Approximate Bayesian Computation MCMC

Federico Di Gennaro, Leonardo Bruno Trentini, Michele Lupini
Stochastic Simulation, MATH-414
GitHub repository

1 Introduction

Many problems in data science involve estimating a set of parameters $\theta \in \Theta$ of a model M that describes the processes underlying the problem of interest. In the Bayesian inference paradigm, the uncertainty over such a set of parameters is quantified by means of a *posterior distribution*, which is often described through sampling techniques. In this context, one assumes the parameters to follow a *prior distribution* $\pi(\theta)$, which outlines the current belief on the problem at hand, *e.g.*, based on available expert knowledge. After having observed some data $\mathcal{D}$, this prior belief is updated using the *likelihood function* $\mathbb{P}(\mathcal{D}|\theta)$, which describes the plausibility of having generated such data under all possible different values of θ. The posterior distribution of interest, $f(\theta|\mathcal{D})$, is then determined by Bayes' rule

$$f(\theta|\mathcal{D}) = \mathbb{P}(\mathcal{D}|\theta)\pi(\theta)/\mathbb{P}(\mathcal{D})$$

where $\mathbb{P}(\mathcal{D}) = \int_\Theta \mathbb{P}(\mathcal{D}|\theta)\pi(\theta)d\theta$, called the evidence, represents the normalizing constant.
Stochastic simulation approaches for generating observations from the posterior distribution $f(\theta|\mathcal{D})$ often depend on knowing explicitly the likelihood function $\mathbb{P}(\mathcal{D}|\theta)$, possibly up to a multiplicative constant (*i.e.* being able to evaluate it for any θ and D). However, for many complex probabilistic models, such likelihoods are either inaccessible or computationally prohibitive to evaluate, so one has to resort to the so-called likelihood-free methods [1], of which, most notably the **Approximate Bayesian Computation** (ABC) [2].

1.1 ABC algorithm

In brief, ABC algorithms sample candidate parameters θ^* from the prior distribution $\pi(\theta)$, generate a data sample $\mathcal{D}^*$ given the candidate parameters θ^* and then compare it with the observed data $\mathcal{D}$ according to some pre-defined discrepancy metric $\rho(\cdot,\cdot)$ and tolerance ϵ. In the acceptance/rejection step, the candidate parameters are accepted as samples from the posterior if the simulated data is similar enough (in terms of the chosen $\rho(\cdot,\cdot)$ and ϵ) to the observed data $\mathcal{D}$. Otherwise, a new candidate is sampled from the prior. This scheme is then repeated until a (sufficiently large) sample of size N, distributed approximately as the posterior, is obtained. An outline of an ABC rejection algorithm scheme can be found in Algorithm 1 below.

This approach requires suitable choices of the metric ρ and tolerance ϵ. This is due to the fact that Algorithm 1 provides samples from the approximated posterior $f(\theta|\rho(\mathcal{D}^*,\mathcal{D}) < \epsilon)$. As $\epsilon \to \infty$, the algorithm generates observations from the prior. If, on the contrary, $\epsilon = 0$, and the observation $\mathcal{D}^*$ is accepted only if $\mathcal{D}^* = \mathcal{D}$, then the accepted observations come from the true posterior density $f(\theta|\mathcal{D})$. The choice of ϵ therefore involves a trade-off between the computability and the accuracy of the method [3].

Algorithm 1 Basic ABC Rejection Method

1: **for** $i = 1, ..., N$ **do**
2: Sample candidate parameters from the prior distribution $\theta^* \sim \pi(\cdot)$
3: Generate data from the underlying model given $\theta^*, \mathcal{D}^* \sim \mathrm{P}(\cdot|\theta^*)$
4: **if** $\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon$ **then**
5: Set $\theta^i \leftarrow \theta^*$
6: **else**
7: Go back to Step 2.
8: **end if**
9: **end for**

Lemma 1.1. *ABC algorithm generates samples distributed as* $f(\theta|\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon)$.

Proof. We want to show that ABC samples are distributed as:

$$f(\theta|\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon) = \frac{\mathbb{P}(\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon|\theta)\pi(\theta)}{\mathbb{P}(\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon)}$$

Firstly, similarly to what we observed in the *Acceptance-Rejection* method [4], we can observe that the distribution of θ is the distribution of θ^* conditional to the event $\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon$:

$$\theta \sim \theta^*|\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon$$

Hence:

$$\mathbb{P}(\theta) = \mathbb{P}(\theta^*|\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon)$$

Then, for any $T \subseteq \mathbb{R}^n$ we can use Bayes rule and write the following:

$$\begin{aligned}\mathbb{P}(\theta \in T) = \mathbb{P}(\theta^* \in T|\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon) &= \frac{\mathbb{P}(\theta^* \in T, \rho(\mathcal{D}^*, \mathcal{D}) < \epsilon)}{\mathbb{P}(\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon)} \\ &= \int_T \frac{\mathbb{P}(\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon|\theta^* = t)\pi(t)dt}{\mathbb{P}(\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon)} \\ &= \int_T f(\theta = t|\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon)dt\end{aligned}$$

Hence, we showed that the density function is exactly $f(\theta|\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon)$ as requested.

□

1.2 ABC-MCMC algorithm

The sampling strategy outlined in Algorithm 1 poses challenges on its own, even for simple models. Acceptance rates can be very low as candidate parameter vectors are generated from the prior $\pi(\theta)$, which may be quite different from the posterior. Thus, several accelerating techniques have been proposed, for instance, to embed the ABC scheme within the well-known *Metropolis-Hastings* framework, to get an **ABC-Markov Chain Monte Carlo** (ABC-MCMC, see Algorithm 2) [5]. In this setting, the algorithm generates a sequence of serially correlated samples from $f(\theta|\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon)$. Determination of the chain length, N, is therefore obtained through a careful assessment of the convergence of the chain and considerations of the chain's ability to explore the parameter space Θ (i.e., chain mixing).
Observe that the candidate vector of parameters is generated from an arbitrary proposal transition density $q(\cdot, \cdot)$ and accepted with a Metropolis-Hastings type acceptance probability, in

Algorithm 2 ABC-MCMC

Initialize θ_0
for $i = 1, ..., N$ **do**
  Sample candidate parameters θ^* from a proposal transition density $q, \theta^* \sim q(\theta_i, \cdot)$
  Generate data from the underlying model given $\theta^*, \mathcal{D}^* \sim \mathrm{P}(\mathcal{D}|\theta^*)$
  if $\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon$ **then**
    Set $\theta_{i+1} \leftarrow \theta^*$ with probability $\alpha = \min\left(1, \frac{\pi(\theta^*)q(\theta^*, \theta_i)}{\pi(\theta_i)q(\theta_i, \theta^*)}\right)$ and $\theta_{i+1} \leftarrow \theta_i$ otherwise
  else
    Set $\theta_{i+1} \leftarrow \theta_i$
  end if
end for

which, however, the (intractable) likelihood ratio $\mathrm{P}(\cdot|\theta^*)/\mathrm{P}(\cdot|\theta_i)$ is coarsely approximated by 1 under the assumption that the simulated and observed data are sufficiently close according to the chosen metric $\rho(\cdot, \cdot)$, and 0 otherwise.

Lemma 1.2. *$f(\theta|\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon)$ is the stationary distribution of the chain generated by ABC-MCMC.*

Proof. In order to prove that $f(\theta|\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon)$ is the stationary distribution of the chain generated by ABC-MCMC, we can show that the transition kernel P of the Algorithm 2 is in detailed balance with the probability density $f(\cdot|\rho(\mathcal{D}^*, \mathcal{D})$. Which implies that $f(\cdot|\rho(\mathcal{D}^*, \mathcal{D})$ is an invariant probability density for P.
First, let's denote the acceptance rate α as

$$\alpha(\theta, \theta^*) = \min\left(1, \frac{\pi(\theta^*)q(\theta^*, \theta_i)}{\pi(\theta_i)q(\theta_i, \theta^*)}\right).$$

where $q(\cdot, \cdot)$ is the proposal transition density of the algorithm and $\pi(\cdot, \cdot)$ is the prior.
Then, if we denote the chain generated by the algorithm as $\{X_n\}$, by definition of the transition kernel one has that $P(\theta, A) = \mathbb{P}(X_{n+1} \in A|X_n = \theta)$, for any $\theta \in \Theta$, $A \in \mathcal{B}(\Theta)$. Moreover, we observe that if $\theta \notin A$:

$$P(\theta, A) = \int_A q(\theta, \theta^*)\mathbb{P}(\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon|\theta^*)\alpha(\theta, \theta^*)\, d\theta^*.$$

Conversely, if $\theta \in A$:

$$\begin{aligned} P(\theta, A) = &\int_A q(\theta, \theta^*)\mathbb{P}(\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon|\theta^*)\alpha(\theta, \theta^*)\, d\theta^* \\ &+ \int_\Theta q(\theta, \theta^*)\left[1 - \mathbb{P}(\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon|\theta^*)\alpha(\theta, \theta^*)\right] d\theta^*. \end{aligned}$$

Therefore, we can define $\tilde{\alpha}(\theta, \theta^*) = \mathbb{P}(\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon|\theta^*)\alpha(\theta, \theta^*)$ and $\tilde{\alpha}^*(\theta) = \int_\Theta \tilde{\alpha}(\theta, \theta^*)q(\theta, \theta^*)\, d\theta^*$, so that the transition kernel P and its density p are given by

$$\begin{aligned} P(\theta, A) &= \int_A \tilde{\alpha}(\theta, \theta^*)q(\theta, \theta^*)\, d\theta^* + \mathbb{1}_A(\theta)\left(1 - \alpha^*(\theta)\right), \\ p(\theta, \theta^*) &= \tilde{\alpha}(\theta, \theta^*)q(\theta, \theta^*) + \delta_\theta(\theta^*)\left(1 - \alpha^*(\theta)\right). \end{aligned}$$

Now, notice that

$$
\begin{aligned}
&f(\theta|\rho(\mathcal{D}^*,\mathcal{D})q(\theta,\theta^*)\tilde{\alpha}(\theta,\theta^*)\\
&\quad= f(\theta|\rho(\mathcal{D}^*,\mathcal{D})<\epsilon)q(\theta,\theta^*)\mathbb{P}(\rho(\mathcal{D}^*,\mathcal{D})<\epsilon|\theta^*)\alpha(\theta,\theta^*)\\
&\quad= \frac{\mathbb{P}(\rho(\mathcal{D}^*,\mathcal{D})<\epsilon|\theta)\pi(\theta)}{\mathbb{P}(\rho(\mathcal{D}^*,\mathcal{D})<\epsilon)}q(\theta,\theta^*)\mathbb{P}(\rho(\mathcal{D}^*,\mathcal{D})<\epsilon|\theta^*)\min\left\{1,\ \frac{\pi(\theta^*)q(\theta^*,\theta)}{\pi(\theta)q(\theta,\theta^*)}\right\}\\
&\quad= \frac{\mathbb{P}(\rho(\mathcal{D}^*,\mathcal{D})<\epsilon|\theta)}{\mathbb{P}(\rho(\mathcal{D}^*,\mathcal{D})<\epsilon)}\mathbb{P}(\rho(\mathcal{D}^*,\mathcal{D})<\epsilon|\theta^*)\min\left\{\pi(\theta)q(\theta,\theta^*),\ \pi(\theta^*)q(\theta^*,\theta)\right\}\\
&\quad= \frac{\mathbb{P}(\rho(\mathcal{D}^*,\mathcal{D})<\epsilon|\theta^*)\pi(\theta^*)}{\mathbb{P}(\rho(\mathcal{D}^*,\mathcal{D})<\epsilon)}q(\theta^*,\theta)\mathbb{P}(\rho(\mathcal{D}^*,\mathcal{D})<\epsilon|\theta)\min\left\{\frac{\pi(\theta)q(\theta,\theta^*)}{\pi(\theta^*)q(\theta^*,\theta)},\ 1\right\}\\
&\quad= f(\theta^*|\rho(\mathcal{D}^*,\mathcal{D})<\epsilon)q(\theta^*,\theta)\mathbb{P}(\rho(\mathcal{D}^*,\mathcal{D})<\epsilon|\theta)\alpha(\theta^*,\theta)\\
&\quad= f(\theta^*|\rho(\mathcal{D}^*,\mathcal{D})<\epsilon)q(\theta^*,\theta)\tilde{\alpha}(\theta^*,\theta)
\end{aligned}
$$

Finally, proceeding as in Lemma 8.11 of [4] one can easily show that the detailed balance holds:

$$\int_A P(\theta,B)f(\theta|\rho(\mathcal{D}^*,\mathcal{D})<\epsilon)\,d\theta = \int_B P(\theta^*,A)f(\theta^*|\rho(\mathcal{D}^*,\mathcal{D})<\epsilon)\,d\theta^*.$$

□

2 Academic Example

As the first step of the project, we considered an academic example where the likelihood function and true posterior distribution f are actually known, so that we can compare the distributions obtained for different values of tolerance ϵ with basic ABC (Algorithm 1) and ABC-MCMC (Algorithm 2) with the true posterior f.
In this example, the observed data $\mathcal{D} = \{x_i\}_{i=1}^N \subset \mathbb{R}$ is an *i.i.d.* sample drawn with probability $1/2$ from $\mathcal{N}(\theta,\sigma_1^2)$ and with probability $1/2$ from $\mathcal{N}(\theta+\alpha,\sigma_1^2)$. As a prior we took $\pi = \mathcal{N}(0,\sigma^2)$. Then, the posterior distribution was a Gaussian mixture given by

$$f(\theta|\mathcal{D}) = \alpha\mathcal{N}\left(\frac{\sigma^2}{\sigma^2+\sigma_1^2/M}\overline{x},\frac{\sigma_1^2}{M+\sigma_1^2/\sigma^2}\right)+(1-\alpha)\mathcal{N}\left(\frac{\sigma^2}{\sigma^2+\sigma_1^2/M}(\overline{x}-\alpha),\frac{\sigma_1^2}{M+\sigma_1^2/\sigma^2}\right) \quad (1)$$

with

$$\alpha = \frac{1}{1+\exp\left\{a(\overline{x}-\frac{a}{2})\frac{M}{M\sigma^2+\sigma_1^2}\right\}},$$

where $\overline{x} = \frac{1}{M}\sum_{i=1}^M x_i$ denotes the sample mean of the data and $\mathcal{N}(\mu,\sigma^2)$ denotes the density of a Gaussian random variable with mean μ and variance σ^2.

We considered the following parameters: $M = 100$, $\sigma_1^2 = 0.1$, $\sigma^2 = 3$, $a = 1$ and we assume that the sample mean of the data is exactly $\overline{x} = 0$.

2.1 Basic ABC: results

We implemented the basic ABC rejection algorithm described in Algorithm 1 for tolerances $\epsilon = \{0.75, 0.25, 0.1, 0.025\}$, until $N = 500$ samples were accepted, with discrepancy metric defined as

$$\rho(S(\mathcal{D}^*),S(\mathcal{D})) = |\overline{x}^* - \overline{x}|, \quad (2)$$

where $\overline{x}^* = \frac{1}{M}\sum_{i=1}^{M} x_i^*$ is the sample mean of the generated data $\mathcal{D}^* = \{x_i^*\}_{i=1}^{M}$ according to the mechanism described above.

We then plotted the histogram of our samples along with the true mixture distribution in 1 for the 4 considered tolerances $\epsilon_i,\ i = 1, ..., 4$.

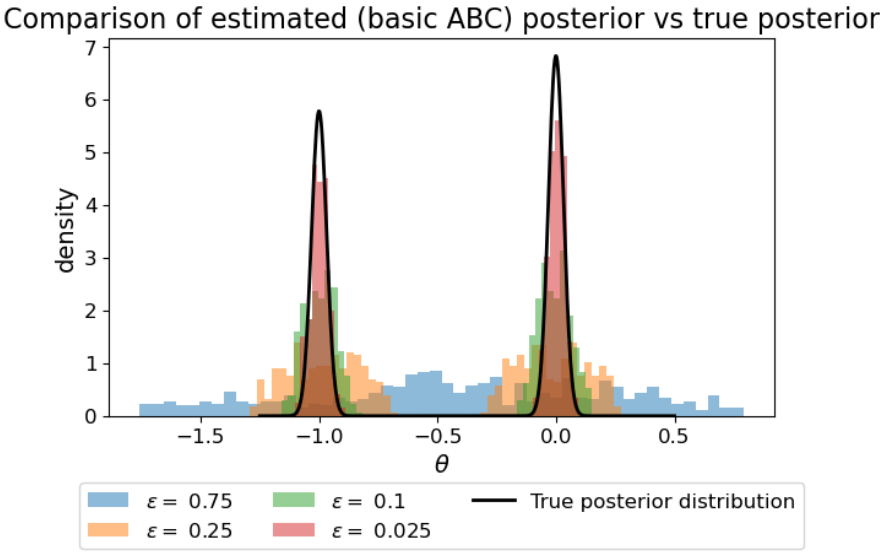

Figure 1: Samples and true mixture distribution for ABC algorithm.

The acceptance rates observed for each of the considered tolerances are shown in Table 1. It is possible to notice that, as expected, if we decrease the tolerance the acceptance rate decreases as well. We averaged over different experiments the results obtained for the acceptance rates so that we can plot them as in 2 to observe also the confidence intervals of the rate of increase.

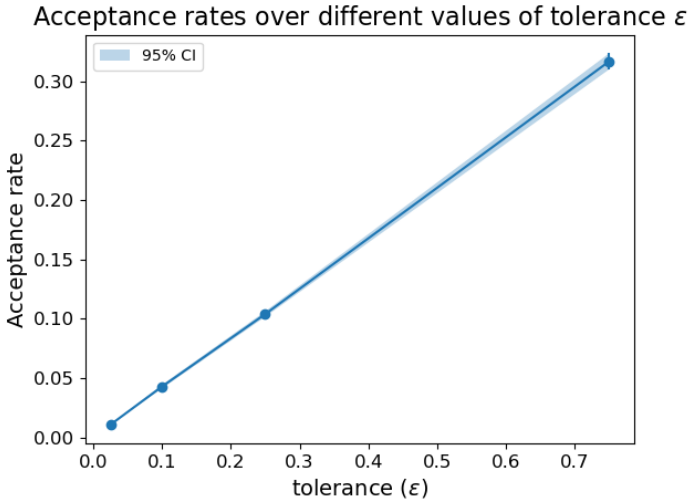

Figure 2: Acceptance rate against ϵ for basic ABC algorithm.

2.2 ABC-MCMC: results

Similarly, we implemented the ABC-MCMC method described in Algorithm 2, with random walk proposal $q(\theta, \cdot) = \mathcal{N}(\theta, \nu^2)$, initial state $\theta_0 = 0$ and discrepancy metric defined in Eq. 2.

We tried different values of ν^2. We run the chain long enough to have an effective sample size $N_{eff} \approx 500$.

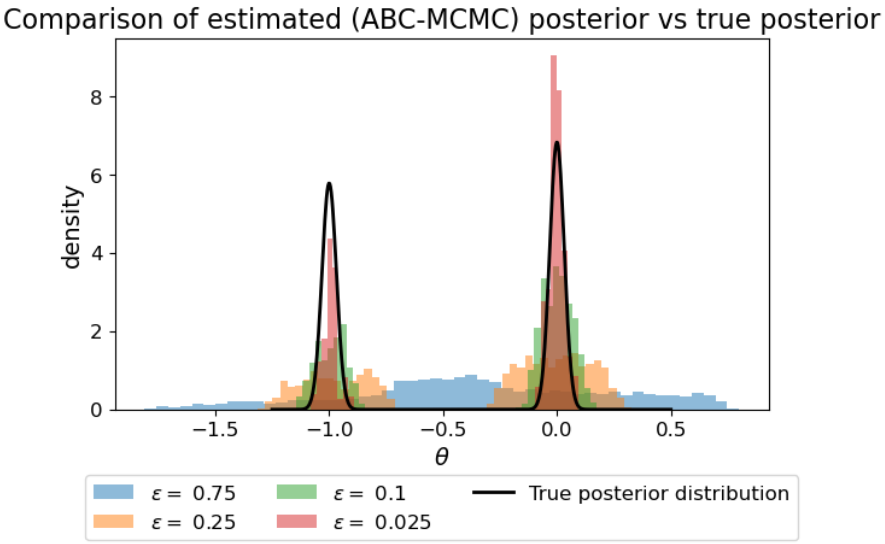

Figure 3: Samples and true mixture distribution for ABC-MCMC algorithm.

As expected, the estimated ABC-MCMC posterior distribution gets closer to the true posterior at the decrease of the acceptance rate ϵ. The results, in terms of approximation of the true posterior, are comparable to the ABC ones, as expected. The real gain of ABC-MCMC comes in terms of acceptance rates. A comparison is drawn in Table 1.

tolerance	acceptance rate basic ABC	acceptance rate ABC-MCMC
0.75	0.316769 ± 0.006739	0.385838 ± 0.006739
0.25	0.103813 ± 0.002282	0.141390 ± 0.002282
0.1	0.042757 ± 0.001535	0.056511 ± 0.001535
0.025	0.010640 ± 0.000279	0.014395 ± 0.000279

Table 1: Comparison of acceptance rates for ABC and ABC-MCMC algorithm.

As it is possible to observe from Table 1, on equal terms (*i.e.*, same tolerance ϵ) the ABC-MCMC algorithm leads to higher acceptance rates than the ABC one. This fact is even more perceivable if we plot the acceptance rates (with the 95% confidence intervals) against the tolerance ϵ, for the two algorithms under analysis (see Figure 4)

3 Pharmacokinetic model and summary statistics

We want to apply the ideas described above to a dynamic model of the pharmacokinetics of Theophylline, a drug used in the treatment of asthma and chronic obstructive pulmonary disease. In pharmacokinetics one aims to study a drug of interest by describing its absorption, distribution, metabolism, and excretion mechanisms from the body. A fundamental concept in pharmacokinetics is drug clearance, that is, the elimination of drugs from the body. Let X_t be the level of Theophylline concentration in blood at time t, then the evolution of X_t over time can be modeled using the following Stochastic Differential Equation (SDE):

$$dX_t = \left(\frac{DK_aK_e}{Cl} e^{-K_a t} - K_e X_t \right) dt + \sigma dW_t, \tag{3}$$

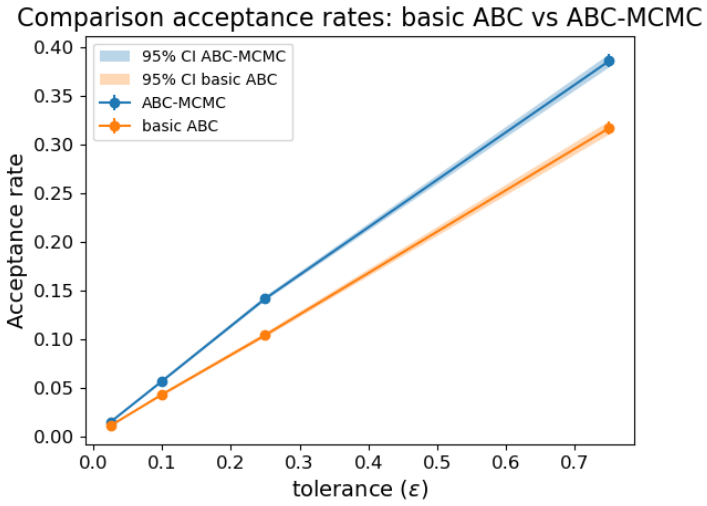

Figure 4: Comparison of acceptance rate againts ϵ for the two algorithms.

where D is the known drug oral dose received by a subject, K_e is the elimination rate constant, K_a the absorption rate constant, Cl the clearance of the drug, and σ the intensity of intrinsic stochastic noise driven by the Brownian motion W_t.
The experimental design for a single hypothetical subject considers nine blood samples taken at $\{t_1, ..., t_9\} = \{0.25, 0.5, 1, 2, 3.5, 5, 7, 9, 12\}$ hours after dosing. The drug oral dose is chosen to be $D = 4$ mg and is administered starting from $t_0 = 0^+$. The initial drug concentration in blood is $X_0 = 0$. Inference is based on data $\{x_i\}_{i=1}^9$ collected at times $\{t_1, ..., t_9\}$ and the parameters of interest are $\theta = (K_e, K_a, Cl, \sigma)$, for which the following priors are considered: $logK_e \sim \mathcal{N}(-2.7, 0.6^2)$, $\log K_a \sim \mathcal{N}(0.14, 0.4^2)$, $\log Cl \sim \mathcal{N}(-3, 0.8^2)$, $\log \sigma \sim \mathcal{N}(-1.1, 0.3^2)$.

We generated synthetically the data $\mathcal{D} = \{x_i\}_{i=1}^9$ by simulating the model 3 with parameters $\theta = (0.08, 1.5, 0.04, 0.2)$ and recording the solution at the sampling times $\{t_1, ..., t_9\}$ to get $n = 9$ values for the process X_t.

Various solutions are possible to solve the SDE in Eq. 3. For this project, we decided to use **Euler-Maruyama** method (Algorithm 3), with time step $\Delta t = 0.05$. It is proved that the Euler-Maruyama method, in general, has a strong order of convergence $\frac{1}{2}$ (see Definition 1). For this reason, in certain cases, it could be more appropriate to simulate the stochastic process X_t with a method that has a higher strong order of convergence. Such a method could be for example the *Milstein-Platen* scheme (Algorithm 4), which has strong order of convergence 1. However, for this particular setting the additional term of this Milstein-Platen scheme simplifies, since the term $g(t, X_t)$ in Algorithm 4 would be constant and independent of t or of X_t, and hence update rule in Line 5 of Algorithm 4 gets rid of the last term. Finally, this process could be simulated also by using algorithms for Gaussian process generation, since the pharmacokinetic model in Eq. 3 is a Gaussian process.

In Figure 5, it is possible to observe a simulation of the process using *Euler-Maruyama* method. Then, to construct the summary statistics $S(\mathcal{D})$, we fitted the following multivariate linear regression model

$$\theta = \beta_0 + \beta_1 x_1 + ... + \beta_9 x_9 + \xi \tag{4}$$

where $\theta = (\theta_1, ..., \theta_4)$ is the vector of parameters, $\beta_i \in \mathbb{R}^4$, $i = 0, ..., 9$, are unknown regression coefficients, $\{x_i\}_{i=1}^9$ are the generated data $\mathcal{D}$ and $\xi = (\xi_1, ..., \xi_4)$ is a random vector with zero mean, independent components and constant variance. For this task, we generated 10000

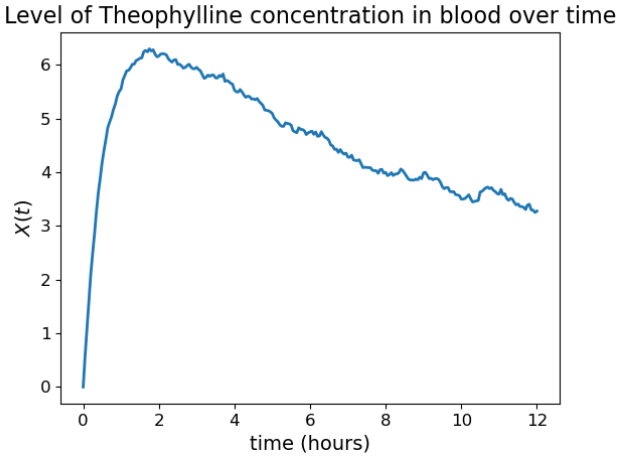

Figure 5: Theophylline concetration evolution over time

values of the parameters $\theta^{(1)}, ..., \theta^{(p)}$ from the prior distribution, and generate corresponding data $\mathcal{D}^{(1)}, ..., \mathcal{D}^{(p)}$ to build the linear regression model.
If $\hat{\beta}_i$, $i = 0, ..., 9$, denote the estimated regression coefficients, then the summary statistics reads

$$S(\mathcal{D}) = \mathbb{E}[\theta|\mathcal{D}] = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \\ \theta_4 \end{bmatrix} = \begin{bmatrix} \hat{\beta}_0^{(1)} \\ \hat{\beta}_0^{(2)} \\ \hat{\beta}_0^{(3)} \\ \hat{\beta}_0^{(4)} \end{bmatrix} + \begin{bmatrix} \hat{\beta}_1^{(1)} & ... & \hat{\beta}_9^{(1)} \\ \hat{\beta}_1^{(2)} & ... & \hat{\beta}_9^{(2)} \\ \hat{\beta}_1^{(3)} & ... & \hat{\beta}_9^{(3)} \\ \hat{\beta}_1^{(4)} & ... & \hat{\beta}_9^{(4)} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_9 \end{bmatrix} \tag{5}$$

Let $\{x_i\}_{i=1}^9$ the data obtained by *Euler-Maruyama* discretization with step-size $\Delta t = 0.05$ of Eq. 3 with parameters $\theta = (0.08, 1.5, 0.04, 0.2)$. Once estimating $\{\hat{\beta}_i\}_{i=0}^9$ using the Linear regression explained above, we obtain the following summary statistic $S(\mathcal{D})$ of $\{x_i\}_{i=1}^9$:

$$S(\mathcal{D}) = [0.0745,\ 1.3483,\ 0.0640,\ 0.3439].$$

4 ABC-MCMC for the pharmacokinetic model

For tolerances $\epsilon = \{0.25, 0.7, 1\}$, we run $N = 10000$ iterations of the ABC-MCMC algorithm (Algorithm 2) with initial state $\theta_0 = (0.07, 1.15, 0.05, 0.33)$ and discrepancy metric

$$\rho(S(\mathcal{D}^*), S(\mathcal{D})) = \|S(\mathcal{D}^*) - S(\mathcal{D})\|, \tag{6}$$

where $\|\theta\|^2 = \sum \frac{\theta_i^2}{(\theta_0)_i^2}$is a weighted euclidean norm in $\mathbb{R}^4$.

This section aims to compute the approximated posterior distribution $f(\theta|\rho(\mathcal{D}^*, \mathcal{D}) < \epsilon)$. To do so, we used the ABC-MCMC algorithm (Algorithm 2).
In order to define this algorithm, we needed to suitably choose the transition density q. Since the parameters θ are expected to be positive, we decided to use as transition density q_i for the i-th parameter θ_i a *lognormal distribution* coming from a normal distribution with mean $\mu = \theta_i$ and variance $\sigma^2 = \sigma_i^2$, where σ_i^2 is the variance of the normal distributions used to define the corresponding lognormal distribution of the prior. Notice that this approach can be considered *anisotropic* since the transition probability q_i is different for each of the four parameters. Further, to calculate α in Line 6 of Algorithm 2, we assumed that our four parameters $\theta = (K_e, K_a, Cl, \sigma)$ were independent of each other. Hence, the priors calculated are simply the

product of all the priors of the four parameters.

Now we discuss the influence of the tolerance on the acceptance rate and the obtained posterior distribution. To do so, we plotted the marginal distribution of each of the four parameters $\theta = (K_e, K_a, Cl, \sigma)$ in Figure 6.

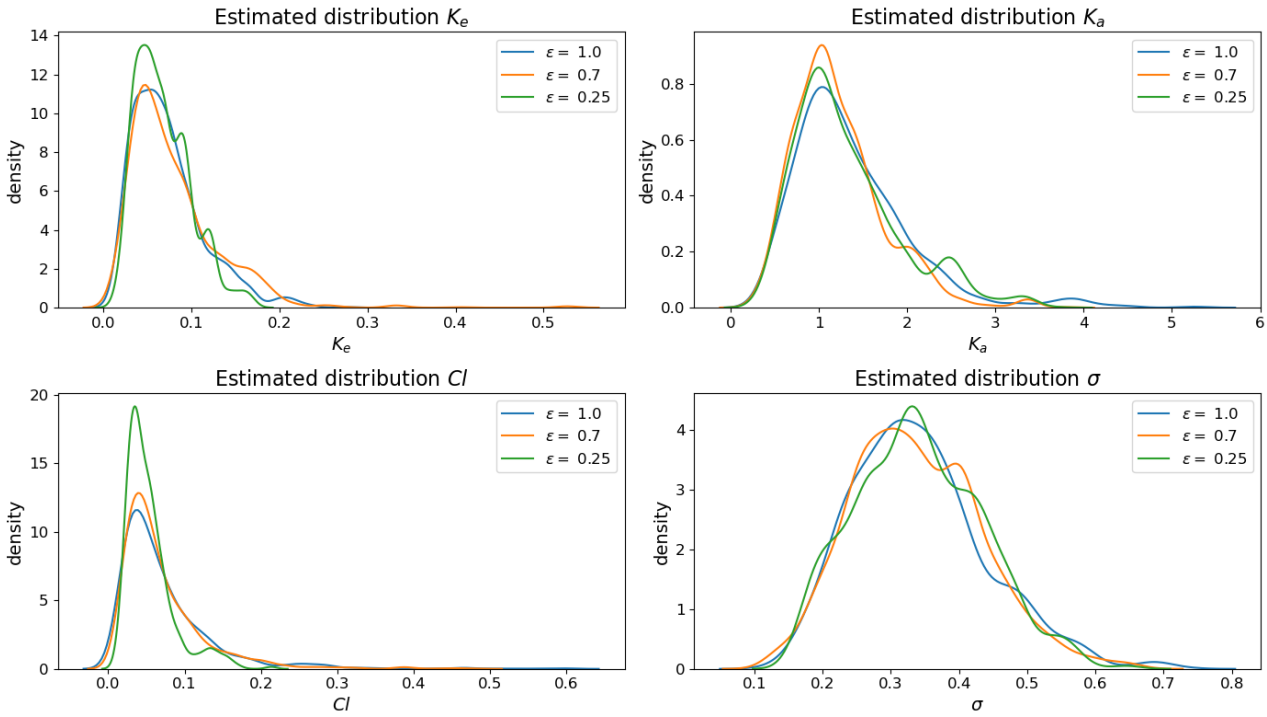

Figure 6: Marginal distributions of the parameters in θ for different values of tolerance ϵ.

Also in this case it is interesting to plot the acceptance rates in function of the tolerance ϵ.

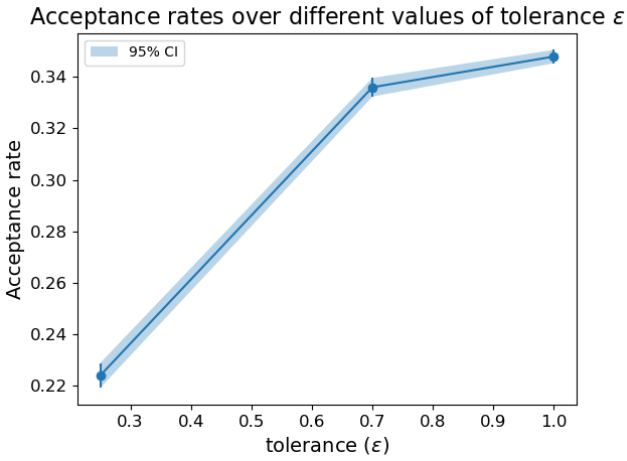

Figure 7: Acceptance rate for different values of tolerance ϵ.

As expected, from Figure 7 it is possible to observe that the acceptance rate decreases as ϵ decreases.
Further, looking at the marginal distributions in Figure 6, it is also possible to notice that almost all the marginal distributions of θ_i are more skewed for smaller ϵ. This behavior can be

tolerance	acceptance rate ABC-MCMC
1	0.347872 ± 0.002595
0.7	0.335957 ± 0.003621
0.25	0.224034 ± 0.004835

Table 2: Acceptance rates for ABC-MCMC algorithm applied to the pharmacokinetics model.

explained by looking at the definition of discrepancy metric $\rho(\mathcal{D}, \mathcal{D}^*)$ we used (see Eq. 6). As ϵ decreases in fact, the algorithm selects parameters whose simulation of X_t produces a summary statistic $S(\mathcal{D}^*)$ as close as possible to $S(\mathcal{D})$; we then expect that such parameters are as close as possible to the parameters θ that produced the data $\mathcal{D} = \{x_i\}_{i=1}^9$ and that have been used to compute the summary statistic $S(\mathcal{D})$.

Finally, we provide an estimate for the posterior mean θ^{PM} of the four parameters. We firstly observed that even initializing the parameters θ_0 in the ABC-MCMC algorithm with random values, the speed of convergence to the approximated posterior distribution was decisively fast. For this reason, since $\theta_0 = (0.07,\ 1.15,\ 0.05,\ 0.33)$ is already an "adequate" vector of parameters (i.e. $\rho(\mathcal{D}_0, \mathcal{D})$ is already low), we decided to provide the following estimate for the posterior mean θ^{PM}, considering the last 70% of parameters θ coming from the algorithm:

$$\theta_i^{PM} = \frac{1}{N} \sum_{j=floor(0.3N)}^{N} \theta_i^{(j)}, \quad i = 1, ..., 4.$$

Notice that $\theta_i^{(j)}$ is the j-th value produced by the ABC-MCMC algorithm for the i-th parameter. You can find these estimates for different values of ϵ in Table 3. Recall that the parameters θ

ϵ	K_e	K_a	Cl	σ
1	0.074563	1.234791	0.070771	0.355616
0.7	0.075951	1.238365	0.076815	0.353635
0.25	0.068989	1.252490	0.063467	0.341178

Table 3: Estimates of the posterior mean θ^{PM} for different values of ϵ.

from which $S(\mathcal{D})$ was calculated were $\theta = (0.08,\ 1.5,\ 0.04,\ 0.2)$ and that the starting point of the algorithm was $\theta = (0.07,\ 1.15,\ 0.05,\ 0.33)$.

4.1 Monte Carlo estimator for $\mathbb{E}[X_9]$

Using the posterior mean computed at the previous point in the Model 3, the goal was to propose an efficient Monte Carlo estimator to estimate $\mathbb{E}[X_9]$, the expected concentration of Theophylline after 12 hours.

Crude Monte Carlo

We first computed the estimate of $\mu = \mathbb{E}[X_9]$ using the Crude Monte Carlo approach, that consists in generating N *i.i.d.* replicas $X_9^{(1)}, ..., X_9^{(N)}$ and then computing

$$\hat{\mu}_{CMC} = \frac{1}{N} \sum_{i=1}^{N} X_9^{(i)}. \tag{7}$$

As seen in the lectures, by the Central Limit Theorem

$$|\mu - \hat{\mu}_{CMC}| \leq c_{1-\alpha/2}\frac{\hat{\sigma}_N}{\sqrt{N}},$$

with probability $1-\alpha$ and asymptotically as $N \to \infty$. Recall also that $c_{1-\alpha/2}$ is the $1-\alpha/2$ quantile of the standard normal distribution satisfying $\Phi(c_{1-\alpha/2}) = 1-\alpha/2$ and $\hat{\sigma}_N$ is the sample variance estimator computed using the same sample $(X_9^{(1)}, ..., X_9^{(N)})$,

$$\hat{\sigma}_N^2 = \frac{1}{N-1}\sum_{i=1}^{N}\left(X_9^{(i)} - \hat{\mu}_{CMC}\right)^2.$$

Using the Crude Monte Carlo method for estimating $\mathbb{E}[X_9]$, i.e. the expected concentration of Theophylline after 12 hours we obtain the following results:

ϵ	$\hat{\mu}_{CMC}$	$Var(\hat{\mu}_{CMC})$
1	1.734999	0.666171
0.7	1.640881	0.687781
0.25	1.932543	0.683048

Table 4: Estimate of $\mu = \mathbb{E}[X_9]$ Crude Monte Carlo and variance of the estimate.

Variance reduction with Antithetic Variables

The Crude Monte Carlo approach presents some drawbacks. One of them is given by the relatively high variance of the obtained estimate. Monte Carlo variance can be lowered by various variance reduction techniques, such as *Antithetic Variables*.
Suppose N even. Instead of generating N iid replicas of X_9, the idea of antithetic sampling is to generate $N/2$ iid pairs of negatively correlated random variables

$$(X_9^{(1)}, X_{9,AV}^{(1)}), ..., (X_9^{(N/2)}, X_{9,AV}^{(N/2)}),$$

where all $X_9^{(i)}, X_{9,AV}^{(i)}$ have the same distribution as X_9 but $Cov(X_9^{(i)}, X_{9,AV}^{(i)}) < 0,\ i = 0, ..., N/2$.
If we now consider the estimator

$$\hat{\mu}_{AV} = \frac{1}{N/2}\sum_{i=1}^{N/2}\frac{X_9^{(i)} + X_{9,AV}^{(i)}}{2}, \tag{8}$$

it follows that

$$\mathbb{E}[\hat{\mu}_{AV}] = \mathbb{E}[X_9] = \mu$$

and

$$Var(\hat{\mu}_{AV}) = \frac{Var(X_9) + Cov(X_9^{(i)}, X_{9,AV}^{(i)})}{N} < Var(\hat{\mu}_{CMC}).$$

In the framework of the pharmacokinetic model of Eq. 3, we can generate pairs of negatively correlated variables $(X_9^{(i)}, X_{9,AV}^{(i)})$ by simply changing the sign of the stochastic term of the model. Hence,

$$dX_{t,AV} = \left(\frac{DK_aK_e}{Cl}e^{-K_at} - K_eX_{t,AV}\right)dt - \sigma dW_t. \tag{9}$$

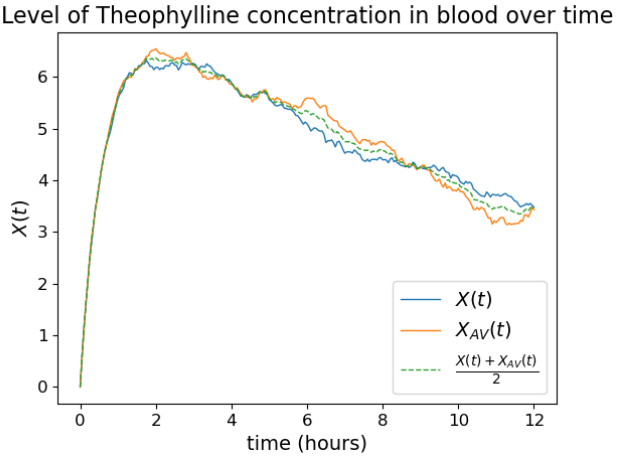

Figure 8: Example of realization of X_t, $X_{AV,t}$ and the mean between these two.

ϵ	$\hat{\mu}_{AV}$	$Var(\hat{\mu}_{AV})$
1	1.797849	0.343546
0.7	1.631523	0.341838
0.25	1.950612	0.347485

Table 5: Estimate of $\mu = \mathbb{E}[X_9]$ using Antithetic Variables and variance of the estimate.

By looking at Figure 8, we can observe an example of one realization of X_9, one of $X_{9,AV}$ and another one of the variable $\frac{X_9+X_{9,AV}}{2}$ we use in our estimate $\hat{\mu}_{AV}$. This plot already suggests that variables $\frac{X_9^{(i)}+X_{9,AV}^{(i)}}{2}$ have fewer spikes and hence they should also have smaller variance compared to the variance of X_9 used to compute $\hat{\mu}_{CMC}$.
Finally, the results in the estimation of $\mathbb{E}[X_9]$ using *Antithetic Variables* were the following: From a comparison between Table 4 and Table 5, we can observe that the variance of the Antithetic Variables estimate is almost half of the variance of the Crude Monte Carlo estimate for all the values of tolerance ϵ.

5 Conclusion

In this project, we studied the ABC-MCMC (Algorithm 2) algorithm for generating observations from the posterior distribution of a set of parameters. Firstly, we compared this algorithm with the Basic ABC through an academic example with a known posterior distribution (Section 2). Afterwards, in Section 3 we applied ABC-MCMC in a real-world model defined in Eq 3. This *likelihood-free* algorithm allowed us to approximate the posterior distribution of the parameter of the pharmacokinetics model in Eq 3, namely $\theta = (K_e,\ K_a,\ Cl,\ \sigma)$. Using the θ^* sampled by the algorithm, we then provide an estimate of the posterior mean θ^{PM}. We finally use this estimate to simulate N times the stochastic process in Eq. 3, in order to compute the expected concentration of Theophylline after 12 hours, $\mathbb{E}[X_9]$ using a *Crude Monte Carlo* estimate (see Eq. 7). Finally, we successfully reduced the variance of our Monte Carlo estimate through the implementation of the *Antithetic Variables* technique (see Eq. 8).

References

[1] S. A. Sisson, Y. Fan, and M. M. Tanaka, "Sequential monte carlo without likelihoods," *Proceedings of the National Academy of Sciences*, vol. 104, no. 6, pp. 1760–1765, 2007.

[2] U. Picchini, "Inference for sde models via approximate bayesian computation," *Journal of Computational and Graphical Statistics*, vol. 23, no. 4, p. 1080–1100, Oct. 2014.

[3] S. Sisson, Y. Fan, and M. Beaumont, *Handbook of Approximate Bayesian Computation*, ser. Chapman & Hall/CRC Handbooks of Modern Statistical Methods. CRC Press, 2018.

[4] F. Nobile, *Lecture Notes - Stochastic Simulation*. EPFL, 2023.

[5] P. Marjoram, J. Molitor, V. Plagnol, and S. Tavaré, "Markov chain monte carlo without likelihoods," *Proceedings of the National Academy of Sciences*, vol. 100, no. 26, pp. 15 324–15 328, 2003.

Appendix

Definition 1 (Strong order of convergence)**.** *A numerical method* $\{X_n\}_n$ *is said to have strong order of convergence* r*, if there exists a constant* C *such that*

$$\mathbb{E}[|X_n - X(t_n)|] \leq C(\Delta t)^r,$$

for any $t_n = n\Delta t \in [0, T]$ *and* Δt *small enough.*

Algorithm 3 Euler-Maruyama

1: Consider the SDE $dX_t = f(X_t, t)dt + g(X_t, t)DW_t$ with initial condition $X_0 = x_0$.
2: Define the time-domain $[0, T]$, the number of point N and the time step $\Delta t = \frac{T}{N}$; $t_n = n\Delta t$.
3: **for** $i = 0, ..., N-1$ **do**
4: $\quad X_{n+1} = X_n + f(X_n, t_n)\Delta t + g(X_n, t_n)\sqrt{\Delta t}\epsilon_n$, where $\epsilon_n \overset{\text{iid}}{\sim} \mathcal{N}(0, 1)$.
5: **end for**

Algorithm 4 Milstein-Platen

1: Consider the SDE $dX_t = f(X_t, t)dt + g(X_t, t)DW_t$ with initial condition $X_0 = x_0$.
2: Define the time-domain $[0, T]$, the number of point N and the time step $\Delta t = \frac{T}{N}$; $t_n = n\Delta t$.
3: **for** $i = 0, ..., N-1$ **do**
4: $\quad Z = X_n + f(X_n, t_n)\Delta t + g(X_n, t_n)\sqrt{\Delta t}$
5: $\quad X_{n+1} = X_n + f(X_n, t_n)\Delta t + g(X_n, t_n)\Delta W_n + \frac{|g(Z,t_n) - g(X_n,t_n)|}{\sqrt{\Delta t}}\left(\frac{(\Delta W_n)^2 - h}{2}\right)$,
6: $\quad$ where $\Delta W_n = W(t_{n+1}) - W(t_n) \sim \sqrt{\Delta t}\epsilon_n$, with $\epsilon_n \overset{\text{iid}}{\sim} \mathcal{N}(0, 1)$.
7: **end for**

Code

The main functions used to obtain the results of this report were the following.

```
def discrepancy_metric(x, sample_mean=0):
    '''
    Computation of discrepancy metric p = |mean(x)-sample_mean|.
    '''
    # rho(S(D*),S(D)) = abs(mean(x*) - mean(x)), we assume that mean(x)=0 (sample mean)
    return abs(np.mean(x) - sample_mean)


def true_mixture_distribution(x, M, a, var, var_1, sample_mean=0):
    '''
    Computation of true mixture distribution of the academic example.
    '''
    alfa = 1./( 1 + np.exp( a * (sample_mean - 0.5*a) * M / (M*var + var_1) ) )
    pdf_1 = alfa * st.norm.pdf(x=x, loc = var/(var+var_1/M)*sample_mean, scale = np.sqrt(var_1/(M+var_1/var)))
    pdf_2 = (1-alfa) * st.norm.pdf(x=x, loc = var/(var+var_1/M)*(sample_mean-a), scale = np.sqrt(var_1/(M+var_1/var)))
    # f = alfa * st.norm(loc = var/(var+var_1/M)*sample_mean, scale = var_1/(M+var_1/var)) + (1-alfa) * st.norm(loc = var/(var+var_1/M)*(sample_mean-a), scale = var_1/(M+var_1/var))
    f = pdf_1 + pdf_2
```

```
    return f


def basic_abc(N, M, eps, var, var_1, a):
    '''
    Implementation of basic ABC algorithm for the academic example.
    N = number of iterations
    M = data for each iteration
    eps = tolerance
    var, var_1, a = data related to the academic example
    '''
    theta = np.zeros(N) #at the end of the algorithm it has to be full (algorithm ends when we have selected N samples)
    pi_rv = st.norm(loc=0, scale=np.sqrt(var))
    rejection_count = 0
    n1_count = 0  #DEBUG
    n2_count = 0  #DEBUG
    i = 0 # count of the number of selected samples

    while i < N:
        # sample candidate parameters from the prior distribution pi = N(0, var)
        theta_star = pi_rv.rvs()

        # generate data from the underlying model given theta_star
        # D observed data, is an iid sample drawn with prob=0.5 from N(theta,var_1), o/w from N(theta+a,var_1)
        N1_rv = st.norm(loc=theta_star, scale=np.sqrt(var_1))
        N2_rv = st.norm(loc=theta_star + a, scale=np.sqrt(var_1))
        D = np.zeros(M)

        #build a set of M observation from the underlying model given theta_star
        if np.random.choice([0, 1]) == 0:
            for j in range(M):
                D[j] = N1_rv.rvs()
                n1_count += 1
        else:
            for j in range(M):
                D[j] = N2_rv.rvs()
                n2_count += 1

        if discrepancy_metric(D) < eps:
            theta[i] = theta_star
            i += 1 # a new sample is selected
        else:
            rejection_count += 1

    print('acceptance rate:', N/(N + rejection_count))
    return theta, N/(N + rejection_count)


def abc_mcmc(N_max, M, eps, nu_squared, var, var_1, a):
    '''
    Implementation of ABC MCMC algorithm for the academic example.
    N = number of iterations
    M = data for each iteration
    eps = tolerance
    nu_squared = variance of (Gaussian) transition probability
    var, var_1, a = data related to the academic example
```

```
    '''
    theta = np.zeros(N_max+1)
    pi_rv = st.norm(loc=0, scale=np.sqrt(nu_squared))  # prior

    acceptance_count = 0
    rejection_count = 0
    entered = 0
    i = 0
    metropoli_ratio_list = []

    while i < N_max:
        # sample candidate parameters from a proposal transition density q(theta_i, )
        # random walk proposal q(theta, ) = N(theta,nu_squared)
        q_i = st.norm(loc=theta[i], scale=np.sqrt(nu_squared))
        theta_star = q_i.rvs()

        # generate data from the underlying model given theta_star
        N1_rv = st.norm(loc=theta_star, scale=np.sqrt(var_1))
        N2_rv = st.norm(loc=theta_star + a, scale=np.sqrt(var_1))

        D = np.zeros(M)
        if np.random.choice([0, 1]) == 0:
            for j in range(M):
                D[j] = N1_rv.rvs()
                # n1_count = n1_count + 1
        else:
            for j in range(M):
                D[j] = N2_rv.rvs()
                # n2_count = n1_count + 1

        if discrepancy_metric(D) < eps:
            q_star = st.norm(loc=theta_star, scale=np.sqrt(nu_squared))
            comp = pi_rv.pdf(theta_star)*q_star.pdf(theta[i])/( pi_rv.pdf(theta[i])*q_i.pdf(theta_star) )
            metropoli_ratio_list.append(min(1.,comp))

            entered += 1

            if st.uniform.rvs() < comp:
                theta[i+1] = theta_star
                acceptance_count += 1
            else:
                theta[i+1] = theta[i]
                rejection_count += 1
        else:
            theta[i+1] = theta[i]
            rejection_count += 1

        i += 1

    print('acceptance rate:', acceptance_count/(acceptance_count+rejection_count))
    return theta, acceptance_count/(acceptance_count+rejection_count)


def simulate_Xt_times(K_e, K_a, Cl, sigma):
    '''
    Simulation of the pharmacokinetics model with Euler-Maruyama and selection of the 9 times we are interested in.
```

```
    Return only the value of the process at those times.
    '''
    dt = 0.05  # Time step
    T = 12  # Total time
    n = int(T / dt)+1  # Number of time steps
    D = 4

    t = np.arange(0, 12.05, 0.05)  # Vector of times

    X_t = np.zeros(n) # Recall that X_0 = 0
    for i in range(n - 1):
        X_t[i + 1] = X_t[i] + dt*( (D*K_a*K_e)/Cl * np.exp(-K_a*t[i+1]) - K_e*X_t[i] ) + sigma*np.sqrt(dt)*np.random.randn()

    times = [0.25, 0.5, 1, 2, 3.5, 5, 7, 9, 12]
    indices = np.where(np.isin(t, times))
    Data = X_t[indices]
    return Data


def simulate_Xt_full(K_e, K_a, Cl, sigma):
    '''
    Simulation of the pharmacokinetics model with Euler-Maruyama.
    Return all the times and all the values of the stochastic process X_t for each time evaluated.
    '''
    dt = 0.05  # Time step
    T = 12  # Total time
    n = int(T / dt)+1  # Number of time steps
    D = 4

    t = np.arange(0, 12.05, 0.05)  # Vector of times

    X_t = np.zeros(n) # Recall that X_0 = 0
    for i in range(n - 1):
        X_t[i + 1] = X_t[i] + dt*( (D*K_a*K_e)/Cl * np.exp(-K_a*t[i+1]) - K_e*X_t[i] ) + sigma*np.sqrt(dt)*np.random.randn()

    return t, X_t


def new_discrepancy_metric(D_star, S_D, intercept, coefficients, theta_0):
    '''
    Computation of the discrepancy metric p = ||S(D)-S(D*)||.
    '''
    S_D_star = intercept + coefficients@D_star

    diff = S_D_star - S_D
    metric = 0

    for i in range(len(diff)):
        metric += diff[i]**2/(theta_0[i])**2

    return metric


def jointly_prior_sample(prior_K_e, prior_K_a, prior_Cl, prior_sigma, theta):
    '''
    Return the jointly prior evaluated in theta assuming independence.
```

```
    '''
    # assuming independence
    return prior_K_e.pdf(theta[0])*prior_K_a.pdf(theta[1])*prior_Cl.pdf(theta[2])*prior_sigma.pdf(theta[3])


def abc_mcmc_pharma(N_max, M, eps, S_D, intercept, coefficients):
    '''
    Implementation of ABC-MCMC algorithm for the pharmacokinetics model.
    N_max = maximum number of iterations
    M = number of data at each iteration
    eps = tolerance
    S_D = summary statistic of the data of point 4
    intercept = beta_0 obtained in point 4
    coefficients = beta_i, i=1,...,9 obtained in point 4
    '''
    theta = np.zeros((N_max + 1, 4))
    theta_0 = [0.07, 1.15, 0.05, 0.33]
    #theta_0 = [5., 5.0, 5.0, 5.0]
    theta[0,:] = theta_0

    prior_K_e = st.lognorm(s=0.6, scale=np.exp(-2.7))
    prior_K_a = st.lognorm(s=0.4, scale=np.exp(0.14))
    prior_Cl = st.lognorm(s=0.8, scale=np.exp(-3))
    prior_sigma = st.lognorm(s=0.3, scale=np.exp(-1.1))

    acceptance_count = 0
    rejection_count = 0
    entered = 0
    i = 0

    while i < N_max:
        # sample candidate parameters from a proposal transition density q(theta_i, )
        # anisotropic approach

        q_i_list = []
        theta_star_list = []
        s_list = [0.6, 0.4, 0.8, 0.3]
        for j in range(4):
            q_i = st.lognorm(s=s_list[j], scale=theta[i,j]) #lognormal 4 dimensional with different scale and same variance s
            q_i_list.append(q_i)
            theta_star=q_i.rvs() #sample from the lognormal
            theta_star_list.append(theta_star)

        # generate data from the underlying model given theta_star
        K_e = theta_star_list[0]
        K_a = theta_star_list[1]
        Cl = theta_star_list[2]
        sigma = theta_star_list[3]

        D_star = simulate_Xt_times(K_e, K_a, Cl, sigma) #generate data from underlying model given theta star

        if new_discrepancy_metric(D_star, S_D, intercept, coefficients, theta_0) < eps:

            prior_K_e_star = st.lognorm(s=s_list[0], scale=theta_star_list[0])
```

```
            prior_K_a_star = st.lognorm(s=s_list[1], scale=theta_star_list
    [1])
            prior_Cl_star = st.lognorm(s=s_list[2], scale=theta_star_list
    [2])
            prior_sigma_star = st.lognorm(s=s_list[3], scale=theta_star_list
    [3])

            q_star_pdf = jointly_prior_sample(prior_K_e_star, prior_K_a_star
    , prior_Cl_star, prior_sigma_star, theta[i,:])
            q_i_pdf = jointly_prior_sample(q_i_list[0], q_i_list[1],
    q_i_list[2], q_i_list[3], theta_star_list)
            pi_star = jointly_prior_sample(prior_K_e, prior_K_a, prior_Cl,
    prior_sigma, theta_star_list)
            pi_i = jointly_prior_sample(prior_K_e, prior_K_a, prior_Cl,
    prior_sigma, theta[i,:])

            comp = pi_star * q_star_pdf/(pi_i * q_i_pdf)
            comp = min(1,comp)

            #comp=0.85
            entered += 1

            if st.uniform.rvs() < comp:
                theta[i+1,:] = theta_star_list
                acceptance_count += 1
            else:
                theta[i+1,:] = theta[i,:]
                rejection_count += 1
        else:
            theta[i+1,:] = theta[i,:]
            rejection_count += 1

        i += 1

    print('acceptance rate:', acceptance_count/(acceptance_count+
    rejection_count))
    return theta, acceptance_count/(acceptance_count+rejection_count)


def simulate_Xt_times_AV(K_e, K_a, Cl, sigma):
    '''
    Simulation of the ANTITHETIC pharmacokinetics model with Euler-Maruyama.
    Return only the values values of the stochastic process X_t for the
    times we are interested in.
    '''
    dt = 0.05  # Time step
    T = 12  # Total time
    n = int(T / dt)+1  # Number of time steps
    D = 4

    t = np.arange(0, 12.05, 0.05)  # Vector of times

    X_t = np.zeros(n) # Recall that X_0 = 0
    for i in range(n - 1):
        X_t[i + 1] = X_t[i] + dt*( (D*K_a*K_e)/Cl * np.exp(-K_a*t[i+1]) -
    K_e*X_t[i] ) - sigma*np.sqrt(dt)*np.random.randn()

    times = [0.25, 0.5, 1, 2, 3.5, 5, 7, 9, 12]
    indices = np.where(np.isin(t, times))
    Data = X_t[indices]
```

```
    return Data


def simulate_Xt_times_AV_full(K_e, K_a, Cl, sigma):
    '''
    Simulation of the ANTITHETIC pharmacokinetics model with Euler-Maruyama.
    Return all the times and all the values of the stochastic process X_t
    for each time evaluated.
    '''
    dt = 0.05  # Time step
    T = 12  # Total time
    n = int(T / dt)+1  # Number of time steps
    D = 4

    t = np.arange(0, 12.05, 0.05)  # Vector of times

    X_t = np.zeros(n) # Recall that X_0 = 0
    for i in range(n - 1):
        X_t[i + 1] = X_t[i] + dt*( (D*K_a*K_e)/Cl * np.exp(-K_a*t[i+1]) -
    K_e*X_t[i] ) - sigma*np.sqrt(dt)*np.random.randn()

    return t, X_t
```

Listing 1: Python implementation of the algorithms used in this project.