abc-sde

A MATLAB toolbox for approximate Bayesian computation (ABC) in stochastic differential equation models

https://sourceforge.net/projects/abc-sde/

Umberto Picchini

www.maths.lth.se/matstat/staff/umberto/

User's Guide and Reference Manual for release 0.1.1

Contents

Credits

The `abc-sde` toolbox has been conceived and coded by Umberto Picchini (Centre for Mathematical Sciences, Lund University, Sweden), but for the notable exceptions listed below.
The creation of this software has been partially supported by the Faculty of Science at Lund University, Sweden, under the grant "Money Tools" (*verktygspengar*) during years 2012–2013.

`abc-sde` includes the following software: (i) `glmnet` for MATLAB [1] for a Lasso type regularisation [2]; (ii) `ARESLab` [3] for an implementation of multivariate regression splines [4]. For an example using exact simulation from the Lotka–Volterra model `GillespieSSA` [5] has been used. The following utilities are available for processing the output produced by `abc-sde`: `acf.m` (autocorrelation function) [6] and `perctile.m` (empirical percentiles) [7].

`abc-sde` is released under the GPLv3 license. If you have found it useful for your research please give credit where credit is due:

Cite the paper

U. Picchini (2013). Inference for SDE models via approximate Bayesian computation. `arXiv:1204.5459`.

Cite the software

U. Picchini (2013). `abc-sde`: a MATLAB toolbox for approximate Bayesian computation (ABC) in stochastic differential equation models, `https://sourceforge.net/projects/abc-sde/`.

Preface

As it is often the case the creation of a software, a package or a toolbox released for public use, is motivated by the possibility to expand a set of files initially meant for its author private use. The same circumstances led to the development of this first release of `abc-sde`[1], which has expanded out of a program developed to implement the methodology in [8]. The mentioned reference is often cited in this guide, however results reported there have been obtained using routines not exactly similar to those now available in `abc-sde`, and as such results in [8] are not exactly replicable by this package.

The purpose of `abc-sde` is to make available some methods for conducting Bayesian inference for parameters of models defined by stochastic differential equation (SDEs). However, since inference for SDEs is notoriously complicated (not only in the Bayesian framework but also in the "frequentist" one), particularly when dealing with multidimensional SDE systems, considering the possibility of applying the flexible tools offered by approximate Bayesian computation (ABC) (e.g. [9, 10]) comes naturally. Of course we are not claiming that ABC has to be considered the inferential framework of choice, as the problem of conducting parameter estimation for SDEs has generated a vast literature. However, when it comes to considering multidimensional systems, with discretely observed measurements affected with error, or when the observations given the latent states are not conditionally independent, it is very difficult and often computationally demanding to conduct exact inference using conventional tools (e.g. likelihood maximization or exact Bayesian inference). Therefore in some cases ABC can offer some relief to deal with complex problems, even though the resulting inference is not exact (except under some extreme conditions such as knowledge of sufficient summary statistics for the unknown parameters and using a zero "tolerance" for statistics comparison).

`abc-sde` applies ABC to multidimensional (even partially observed) SDE models observed with error by considering an MCMC algorithm, where simulations are accelerated using an "early rejection" trick as described in [8] (although when specifying uniform priors for the ABC-MCMC procedure, i.e. while running `abc_mcmc`, there might be no significant acceleration, see section 4 in [8]) and regression based methods are used to determine summary statistics, as suggested in [11]. Also, the ABC tolerance can be chosen *a-posteriori* after the end of the MCMC simulation. Two examples are illustrated in this guide and although I am aware that these are rather low-dimensional compared to many applications of stochastic modelling, it has to be considered that: (i) SDE modelling, because of its inferential complexity, is rarely considered beyond dimensions of, say, order 5 (unless model simulation is the only requirement, instead of inference) and (ii) for illustration purposes it is not feasible to consider high dimensional examples, as our aim is to

[1] In a sweep of intellectual inspiration the package was christened `abc-sde` to denote (hold your breath) approximate Bayesian computation (ABC) for stochastic differential equations (SDEs).

show the potential of the methodology while allowing algorithms to run smoothly on a common pc. See for example [8] for an application considering a four-dimensional SDE.

Summary statistics for ABC are computed following a regression approach [11], using multivariate adaptive regression splines [4] or a lasso-type regularisation [2]. Regression splines is enabled by interfacing `abc-sde` with `ARESLab` [3], and lasso with `glmnet` [1]. Both packages are provided within `abc-sde`. However since the main purpose of `abc-sde` is ABC inference for SDEs and not multivariate regression, we provide hard-coded defaults for `ARESLab` and `glmnet`, useful for a typical usage (for example it is in principle possible to use the more general *elastic-net* approach implemented in `glmnet`, which includes "'lasso" and "ridge regression" as important special cases). Users accustomed to nonlinear regression modelling might wish to customize these defaults and in such case the functions to modify are:

`abc_stats.m`: modify input parameters to `aresparams`, `aresbuild` and `cvglmnet`;

`abc_mcmc.m` modify input parameters to `arespredict` and `glmnetPredict`.

For detailed information users are encouraged to read the documentation provided by the original authors for `ARESLab` and `glmnet`, the former being available into `\...\abc-sde\ARESLab` while for `glmnet` relevant references are provided at `http://www-stat.stanford.edu/~tibs/glmnet-matlab/`.

Umberto Picchini
Lund, May 2013
umberto@maths.lth.se
www.maths.lth.se/matstat/staff/umberto/

Introduction

We present `abc-sde`, a software implementing a Markov chain Monte Carlo (MCMC) algorithm for approximate Bayesian computation (ABC) allowing inference for stochastic differential equation models. The reference for the relevant methodology is [8], please consult it for motivational aspects, background and useful references. Important reviews on ABC methodology are [9, 10].

Goal of our work is to enable inference for parameters of stochastic models having latent states expressed by a d-dimensional Itô stochastic differential equation (SDE). A time-inhomogeneous SDE can be written as

$$dX_t = \mu(X_t, t, \theta)dt + \sigma(X_t, t, \theta)dW_t, \qquad X_{t_0} \in \mathbb{R}^d, \quad t \geq t_0 \geq 0 \tag{1}$$

where $\mu(\cdot) : \mathbb{R}^d \times \mathbb{R}^+ \times \mathbb{R}^p \to \mathbb{R}^d$ is the *drift* of the SDE, $\sigma(\cdot) : \mathbb{R}^d \times \mathbb{R}^+ \times \mathbb{R}^p \to \mathbb{R}^{d\times m}$ is the *diffusion* of the SDE, and W_t is an m-dimensional process having m independent (standard) Brownian motions.

We assume the availability of *noisy* observations: this means that data, sampled at $n+1$ observational times $0 \leq t_0 < t_1 \cdots < t_n$, are not directly modelled via solutions to (1), but instead some form of "error" is superimposed to the SDE solution. We do not consider a specific interpretation for what such error actually is (see [12] for interesting implications of considering the error as "model-error" or "measurement-error").

A generic noisy measurement at sampling time t_i is denoted with y_i ($\equiv y_{t_i}$), where the latter is a draw from some probability distribution underlying the following error–model

$$Y_i = f(X_i, \varepsilon_i), \qquad i = 0, 1, ..., n; \quad Y_i \in \mathbb{R}^{d_i}, \quad d_i \leq d \tag{2}$$

where $f(\cdot)$ is a known real-valued function and the ε_i are independent draws from a probability distribution independent of X_i ($\equiv X_{t_i}$). Therefore our data (sometimes denoted as "'measurements" or "observations") is the sequence $\{y_i\}_{i=0,...,n}$. We will often consider the case where $f(\cdot)$ induces an additive relationship, for example for scalar Y_i

$$Y_i = X_i + \varepsilon_i, \qquad i = 0, 1, ..., n \tag{3}$$

where the ε_i are assumed i.i.d. Gaussian[2] $\varepsilon_i \sim N(0, \sigma_\varepsilon^2)$ with ε_i independent of X_0 and W_t. Although this is a common setup in many practical situations we stress that in `abc-sde` we are *not* forcing such assumptions on model (2), i.e. we do not need to assume (3) to hold, nor assume Gaussian errors. It is possible to consider correlated errors and also measurements which are

[2]Here and in the following $x \sim N(a, b)$ denotes a scalar random variable x having Gaussian distribution with mean a and variance b.

not conditionally independent given the latent process X_t, see section 3.1.3. Therefore we are *not* confined in the so called "state-space" modelling framework (also known as Hidden Markov Model, see [13]), although this is an important topic which is studied in the two considered applications.

The model object of study is

$$\begin{cases} dX_t = \mu(X_t, t, \theta)dt + \sigma(X_t, t, \theta)dW_t, & X_{t_0} \in \mathbb{R}^d \\ Y_i = f(X_i, \varepsilon_i), \quad \varepsilon_i \sim \pi(\varepsilon_i; \sigma_\varepsilon), \quad i = 0, 1, ..., n. \end{cases} \tag{4}$$

where $\pi(\varepsilon_i; \sigma_\varepsilon)$ denotes the (known) distribution of the error term, depending on some (typically unknown) variance components coded into σ_ε.

Depending on the application scenario the initial state of the system X_0 might be known and set equal to a constant $X_0 = x_0$ otherwise we consider it as an unknown parameter. Similarly the components of the distribution set on ε can be considered as unknowns. In next chapter we establish some additional notation used in `abc-sde` and we anticipate that the vector of unknown pararameters object of inference (typically including θ and which may also include X_0 and variance components) is denoted with `theta`. We also introduce a larger vector named `bigtheta` containing *all* structural parameters involved in (4), that is *also* including known constants. Of course `theta` is a subset of `bigtheta` and only components in `theta` will be estimated via ABC-MCMC. A special remark is required for the parameter δ (see [8]), denoting the tolerance/bandwidth for ABC, i.e. a bound to the acceptable discrepancy between summary statistics computed on real data and synthetic data: although δ is considered unknown and it has its own Markov chain automatically constructed in `abc-sde`, it *should not* be specified in `bigtheta` nor in `theta`, see instead the section on ABC-MCMC-related parameters on page 17 and section 3.4.

Therefore object of the inference is `theta`, and the statistical framework of choice here is Bayesian, thus we are interested in obtaining draws from some approximation of the posterior distribution of `theta` given available data.

Depending on the experimental/sampling setup available for the problem at hand we can have two types of observational schemes:

fully observed system
: In this case the dimension of Y_i at *each* sampling time t_i is exactly d (same as the dimension of process $\{X_t\}$), that is to say all coordinates of the system have been observed.

partially observed system
: With "partially observed" we mean an experiment for which *not all* the d coordinates of the multidimensional process $\{X_t\}$ are observed at each sampling time t_i. That is at time t_i the observed y_i has dimension $d_i \leq d$ where d_i is the number of observed coordinates at t_i, see equation (2).

`abc-sde` is able to handle both scenarios: the fully observed case is considered in both examples discussed in this guide (chapters 3 and 4) while the partially observed case is considered only for the second example (section 4.5).

Part I

abc-sde

Chapter 1

Installation and limitations

1.1 Installation

Go to the `abc-sde` project webpage `https://sourceforge.net/projects/abc-sde/` and download the latest archive. Extract the archive in you preferred location then add it to the MATLAB search path together with all its subfolders: i.e. from the MATLAB File Menu, select *Set Path*, then choose *Add with Subfolders* then select the `abc-sde` folder. For a typical usage it is possible to avoid loading the `gillespiessa` folder, which is only required to run the example in Chapter 4.

1.2 Limitations and dependencies

- This is an alpha-release of `abc-sde`. At this development stage, in order to avoid serious issues related to delicate tasks such as random numbers generation, *the present release depends on the* MATLAB STATISTICS TOOLBOX. However such dependency simply requires access to very few basic statistical functionalities, essentially pseudo-random number generation, pdf's and cdf's[1]. Therefore the dependency will be "soon" removed in next releases. In the meantime users can substitute such functions with similar ones readily available on the internet.

- `abc-sde` uses `glmnet` for MATLAB [1] and `ARESLab` [3] to perform certain tasks. We cannot provide support for `glmnet` and `ARESLab` in case of malfunction. Moreover here follows a remark from the `glmnet` authors as from `http://www-stat.stanford.edu/~tibs/glmnet-matlab/`:

  ```
  The distribution in the file glmnet_matlab.zip contains a dll for Windows XP
  (tested with Matlab R13) and Windows Vista (tested with Matlab R14), a 64-bit
  Linux version (tested with Matlab R2008b), and a 64-bit build of glmnet on Windows
  XP (tested on Matlab R2009a and R2010a).  We may add precompiled versions for
  other platforms such as 32 bit Linux in the future.  For now, users of those
  platforms can easily compile the code themselves.
  ```

[1] This is the list of functions from the Matlab Statistics Toolbox which are accessed by `abc-sde`: `mvnrnd`, `normrnd`, `exppdf`, `normcdf`, `normpdf`, `unifrnd`, `randsample` (used in glmnet).

- `abc-sde` has been developed and tested with MATLAB R2011a for Windows (32 and 64-bit versions). Please let us know if you experience problems under different setups.
- the `TO-DO.txt` file gives a wish-list for some improvements.
- the `LOG.txt` file gives the list of changes.
- the development and maintenance of `abc-sde` is a one-man-effort. Also, it is an entirely voluntary task, so please be patient if the author does not succeed in implementing new features, fix bugs, expand the documentation etc. as rapidly as you (and we all) would hope.

Chapter 2

List of relevant variables

Here we list and describe all the *relevant* variables, structures etc. required for using `abc-sde`. We divide them according to the framework where they operate. With "relevant" we mean those variables that need to be created by the user and manually supplied to functions. Documentation for non-user-supplied variables is hard-coded in the corresponding functions: type `help name_of_function` at the prompt to read the documentation for function `name_of_function`.

SDE-related variables

`sde_param` a *structure*: it contains all the settings required to simulate a trajectory of the (approximated or exact) SDE solution. `sde_param` should always be constructed by respecting the following scheme
`sde_param = {bigtheta,problem,owntime,time,numdepvars,vrbl,integrator};`

`bigtheta` an array containing all structural parameters required in system (4), i.e. θ, X_0 and components for the distribution of ε_t. Some of those may be known constants and others unknown quantities. Notice the bandwidth/tolerance δ value *should not* be specified in `bigtheta`, see instead the ABC-MCMC-related parameters on page 17 and section 3.4.

`problem` a user defined string identifying the name of the "problem" under consideration, e.g. `'theophylline'` (example in section 3.6), `'lv'` (chapter 4) etc. All file-names corresponding to a given problem should start with `myproblem_` where `myproblem` is the string defined into `problem`: e.g. `lv_sdefile.m`, `lv_prior.m` etc.

`owntime` an array containing the time-grid for the numerical discretization of the SDE solution (exact or approximated). If $t_0, t_1, ..., t_n$ are the measurement-times, then a possible choice is `owntime=[t0:h:tn]` where `t0` is t_0, `tn` is t_n and `h` is the *stepsize* for the chosen SDE integration method. Of course `h` should be much smaller than $\min_i |t_{i+1} - t_i|$.

`time` the sequence of sampling times, with or without repetitions. If `vrbl` (see below) is a sequence of `1`'s then the sampling times necessarily correspond to the only observed state variable, thus `time` is the sequence $\{t_0, t_1, ..., t_n\}$. If `vrbl` contains entries other than `1`'s, for example if `vrbl = [1 2 1 2 1 2 1 2]` then we are observing alternatively two different variables and therefore `time` contains the corresponding sampling times, say `time=[0 0 1.5 1.8 2 2 3 3.5]`, the first element (0) being the first sampling time for variable 1,

the second one (0) the first sampling time for variable 2, the third one (1.5) the second sampling time for variable 1 etc. Notice that different variables do not necessarily have to be observed at the same sequence of sampling times.

`numdepvars` the dimension of the SDE, i.e. `numdepvars` must equal d, see eq. (1).

`vrbl` an array containing the (time-dependent) sequence of integers indexing the *observed* variables $\{y_i\}_{i=0,\ldots,n}$; it has same length as `time`. For a 1-dimensional system (`numdepvars = 1`) we have `vrbl=[1 1 1 ... 1]`. For larger systems several patterns can be specified. Some examples follow (this is a non-exhaustive list):

- `numdepvars=2`, `time=[0, 0, .5, .5, 1, 1, 1.5, 1.5]`, `vrbl=[1, 2, 1, 2, 1, 2, 1, 2]` both variables of a 2-dimensional SDE have been observed at exactly the same sequence of sampling times (fully observed system).
- `numdepvars=2`, `time=[0, 0.5, 1, 1.5]`, `vrbl=[1, 1, 1, 1]` only the first variable of a 2-dimensional SDE has been measured and its sampling times are all those contained into `time` (partially observed system).
- `numdepvars=2`, `time=[0, .5, 1, 1.5]`, `vrbl=[2, 2, 2, 2]` only the second variable of a 2-dimensional SDE has been measured and its sampling times are all those contained into `time` (partially observed system).
- `numdepvars=3`, `time=[0, .5, .5, 1, 1.5, 1.5]`, `vrbl=[1, 1, 3, 3, 1, 3]`, so we have three state variables but only the first and third variable have been observed (partially observed system). In this case we have observed the first variable at time 0 (but not the second nor third variable), then we have measured both the first and third variable at time 0.5, then the third variable has been measured at time 1, then both first and third variable have been measured at time 1.5.

`yobs` an array having the same length as `time`. It contains the (time-ordered) sequence of observed data $\{y_i\}_{i=0,\ldots,n}$; it must be consistent with the indices contained in `vrbl`. For example if `vrbl=[1 2 1 2]` and `yobs=[1.1 2.3 1.7 2.5]` then 1.1 is the first observed value of the first variable, 2.3 is the first observed value of the second variable, 1.7 is the second observed value of the first variable etc.[1]

`integrator` a string declaring whether numerical integration of the SDE should be carried using the Euler-Maruyama approximation (`integrator='em'`) or exactly (`integrator='exact'`). In the former case it is necessary to specify an sde-file (e.g. `lv_sdefile.m`), in the latter case the exact solution must be coded in a exact-solution-file (e.g. `theophylline_exactsol.m`).

[1] Technically `yobs` should not be listed in a section reserved to SDEs-related commands, as `yobs` is "raw data" and not necessarily the output of an SDE simulation (and even if it has been produced from an SDE model, it might incorporate "measurement error"). However there does not seem to exist a more suitable section where to introduce `yobs`.

Utilities for free/constant parameters

`bigtheta_start` an array containing starting values for `bigtheta`[2]. Some of those values may be kept constant during the MCMC phase depending on the corresponding values in `parmask`, see below. Notice starting values for the bandwidth/tolerance δ *should not* be specified in `bigtheta_start`, see instead the ABC-MCMC-related parameters below and section 3.4.

`parmask` an array having the same length as `bigtheta_start`. It specifies which elements in `bigtheta_start` (or more generally in `bigtheta`) have to be considered as constant and which ones free-to-vary (i.e. to be estimated); a `0` is used to denote the constant ones (will not vary during the ABC-MCMC procedure), whereas for those considered as unknown and to be estimated a `1` is reserved.

`parbase` an array having the same length as `bigtheta_start`, it contains "baseline" values for all parameters, therefore typically `parbase=bigtheta_start` (although this is not a rule). During the MCMC procedure `abc-sde` automatically separate free-to-vary (unknown) parameter from those that are kept constant (thanks to the `parmask` vector). However when `abc-sde` requires to re-construct the updated `bigtheta` array, containing the working values for free parameters (as produced by the MCMC proposal mechanism) in addition to the constant ones then, for the latter, values specified in `parbase` are plugged in.

`theta` an array containing the current values of the unknown parameters. As such it has length equal to the number of non-zero entries in `parmask`. It is specified by the user into "prior" files (e.g. `theophylline_prior.m`) as only unknown parameters require a prior density. Of course `theta` is a subset of `bigtheta`.

ABC-MCMC setup

The main function for ABC inference via MCMC is `abc_mcmc.m` which returns a matrix `MCMCstates` having as many columns as the length of `theta` plus one (the additional one being the chain for the ABC threshold/bandwidth δ) and a varying number of rows, depending on the settings below.

`R_mcmc` the number of MCMC draws to be simulated. Not necessarily all `R_mcmc` draws are returned by `abc_mcmc.m` or stored in the MATLAB workspace, depending on the `thin` value, see below.

`thin` the "thinning" factor, i.e. a non-negative integer j specifying whether the simulated chain of length `R_mcmc` should be returned in its entirety or instead each jth simulated draw should be returned. `thin` must be either `0` (no thinning, the full sequence of `R_mcmc` draws is returned) or an integer strictly larger than 1. If a value `thin` > 1 is specified then the matrix `MCMCstates` has `floor(R_mcmc/thin)` rows. If `thin=0` then `MCMCstates` has `R_mcmc+1` rows (including the starting values for free-parameters).

[2]There is space here for a little confusion: the reader might argue why there is any need to specify starting values for some quantities which are kept fixed (not estimated) during the MCMC phase. For fixed parameters `bigtheta_start` provides values for such constants, see also `parbase`.

`step_rw` the array of initial standard deviations for the Metropolis random walk updates; it has length equal to the number of free parameters. Each entry in `step_rw` is used as the standard deviation for the Gaussian proposal for the corresponding parameter during the initial MCMC steps[3]. Afterwards, adaptive Metropolis random walk [14] kicks-in.

`mean_bandw` the mean value for the exponential distribution of the "bandwidth" δ (where δ is the tolerance value for ABC).

`bandw_start` a starting value for the bandwidth δ, this is because in our approach a chain for δ is constructed.

`maxbandwidth` an upper bound for the bandwidth δ. If no upper bound is desired, set `maxbandwidth=inf`.

`numsimpar` the number of simulated pairs of parameters/synthetic-data over which summary statistics are obtained by regression.

`trainitermax` the number of repetitions of `numsimpar` simulations during the pilot procedure.

`var_training` an array having length equal to the number of free parameters. It is returned by `abc_training.m` and its purpose is to assign appropriate weights to each free parameter[4].

`regrtype` a string specifying the type of regression used to obtain summary statistics. It can be `regrtype='mars'` or `regrtype='lasso'`. If `'mars'` is specified then multivariate regression splines [4] is employed as provided in `ARESLab` [3]. If `'lasso'` is specified then a lasso-type regularisation is employed [2], as provided in `glmnet` [1].

`maxbasis` the maximum number of basis functions, to be specified only when `regrtype='mars'` is set.

[3] In the current implementation, when the number of free parameters is less than 15 fixed-step Metropolis random walk using `step_rw` is run for the initial 300 ABC-MCMC draws before adaptive Metropolis random walk start. Otherwise for a larger number of free parameters it is run for the initial 1000 ABC-MCMC draws

[4] `var_training` is the inverse of the diagonal of matrix A in [11, 8]. `var_training` can be set to be a vector of `1`'s if unknown parameters are supposed to have comparable sizes.

Part II

Model–Setup and Examples

This part of the guide describes how to set the mathematical and probabilistic features of the model under investigation; then two simulation studies are presented. The first example considers a stochastic extension of a one-dimensional model often studied in pharmacokinetic/pharmacodynamic (PK/PD) research. The second one is a two-dimensional stochastic predator–prey model. In both cases synthetic data are considered.

Notice that approximate Bayesian computation (ABC) is not strictly necessary for conducting inference in the presented examples. Exact Bayesian inference can be carried-out for both applications, for example using particle MCMC (pMCMC) [15] using the "adaptation" proposed in [16]. ABC as a body of methods finds its *raison d'être* in the analysis of more challenging models. However the application of exact Bayesian methodology such as pMCMC in some cases is computationally very demanding and therefore even for relatively simple models, such as those considered here, ABC can still be useful (also, our "early–rejection" algorithm provide an additional boost[5] of 40-50% in terms of computational time compared to a *naïve* Metropolis random walk version).

Although in the following sections guidance for coding practice is given, the reader is encouraged to read through the "run" scripts (`theophylline_run.m` and `lv_run.m`) for the two examples in order to understand e.g. how synthetic data are produced and how the most common tasks can be accomplished.

Also, remember the limitations of the current release of the `abc-sde` toolbox, as from Chapter 1.

[5]However when specifying uniform priors for the ABC-MCMC procedure, i.e. while running `abc_mcmc`, there might be no significant acceleration, see section 4 in [8].

Chapter 3

General settings and a one-dimensional example

3.1 Setting up the mathematical and probabilistic model

Here we describe how to define a model in separate files, each describing different features – mathematical or probabilistic – of the model under study. The easiest approach is to consider the files corresponding to implemented examples and use those as templates for custom models. In this chapter we illustrate how to implement a one-dimensional model. See Chapter 4 for a two-dimensional fully ad partially observed model.

Files pertaining the one-dimensional example are stored in the `\...\abc-sde\theophylline` folder.

3.1.1 SDE model

Here we specify the functional shape of the dynamical model (1). Probabilistic features are instead detailed in the `prior` file. Notice sections below on one/multidimensional SDEs illustrate the case where we need to set files for an SDE model having unknown closed-form solution (which is typically the case): when an exact solution is available, the creation of such files is not necessary, go instead to section 3.2.2.

One-Dimensional SDEs

Copy the file `theophylline_sdefile.m` in your working directory and rename it into e.g. `mySDE_sdefile.m`. Modify the code according to the following instructions (it is possible to change the names of variables and parameters):

1. change the first line of `mySDE_sdefile.m` into

```
function [out1,out2] = mySDE_sdefile(t,x,flag,bigtheta)
```

2. store all the parameters necessary to define the SDE (including its initial state) into the `bigtheta` array, e.g.

```
X0  = bigtheta(1); % this is the mySDE initial condition X_0
p1  = bigtheta(2); % this is a model parameter for mySDE
p2  = bigtheta(3); % this is another model parameter for mySDE
...
```

It is up to the user to take care of the correct scale used to represent the parameters stored in `bigtheta`. For example it is sometimes the case that their natural logarithms are considered (e.g. in the "theophylline" example we have `log_Ke` in place of `Ke`), as the MCMC proposals are based on Gaussian random walks and therefore a logarithmic transformation is suitable for positive parameters. See next section on multidimensional SDEs for a more explicit example.

3. define the `driftX` and the `diffusionX` part of the SDE into the appropriate slots (see the Introduction for definitions of *drift* and *diffusion*);

4. check that the following two lines are present (it should be the case if the provided sde-files are used as template)

```
out1 = driftX;
out2 = diffusionX;
```

5. insert the SDE initial condition: go to the bottom of the file and change the line
   `out2 = xzero;  % write here the SDE initial value(s)`
   into
   `out2 = X0;  % write here the mySDE initial value`

Therefore, depending on the current time-value, `mySDE_sdefile` returns the current drift and diffusion values (which will be automatically plugged in the Euler-Maruyama integration), or the SDE initial value.

Multi-Dimensional SDEs

Here we assume familiarity with the previous section on one-dimensional SDEs. Suppose we wish to consider the following model:

$$dX_t = \beta(\alpha - X_t)dt + g \cdot dW_t, \qquad X_0 = x_0$$

where $X_t = (X_t^1, X_t^2)^T$, $\alpha = (\alpha_1, \alpha_2)^T$, $W_t = (W_t^1, W_t^2)^T$, $x_0 = (x_0^1, x_0^2)^T$

$$\beta = \begin{pmatrix} \beta_{11} & \beta_{12} \\ \beta_{21} & \beta_{22} \end{pmatrix} \qquad g = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \end{pmatrix}$$

and T denotes transposition. Thus X_t is 2-dimensional, and this means that we have somewhere specified `numdepvars=2`, e.g. before constructing the vector of settings for the SDE model `sde_param` (see `lv_run.m` for an illustration). Suppose all values in x_0, α, β and g are positive.

1. create the file `mySDE_sdefile.m` as previously described and make the modifications listed below.

2. define the parameters (remember for this example we have assumed positive parameters, therefore a log-transformation is recommended, see the remark in the one-dimensional SDEs section):

```
log_x01     = bigtheta(1); % the log of mySDE's initial condition X_01
log_x02     = bigtheta(2); % the log of mySDE's initial condition X_02
log_alpha1  = bigtheta(3);
log_alpha2  = bigtheta(4);
...
```

3. In some cases we might want to separate the different components of the current state `x` into

```
x1 = x(1);
x2 = x(2);
```

   however *this is not always necessary.*

4. define the `driftX` and `diffusionX` components of the SDE, which are matrices. For example here we have

```
beta   = [exp(log_beta1) exp(log_beta2); exp(log_beta3) exp(log_beta4)];
alpha  = [exp(log_alpha1) ; exp(log_alpha2)];
driftX = beta*(alpha-[x1;x2]);
diffusionX = [exp(log_sigma1) 0; 0 exp(log_sigma2)];
```

   then copy such quantities into `out1` and `out2`

```
  out1 = driftX;
  out2 = diffusionX;
```

5. at the bottom of `mySDE_sdefile.m` insert the SDE initial conditions into a *row vector*, e.g. for a two-dimensional SDE with initial conditions `x01` and `x02` write:

```
  out2 = [exp(log_x01) exp(log_x02)];   % the initial states x01 and x02
```

Of course when the initial states (x_0^1, x_0^2) – or any other known parameter – are considered as positive known constants (i.e. do not need to be estimated) then, provided the corresponding entries in `parmask` have been correctly set to `0`, we can safely avoid transforming those to

logarithmic scale as there will be no MCMC chain created for such parameters. Therefore in this situation it is completely safe to specify above

```
x01     = bigtheta(1);
x02     = bigtheta(2);
```

and

```
  out2 = [x01 x02];
```

Similar considerations apply to any parameter considered as fixed.

3.1.2 `Prior` files

Prior densities and sampling (simulation) from such densities are specified in "prior files", e.g. `lv_prior.m`. A prior file contains the setup for prior densities and pseudo-random numbers generation, Prior densities are used during the ABC-MCMC phase, whereas pseudo-random numbers from probability distributions are necessary for the ABC "pilot" and "training" phases [11, 8].

If the pilot (which is an optional task) is performed then the corresponding prior densities have to be specified at the end of the file, in the section clearly labelled as "pilot", e.g. in `lv_prior.m` we have

```
 log_c1 = unifrnd(-3,2);
 log_c2 = unifrnd(-7,0);
...
```

Instead the remaining sections of the file are reserved for (i) prior densities for MCMC and (ii) simulation from the priors specified in (i). Again by looking for reference at `lv_prior.m`, phase (i) is labelled as "MCMC phase" and we have

```
    log_c1_prior = normpdf(theta(1),-1.55,0.4);
    log_c2_prior = normpdf(theta(2),-6,0.3);
    log_c3_prior = normpdf(theta(3),-1.45,0.5);
```

Therefore, for the MCMC phase we have specified Gaussian priors for each parameter to be estimated[1] (see Chapter 4 for a rationale). Then we may assume independence between parameters and therefore the joint prior is the product of marginal priors, hence we write

```
out = log_c1_prior*log_c2_prior*log_c3_prior;
```

Finally, corresponding to (ii) we have instructions specific to the "training phase"

```
    log_c1 = normrnd(-1.55,0.4);
    log_c2 = normrnd(-6,0.3);
```

[1] In fact, as previously remarked, parameters in the `theta` array are unknown parameters object of our inference, as opposed to `bigtheta`, the latter containing *all* structural model parameters.

```
log_c3 = normrnd(-1.45,0.5);

out = [log_c1,log_c2,log_c3];
```

Of course the objects specified for (i) and (ii) must be consistent one with the other (we do not provide a tool for automatic checking).

3.1.3 Error-model

This section closes the list of files necessary to set the mathematical/statistical model. In the "error-model file" we specify the probabilistic mechanism perturbing simulated values from the SDE model with some form of "noise" (e.g. measurement error), same as in (2). As an example assume that scalar data $\{y_i\}$, collected at $n+1$ sampling times $\{t_0, t_1, ..., t_n\}$, are modelled as

$$y_i = X_i + \varepsilon_i, \qquad \varepsilon_i \sim N(0, \sigma_\varepsilon^2), \quad i = 0, ..., n \tag{3.1}$$

that is data are the outcome of an additive relationship between a realization of the diffusion process modelled via an SDE and measurement error ε (here assumed Gaussian homoscedastic). However more general types of relations between $\{y_i\}$, $\{X_i\}$ and $\{\varepsilon_i\}$ can be postulated in a error-model file. Notice in particular we do not need to assume conditional independence of y_i's given X_i, see the Introduction.

Examples are `theophylline_errormodel.m` and `lv_errormodel.m`. The content of such type of file does not have a prescribed structure. It just needs to take as input the complete array of model parameters `bigtheta`, the realization `xhatpredict` (i.e. $\{X_i\}$) of the SDE solution extrapolated at sampling times $\{t_0, .., t_n\}$ (this is all done automatically by `abc-sde`), and other inputs which it's not necessary to describe here. Therefore we simply need to describe the functional relation between such quantities, for example

```
yobssim = xhatpredict + normrnd(0,sigmaepsilon*ones(length(xhatpredict),1));
```

The line above implements model (3.1) for scalar y_i's for each $i = 0, 1, ..., n$.

3.2 Optional files

For a given problem, the creation of files described in this section is optional. In section 3.2.1 we discuss how to consider "admissible" trajectories, while in section 3.2.2 we illustrate how to implement the exact solution for a given SDE model.

3.2.1 Admissible trajectories

A fundamental step in an approximate Bayesian computation (ABC) algorithm is to generate parameters from prior distributions, then conditionally on such draws observations from the postulated model are simulated and then summary statistics are obtained on such simulated data ([9], [10]). However it is sometimes the case that for a given simulation of unknown parameters from vague priors, numerical infelicities may occur in the corresponding generated trajectory from the SDE model: e.g. negative trajectories for states which should be non-negative (e.g. time-course of concentrations), inadmissible values (`NaN`, `inf`) or complex values. Of course in some cases such problems can be alleviated when informative priors are used, however most

often we do not have enough confidence in our prior knowledge and a qualitative study of the *a-priori* model behaviour might be impractical. However such issues are not of immediate concern for the MCMC phase of ABC, as the Metropolis-Hastings acceptance/rejection mechanism will automatically reject parameters producing the situations above. What is instead problematic is how to deal with the creation of summary statistics which takes place prior to MCMC start, as `abc-sde` consider regression-based methods where the simulated parameters are regressed against the corresponding trajectories, and of course computations cannot be carried out if inadmissible values have been generated. We can therefore specify an "admissible-trajectories-file": if such file is created (again, this is optional) `abc-sde` will keep simulating parameters *during the pilot and training phases* (not MCMC!) until some criteria are met. Such criteria are specified by the user. In the end the `abc_training` output will only contain those parameters corresponding to "admissible trajectories".

For example the file `theophylline_admisstraj.m` takes as argument the SDE trajectory generated by the current vector of parameters at a given iteration of the pilot procedure:

```
function ok = theophylline_admisstraj(xhat)

 ok = 1;

% we consider as "unwanted" trajectories all those having at least a NaN,
% inf, complex or negative value
 if  ~isreal(xhat) || any(isnan(xhat(:))) || any(isinf(xhat(:))) || any(xhat(:) < 0)
     ok = 0;
 end
```

The function returns a value `ok` which can be 1 or 0. If it's 0 then at least a non-real value in the trajectory has been detected, or a `NaN`, `inf` or negative value. Since the modelled data represent non-negative concentrations we only desire to work with parameters producing non-negative trajectories. Therefore in general the user can set its own specific criteria in such file: the only requirement is that the `ok` value should return a 1 (all went fine) or a zero (something got wrong).

Remember: such file is not invoked during the MCMC phase, therefore the posterior inference we obtain from running `abc_mcmc` is perfectly legitimate as "inadmissible" trajectories will be automatically rejected by the Metropolis-Hastings ratio, as motivated above. Therefore the MCMC phase will not be conditioned on what is specified in the admissible-trajectories-file. Of course summary statistics are instead computed conditional to restrictions placed in the admissible-trajectory-file.

3.2.2 Exact solution of SDEs

Whenever the explicit solution to an SDE is known this should be coded in an "exact-solution-file", see e.g. `theophylline_exactsol.m`. Provided such file has been created, exact integration can be carried out by writing

```
integrator = 'exact';
```

in your "run" script file, see e.g. `theophylline_run.m`. Otherwise using `integrator = 'em'` the Euler-Maruyama approximation is employed. Notice that whenever the exact integrator is used there is no need to create an sde-file (e.g. `theophylline_sdefile.m`), the latter being necessary when Euler-Maruyama is adopted. Now, since an SDE has rarely a solution available in closed-form, there is no recipe for writing an exact-solution-file. See `theophylline_exactsol.m` for an example of coding the (available) exact solution for the theophylline example. The function must return an array having length `length(owntime)` if the SDE is one-dimensional, or a matrix having size `length(owntime)` $\times$ `numdepvars`, where `numdepvars` is the dimension d of the SDE, see the notation in the Introduction.

3.3 Working with external data and data-format conventions

Should data be loaded from an external source, this is of course possible and here we provide some recommendation on how the data should be arranged. For examples see the ASCII tab-delimited `data_full.dat` and `data_partial.dat` files into `\...\Lotka-Volterra`; see also section 2 for variables definitions and section 4.5. If data are stored into a single file, then these *must* be arranged in three columns: one column must contain the sampling times `time` sorted in non-decreasing order (repeated values might be present, if more than one variable is being observed), another column is reserved for the corresponding measured values from the dynamical process being observed (`yobs`) and finally there should exist a column for the numerical labels identifying the state variables (`vrbl`). Arrays `time`, `yobs` and `vrbl` must have the same length.

3.4 Choosing the tolerance

The tolerance/bandwidth δ is a fundamental building-block for inference via ABC, see [8]. In our toolbox δ is considered as an unknown parameter, and as such it has its own prior distribution and a dedicated MCMC trajectory is built, following [17]. A starting value `bandw_start` for δ must be supplied to the `abc_mcmc` function then, considering that we always assume δ to have an exponential distribution with mean λ, we also need to specify a value for λ via `mean_bandw`; proposals for δ are then automatically generated by `abc-sde`[2]. Also, an upper bound for δ must be set via `maxbandwidth`; this way no proposed δ will exceed such bound (it is also possible to set `maxbandwidth=inf` if no explicit bound is required).

Ideally `mean_bandw` should be such that small values for δ are proposed, while at the same time keeping the acceptance rate at reasonable levels. In MCMC for ABC we are not aware of optimality results regarding suitable acceptance rates, however these should be not "too large" otherwise the prior distribution is targeted instead of the posterior! Also notice the warning box below.

In some case setting a low value for `maxbandwidth` might produce a long series of rejected proposals. As such the chain might get stuck for long iterations in the same point. Although to some extent this is an expected behaviour in ABC, our implementation of the adaptive MCMC scheme proposed in [14] might "fail", in the sense that the dynamically updated covariance matrix

[2]More in detail: a generic proposal is generated for $\log \delta$ using Metropolis random walk with $N(0, \lambda^2)$ increments, then the generated value is reverted to the δ scale.

for generating the increments in the proposed values, after many rejections would contain very small values. Therefore the chain keeps being stuck in a pathological way. If this happens (it is detectable by noticing decreasing acceptance rates printed on the computer screen during the MCMC simulation) `maxbandwidth` and/or `mean_bandw` should be enlarged.

3.5 Example: theophylline drug pharmacokinetic

This first example considers the theophylline drug pharmacokinetic, which has often been studied in literature devoted to longitudinal data with random parameters (mixed–effects models), see [18, 19], [8] and references therein. Files implementing this example are stored in the `\...\abc-sde\theophylline` folder.

By denoting with X_t the level of drug concentration at time t in a subject, after receiving orally a *Dose* of drug, we modellize the drug kinetic via the following SDE:

$$dX_t = \left(\frac{Dose \cdot K_a \cdot K_e}{Cl} e^{-K_a t} - K_e X_t \right) dt + \sigma dW_t. \tag{3.2}$$

This example is fully discussed in [8] and the interested reader is redirected there for further details. Synthetic data are generated following the setup described hereafter. We consider nine sampling times t_i's at 15 min, 30 min, 1, 2, 3.5, 5, 7, 9 and 12 hrs after dosing, same as in [18, 19]. Since an exact solution to (3.2) can be obtained (see [20]), a single trajectory is simulated on a time grid (`owntime`) built by placing 50 artificial linearly spaced points between and including t_i and t_{i+1}. Finally, the generated values are perturbed with measurement error according to the additive error-model $y_i = X_i + \varepsilon_i$ with $\varepsilon_i \sim N(0, \sigma_\varepsilon^2)$ i.i.d., $i = 1, ..., 9$. For data generation we used

$$(\log K_e, \log K_a, \log Cl, \log \sigma, \log \sigma_\varepsilon) = (-2.52, 0.40, -3.22, \log \sqrt{0.2}, \log \sqrt{0.1}).$$

At time $t_0 = 0$ the drug is administered and therefore at t_0 the drug concentration in blood is zero, i.e. $X_0 = x_0 = 0$; thus it's reasonable to assume the SDE initial state to be known and as such will not be estimated. Parameters of interest are $(K_e, K_a, Cl, \sigma, \sigma_\varepsilon)$ however in practice to prevent the parameters from taking unrealistic negative values their natural logarithm is considered, thus we set `theta` $= (\log K_e, \log K_a, \log Cl, \log \sigma, \log \sigma_\varepsilon)$. *Dose* is assumed known and equal to 4 [mg/KgBW]. In one case σ_ε will be considered known and in such case it will be `theta` $= (\log K_e, \log K_a, \log Cl, \log \sigma)$.

3.6 Results

ABC inference for model (3.2) with noisy measurements $y_i = X_i + \varepsilon_i$ has been performed by using nine data points as obtained from `theophylline_run`. Data have been generated by setting $(\log K_e, \log K_a, \log Cl, \log \sigma, \log \sigma_\varepsilon) = (-2.52, 0.40, -3.22, \log \sqrt{0.2}, \log \sqrt{0.1})$.

3.6.1 Identification of a "training" region (optional)

We first consider performing inference for the case where σ_ε is known and equal to $\sqrt{0.1}$. Notice that in order to exclude some parameters from the estimation process we can simply put a 0 in the corresponding `parmask` entry, that is in this case we write

```
parmask  = [0,   1,   1,   1,   1,   0];
```

corresponding to parameters listed in the following order: X_0, $\log K_e$, $\log K_a$, $\log Cl$, $\log \sigma$, $\log \sigma_\varepsilon$, and where `parmask` contains flags for unknown parameters to be estimated (`1`) and for parameters assumed to be known and fixed (`0`).

ABC summary statistics are determined by simulating a large number of parameters and synthetic data from some prior distribution. It is therefore important to first identify an appropriate support over which to place our prior. Such identification procedure is named "pilot", but it is not strictly necessary if a reasonable region is already known (e.g. from previous experiments).

Notice that the pilot procedure is potentially time consuming, depending if the SDE model has a closed form solution or requires numerical integration and whether the number of observations is large. Timing is particularly affected by settings in the "`admisstraj`" file (`theophylline_admisstraj.m` in this example). However in this specific case it is not an issue.

We initially specify a diffuse but proper prior $\pi_0(\texttt{theta})$ and by using a pilot we aim at identify subsets of the real line on which to place most of the mass for each individual prior $\pi(\texttt{theta}_j)$ that will actually be used during the MCMC phase. Notice the different notation employed for different priors: $\pi_0(\cdot)$ corresponds to the prior used in the pilot, while $\pi(\cdot)$ is used in the MCMC phase.

We choose uniform initial priors $\pi_0(\theta)$: $\log K_e \sim U(-4.5, -1.2)$, $\log K_a \sim U(-1.2, 1.4)$, $\log Cl \sim U(-5.3, -1.6)$, $\log \sigma \sim U(-2.3, 0.4)$. Since the pilot procedure aims at drawing samples from $\pi_0(\theta)$, we set the appropriate commands into `theophylline_prior.m` (bottom lines, in the section named "pilot stage") as

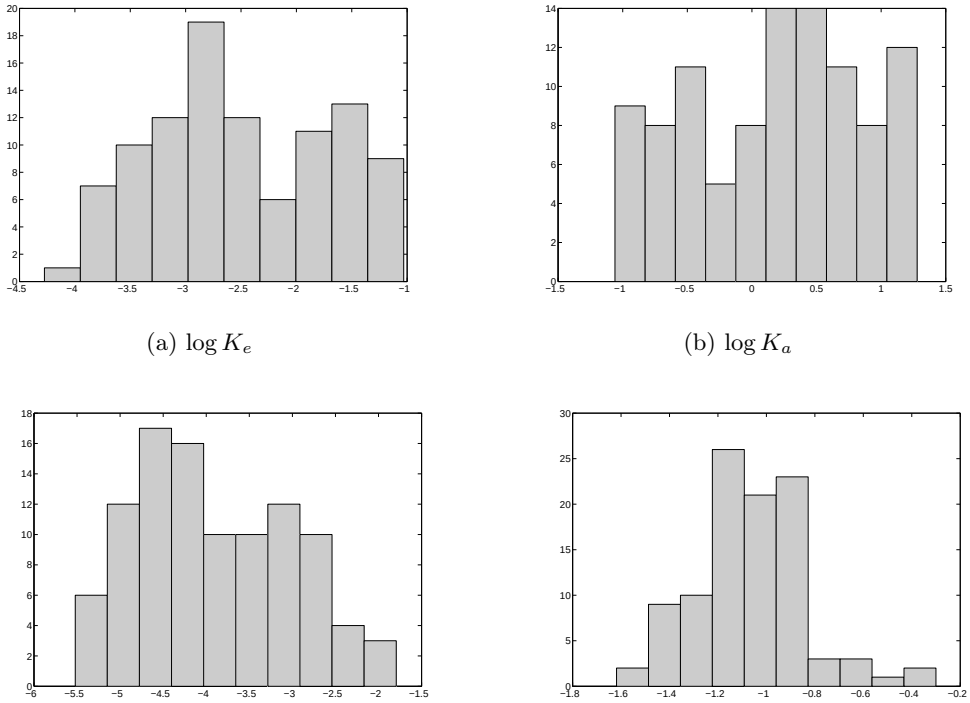
```
log_Ke = unifrnd(-4.5,-1.2);
log_Ka = unifrnd(-1.2,1.4);
log_Cl = unifrnd(-5.3,-1.6);
log_sigma = unifrnd(-2.3,0.4);
```

The pilot is started by invoking `abc_training.m` and here we use multivariate adaptive regression splines for the determination of summary statistics, `regrtype='mars'` (see `theophylline_run.m` for the detailed setup)

```
[training,var_training] =  abc_training(trainitermax,numsimpar,sde_param,...
            parmask,parbase,regrtype);
```

This returns a matrix `training` whose columns should be explored using e.g. graphical tools like histograms (or `ksdensity()` or `normfit()` etc., if the STATISTICS TOOLBOX is available). If

we specify `trainiter=100` we get a 100 x 4 matrix whose columns have the sample distributions shown in Figure 3.1. Using such information we decide to impose the following priors $\pi(\texttt{theta})$ which will be used for ABC-MCMC: $\log K_e \sim N(-2.5, 0.8)$, $\log K_a \sim N(0.18, 0.7^2)$, $\log Cl \sim N(-4, 0.9)$, $\log \sigma \sim N(-1, 0.23^2)$. Of course the histograms in Figure 3.1 are hardly Gaussian, however a pilot simply indicate the location over which to place most of the probability mass of the prior distribution we simulate synthetic-data/parameters from during summary statistics construction. The second output from `abc_training.m` is the array `var_training` which we can plug directly into the main ABC function `abc_mcmc.m` described in next section[3].



(a) $\log K_e$ (b) $\log K_a$

(c) $\log Cl$ (d) $\log \sigma$

Figure 3.1: Theophylline model with know measurement error: sampling distributions of $\log K_e$, $\log K_a$, $\log Cl$ and $\log \sigma$ from the pilot procedure.

It is important to recognize that a pilot as implemented in `abc_training` (and as discussed in [8]) can only provide some indication of where the *expected value* for the approximated posterior is likely to be, not the support for the approximated posterior distribution itself. As such the regions we deduce from the pilot have to be considered "with a grain of salt". In [11] it is suggested to consider as a pilot a first round of ABC-MCMC: of course this approach would be more reliable, but for the considered class of models it is difficult to imagine how to obtain the summary statistics necessary to run such a preliminary ABC analysis.

[3] `var_training` is used to build the matrix of weights denoted with A in [8]. See also its definition in section 2.

During the pilot the working `training` matrix is automatically and periodically saved (at each completed `trainitermax` iteration) with `.mat` format into the working directory, with names such as `training_lasso_temp` or `training_mars_temp` depending on the setting provided for the `regrtype` parameter. At the end of all the `trainitermax` simulations, the resulting `training` matrix is saved with names such as `training_lasso_final` or `training_mars_final`; `var_training` is also saved as `var_training_lasso` or `var_training_mars`. In this version of `abc-sde` we are not providing a mechanism to prevent such files to be created: the user might want to comment-out the appropriate lines towards the end of the `abc_training.m` file.

3.6.2 ABC–MCMC

We use the means from the Gaussian priors determined in section 3.6.1 as starting values for $(\log K_e, \log K_a, \log Cl, \log \sigma)$ for the MCMC phase. We store such values into `bigtheta_start`, together with constant parameters (again in the following order: X_0, $\log K_e$, $\log K_a$, $\log Cl$, $\log \sigma$, $\log \sigma_\varepsilon$):

```
bigtheta_start = [0,  −2.5,   0.2,    −4,    −1,    −1.15];
parmask        = [0,     1,     1,     1,     1,        0 ];
parbase = bigtheta_start;
sde_param{1} = bigtheta_start;
```

For parameters assumed to be known their starting values are used throughout the simulations, therefore we deduce that in this initial example X_0 is not estimated and is set equal to 0; also $\log \sigma_\varepsilon$ is not estimated and is set equal to -1.15 (which is also its true value).

Before running the MCMC we first need to modify the `theophylline_prior` function in order to plug the priors $\pi($`theta`$)$. The appropriate modifications should be performed in two sections: (i) specify prior densities used in MCMC and (ii) drawing/sampling from such densities. In `theophylline_prior` case (i) is denoted with "MCMC phase" while case (ii) with "Training phase".

```
  [...]

%::: MCMC PHASE: SPECIFY HERE PRIOR DENSITIES
log_Ke_prior    = normpdf(theta(1),−2.5,0.8);
log_Ka_prior    = normpdf(theta(2),0.18,0.7);
log_Cl_prior    = normpdf(theta(3),−4,0.9);
log_sigma_prior = normpdf(theta(4),−1,0.23);

out = log_Ke_prior*log_Ka_prior*log_Cl_prior*log_sigma_prior;

  [...]

%::: TRAINING PHASE: SPECIFY HERE SAMPLING FROM PRIORS USED IN MCMC

log_Ke    = normrnd(−2.5,0.8);
log_Ka    = normrnd(0.18,0.7);
log_Cl    = normrnd(−4,0.9);
```

```
log_sigma = normrnd(-1,0.23);

out = [log_Ke,log_Ka,log_Cl,log_sigma];
```

We set a maximum value for the bandwidth `maxbandwidth = 1.2`, which has been set by trial-and-error. Also we set `mean_bandw = 0.6` and `bandw_start = 0.6`.

```
integrator = 'exact'; % can be "EM" (Euler-Maruyama) or "exact"
R_mcmc = 2000000; % number of MCMC draws
step_rw = [0.55,0.2,0.7,0.22]; % starting SD for Metropolis random walk
mean_bandw = 0.6; % bandwidth mean value
bandw_start = 0.6; % starting bandwidth value
maxbandwidth = 1.2; % upper bound for the bandwidth
numsimpar = 30*length(sampletime);  % number of simulated datasets
                                    % for computing summary statistics
thin = 50;  % the "thinning" value
maxbasis = 50;  % max number of basis for MARS
regrtype = 'mars';  % can be "lasso" or "mars"
```

We run `abc_mcmc.m` for `R_mcmc = 2000000` with the settings above (however see `theophylline_run.m` for full details), including `var_training=[0.8446,0.5359,1.0270,0.5762]` obtained in section 3.6.1.

```
MCMCstates = abc_mcmc(yobs,sde_param,R_mcmc,step_rw,mean_bandw,bandw_start,...
             var_training,numsimpar,maxbandwidth,thin,parmask,parbase,...
             regrtype,maxbasis);
```

We obtained an average acceptance rate of about 11% during the simulation, which was completed in about 75 minutes[4]. The resulting matrix `MCMCstates` contains in its first 4 columns the (thinned) ABC-MCMC draws for $(\log K_e, \log K_a, \log Cl, \log \sigma)$ and in the last (fifth) column the chain for the bandwidth δ.

Depending on user's requirements, obtained draws could then be exported and treated using advanced diagnostic tools[5] which are not available in `abc-sde`.

During ABC-MCMC the matrix containing the created chains is automatically and periodically saved (every 1% of the total run) with `.mat` format into the working directory, with names such as `mcmcstates_lasso_temp` or `mcmcstates_mars_temp` depending on the setting provided for the `regrtype` parameter. This way a relatively recent version of the produced chain is available for safety reasons or to allow the exploration of partial results by opening another instance of MATLAB or another software. At the end of all the `R_mcmc` simulations, the complete chain

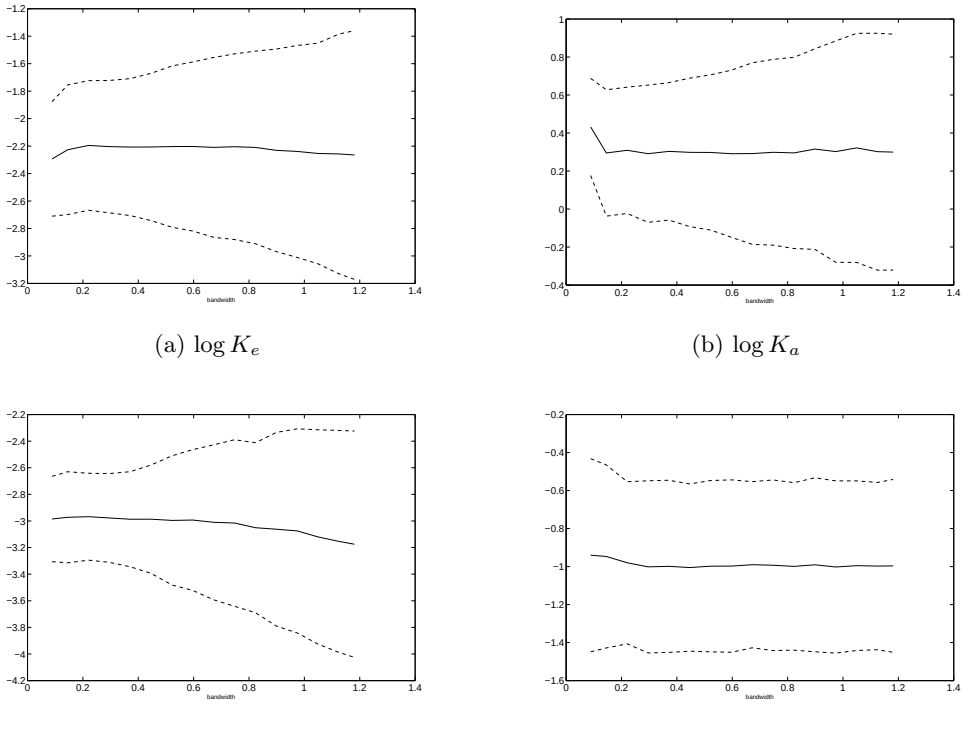
[4] Using an Intel Core i7-2600 CPU 3.40 GHz.

[5] Some software for MCMC diagnostics:

- for Matlab:
  - `http://becs.aalto.fi/en/research/bayes/mcmcdiag/`
  - `http://helios.fmi.fi/~lainema/mcmc/`
- for R: `http://cran.r-project.org/web/packages/coda/`

is saved with names such as `mcmcstates_lasso_final` or `mcmcstates_mars_final`. In this version of `abc-sde` we are not providing a mechanism to prevent such files to be created: the user might want to comment-out the appropriate lines towards the end of the `abc_mcmc.m` file.

We remove the initial 2,000 draws (burn-in), i.e. the first 2,000 rows in `MCMCstates`, corresponding to the first 100,000 iterations (because we used a thinning of size 50), see the code below. Then we examine the plots automatically produced via `abc_posthoc()` and reported in Figure 3.2: each instance of `abc_posthoc()` creates a plot by taking as first input the matrix of draws we want to inspect, as second argument the index of the column of such matrix corresponding to a given unknown parameter, as third argument the index of the column corresponding to the chain for δ, and finally an optional integer (default is 10) specifying the number of "bins" required to subdivide the values of δ into separate classes, same as when constructing an histogram. The resulting plots represent posterior means (solid lines) for varying δ's: we



(a) $\log K_e$ (b) $\log K_a$

(c) $\log Cl$ (d) $\log \sigma$

Figure 3.2: Theophylline model: posterior means for varying bandwidth δ.

notice in some cases the posterior mean changes for varying δ, notably for $\log K_e$ and $\log Cl$, and we want to select draws corresponding to small δ's, such that the posterior means result "stable" for the chosen δ. Here we decide to select draws corresponding to $\delta \leq 0.3$ and exclude the others from our inferential results.

```
MCMCredux = MCMCstates(2000:end,:);  % remove the burn-in
% check posterior means vs bandwidth
```

	K_e	K_a	C_l	σ	σ_ε
True value	0.08	1.49	0.04	0.45	0.32
ABC with Mars (*)	0.110 [0.068,0.168]	1.357 [0.965,1.913]	0.051 [0.036,0.070]	0.377 [0.244,0.584]	–
ABC with Lasso (*)	0.108 [0.068,0.168]	1.068 [0.357,2.643]	0.046 [0.030,0.071]	0.363 [0.240,0.572]	–
ABC with Mars	0.093 [0.053,0.141]	1.121 [0.772,0.160]	0.039 [0.026,0.063]	0.371 [0.209,0.648]	0.363 [0.249,0.536]
ABC with Lasso	0.101 [0.068,0.141]	1.053 [0.627,1.469]	0.046 [0.037,0.058]	0.326 [0.226,0.478]	0.272 [0.244,0.306]

Table 3.1: Theophylline example: posterior means and 95% credible intervals. Asterisks denote results obtained when σ_ε has been kept constantly equal to its true value (and thus it has not been estimated).

```
abc_posthoc(MCMCredux,1,5,15);
abc_posthoc(MCMCredux,2,5,15);
abc_posthoc(MCMCredux,3,5,15);
abc_posthoc(MCMCredux,4,5,15);
% remove draws having bandwidth > 0.3, then remove the last column (bandwidth)
MCMCfinal = MCMCredux(MCMCredux(:,end)<0.3,1:4);
% check autocorrelations
acf(MCMCfinal(:,1),40);
acf(MCMCfinal(:,2),40);
acf(MCMCfinal(:,3),40);
acf(MCMCfinal(:,4),40);
```

For the selected $\sim 1,950$ draws we check their autocorrelation using `acf()`: plots show rapidly decaying autocorrelation functions (not reported here) and therefore these have all been used for our inference. Finally we exponentiate means and percentiles in order to produce results for the original untransformed parameters, reported as "ABC with MARS (*)" in Table 3.1.

```
% Obtain inferential results on non-log scale
>> exp(mean(MCMCfinal(:,1:4)))

ans =

    0.1097    1.3572    0.0511    0.3772
>> exp(perctile(MCMCfinal(:,1:4),2.5))

ans =

    0.0677    0.9652    0.0358    0.2443

>> exp(perctile(MCMCfinal(:,1:4),97.5))

ans =

    0.1680    1.9127    0.0697    0.5837
```

We now use the same prior $\pi(\texttt{theta})$ to produce ABC inference for when Lasso is employed in summary statistics computations. We simply need to specify `regrtype = 'lasso'` and then run `abc_mcmc` same as before. However using the same settings as above the acceptance rate grows to 25% which is too high: we notice in this case we are able to reduce drastically the settings pertaining δ without compromising the exploration of the approximated posterior (as warned in section 3.4). We thus set

```
mean_bandw   = 0.07;
```

```
bandw_start   = 0.07;
maxbandwidth = 0.15;
regrtype      = 'lasso';
```

while still obtaining an acceptance rate of 14%. Results are again in Table 3.1, obtained by selecting those draws having $\delta < 0.04$.

The case of σ_ε unknown

We now want to consider the case where σ_ε needs to be estimated. This is not only interesting from the inferential point of view, but it also gives a motivation to show how to modify our code to accommodate such case. We need to:

- modify `step_rw` to include the initial value for the standard deviation of Gaussian innovations for Metropolis random walk corresponding to σ_ε;
- write a `1` in the `parmask` entry for σ_ε;
- modify `var_training` by adding a corresponding entry for σ_ε (ideally such value should be obtained by running a new pilot).
- modify the function `theophylline_prior` accordingly, see below.

We set:

```
step_rw = [0.55,0.2,0.7,0.22,0.1];
parmask = [0, 1, 1, 1, 1, 1];
```

Then, since we want to re-run the pilot to obtain a training region for σ_ε we modify the appropriate section in `theophylline_prior` and add

```
log_sigmaepsilon = unifrnd(-2.3,-0.22).
out = [log_Ke,log_Ka,log_Cl,log_sigma,log_sigmaepsilon];
```

We can now run the pilot with the same settings as before using the `abc_training` function while considering MARS for producing summary statistics (`regrtype = 'mars'`). `abc_training` returns among other things a vector `var_training`, this contains the values $[0.8417, 0.5464, 1.0298, 0.5765, 0.3559]$ which we will feed later on to `abc_mcmc`. Also the matrix `training` is created. By graphical examination of each of its columns we deduce the following priors to be used in the ABC-MCMC procedure: $\log K_e \sim N(-2.7, 1)$, $\log K_a = N(0.3, 0.8^2)$, $\log Cl = N(-3.5, 1)$, $\log \sigma = N(-1, 0.3^2)$, $\log \sigma_\varepsilon = N(-1, 0.2^2)$. Therefore we now need to modify the `theophylline_prior` function in order to plug the new priors for ABC-MCMC, same as previously shown:

```
  [...]

  %::: MCMC PHASE: SPECIFY HERE PRIOR DENSITIES

     log_Ke_prior = normpdf(theta(1),-2.7,1);
     log_Ka_prior = normpdf(theta(2),0.3,0.8);
     log_Cl_prior = normpdf(theta(3),-3.5,1);
```

```
    log_sigma_prior = normpdf(theta(4),-1,0.3);
    log_sigmaepsilon_prior = normpdf(theta(5),-1,0.2);


    out = log_Ke_prior*log_Ka_prior*log_Cl_prior*log_sigma_prior*...
          log_sigmaepsilon_prior;

  [...]

  %::: TRAINING PHASE: SPECIFY HERE SAMPLING FROM PRIORS USED IN MCMC

    log_Ke = normrnd(-2.7,1);
    log_Ka = normrnd(0.3,0.8);
    log_Cl = normrnd(-3.5,1);
    log_sigma = normrnd(-1,0.3);
    log_sigmaepsilon = normrnd(-1,0.2);

    out = [log_Ke,log_Ka,log_Cl,log_sigma,log_sigmaepsilon];

  [...]
```

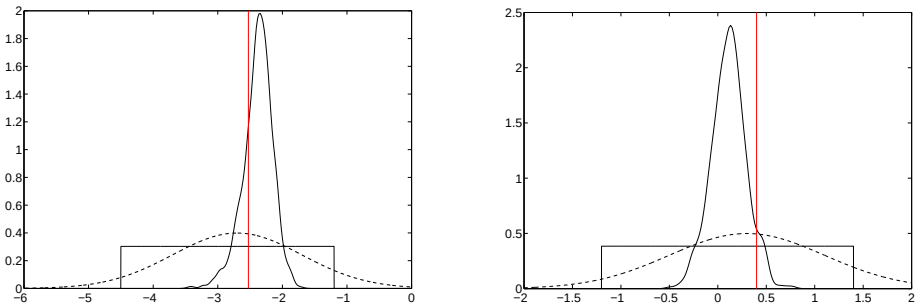
We plug the means of the Gaussian priors into `bigtheta_start` then we can finally run ABC-MCMC via `abc_mcmc` as usual.

```
mean_bandw   = 0.6;
bandw_start  = 0.6;
maxbandwidth = 1.2;
bigtheta_start = [0, -2.7,     0.3,  -3.5,   -1,    -1 ];
parmask        = [0 ,   1,       1,     1,    1,     1 ];
parbase = bigtheta_start;
sde_param{1} = bigtheta_start;

var_training = [0.8417    0.5464    1.0298    0.5765    0.3559];
```

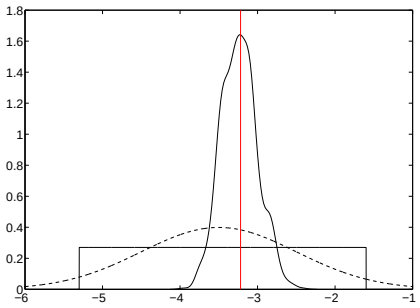
Results are in Table 3.1, produced from a set of about 2,070 draws corresponding to $\delta < 0.3$ (we discarded the same amount of draws from the "burn-in" as in previous analyses). For some parameters Figure 3.3 gives a comparison between their true values, their kernel smoothing estimates[6] of the ABC marginal posterior densities, their Gaussian priors $\pi(\texttt{theta})$ used during the MCMC and the uniform priors $\pi_0(\texttt{theta})$ used in the pilot.

Finally we repeated the whole procedure again (including a new pilot and thus deduced new priors for ABC-MCMC) for the case `regrtype = 'lasso'`; for brevity we only report inferential results, see Table 3.1.

[6]This functionality is not provided in `abc-sde`.



(a) $\log K_e$ (b) $\log K_a$



(c) $\log Cl$

Figure 3.3: Theophylline example when using MARS with σ_ε unknown: approximate posteriors (solid curves), Gaussian priors (dashed lines) and uniform priors used during the pilot. The vertical lines correspond to true parameter values.

Chapter 4

A two-dimensional example

Here we consider a two-dimensional example implementing a stochastic version of the Lotka-Volterra (LV) *predator-prey* model, often considered as a toy-model for chemical kinetics (see e.g. [16], [21]). Files pertaining this example are stored in the `\...\abc-sde\Lotka-Volterra` folder.

Because of its use in chemical kinetics we can consider prey ($\mathcal{X}_1$) and predators ($\mathcal{X}_2$) as two interacting chemical "species" of "reactants" subject to three "reactions" $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$ defined as:

$$\begin{aligned}
\mathcal{R}_1 &: \mathcal{X}_1 \xrightarrow{c_1} 2\mathcal{X}_1 \\
\mathcal{R}_2 &: \mathcal{X}_1 + \mathcal{X}_2 \xrightarrow{c_2} 2\mathcal{X}_2 \\
\mathcal{R}_3 &: \mathcal{X}_2 \xrightarrow{c_3} \emptyset.
\end{aligned}$$

Reaction $\mathcal{R}_1$ represents prey reproduction. $\mathcal{R}_2$ captures predator-prey interaction (consumption of prey by predator, in turn influencing predator reproduction rate), while $\mathcal{R}_3$ represents death of predators due to natural causes. Such reactions occur at rates specified by some unknown constant c_1, c_2 and c_3. For given values of the three constant rates and by specifying initial values $X_0 = (x_{01}, x_{02})$ for the amount of populations $\mathcal{X}_1$ and $\mathcal{X}_2$ respectively, the dynamics of the system is completely specified and realizations from the LV model can be simulated *exactly* via the "Gillespie algorithm" [22]. The "size of populations", i.e. the number of molecules, for the two chemical species at a generic time $t > 0$ will be denoted by $X_t = (X_{t,1}, X_{t,2})^T$ and of course $X_{t,j}$ takes non-negative integer values ($j = 1, 2$).

We can simulate a realisation of the stochastic LV model in MATLAB using for example `GillespieSSA` [5] (included within `abc-sde`), instructions are below, however since the task itself might be of no special interest to the reader of this guide, alternatively it is possible to just run the following:

```
load('lv_full_seed')
% restore the state for the pseudorandom numbers used to create the data-file
rng(lv_full_seed)
load('data_full.dat')
time = data_full(:,1);
yobs = data_full(:,2);
```

```
vrbl = data_full(:,3);
```

Such instructions load the standard data-arrays and restore the state producing the stream of pseudo-random numbers which have been used to simulate data as described in the section below (such section can be skipped).

Notice MATLAB's built-in `rng` function might be unavailable for versions of MATLAB older than 7.12: in such case an error will be issued. It is completely safely to comment such line, however it will not be possible to recover the same stream of pseudo-random numbers used to generate data in this example, and as such results might differ with those reported here.

4.1 Data generation (optional)

We first need to specify the matrices containing the coefficients of reactants (left side of the reactions above) and products (right side of the reactions).

```
% c_reac(j,i) = coefficient for the j-th chemical species at the
% reactant side of the i-th reaction
c_reac = [1     1     0     ;
          0     1     1    ];
% c_prod(j,i) = coefficient for the j-th chemical species at the
% product side of the i-th reaction
c_prod = [2     0     0     ;
          0     2     0     ];
```

Then we use the same setup as in [16] and set the initial states to be $X_0 = (x_{01}, x_{02}) = (100, 100)$ and constant rates $(c_1, c_2, c_3) = (0.5, 0.0025, 0.3)$. We want to simulate a realisation of the stochastic LV system from time $t = 0$ to time $t = 50$: it is sufficient to invoke the function `gillespiessa` and specify

```
X0 = [100;100];  % the initial states
const_rates = [0.5, 0.0025, 0.3]; % the constant rates
t_end = 50; % the end-time for the simulation
[nummol,time_reac] = gillespiessa(const_rates, c_reac, c_prod, X0, t_end);
plot(time_reac,nummol)
xlabel('Time')
```

The output returns the times corresponding to when reactions take place and the corresponding number of molecules for each species. The resulting plot is given in Figure 4.1.

We use the trajectories previously obtained to extract data at integer sampling times in $[0, 50]$. The resulting points (50 for $X_{t,1}$ and 50 for $X_{t,2}$) are then corrupted with i.i.d. Gaussian measurement noise with mean zero and variance 10, $\varepsilon_{i,j} \sim N(0, 10)$, $i = 0, ..., 49$, $j = 1, 2$, that is

$$y_{i,1} = X_{i,1} + \varepsilon_{i,1} \tag{4.1}$$

$$y_{i,2} = X_{i,2} + \varepsilon_{i,2} \tag{4.2}$$

where $\varepsilon_{i,1}$ is independent of $\varepsilon_{i,2}$ for every i and they are both independent of $X_{t,1}$ and $X_{t,2}$ for every t.

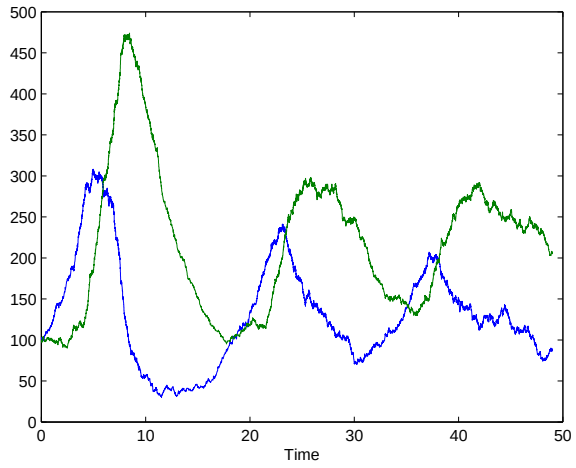


Figure 4.1: A realization from the stochastic LV model: $X_{t,1}$ (blue) and $X_{t,2}$ (green).

4.2 An SDE model

As shown in section 4.1 it is possible to simulate exactly a discrete Markov process for the three considered reactions, however exact simulation implies computationally costly inference. A continuous time approximation is often preferred and a stochastic differential equation is formulated. In this case we consider the following (*chemical Langevin equation*)[1]

$$dX_t = Sh(X_t, c)dt + S\text{diag}\{\sqrt{|h(X_t, c)|}\}dW_t \tag{4.3}$$

where $\text{diag}\{z\}$ denotes a diagonal matrix having values from vector z on its main diagonal, $X_t = (X_{t,1}, X_{t,2})^T$, $dW_t = (dW_{t,1}, dW_{t,2}, dW_{t,3})^T$ has independent (standard) Brownian motions as components, $c = (c_1, c_2, c_3)^T$ and we take $h(X_t, c) = (c_1 X_{t,1}, c_2 X_{t,1} X_{t,2}, c_3 X_{t,2})^T$. $h(\cdot)$ is the *hazard function* and S is the *stoichiometry matrix* defined as

$$S = \begin{pmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \end{pmatrix}.$$

Goal is to estimate (c_1, c_2, c_3) given noisy data $\{y_{i,1}, y_{i,2}\}_{i=0,\dots,49}$ as from (4.1)-(4.2), where the unobserved latent states are modelled via (4.3). We assume residual variability and initial state values to be known constants equal to $\sigma_\varepsilon = \sqrt{10}$ and $(x_{0,1}, x_{0,2}) = (100, 100)$ respectively[2]. The actual inference will be performed on the log-transformed rates, therefore in practice we consider `theta` $= (\log c_1, \log c_2, \log c_3)$.

[1] Such equation can be expressed in several equivalent ways, see section 7 in [8] and section 8.3.3 in [21] for a discussion. Some hints for an efficient implementation are given in `lv_sdefile.m`. Also absolute values in (4.3) are not strictly necessary, only useful for numerical stability (as in [23]).

[2] Same as with the theophylline example, we can consider the possibility to estimate the measurement error variability σ_ε, see page 36. Similarly, it is also possible to estimate the initial states $(X_{0,1}, X_{0,2})$.

Notice this is quite a different example compared to the theophylline model considered in section 3. In fact here the diffusion process X_t underlying (4.3) is not the one generating data (in the hypothesis of no measurement error), but instead here X_t is a continuous approximation to the true (discrete) stochastic process we considered to simulate chemical reactions (the Gillespie algorithm). Therefore here the postulated underlying latent state is itself an imperfect representation of reality, regardless of the presence of measurement error.

4.3 Identify a "training" region (optional)

In order to run a pilot, we specify a diffuse but proper prior $\pi_0(\texttt{theta})$: $\log c_1 \sim U(-3, 2)$, $\log c_2 \sim U(-7, 0)$, $\log c_3 \sim U(-3, 2)$. Since the pilot procedure aims at drawing samples from $\pi_0(\texttt{theta})$, we set the appropriate commands into `lv_prior.m` (bottom lines) as

```
log_c1 = unifrnd(-3,2);
log_c2 = unifrnd(-7,0);
log_c3 = unifrnd(-3,2);
```

This example is computationally intense as the SDE does not have a closed-form solution and therefore we are required to use the Euler-Maruyama integration via `integrator = 'em'`. We specified `trainiter=100`, `numsimpar = 2*100` and `regrtype='lasso'` (warning: this may require about 2,5 hrs on a performing desktop computer!). The pilot starts by invoking `abc_training.m` with

```
[training,var_training] =  abc_training(trainitermax,numsimpar,sde_param,...
            parmask,parbase,regrtype);
```

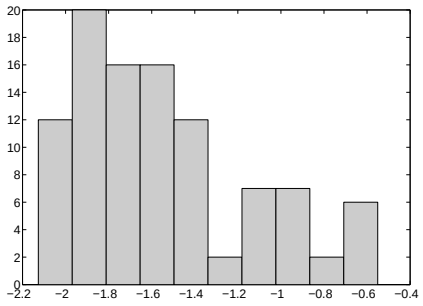
and returns a matrix `training` whose columns should be graphically explored.

`training` is a 100 x 3 matrix whose columns have the sampling distributions in Figure 4.2. We can see that such distributions have tighter supports than those for $\pi_0(\texttt{theta})$. By examining Figure 4.2 we decide to impose the following priors $\pi(\texttt{theta})$ which will be used for ABC-MCMC: $\log c_1 \sim N(-1.55, 0.4^2)$, $\log c_2 \sim N(-6, 0.3^2)$, $\log c_3 \sim N(-1.45, 0.5^2)$. See section 3.6.2 for guidance on how to code such densities, or just look at `lv_prior.m`.

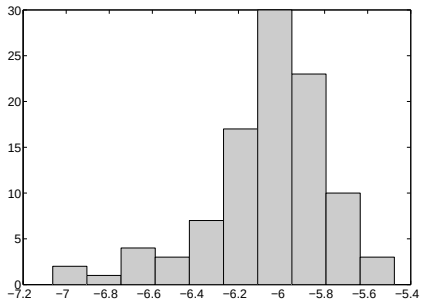
4.4 ABC–MCMC

We plug the means $(-1.55, -6, -1.45)$ from the Gaussian distributions determined from the pilot into $(\log c_1, \log c_2, \log c_3)$ in the ABC-MCMC procedure, that is by considering also constant parameters we set

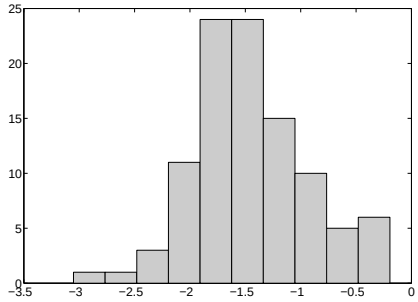
```
bigtheta_start = [log(100),  log(100),  -1.55,  -6,  -1.45,  log(sqrt(10))];
parmask        = [0       ,         0,      1,   1,      1,         0      ];
parbase = bigtheta_start;
sde_param = {bigtheta_start,problem,owntime,time,numdepvars,vrbl,integrator};
```



(a) $\log c_1$



(b) $\log c_2$



(c) $\log c_3$

Figure 4.2: LV model pilot: sampling distributions of $\log c_1$, $\log c_2$ and $\log c_3$ from the pilot procedure.

We run the ABC-MCMC for `R_mcmc = 2000000` iterations, using `numsimpar = 20*100` (ten times larger than for the pilot, as here summary statistics are computed only once) and setting a maximum value for the bandwidth `maxbandwidth = 1.1` which has to be set by trial-and-error. Also we set `mean_bandw = 0.6` and `bandw_start = 0.6` and obtained an acceptance rate of about 13% (interestingly if previous values are set to 1, 0.5 and 0.5 respectively we get a 0% acceptance rate! So our algorithm is very sensitive to such settings). Notice you might get some warnings of the type

```
Warning:  NaN found in Y, interpolation at undefined values will result in undefined
values.
```

You can safely ignore such warnings (or type `warning off` at the prompt): these are produced because some simulated trajectory contains `NaN`'s. However those trajectories will be automatically rejected by the Metropolis-Hastings acceptance ratio. After about 4 hrs of computation[3] we obtained our MCMC sample `MCMCstates`.

We retrieve the produced chain of length 40001 for each unknown parameter (remember the chain has been thinned with `thin=50` therefore it corresponds to $40000 * 50 + 1 = 2000001$ draws, the starting value and additional two-million draws) by accessing the `MCMCstates` matrix.

[3]On a Intel Core i7-2600 CPU 3.40 GHz...I know, the timing does not look very appealing for a toy model meant to be used as an illustrative example...but again, the SDE has to be integrated numerically.

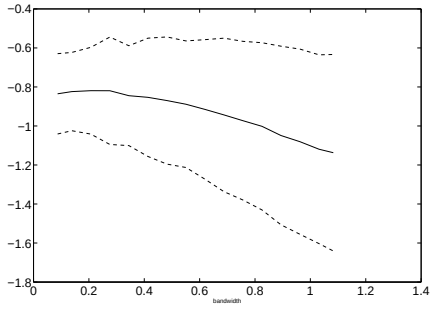
We then remove the initial 2,500 iterations (corresponding to a burn-in of `2500*50=125000` iterations)

```
>> MCMCredux = MCMCstates(2500:end,:);
```

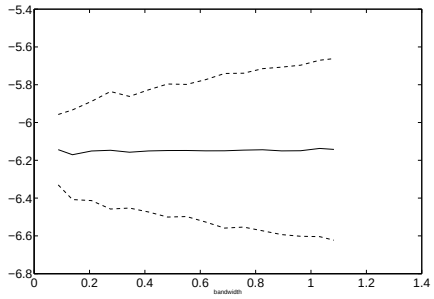
Then we explore each chain for varying bandwidths by inspecting the plots produced by

```
>> abc_posthoc(MCMCredux,1,4,15);
>> abc_posthoc(MCMCredux,2,4,15);
>> abc_posthoc(MCMCredux,3,4,15);
```

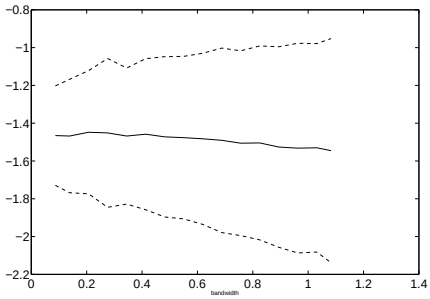
By looking at Figure 4.3 we deduce that we should keep draws corresponding to $\delta < 0.25$ as for larger bandwidth posterior means vary markedly (particularly $\log c_1$). We retain draws



(a) $\log c_1$



(b) $\log c_2$



(c) $\log c_3$

Figure 4.3: LV model: posterior means for varying bandwidth δ.

satisfying the criterion above via the following (notice we also get rid of the last (fourth) column corresponding to the bandwidth δ, which is of no use anymore):

```
>> MCMCfinal = MCMCredux(MCMCredux(:,end)<0.25,1:3);
```

we are thus left with a sample of about 1120 draws for each chain. We now check whether the autocorrelation function for each chain decays rapidly: we do this via the `acf` function:

```
>> acf(MCMCfinal(:,1),30);
>> acf(MCMCfinal(:,2),30);
>> acf(MCMCfinal(:,3),30);
```

We notice a rapid decay in the autocorrelations (not reported) and therefore there is no need for further thinning and we can directly use the chains to produce posterior inference. We exponentiate means and percentiles in order to make inference for the original untransformed parameters (c_1, c_2, c_3).

```
>> exp(mean(MCMCfinal(:,1:3)))

ans =

    0.4398    0.0021    0.2331

>> exp(perctile(MCMCfinal(:,1:3),2.5))

ans =

    0.3478    0.0017    0.1709

>> exp(perctile(MCMCfinal(:,1:3),97.5))

ans =

    0.5521    0.0028    0.3279
```

Therefore posterior means, 2.5th–97.5th posterior percentiles are: 0.44 [0.35,0.55], 0.0021 [0.0017, 0.0028], 0.23 [0.17, 0.33] for c_1, c_2 and c_3 respectively, where the true values used to produce data are $(c_1, c_2, c_3) = (0.5, 0.0025, 0.3)$.

By considering the vagueness of the initial prior $\pi_0(\texttt{theta})$ and the several approximations affecting our results (continuous approximation to a discrete process, Euler-Maruyama discretisation, measurement noise, $\delta > 0$, non-sufficient statistics) the results are quite satisfactory. Figure 4.4 gives a comparison between the true value of $\log c_1$, the kernel smoothing estimate of the ABC posterior density for $\log c_1$, its Gaussian prior $\pi(\log c_1)$ used during the MCMC and the uniform prior $\pi_0(\log c_1)$ used in the pilot.

4.5 A challenging case: partially observed systems

In this section we consider the case where only the first coordinate $X_{t,1}$ is observed (again with known measurement error). This means that during the ABC inference we still simulate from the 2-dimensional SDE system, however only summary statistics for the first coordinate are used (of course after adding simulated measurement noise to $X_{t,1}$) for comparison against the summary statistics for real data.

If we only consider, for simplicity, the possibility to load our data-file, then there is no modification to be made to the code considered above for the fully observed case. That is to say, we simply removed from `data_full.dat` the rows corresponding to the second coordinate: the reduced data-file is available as `data_partial.dat` to reflect that we are now dealing with a *partially observed system*. The only relevant difference with the code for the fully observed system is the use of

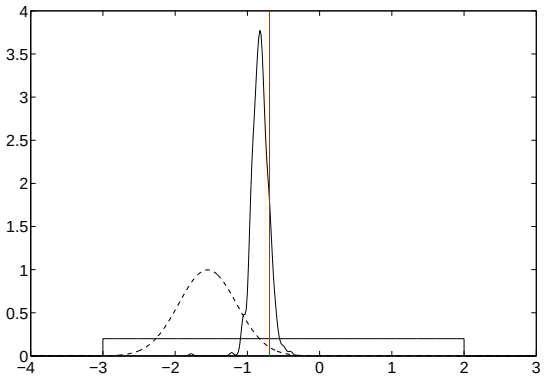


Figure 4.4: LV model: approximate posterior for $\log c_1$ (solid curve), Gaussian prior (dashed line) and uniform prior used during the pilot. The vertical line corresponds to the true value of $\log c_1$.

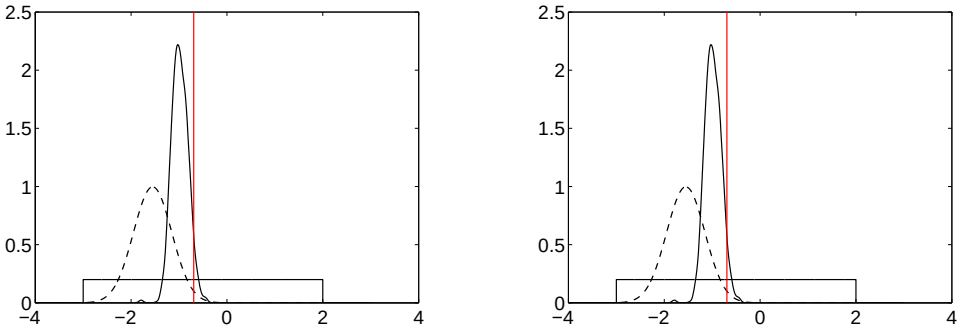
```
load('lv_full_seed')
% restore the state for the pseudorandom numbers used to create the data-file
rng(lv_full_seed)
load('data_partial.dat')
time = data_partial(:,1);
yobs = data_partial(:,2);
vrbl = data_partial(:,3);
```

All other instructions can be left unchanged and the code can be run. However for ease of use we created a separate run-file, named `lv_partial.run`. For simplicity we use the same settings as in the previous (fully observed) case, using the same priors etc. although a new pilot could be performed leading to different training regions. After running the MCMC simulations via `abc_mcmc` we filter the obtained `MCMCstates` matrix:

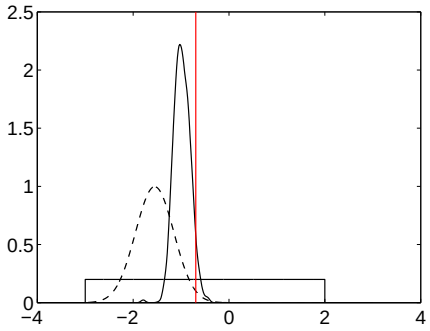
```
>> MCMCredux = MCMCstates(2500:end,:); % get rid of burn-in
>> abc_posthoc(MCMCredux,1,4,15);
>> abc_posthoc(MCMCredux,2,4,15);
>> abc_posthoc(MCMCredux,3,4,15);
% from the plots created above we decide to select delta < 0.2
>> MCMCfinal = MCMCredux(MCMCredux(:,end)<0.2,:);
>> acf(MCMCfinal(:,1),40);
>> acf(MCMCfinal(:,2),40);
>> acf(MCMCfinal(:,3),40);
% no serious autocorrelation is detected
>> exp(mean(MCMCfinal(:,1:3)))

ans =

    0.3749    0.0019    0.2515
>> exp(perctile(MCMCfinal(:,1:3),2.5))

ans =

    0.2780    0.0013    0.1744
>> exp(perctile(MCMCfinal(:,1:3),97.5))
```

```
ans =

    0.5283    0.0029    0.3793
```

Therefore our estimate for c_1 is 0.37 [0.28,0.53], for c_2 is 0.0019 [0.0013,0.0029] and for c_3 is 0.25 [0.17,0.38]; again, true values are $(c_1, c_2, c_3) = (0.5, 0.0025, 0.3)$. Comparisons between ABC posteriors, priors used during MCMC and priors used during the pilot are in Figure 4.5.



(a) $\log c_1$ (b) $\log c_2$



(c) $\log c_3$

Figure 4.5: Partially observed LV model: approximate posteriors (solid curves), Gaussian priors (dashed lines) and uniform priors used during the pilot. Vertical lines corresponds to true parameter values.

Bibliography

[1] J. Friedman, T. Hastie, and R. Tibshirani. Glmnet for MATLAB: Lasso (L1) and elastic-net regularized generalized linear models, 2010. Matlab wrapper maintained by H. Jiang. `http://www-stat.stanford.edu/~tibs/glmnet-matlab/`.

[2] J. Friedman, T. Hastie, and R. Tibshirani. Regularization paths for generalized linear models via coordinate descent. *Journal of Statistical Software*, 33(1), 2010.

[3] G. Jekabsons. ARESLab: Adaptive regression splines toolbox for Matlab/Octave, 2011. Version 1.5.1 `http://www.cs.rtu.lv/jekabsons/regression.html`.

[4] J. Friedman. Multivariate adaptive regression splines. *The Annals of Statistics*, pages 1–67, 1991.

[5] T. Vejchodský. GillespieSSA for MATLAB, 2008. Release 1, `http://www.math.cas.cz/~vejchod/gillespiessa/gillespiessa.html`.

[6] C. Price. Autocorrelation function (ACF). `http://www.mathworks.com/matlabcentral/fileexchange/30540-autocorrelation-function-acf` (MATLAB Central File Exchange), retrieved 11 March 2013.

[7] P.J. Acklam. Percentiles. `http://home.online.no/~pjacklam/matlab/software/util/statutil/perctile.m`, retrieved 11 March 2013.

[8] U. Picchini. Inference for SDE models via approximate Bayesian computation. `arXiv:1204.5459`.

[9] S.A. Sisson and Y. Fan. *Handbook of Markov chain Monte Carlo*, chapter Likelihood-free Markov chain Monte Carlo. CRC Press, 2011. Available as `arXiv:1001.2058`.

[10] J.M. Marin, P. Pudlo, C.P. Robert, and R.J. Ryder. Approximate Bayesian computational methods. *Statistics and Computing*, 22(6):1167–1180, 2012.

[11] P. Fearnhead and D. Prangle. Constructing summary statistics for approximate Bayesian computation: semi-automatic approximate Bayesian computation. *Journal of the Royal Statistical Society: Series B*, 74(3):419–474, 2012.

[12] R.D. Wilkinson. Approximate Bayesian computation (ABC) gives exact results under the assumption of model error. To appear in Statistical Applications in Genetics and Molecular Biology, available as `arXiv:0811.3355`, 2013.

[13] O. Cappé, E. Moulines, and T. Rydén. *Inference in Hidden Markov Models*. Springer, 2005.

[14] H. Haario, E. Saksman, and J. Tamminen. An adaptive Metropolis algorithm. *Bernoulli*, 7(2):223–242, 2001.

[15] C. Andrieu, A. Doucet, and R. Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B*, 72(3):269–342, 2010.

[16] A. Golightly and D.J. Wilkinson. Bayesian parameter inference for stochastic biochemical network models using particle Markov chain Monte Carlo. *Interface Focus*, 1(6):807–820, 2011.

[17] P. Bortot, S.G. Coles, and S.A. Sisson. Inference for stereological extremes. *Journal of the American Statistical Association*, 102(477):84–92, 2007.

[18] J. Pinheiro and D. Bates. Approximations to the log-likelihood function in the nonlinear mixed-effects model. *Journal of Computational and Graphical Statistics*, 4(1):12–35, 1995.

[19] J. Pinheiro and D. Bates. *Mixed-Effects Models in S and S-PLUS*. Springer, 2000.

[20] P.E. Kloeden and E. Platen. *Numerical Solution of Stochastic Differential Equations*. Springer, 1992.

[21] D.J. Wilkinson. *Stochastic Modelling for Systems Biology*. CRC Press, second edition, 2012.

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

[23] D.J. Higham. Modeling and simulating chemical reactions. *SIAM review*, 50(2):347–368, 2008.