ABC-SysBio : Approximate Bayesian Computation in Python with GPU support

Juliane Liepe, Chris Barnes, Erika Cule

July 29, 2011

Contents

Chapter 1

Introduction

During the last decade advanced tools to investigate and simulate biological systems have been developed. Now it is standard practice to investigate biological models and their dynamical behaviour *in silico*. One major problem of this strategy is the estimation of system parameters and the exclusion of overfitting a model to biological data. Several statistical approaches have been introduced to this topic. With these statistical methods it is possible to infer parameter values of a biochemical system. One limitation of these approaches is that most of them provide a single best estimate for each parameter value, when it is preferable to obtain a probability distribution of likely parameter values. One algorithm for obaining such a distribution is the ABC (approximate Bayesian computation) rejection sampler (Pritchard *et al.*, 1999). Starting from a given prior distribution the ABC rejection method samples parameter values and tests if their simulation result fits to the experimental data. However, this algorithm is computationally expensive, especially for large prior distribution. A computationally more efficient algorithm was published by Toni *et al.* in 2009, using ABC combined with a sequential Monte-Carlo approach (SMC). (Toni *et al.*, 2009; Toni & Stumpf, 2010) also shown that the ABC SMC allows for model selection.

Here we present a Python package, `abc-sysbio`, that implements parameter inference and model selection in dynamical systems. `abc-sysbio` combines three algorithms (see Figure 1.1). The first and most simple one is the ABC rejection sampler (Pritchard *et al.*, 1999). It is automatically used when only one model and one maximal distance is given. The second algorithm is the ABC SMC for parameter inference (Toni *et al.*, 2009) and it is used when only one model but several maximal distances are given. If several models and maximal distances are given the last implemented algorithm, the ABC SMC for model selection (Toni *et al.*, 2009), is applied. The first two methods are optimized for the inference of parameter values. However, with an optimal chosen series of maximal distances and a large enough population size the third one can be used for a simultaniously parameter inference and model selection.

`abc-sysbio` is designed to work with models written in Systems Biology Markup Language (SBML), a data exchange format based on XML. Models written in SBML can be obtained from the BioModels database (Le Novere *et al.*, 2006). Typical biochemical systems represented in the BioModels database include regulatory networks, metabolic networks and cell-signalling pathways. The thirteenth release of the BioModels database contains 211 curated and 124 non-curated models. The `abc-sysbio` package is able to analyze all dynamical models taken from the curated database that do not include algebraic rules. Non-dynamical models can not be investigated with this package. It is also possible to investigate other dynamical models that are not in SBML format, but this takes a more advanced usage of the package as a Python library.

We advise that you read through the Examples to get a feel for how the package works. The SBML models and user input files used in the examples are distributed with the package enabling you to reproduce the examples yourself. Thus familiarised, you will be ready to use the package to perform ABC algorithms on your own models. For full instructions on how to use the package in a general case, see Chapter 3.

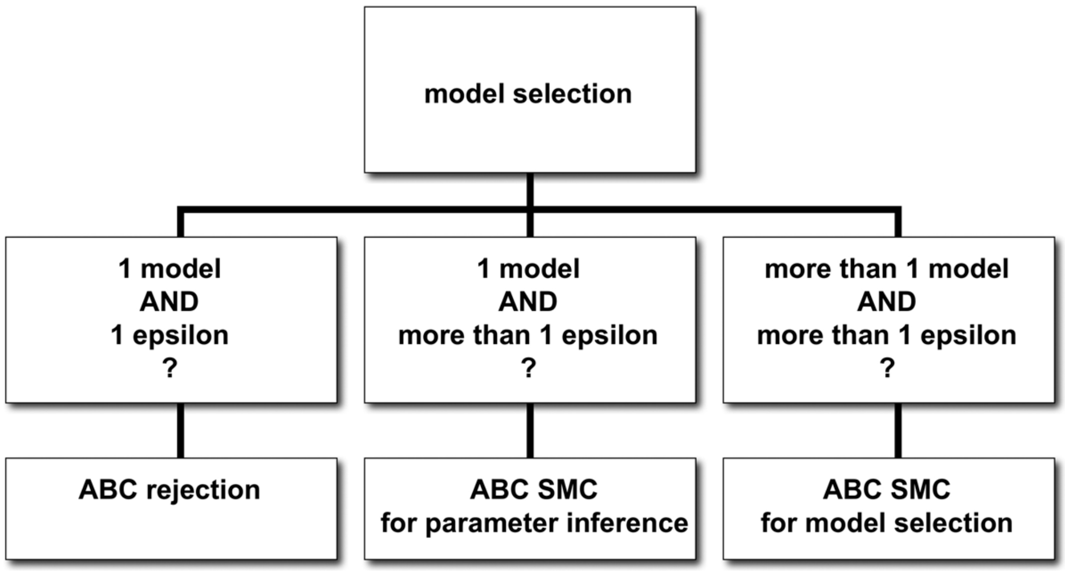
model selection
1 model AND 1 epsilon ?
1 model AND more than 1 epsilon ?
more than 1 model AND more than 1 epsilon ?
ABC rejection
ABC SMC for parameter inference
ABC SMC for model selection

Figure 1.1: Schematic overview of the implemented ABC algorithms.

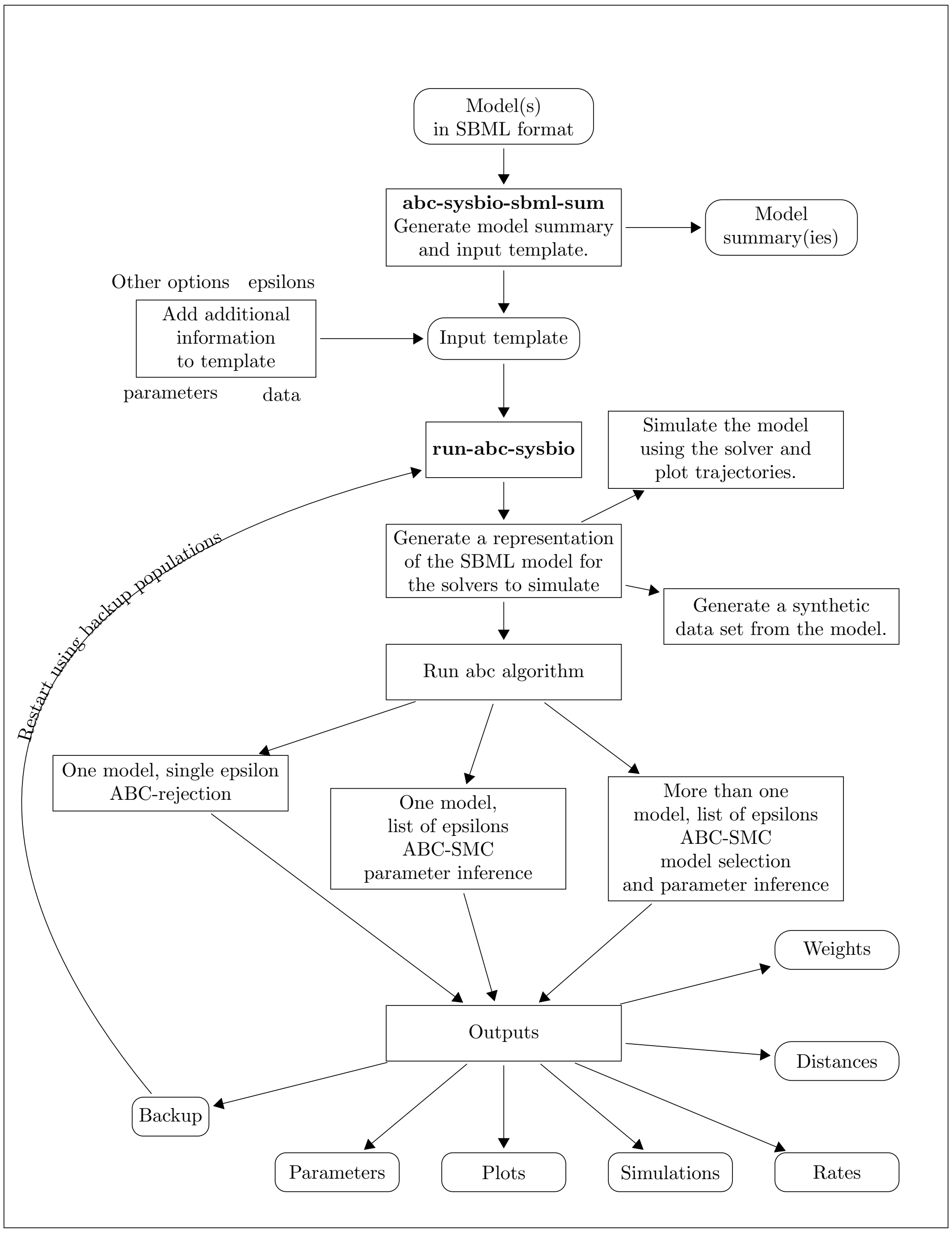
Model(s) in SBML format
abc-sysbio-sbml-sum Generate model summary and input template.
Model summary(ies)
Other options
epsilons
Add additional information to template
parameters
data
Input template
run-abc-sysbio
Simulate the model using the solver and plot trajectories.
Generate a representation of the SBML model for the solvers to simulate
Generate a synthetic data set from the model.
Run abc algorithm
Restart using backup populations
One model, single epsilon ABC-rejection
One model, list of epsilons ABC-SMC parameter inference
More than one model, list of epsilons ABC-SMC model selection and parameter inference
Weights
Outputs
Distances
Backup
Parameters
Plots
Simulations
Rates

Figure 1.2: A user's-eye-view of the package abc-sysbio.

Chapter 2

Installation

The package `abc-sysbio` provides Python library, `abcsysbio` containing functions for parameter inference and model selection. Together with the scripts `abc-sysbio-sbml-sum` and `run-abc-sysbio` it creates a user-friendly tool. The biochemical network simulation algorithms are now implemented in Python, C++ and CUDA through `cuda-sim` (Zhou *et al.*, 2011). It is developed for the Linux operating system but can also run on Mac OS X.

`abcsysbio` has a number of dependencies which change as different parts of the software are used. `numpy` and `matplotlib` are essential. `libsbml` must be installed if import from SBML is desired. We advocate the use of CUDA if possible, for which `cuda-sim` must be installed, or C++ which requires the *GSL* library to be installed. `scipy` must be installed if the Python ODE solver is to be used. To make installation easier, it is advisable to use the package together with the Python Enthought Distribution which contains `matplotlib`, `numpy`, `scipy`.

Once the dependent packages have been installed (see the README.txt in the package distribution), abc-sysbio is installed via the standard `distutils` interface. Download and unpack the code `abc-sysbio-XX.tar.gz` where XX is the version number.

```
$ tar -xzf abc-sysbio-XX.tar.gz
$ cd abc-sysbio-XX
$ python setup.py install
```

2.1 Linux

On linux this will copy the module `abcsysbio` into the `lib/pythonXX/site-packages` directory corresponding to the python version that was used to invoke the installation. In addition it will copy the scripts `run-abc-sysbio` and `abc-sysbio-sbml-sum` into the same `bin` directory as the python version that was used to invoke the installation. For example

```
$ /usr/local/bin/python2.6 setup.py install
```

places `run-abc-sysbio` and `abc-sysbio-sbml-sum` into `/usr/local/bin/` and `abcsysbio` into `/usr/local/lib/python2.6/site-packages`. If `run-abc-sysbio` and `abc-sysbio-sbml-sum` are already in the path then it should now be sufficient to issue the command

```
$ run-abc-sysbio -h
```

to get a list of options. Alternatively if they are not in the path, they can be added by issuing

```
$ export PATH=<dir>:$PATH
```

for 'sh' type shells or

```
$ setenv PATH <dir>:$PATH
```

for 'c' type shells. This must be done each time you open a new shell unless you add the line to your .bashrc / .cshrc files. Alternatively you can just call the scripts using the full path eg `/usr/local/bin/run-abc-sysbio`.

To run the C++ version of the code, you must also set two environment variables, `GSL_INC` and `GSL_LIB`, which point to the directories containing the `GSL` include and libraries respectively.

2.2 Mac OS X

On the Mac the location of the resulting files can be different depending on which flavour of python you are using. The output from the `install` command can be used to identify the location of the scripts and this directory should be added to the PATH as above. See the README.txt in the package distribution for one way to install `abc-sysbio` on the Mac.

Chapter 3

Using the package

3.1 Overview

`abc-sysbio` can be used as a standalone Python library but to facilitate the most common applications of the package, `abc-sysbio` is supplied with two scripts, `abc-sysbio-sbml-sum` and `run-abc-sysbio`. These scripts combine the functions in the package to implement the ABC algorithms.

`run-abc-sysbio` runs all the simulation, parameter inference and model selection algorithms and takes as input a .xml configuration file.

`abc-sysbio-sbml-sum` parses SBML files to produce a model summary containing information on the structure of the models structures. It also provides a template .xml configuration file for the user to edit and supply to `run-abc-sysbio` .

3.2 Conversion of SBML models to Python/C++/CUDA code

When `run-abc-sysbio` is run, model(s) written in SBML format are used to generate an appropriate code module representing the model, via a call to `abcsysbio_parser.ParseandWrite`. The format of the code written depends on the integration type, which also informs the program which solver to use to simulate the model. (See Section 5.2.)

In an SBML model, values can be assigned to compartment sizes, to parameters and to species. When the Python module representing the model is written, the model is inspected to determine how values should be assigned during the course of the simulation. Compartments typically have a constant size and are considered as an additional parameter to the model, albeit one that the user may not want to infer. Some species may be outside of the scope of the reactions in the model, having either a constant value or a value assigned by an assignment rule.

Conversely, SBML has the capacity for rate rules to be used to assign values to species, parameters or compartment sizes. Rate rules are time-dependent rules used to assign values to variables. Because our solvers typically assign values to species over time to compute a trajectory, species, parameters and compartment sizes with rate rules are treated as ‘species’ in a Python module representing the SBML model.

3.3 The user input file

Not all the information required for the algorithms is intrinsic to the model, therefore a .xml file that contains the additional information (such as initial conditions and parameter values) must be supplied to the script. This file, the “user input file”, must be written in a specific format. Examples are included in the “Examples” folder which is supplied along with the package; details of the input file are given here.

3.3.1 Required arguments

modelnumber Number of the model for which details are described in this input file

epsilon This allows for the specification of the tolerance schedules. Often only one epsilon schedule is required but in more complex design problems or with inferences using summary statistics multiple epsilon schedules may be desired. Within the `epsilon` tags each vector of schedules can be specified via a whitespace delimited list of values between two tags eg `<e1> </e1>`. Note that the parser ignores the name of the tag and so will always read them in order. If multiple schedules are provided then the user must specify a custom distance function (see Chapter 5.2).

autoepsilon This is an experimental feature used instead of the **epsilon** option where the epsilon schedule is automated. The vector of final epsilons (whitespace separated) can be specified within `<finalepsilon> </finalepsilon>` tags. The `<alpha>` tag specifies the quantile of the previous population distance distribution to choose as the next epsilon. The optimal value of this parameter will depend on the models and data.

particles Number of particles to accept

beta Number of times to simulate each sampled parameter set. For deterministic systems beta is set to 1. For analysis of stochastic systems beta can be chosen larger than 1.

dt The internal time step for the solvers.

data This section is divided into `<times>` and `<variables>`. Both describe the experimental data for which the parameters or models respectively have to be analyzed.

- **times** The time points are given within `<times>` tags as a whitespace delimited list.
- **variables** The species concentrations (in the case of an ODE or SDE simulation) or molecule numbers (in the case of a Gillespie simulation) are also given as whitespace delimited lists and denoted as `<v1>`, `<v2>` .. `<vN>` for N different species. Note that the names of the tags is ignored and the parser always reads the species in order. Missing data can be specified provided that the first entry in present. Missing data is denoted by 'NA'.

models Each model is contained within tags `<modeli>` ($i = 1, \dots, M$, where M is the total number of models to be investigated.)

- **name** The name of the model which will be the name of the code written (with suitable extension .py, .cpp, .cu) to represent the model in a format that is interpretable by the solvers. The name of the code model file if option `--localcode` is given.
- **source** The name of the .xml file containing an SBML representation of the model. Can be left blank if option `--localcode` is given.
- **type** The simulation type. One of `ODE`, `SDE` or `Gillespie`.
- **fit** Denotes the correspondence between the species defined in the SBML model and the experimental data. If this keyword is not given, or if **fit** is `None`, all the species in the model are fitted to the data in the order that they are listed in the model. Otherwise, a comma-delimited list of fitting instructions with the same length as the dimensions of the data can be supplied. Simple arithmetic operations can be performed on the species from the SBML model. To denote the `N`th species in the SBML model, use `speciesN`. For example, to fit the sum of the first two species in the model, write **fit**: `species1+species2`.

parameters, initial Prior specifications on parameters and initial conditions. Note that the tag names for each parameter and species initial condition are ignored and are always read in the order specified. The prior is specified within the tags via a whitespace delimited list:

- constant x: constant parameter with value x
- normal a b: lognormal distribution with location a and var b
- uniform a b: uniform distribution on the interval $[a, b]$
- lognormal a b: lognormal distribution with location a and var b

3.3.2 Optional arguments

modelkernel Used in model selection. This controls the model perturbation probability (default = 0.7).

modelprior This specifies the prior over the models. The default is a uniform prior over the model space.

kernel The implemented ABC SMC algorithms compute the pertubation kernels after each population, dependent on the previous particle distributions. Implemented distributions for the pertubation kernels are (default uniform)

- uniform : component-wise uniform kernels
- normal : component-wise normal kernels
- multiVariateNormal : multi-variate normal kernel whose covariance is based on all the previous population
- multiVariateNormalKNeigh : multi-variate normal kernel whose covariance is based on the K nearest neighbours of the particle
- multiVariateNormalOCM : multi-variate normal kernel whose covariance is the OCM

rtol, atol For models to be simulated as an ODE system these two keywords can be used to set the relative and absolute error tolerances for the numeric simulation. For stiff models, this may be necessary for successful simulation.

restart Frequently in the implementation of the ABC SMC algorithm, the epsilon schedule selected in the first instance might be sub-optimal, leading to a high acceptance rate and too wide a posterior distribution. In addition this makes parameter inference computationally expensive. To avoid wasting the information from initial attempts at parameter inference, it is possible to make a backup that stores the information about each popualation after it has been completed. With this backup one can stop the program, change the maximum distances or any other parameters and restart the program with the results of the last population. To do this set: `<restart>: True` When restarting from a backup population, it is important not to increase the population size and to keep the structure of the models constant. Permitted changes include **epsilon**, **beta**, **dt**, **rtol** and **atol**, the values in **data** (but not the structure), the initial concentrations, the prior distributions (for constant parameters) and the pertubation kernels. Which of these changes will make the inference more informative, we will leave the user to decide.

3.4 `abc-sysbio-sbml-sum`

This user input file can be written by hand; however, we forsee attempts to hand-write the user input file leading to errors. Therefore, the script `abc-sysbio-sbml-sum` is provided. This script generates two text files. The first one is a summary of all SBML models to be investigated. The second one is a template for the user input file. This template includes all necessary keywords with instructions how to fill in the information. It has already the correct structure for the models and provides default values for several keywords. Additionally if a data file is specified then the data will be passed into the template .xml file.

3.4.1 Command line options

The current options are listed below and be specified using the `--help` option.

–**files** a comma separated list of xml model files

–**data** a data file with columns time, variable1, variable2, variable3 .... (optional)

–**input_file_name** the name of the xml file to write (default input_file_template.xml)

–**summary_file_name** the name of the summary file to write (default model_summary.txt)

For example, to run `abc-sysbio-sbml-sum` with three SBML models, type

```
$ abc-sysbio-sbml-sum --files source1.xml,source2.xml,source3.xml
```

to produce the output files model_summary.txt and input_file_template.xml. The model summary is very useful for examining the model structure and is required to identify the order of the parameters.

3.5 `run-abc-sysbio`

To run `run-abc-sysbio`, use

```
$ run-abc-sysbio -i user_input_file.xml
```

3.5.1 Command line options

When running `run-abc-sysbio`, several options are possible. To see a list of these, type

```
$ run-abc-sysbio --help
```

The current options are listed below. There are two forms for arguments.

-i , –infile declaration of the input file. This input file has to be provided to run the program!

-lc , –localcode do not import model from sbml intead use .py, .cpp or cuda file

-sd , –setseed seed the random number generator in numpy with an integer eg -sd=2, –setseed=2

-tm , –timing print timing information

–**c++**, use C++ implementation

-cu, –cuda use CUDA implementation

-of , –outfolder write results to folder eg -of=/full/path/to/folder (default is _results_ in current directory)

-f , –fulloutput print epsilon, sampling steps and acceptence rates after each population

-s , –save no backup after each population

-S , –simulate simulate the model over the range of timepoints, using paramters sampled from the priors

-d , –diagnostic disable printing of diagnostic plots

-t , –timeseries disable plotting of simulation results after each population

-p , –plotdata disable plotting of given data points

-h , –help print this list of options.

3.5.2 Output

The outputs from running the ABC SMC algorithm are saved in a folder specified via the `-of --outfolder` option.

- `_data.png`, a scatter plot of your input data.
- `rates.txt` containing population number, number of sampled particles, acceptance rate and time to complete in seconds
- `ModelDistribution_1.png` and `ModelDistribution.txt` Histograms of the posterior distribution of accepted models after each population. Above each histogram the population number, epsilon, and acceptance rate for that population are displayed.
- One text file per population, `distance_PopulationN.txt`, listing the distances of the accepted particles together with the model number of the accepted model.
- One text file per population, `traj_PopulationN.txt`, the trajectories of the accepted particles. Each line contains:
  accepted particle number, replicate number (==0 if beta=1), model id, fitted species id, X(t=1), X(t=2) ......
- One sub-folder per model. These sub-folders, suffixed with the model name, contain sub-folders for each population, `population_N`. Each contains:
  - `data_PopulationN.txt`, the accepted parameter sets
  - `data_WeightsN.txt`, the accepted parameter weights
  - `ScatterPlotPopulationN.png`, scatter plots of all accepted parameters. (See Figure 4.3)
  - `TimeseriesPopulationN.png`, simulations of the model using ten accepted parameter sets, to compare with the data. Refer to Figure 4.4.
  - `weightedHistograms_PopulationN.png`, histograms showing accepted parameter distributions.
- `copy` contains in binary data form the information required to restart the ABC SMC algorithm using the last population. These files are not human-readable but are read into Python if the algorithm is being run restarting from a previous population. See Example 2.

3.5.3 Simulation mode

Beside implementing the ABC SMC algorithms the program `run-abc-sysbio` provides an easy way to simulate biochemical systems directly from the SBML source. Simulation mode requires the user input file but information on the `epsilon` schedule and the `variables` section of `data` are not required or ignored if present. Here `particles` specifies the number of simulations to perform, parameters are sampled from their priors and multiple models can be specified, together with `modelprior`, so that model averaging can be performed.

The output folder in simulation mode contains

- particles.txt, the simulated parameter sets
- trajectories.txt, he trajectories of the accepted particles. Each line contains:
  accepted particle number, replicate number (==0 if beta=1), model id, fitted species id, X(t=1), X(t=2) ......
- One .png file for each model plotting the simulated timeseries

Chapter 4

Examples

The following describes the examples contained in the package. By default the package uses Python to simulate models but the examples containing SBML files, namely Example1, Example3 and Example4, can all be run in C++ and CUDA modes.

4.1 Example 1 : Epidemiology of infections diseases using ODEs

4.1.1 Background

We illustrate an example of the ABC SMC algorithm for model selection using a range of simple models that describe the epidemiology of infections diseases. This example reproduces the findings of (Toni *et al.*, 2009). As described by (Toni *et al.*, 2009), SIR models describe the spread of such disease in a population of susceptible (S), infected (I) and recovered (R) individuals (Anderson & May, 1991). The three models used for this example use systems of ordinary differential equations (ODEs) to describe the change in the number of S, I and R indivuduals over time.

Model 1

In the simplest case, the model assumes that every individual can be infected only once and that there is no time delay between the individual getting infected and their ability to infect other individuals.

$$\begin{aligned}\dot{S} &= \alpha - \gamma SI - dS \\ \dot{I} &= \gamma SI - vI - dI \\ \dot{R} &= vI - dR\end{aligned}$$

where $\dot{x}$ denotes the time derivative of x, dx/dt. Individuals, who are born at rate α, are susceptible (S). All individuals die at rate d, γ is the infection rate and v is the recovery rate.

Model 2

To represent the delay between the time an individual gets infected and the time when they become infections, a population of individuals in a latent phase of infection, L, can be included in the model. Individuals in the latent population are infected but cannot infect others. Indiviuals make the transition from the latent phase to the infective phase at rate δ.

$$\begin{aligned}\dot{S} &= \alpha - \gamma SI - dS \\ \dot{L} &= \gamma SI - \delta L - dL \\ \dot{I} &= \delta L - \upsilon I - dI \\ \dot{R} &= \upsilon I - dR\end{aligned}$$

Model 3

Another extension of the basic model allows the recovered individuals to become susceptible again at rate e.

$$\begin{aligned}\dot{S} &= \alpha - \gamma SI - dS + eR \\ \dot{I} &= \gamma SI - \upsilon I - dI \\ \dot{R} &= \upsilon I - dR - eR\end{aligned}$$

4.1.2 Generating the input file

The package `abc-sysbio` is primarily written to handle SBML models. The BioModels Database (Le Novere *et al.*, 2006), a database of SBML models, contains mathematical models of biological interest. These SIR models, however, are not available from the BioModels database, so SBML models have been constructed for the purpose of this example. To construct the models, we make use of SBML shorthand and its conversion tools (Gillespie *et al.*, 2006), and construct the three SIR models, `SIRmodel1.xml`, `SIRmodel2.xml` and `SIRmodel3.xml`. We have also provided a file `data.txt` containing the (simulated) data for this example.

Two scripts are included in the package `abc-sysbio` to automate the most common applications of the modules in the package. The scripts `abc-sysbio-sbml-sum` and `run-abc-sysbio` should be placed in the standard locations upon installation. As can be seen from Figure 1.2, `abc-sysbio-sbml-sum` is the first script to use. At the command line, type

```
$ abc-sysbio-sbml-sum --files SIRmodel1.xml,SIRmodel2.xml,SIRmodel3.xml
--data data.txt
```

In your working directory, two new files will be written: a summary of the model(s), `model_summary.txt`, and a template file, `input_file_template.xml`.

Inspection of `model_summary.txt` will reveal a summary of the properties of the model(s), including the numbers of species and parameters contained therein, how these are handled by the simulators, and the presence of any rules, events and functions in the model. `model_summary.txt` is primarily for information and for verification that the properties of your model are as expected.

`input_file_template.xml` contains, or will contain, all the information to be passed to the ABC SMC algorithm. It might be prudent to rename this template file with a more meaningful name.

```
$ mv input_file_template.xml input_file_SIR.xml
```

`input_file_template.xml` is written with its own comments in it, preceeded by `#` (as in Python). These comments should help the user to complete the template; additional comments can be added, preceded by `#`.

`run-abc-sysbio` has the capacity to simulate the model (see Example 2), to generate a synthetic data set, or to perform the ABC SMC algorithm using the information in the input file.

4.1.3 Editing the input file

To perform ABC SMC, some additional information does need to be added to the input file.

epsilons A whitespace delimited list of floats, these are the distances below which a parameter set is accepted by the algorithm.

particles Integer, the size of each population.

dt Float, the internal timestep. For a stiff model, a small **dt** is required for a successful simulation. However, the smaller the value of **dt**, the longer the simulation will take. The SIR models are not particularly stiff so **dt** can be set to 1.

Some model-specific information must be completed for each of the models.

name The name of the python file that will represent the model. Defaults to `model1`, `model2`, ... ,`modelN`. These have been renamed in the input file we are using.

source The name of the .xml file containing the SBML model. The source file(s) must be in the same working directory as the scripts and the input file.

type The integration type used to simulate the model. Defaults to `ODE`. Other integration types are `SDE` (see Example 2) or `Gillespie` (see Example 3).

fit A string, defaults to `None`. If **fit** is not `None`, it must take the form of a comma-delimited list of fitting instructions describing how to fit the species in the SBML model (denoted `species1, species2, ... , speciesN`) to the variables given in the **data**. Model 2 in this example describes the trajectories of four species, including the latent population L. However, our data only describes three variables. We give the fitting instruction

```
fit: species1, species3, species4
```

to denote that the first species in Model 2 describes variable 1 in the data, the third species in Model 2 describes variable 2 and the fourth species in Model 2 describes variable 3. Fitting instructions can include simple arithmetic operations. For example if data represent the sum of the first two species in a model the fitting instruction for that variable should be written `species1+species2`.

Priors For each parameter and initial condition we specify the distribution:

- constant value
- normal location variance
- uniform lower upper
- lognormal location variance

4.1.4 Running ABC SMC

To recap, in the current working directory we have

- Three SBML models to select from, `SIRModel1.xml`, `SIRModel2.xml` and `SIRModel3.xml`
- The completed input file, `input_file_SIR.xml`
- The scripts `abc-sysbio-sbml-sum`, which we have already used, and `run-abc-sysbio` which we will use now.

`run-abc-sysbio` has a number of options. To see a list of these, use the inbuilt help function.

```
$ run-abc-sysbio -h
```

By default, `run-abc-sysbio` requires one argument, `--infile filename.xml`. So to use our completed input file to run ABC SMC, at the prompt we type

```
$ run-abc-sysbio --infile input_file_SIR.xml
```

Depending on the stiffness of the model(s), the length of each simulation, **dt**, the priors, epsilons and population size, ABC SMC may take from minutes to hours to run. To monitor the progress of the algorithm, run `run-abc-sysbio` with full output:

```
$ run-abc-sysbio --infile input_file_SIR.xml --fulloutput
```

In this example, the data (which are available in `input_file_SIR.xml`) are generated from Model 1 and perturbed with Gaussian noise, $N(0, (0.2)^2)$. From the posterior distributions (Figure 4.2), after eleven populations, the correct model has been selected 1000 times in a population of 1000 particles.

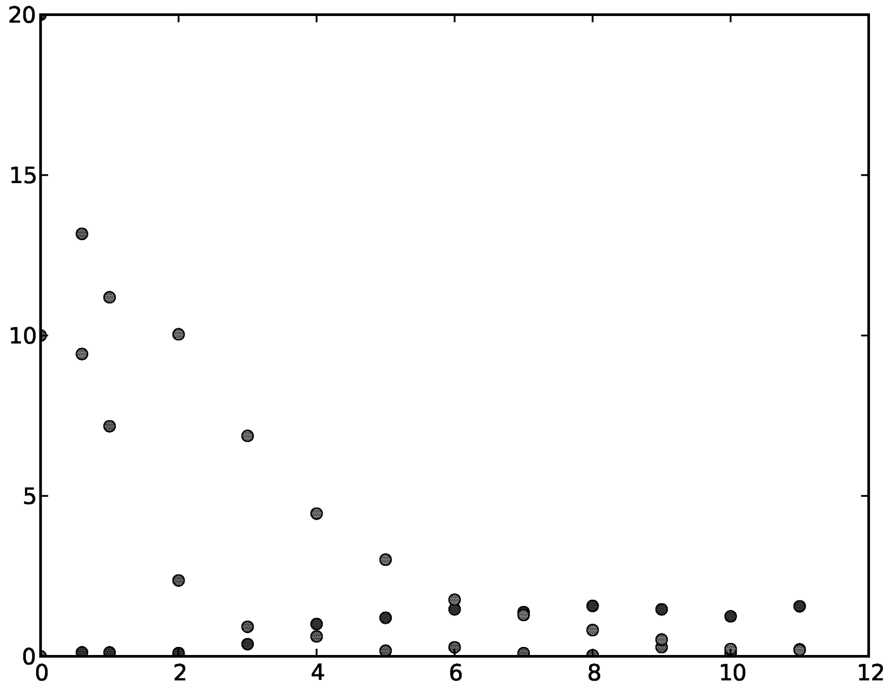

Figure 4.1: Scatterplot of data from an SIR model as input to abc-sysbio.

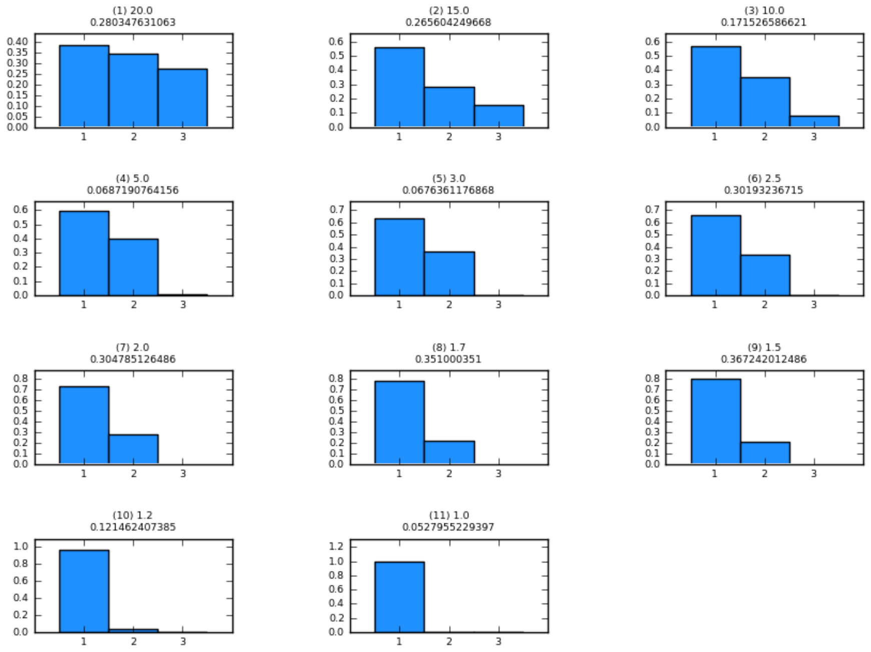

Figure 4.2: Posterior distributions of the three SIR models.

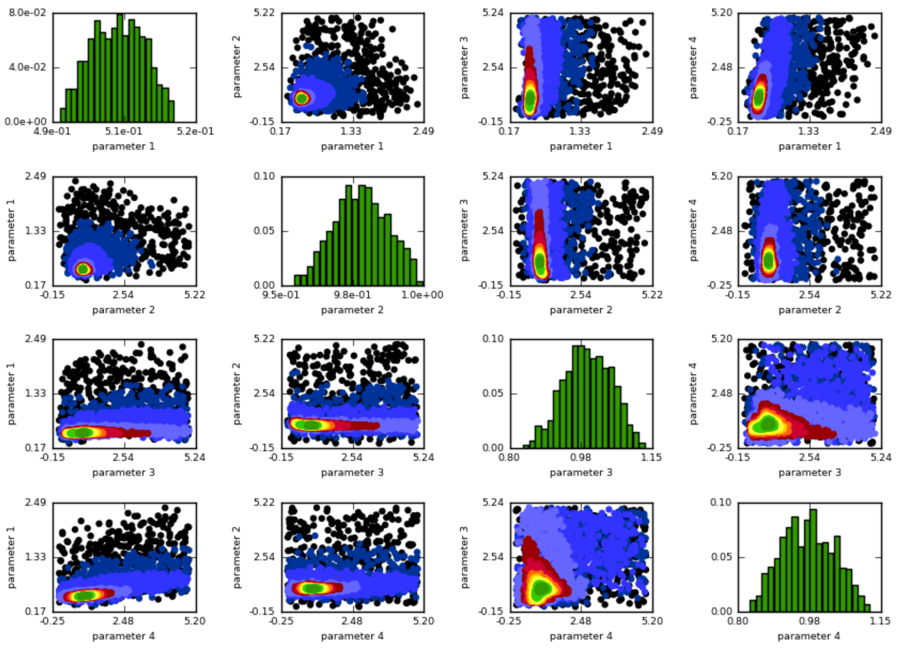

Figure 4.3: Posterior distributions of parameters of SIR Model 1, final population.

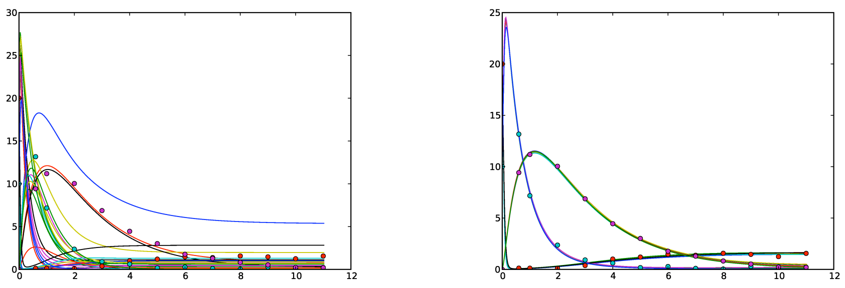

Figure 4.4: Simulations of SIR Model1 using accepted parameters from the first (left) and last (right) populations.

4.2 Example 2: An SDE model for linear growth

As shown in Figure 1.2, if only one model is used, in the absence of model selection the package `abc-sysbio` effectively performs ABC SMC for parameter inference. In this section we perform ABC SMC for parameter inference in a stochastic linear growth model.

This model is an instance when the flexibility of the package is useful. The noise terms in the Python representation of the model automatically generated by the program are too large for the model to be simulated efficiently. (See Chapter 6). In this example the Python file has been edited to reduce the magnitude of the noise terms.

`run-abc-sysbio` has the capacity to generate a synthetic data set from the model. To do this, a series of timepoints at which data are required must be added to the input file, and all the parameters in the input file must be constant. In this example, the data were simulated from the model parameterised with parameters $10.0, 1.0$.

```
$ run-abc-sysbio --infile input_file_lingrow_simulation.xml --simulate -lc
```

In the input file, we input the data and the initial epsilon schedue, $\epsilon = \{5, 4, 3\}$. The population size is 500. The option `-lc` tells the package to use local python files and not try an import from an SBML file.

To run our script, we type

```
$ run-abc-sysbio  --infile input_file_lingrow.xml -lc
```

After running the script, we inspect the outputs. From Figure 4.5 we see that the accepted parameters span the entire prior distribution. Our choice of ϵ was too large for meaningful parameter inference. Using the **restart** option, we can adjust the epsilon schedule and restart the program, making use of the posterior distribution from the last population. To do this, we make a copy of the user input file.

```
$ cp input_file_lingrow.xml input_file_lingrow_restart.xml
```

In the new user input file, we set the **restart** flag in the input file to `True` and change the epsilon schedule. We can then restart the program. The data in the binary data files saved in the folder `copy` will be read into python and used as the posterior distribution from which the parameters are sampled.

```
$ run-abc-sysbio --infile input_file_lingrow_restart.xml -lc
```

By inspecting the output, we can see that after restarting the program with a more appropriate epsilon schedule, more meaningful parameter inference is obtained.

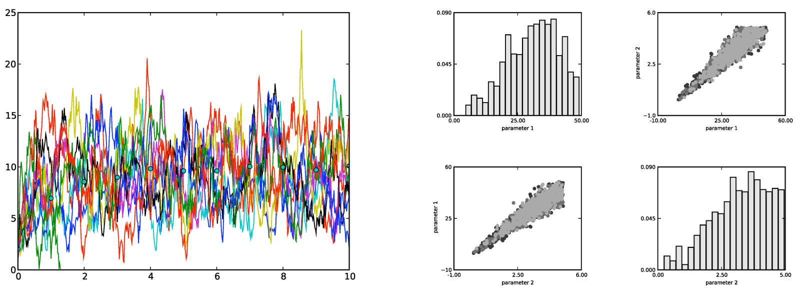
Figure 4.5: Time series and posteriors for the stochastic linear growth model with $\epsilon = 6$. Left, time series using ten accepted parameter sets. Data are turquoise circles; Right, posterior distributions after the third population

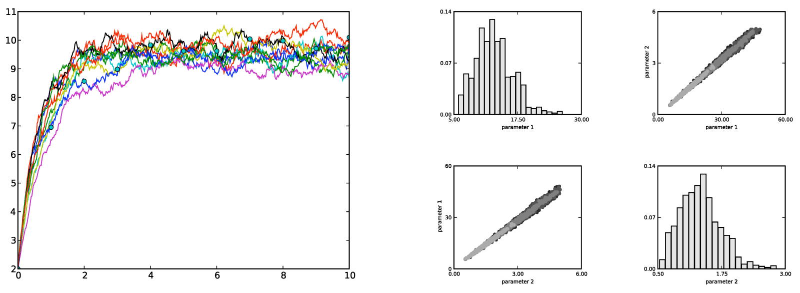
Figure 4.6: Time series and posteriors for the stochastic linear growth model with $\epsilon = 4$. Left, time series using ten accepted parameter sets. Data are turquoise circles; Right, posterior distributions after the fourth population.

4.3 Example 3: Modelling dimerisation using Chemical Master Equation

In an even simpler case than that described in Example 2, specifying a single epsilon will return a single population of accepted parameters, effectively implementing the ABC rejection algorithm (Pritchard *et al.*, 1999). We give an example using the dimerisation model of (Wilkinson, 2006). This model has been written with appropriate rate laws for a discreet stochastic simulation using Gillespie's direct method (Gillespie, 1977).

As in Example 2, in this example the data are data from the model (Figure 4.7). The

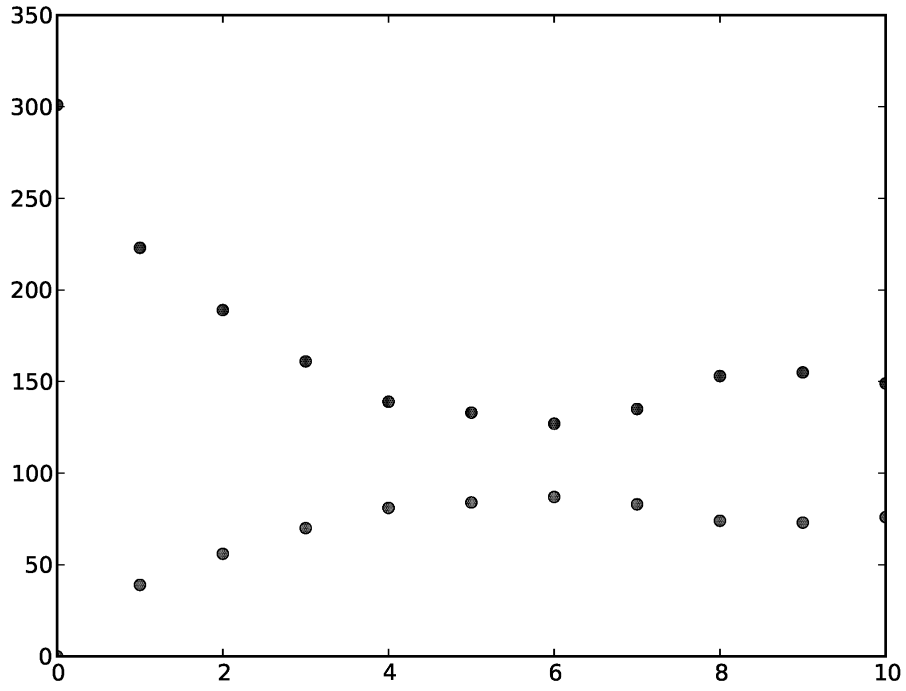

Figure 4.7: Simulation of the dimerisation model of (Gillespie, 1977) using "Gillespie's Direct Method". These data are used in the ABC-Rejection algorithm.

procedure for using `abc-sysbio` to implement the ABC rejection algorithm follows that of the use of the package to implement ABC SMC. By specifying a single epsilon together with a size for the population of accepted particles, a single population of accepted parameters is returned.

Figure 4.8 shows the posterior distributions of the two parameters in the dimerisation model. The true parameter values were $1{\times}10^{-15}$, $1.66{\times}10^{-3}$ and 0.2 for parameters 1,2 and 3 respectively . As discussed by (Toni *et al.*, 2009) and (Sisson *et al.*, 2007), the ABC rejection algorithm is less efficient than ABC SMC. With the ability to restart the algorithm using the posterior distribution of a population of accepted parameters, the user can transition from ABC rejection to ABC SMC to narrow the posterior distribution further with further iteration(s) of ABC.

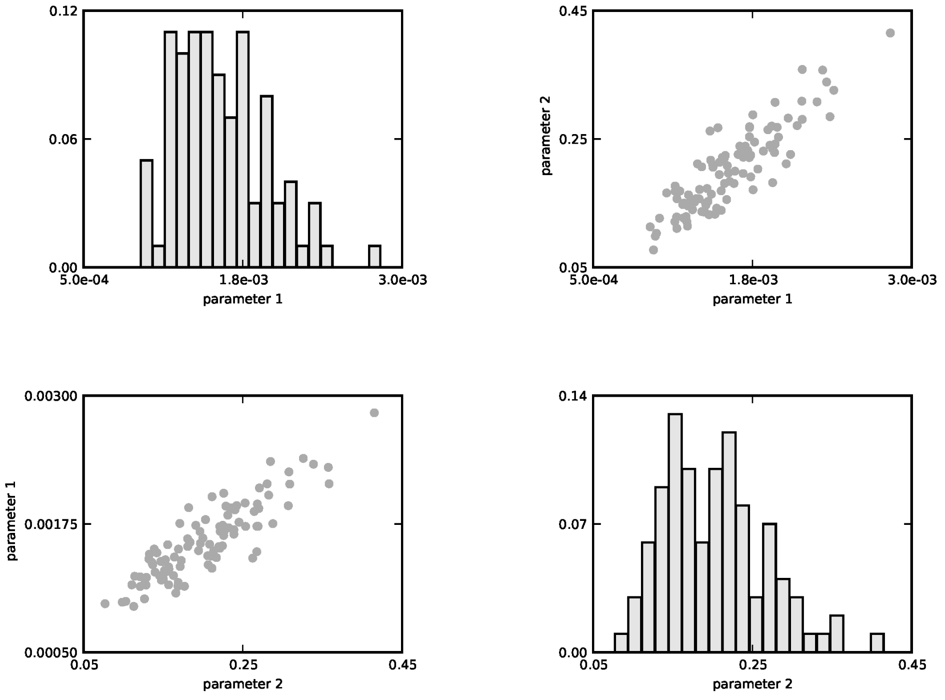

Figure 4.8: Posterior parameter distributions of parameter 1 and parameter 2 in the dimerisation model.

4.4 Example 4: Population growth models modelled with CME, SDE and ODE

Example 4 that comes with the `abc-sysbio` package contains a simple model selection example using all three simulation algorithms. The data has been simulated under a stochastic immigration-death model. We will perform model selection using a stochastic immigration death model (true model) and a stochastic logistic growth model.

Chapter 5

Extending the library

5.1 Using the modules independently of the scripts

`abcsysbio` can be imported into an interactive Python session. By defining the arguments to the functions in the package in the interactive namespace, the functions can be used in the Python shell.

5.2 Writing code to represent models

5.2.1 Python

Examples of the format of Python modules that are compliant with ODE, SDE and Gillespie solvers are supplied with the package. These are the models that were genarated by the package when the Examples were prepared (see Section 4).

Broadly, we use the function definition `modelfunction` to define the callable function solved by the ODE and SDE systems. Functions passed to the ODE solver return the derivatives of the species whose trajectories are described by the function; models for simulation by the SDE solver also return the noise terms for each species. An additional function, `rules`, is obligatory even if empty. `rules` takes as arguments species, parameters and the time, `t`, and returns the species and parameters. The `rules` function and can contain time-dependent rules or events. User-defined functions required by the model can be defined at the top of the module and called within it.

Models for the Gillespie solver are written with one function, `Hazards`, which returns the hazards for each reaction.The reactions themselves are defined as `reaction0`, `reaction1`, . . . , `reactionN`. Each reaction takes all species as arguments and returns all species, unchanged if these are not changed by the reaction. A dictionary, defined `Switch`, is used by the solver to call the chosen reaction.

5.2.2 CUDA

Detailed instructions on how to write CUDA code for the various simulators can be found in the `cuda-sim` package.

5.3 Defining your own distance functions

Defining custom distance functions is straightforward. The user needs to to pass the argument

The module `euclidian` contains a function, `euclidianDistance`, which returns the Euclidian distance between two arrays of equal dimensions. The advanced user could define his own distance function, either in the `euclidian` module or elsewhere. The distance function is called in the `abcSimulator` function within the `abcSMC_model` module; the user would need to call his own distance function to compare the experimental and simulated data.

There is an alternative way to define custom distance functions when using `run-abc-sysbio`. If the argument `--custd` is given, the program looks for a module named `customABC` in the current directory and a function named `distance`. This must handle the same format as `euclidianDistance`, namely take two same sized data matrices and return a scalar.

Chapter 6

Troubleshooting

Before the ABC algorithm is implemented, the package checks that all the supplied information is consistent with each other and with the model(s) being used. If a discrepancy is found an error message is returned and the program is stopped. Despite this, some known issues remain.

- When attempting to simulate a stiff ODE model, the error message

  ```
  Excess work done on this call. Perhaps wrong Dfun type.
  ```

  may be encountered. This error message arises when the model is too stiff to be solved successfully by the `odeint` solver from Scipy. If this error message is encountered, specifying a smaller **dt** and/or a small **rtol** or **atol** (on the order of $1e^{-5}$ to $1e^{-7}$) may allow a stiff model to be successfully simulated; however, such a stiff model may take such a long time to be simulated as to be impractical for ABC SMC.

Bibliography

Anderson, R. M. & May, R. M. 1991 *Infections diseases of humans: dynamics and control* Oxford Science Publications.

Gillespie, C. S., Wilkinson, D. J., Proctor, C. J., Shanley, D. P., Boys, R. J. & Kirkwood, T. B. L. 2006 Tools for the SBML Community *Bioinformatics* **22**, 628–629.

Gillespie, D. T. 1977 Exact stochastic simulation of coupled chemical reactions *The Journal of Physical Chemistry* **81**, 2340–2361.

Klöckner, A., Pinto, N., Lee, Y., Catanzaro, B., Ivanov, P. & Fasih, A. 2009 Pycuda: Gpu run-time code generation for high-performance computing *arXiv* **cs.DC**.

Le Novere, N., Bornstein, B., Broicher, A., Courtot, M. *et al.* 2006 Biomodels database: a free, centralized database of curated, published, quantitative kinetic models of biochemical and cellular systems. *Nucleic Acids Research* **34**, Database issue.

Pritchard, J., Seielstad, M. T., Perez-Lezaun, A. & Feldman, M. W. 1999 Population growth of human Y chromosomes: a study of Y chromosome microsatellites. *Molecular Biology and Evolution* **16**, 1791 – 1798.

Sisson, S. A., Fan, Y. & Tanaka, M. M. 2007 Sequential Monte Carlo without likelihoods *Proceedings of the National Academy of Sciences* **104**, 1760–1765.

Toni, T. & Stumpf, M. P. H. 2010 Simulation-based model selection for dynamical systems in systems and population biology *Bioinformatics* **26**, 104–10.

Toni, T., Welch, D., Strelkowa, N., Ipsen, A. & Stumpf, M. P. H. 2009 Approximate bayesian computation scheme for parameter inference and model selection in dynamical systems *Journal of the Royal Society Interface* **6**, 187 – 202.

Wilkinson, D. J. 2006 *Stochastic Modelling for Systems Biology* Chapman & Hall/CRC.

Zhou, Y., Liepe, J., Sheng, X., Stumpf, M. P. & Barnes, C. 2011 Gpu accelerated biochemical network simulation *submitted* .