical Physics*, 124:044109, 2006.

[15] Paun Gh. Computing with membranes. *Journal of Computer and System Sciences*, 61(1):108–143, 2000.

[16] Priami C. and Plotkin G.(Eds.). P systems, a new computational modelling tool for systems biology. *Transaction on computational Systems Biology VI*, LNBI 4220:176–197, 2006.

[17] Toni T., Welch D., Stumpf M.P.H., and et al. Approximate bayesian computation scheme for parameter inference and model selection in dynamical systems. *in press*, 2008.

[18] Brad Chapman and Jeffrey Chang. Biopython: Python tools for computational biology. *SIGBIO Newsl.*, 20(2):15–19, 2000.

[19] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2007. ISBN 3-900051-07-0.

[20] Tomas Radivoyevitch. A two-way interface between limited systems biology markup language and r. *BMC Bioinformatics*, 5:190, Dec 2004.

[21] Pritchard J.K., Seielstad M.T., Perez-Lezaun A., and Feldman M.W. Population growth of human Y chromosomes: a study of Y chromosome microsatellites. *Molecular Biological Evolution*, 16(12):1791–1798, 1999.

[22] S. A. Sisson, Y. Fan, and Mark M. Tanaka. Sequential monte carlo without likelihoods. *PNAS*, 104(6):1760–1765, 2007.

[23] Del Moral P. and Jasra A. Sequential monte carlo samplers. *Journal of Royal Statistics Society B*, 68(3):411–436, 2006.

[24] Del Moral P. and Jasra A. Sequential monte carlo for bayesian computation. *in press*, 2008.

[25] Robert E. Kass and Adrian E. Raftery. Bayes factors. *Journal of the American Statistical Association*, 90(430):773–795, 1995.

[26] George Marsaglia and Arif Zaman. A new class of random number generators. *The Annals of Applied Probability*, 1(3):462–480, 1991.