Project Report

Speeding up Markov Chain Monte Carlo without Likelihood

Jonas Helfer

Supervised by

Daniel Wegmann and Bernard Moret

June 2010

1 Disclaimer

The purpose of this report is to present the progress made and the effort undertaken during one semester. The amount of time available to students at EPFL for an optional semester project (8 ECTS) is very limited, but it allows the student to participate in and contribute to an ongoing project. This is also the case here. Therefore, you are now reading what can be considered an intermediate report, giving an understanding of the background, presenting the results achieved in the course of one semester and offering a brief outlook on the future of the project.

Contents

2 Introduction

2.1 A biref digression on Bioinformatics

For centuries, biology was a so-called *descriptive science*, aiming to *describe* the existing diversity in nature, as opposed to physics, whose aim it is to *explain* observations by finding the underlying rules. In the 19th century, Darwin's theory of evolution hinted for the first time at a future in which Biology would no longer only describe, but also offer explanations for the observed at the level of Chemistry or Physics. In the 20th century, the development of electron microscopy, radiocristallography and a whole bunch of other techniques allowing to study nature at a much smaller level, opened a broad range of new possibilites for Biology in explaining the observed. While the "traditional" Biogloy continued to exist, this new field developped rapidly and revolutionized the study of nature. Because these new methods were very expensive and rather slow, data was scarce and biologists would use their ingenuity to make up for the lack of it. Slowly but steadily, the old techniques were refined and new ones developed, which made gathering data easier and easier (so-called high-throughput methods, eg shotgun-sequencing). So much easier in fact, that the amount and detail of data generated could no longer be analyzed by classical means, which was to have a biologist look at it and draw the right conclusions. A new type of Scientist has emerged, trying to cover this area. No longer staring through a microscope but sitting behind a screen, he uses the exponentially growing power of computers and algorithms to find the sunken treasures in oceans of data available. But with all the possibilites that computers offer, a vast scope of entirely new problems has arisen: More often than not, the Biologist knows exactly what he needs, but the computer scientist has to tell him that no computer in the world is fast enough to calculate it, because that particular problem is simply 'too hard to solve' (NP-hard). Now the scientist is left with two options, which are usually closely connected: 1. formulate the problem in such a way that it can be solved more efficiently, or 2. find an algorithm which can solve the original problem more efficiently. This opens up a new 'frontier' in Biology: the design of efficient algorithms to solve the problems nature has posed to us.

In this project, we will implement a new method for Approximate Bayesian Computation without likelihoods, which is a solution to one particular kind of problem along this new 'frontier'. We hope to improve the ability to infer parameters from available information (data) when faced with analytically intractable models.

2.2 Bayesian Statistics

The following introduction to Approximate Bayesian Computation is very short and should not be used as a general introduction to the subject, its goal is simply to recall the basics and introduce a common terminology. For further detail, the reader is referred to the original papers.

With the advent of ever more powerful computers and the refinement of algorithms like Markov Chain Monte Carlo (MCMC) or Gibbs sampling, Bayesian statistics has become an important tool for scientific inference during the past two decades.

A brief recall of essential set theory:

$$\mathbb{P}(A|B) := \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)} \tag{1}$$

From this, the Bayesian Theorem follows directly:

$$\mathbb{P}(A|B) := \frac{\mathbb{P}(B|A)\mathbb{P}(A)}{\mathbb{P}(B)} \tag{2}$$

This formula can in turn be used to calculate the probablility of a model's parameters $\boldsymbol{\theta}$ given a set of data $\mathcal{D}$:

$$\pi(\boldsymbol{\theta}|\mathcal{D}) = c \cdot f_{\mathcal{M}}(\mathcal{D}|\boldsymbol{\theta})\pi(\boldsymbol{\theta}), \tag{3}$$

where c is a normalizing constant and $f_{\mathcal{M}}(\mathcal{D}|\boldsymbol{\theta})$ is the so called likelihood of the data $\mathcal{D}$, $\pi(\boldsymbol{\theta})$ the joint prior density and $\pi(\boldsymbol{\theta}|\mathcal{D})$ the posterior probability density. The posterior tells us, which parameter values are likely to generate the observed data.

It's important to notice that the crucial difference to the fairly common maximum likelihood method is the fact that the *a priori* density of the parameters is no longer ignored. This is important especially when reliable prior knowledge about certain parameters exists. The above formula thus expresses the intuitive concept that our prior knowledge of the parameters (called *a priori*) is influenced by the Data $\mathcal{D}$ to give rise to our new *a posteriori* estimate.

While the probability theory to this is straight forward, a direct application of this formula is very often made impossible due to the fact that the likelihood function cannot be calculated analytically. Even in the case of a traceable function, it is not always possible to integrate the likelihood analytically, which is a necessary step in obtaining the normalizing constant c.

A smart workaround for this can be used, when the likelihood ratios (the ratio of likelihoods for two different parameter vectors) can somehow be calculated. In this case, one may probe the posterior distributions with Markov Chain Monte Carlo (MCMC) methods. These methods are often used to numercially solve multi-dimensional integrals in physics.

The Metropolis-Hastings algorithm is a widely used MCMC method. The algorithm uses a Markov chain whose transition depends only on the current state $\boldsymbol{\theta}$. It proposes a move $q(\boldsymbol{\theta} \rightarrow \boldsymbol{\theta})'$ to a new state $\boldsymbol{\theta}'$, which is accepted with probability

$$h(\boldsymbol{\theta} \rightarrow \boldsymbol{\theta}') = \min\left(1, \frac{\mathbb{P}(\mathcal{D}|\boldsymbol{\theta}')\pi(\boldsymbol{\theta}')q(\boldsymbol{\theta}' \rightarrow \boldsymbol{\theta})}{\mathbb{P}(\mathcal{D}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})q(\boldsymbol{\theta} \rightarrow \boldsymbol{\theta}')}\right) \tag{4}$$

If the move is rejected, the algorithm remains at $\boldsymbol{\theta}$. This makes sure that the static distribution of the chain converges to the posterior likelihood.

2.3 Approximate Bayesian Computation

If even the likelihood ratio cannot be calculated analytically (sadly, this is the case for many models in Biology), MCMC can no longer be used and one needs to revert to somewhat cruder methods. Tavare et al. [4] proposed a rejection sampling method in which the posterior is obtained through repeated simulations, retaining only parameter assignments, for which the simulated data corre-

sponds to the data observed in nature. To simplify the equality between simulation and observations, Tavare et al [4] proposed the use of summary statistics. There are two important reasons for using summary statistics: Firstly, it can be prohibitive, if not impossible, to define and calculate a distance measurement over the complete data. Secondly, it is often not necessary to evaluate all of the information available in the data, but only a quantitative summary of it, which is relevant to the problem at hand. Thirdly, the larger and finer grained the data is, the unlikelier it is to obtain exactly the observed values when using a rejection sampling method, thus making it either a very lengthy or inaccurate process. This problem is known in other instances as the 'curse of dimensionality'. The more dimensions our data-space has, the more difficult it becomes to get close to a specific point in space. A well designed set of summary statistics can reduce the dimensionality of the problem while keeping all the important information. The choice of summary statistics is all but trivial. Several statistical methods can be used to find suitable summary statistics, but intuition also plays an important role [5].

Many problems however, are so complex, that even with the use of summary statistics it is nearly impossible to obtain exactly the observed values after a simulation. To alleviate this problem, one may use a threshold distance (or tolerance) δ_ϵ, accepting simulations if the condition $\text{dist}(\mathbf{s}, \mathbf{s}_{obs}) < \delta_\epsilon$ is satisfied. Still, with an increasing number of dimensions, it rapidly becomes more difficult to fall within a range of δ_ϵ from the observed data. To increase the acceptance rate, the tolerance must be increased, which reduces the accuracy of the method. If the chosen tolerance is too large, the posterior distribution will be dominated by the prior distribution [2]. Minor distortions of the posterior distribution can be corrected in part by post-sampling adjustment [1].

2.3.1 MCMC without likelihood

In 2003 Marjoram et al [3] proposed a variant of Metropolis-Hastings which does not require the likelihood ratio to be known, and were able to prove that that the stationary distribution of the Markov Chain converges to the correct posterior distribution.

The 'MCMC without likelihood' algorithm is defined by the following steps:

MCMC1 If now at $\boldsymbol{\theta}$, propose a move to $\boldsymbol{\theta}'$ according to a transition kernel $q(\boldsymbol{\theta} \rightarrow \boldsymbol{\theta}')$.

MCMC2 Simulate $\mathcal{D}'$ using θ_i.

MCMC3 Calculate summary statistics s' on $\mathcal{D}'$.

MCMC4 If $\mathrm{dist}(\mathrm{s}, \mathrm{s}_{obs}) < \delta_\epsilon$ go to 5, otherwise stay at $\boldsymbol{\theta}$ and go to 1.

MCMC5 Accept $\boldsymbol{\theta}'$ with probability

$$h(\boldsymbol{\theta} \rightarrow \boldsymbol{\theta}') = \min\left(1, \frac{\pi(\boldsymbol{\theta}')q(\boldsymbol{\theta}' \rightarrow \boldsymbol{\theta})}{\pi(\boldsymbol{\theta})q(\boldsymbol{\theta} \rightarrow \boldsymbol{\theta}')}\right), \text{ otherwise remain at } \boldsymbol{\theta}. \text{ Go to 1.}$$

The main difference to standard MCMC is, that the transition Kernel does no longer need the likelihood ratio, as was the case in equation 4.

The accuracy of MCMC without likelihood is very sensitive to δ_ϵ and the transition kernel $q(\boldsymbol{\theta} \rightarrow \boldsymbol{\theta}')$, therefore their choice is crucial. Wegmann et al [5] proposed to use an initial set of 10000 simulations for calibration, where parameters are randomly sampled from the prior. One can then calculate δ_ϵ such that a specified fraction of the calibration simulations is accepted. As for the transition kernel, one possible way to find the next proposed parameter vector is is to sample it from a uniform distribution with width φ (expressed in units of standard deviations of the accepted simulations for each parameter). In the ABCtoolbox, φ is specified with the parameter rangeProp. Throughout our experiments we used a value of 0.5 for rangeProp as proposed in [5].

2.3.2 Post-sampling adjustment

A way to increase the precision and try to correct for the bias introduced with large tolerance values is post sampling adjustment [1]. Post sampling adjustment supposes a linear relationship between parameters and their summary statistics. For instance, Wegmann and Leuenberger [2] supposed the likelihood to be a general linear model, for which the parameters can be obtained using standard regression techniques. While this does not hold globally, a linear model can often be used as a good approximation of the values of summary statistics in the narrow range accepted by the sampling method.

2.3.3 SuffStat MCMC without likelihood

In some models, many parameters are correlated with a subset of summary statistics only, meaning that a part of the data is not affected by certain parameters. If all parameters and their corresponding subsets of summary statistics can be isolated (complete independence not being a requirement), it should be possible to run likelihood free samplers with improved speed and/or accuracy thanks to this information.

Such a method (called SuffStat MCMC), along with a mathematical proof, has been proposed by Leuenberger and Wegmann [in preparation]. In this project, we will extend the existing open source package ABCtoolbox [6] with this algorithm, run verification and performance tests on a toy model and compare it to the standard MCMC approach.

3 Materials and Methods

In this section we present the SuffStat MCMC aglorithm, the ABCtoolbox for which we made the implementation, the General Linear model we used to generate our simulated data, features of the implementation and the tests we used to verify the algorithm worked as expected.

3.1 SuffStat MCMC Algorithm

We assume a model $\mathcal{M}$ depending on n parameters $\boldsymbol{\theta}$ and creating m-dimensional data (summary statistics). Let the random variable $T_i = T_i(\mathcal{S})$ be an m_i-dimensional function of $\mathcal{S}$. We call T_i sufficient for the parameter $\boldsymbol{\theta}_i$ if the conditional distribution of $\mathcal{S}$ given T_i does not depend on the i-th component of $\boldsymbol{\theta}$. We may then use the following algorithm to obtain the posterior density:

SUFST1 Choose an index $i = 1, \ldots, n$ according to a probability distribution $(p_1, \ldots, p_n)$

SUFST2 If now at $\boldsymbol{\theta}$, propose a move to $\boldsymbol{\theta}'$ according to a transition kernel $q(\boldsymbol{\theta} \rightarrow \boldsymbol{\theta}')$ where $\boldsymbol{\theta}'$ differs from $\boldsymbol{\theta}$ only in the i-th component: $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_{i-1}, \boldsymbol{\theta}'_i, \boldsymbol{\theta}_{i+1}, \ldots, \boldsymbol{\theta}_n)$.

SUFST3 Simulate $\mathcal{D}'$ using $\boldsymbol{\theta}'$.

SUFST4 Calculate summary statistics s' on $\mathcal{D}'$.

SUFST5 If $\mathrm{dist}(\mathrm{s}', \mathrm{s}_{obs}) < \delta_\epsilon$ go to 6, otherwise stay at $\boldsymbol{\theta}$ and go to 1.

SUFST6 Accept $\boldsymbol{\theta}'$ with probability

$$h(\boldsymbol{\theta} \rightarrow \boldsymbol{\theta}') = \min\left(1, \frac{\pi(\boldsymbol{\theta}')q(\boldsymbol{\theta}' \rightarrow \boldsymbol{\theta})}{\pi(\boldsymbol{\theta})q(\boldsymbol{\theta} \rightarrow \boldsymbol{\theta}')}\right),$$ otherwise remain at $\boldsymbol{\theta}$. Go to 1.

3.2 Implementation

ABCtoolbox was designed to perform Approximate Bayesian Computation (ABC) estimations using various recently published algorithms including MCMC without likelihood and Population Monte Carlo. Due to its potential to interact with almost any command line simulation software, ABCtoolbox can be used to study problems in different areas including genomics or population genetics. [6]

The ABCtoolbox is an open source toolkit written in C++ and published by the population genetics laboratory of the University of Berne, Switzerland [6]. It consists of two main programs: ABCsampler is used to sample from the prior distribution, run simulations and calculate summary statistics, while ABCestimator calculates marginal posterior distributions of parameters with or without post-sampling adjustment. It can be fully controlled by input files or the command-line, which makes it suitable for use on clusters or grid computers.

We used the ABCtoolbox, in particular ABCsampler, as a framework to implement the SuffStat MCMC algorithm, because it offers an open source implementation of other algorithms such as rejection sampling [4] and MCMC without likelihood [3], thereby simplifying the amount of work necessary for the implementation. Any input, output and file handling needed could be done by existing parts of the toolbox, allowing us to concentrate on the implementation of the actual algorithm and its verification. Furthermore, we believe, that the implementation of SuffStat MCMC is easier to use if it fits into an existing framework. The ABCtoolbox has a good documentation, making it easy for other Biologists or computer scientists to understand and use the software. By melding our code into the existing frame, we can spare the effort of writing a separate manual.

Our implementation of SuffStat MCMC requires the user to define the relationship between parameters and statistics for the program to infer the independent components, also called blocks. We chose to model these relationships as linear combinations, because this makes them easy to handle in software. Furthermore, they allow SuffStat MCMC to directly use the results of a partial least squares (PLS) analysis, which is one possible way of finding the independent components. An example of such a file (paramA_file.pls) can be found in the Appendix.

3.3 Model

To test our implementation, we need a fast model and the ability to predict the outcome very accurately. General Linear Models (abbreviated as GLM – not to be confused with the Generalized Linear Models, with which it shares its abbreviation) fulfil both of these requirements. There is a fast built-in GLM module in the ABCtoolbox and we can precisely define which summary statistics

are independent and therefore predict the outcome accurately.

General Linear Models are defined as follows:

$$\mathbf{s}|\boldsymbol{\theta} = \mathbf{C}\boldsymbol{\theta} + \mathbf{c}_0 + \boldsymbol{\epsilon}, \tag{5}$$

where $\mathbf{s}|\boldsymbol{\theta}$ is the vector of summary statistics given the parameter vector, $\mathbf{C}$ is a $n \times m$-matrix of constants, $\mathbf{c}_0$ a $n \times 1$-vector and $\boldsymbol{\epsilon}$ a random vector with a multivariate normal distribution of zero mean and covariance matrix $\boldsymbol{\Sigma}_s$: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}_s)$.

Throughout our experiments, we used the GLM parameters as defined in the file matrix.txt (cf Annex).

3.4 Validation and Comparison

Once the code compiled without errors, we carried out two basic tests to verify the correctness of the algorithm's implementation.

Firstly, we made sure the algorithm converged and determined the length of chain necessary with different settings. Secondly, we checked if the posterior density was correct. We also compared the convergence and posterior to a normal MCMC with the same length of chain and tolerance

3.4.1 Convergence

To test convergence, we ran 10 trials with chains of a length of 10M and 5M and 20 trials with a length of 2.5M. We then plotted the 0.01, 0.05, 0.1, 0.25, 0.5, 0.75, 0.9 , 0.95 and 0.99 quantiles of the parameter values to see if and after how many iterations the chain would converge. Once the chain has converged, we expect the quantiles to stay almost level. To calculate the quantiles, we used the free statistics software R.

3.5 Posterior distribution

To test whether the SuffStat MCMC does indeed correctly sample the posterior distribution, we ran chains of a length which we had determined to be suitable in the previous step. We did this for 1000 (resp 500) pseudo-observed values randomly sampled from the prior. We know that the posterior density function $\mathcal{P}$ is correct, if $\int_{-\infty}^{\mathbf{s}_{obs}} \mathcal{P}(\mathbf{s})dx$ is a uniform distribution over $\mathbf{s}_{obs}$. To verify this fact, we simply counted for each trial the number of times where the chain was below the true value (recall: the stationary distribution of the chain is the posterior). We call this the 'thresholded count'. If it is not uniform, we know that either the chain is too short or the implementation faulty. To check for uniformity, we used a kolmogorov-smirnov test and plotted the estimated density and a histogram for visualization.

The bash scripts used to run the trials and the R scripts used to plot the quantiles and densities can be found in the annex.

4 Results

We show only a fraction of the graphs produced in this project and describe only some of the tests conducted after initial debugging.

4.1 Convergence

The 10 trials with chains of length 10M took a total of about 16 hours at full CPU (on an Intel® CoreTM 2 Duo Processor T5600 (1.83GHz, 2MB L2 Cache, 667MHz FSB)). The tests showed that convergence was usually achieved with a chain of length smaller than 500000. After correcting some minor problems, we ran the simulations on chains of 5M length, which took about 8 hours at full CPU and indicated that a chain of about 100000 to 200000 samples should suffice to achieve convergence, depending on the tolerance. Finally we ran 20 tests of chains of 2.5M with a tolerance of 0.05, which took around 8 hours. One of the runs is depicted in figure 1. The figure shows that especially the middle quantiles converge very quickly, and after about 500K, the chain has essentially converged.

4.2 Posterior distribution

Given the results of the convergence tests, we started simulating 1000 chains of length 100'000. We observed that chains of this length did not cover the posterior well: they resulted in a non-uniform distribution of the 'threshold count'. An attempt with 1000 chains of a length of 100K and a tolerance of 0.1 is shown in figure 2 (we aborted after 800, because the distribution was clearly not uniform). We used an epanechnikov kernel to plot the probability density. A Kolmogorov-Smirnov test for resemblance to a uniform distribution gave p-values of 0.1384645, 0.02378449 and 0.3659787, which is much too low, especially for the second parameter. A quick look at the density and the histogram confirm the suspicion, that the borders are not well explored, leading to a "peak" in the middle. We then ran a longer and longer chains hoping to get closer to the expected uniform distribution.

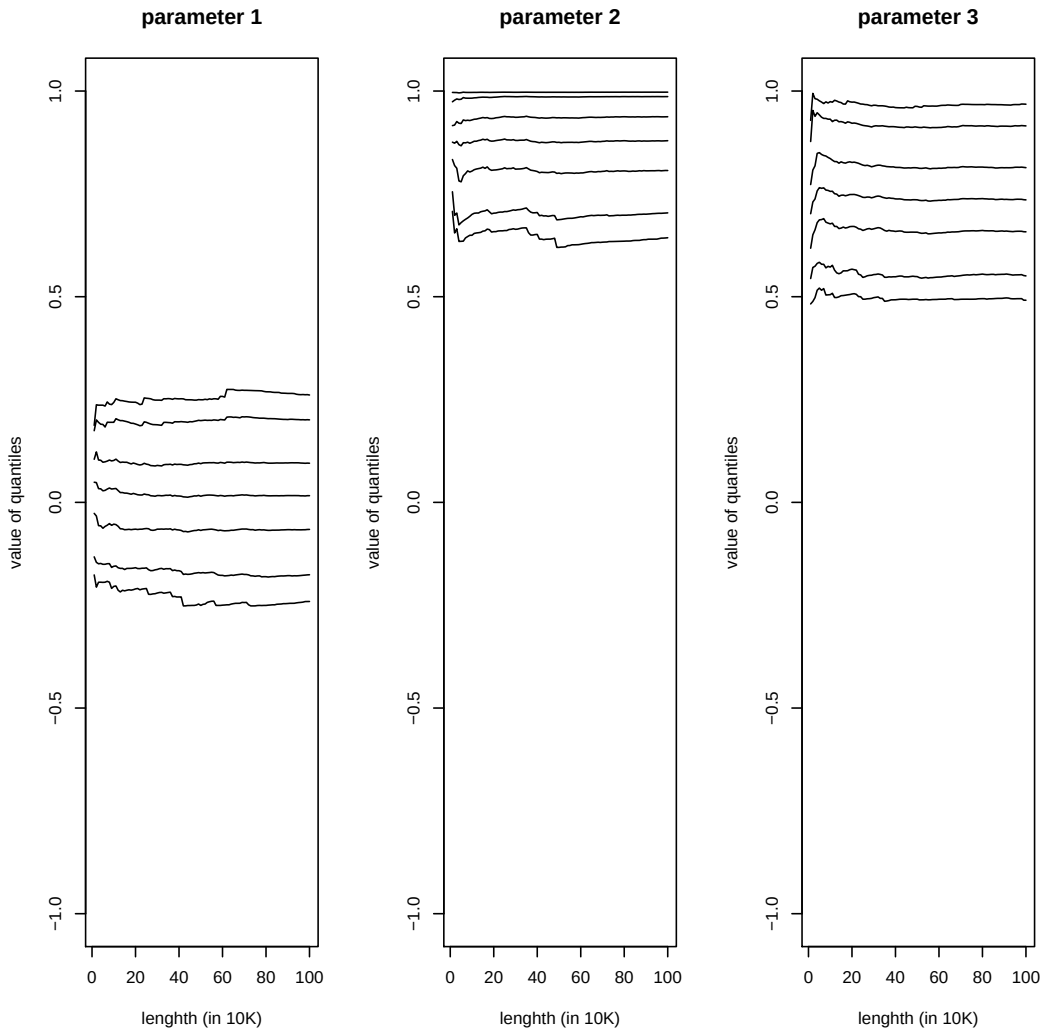

Figure 1: Convergence of a chain cut to length 1M, using 10000 calibration simulations, sampled from 3 uniform priors between -1 and +1. Tolerance 5%. The quantiles were plotted every 10000 steps.

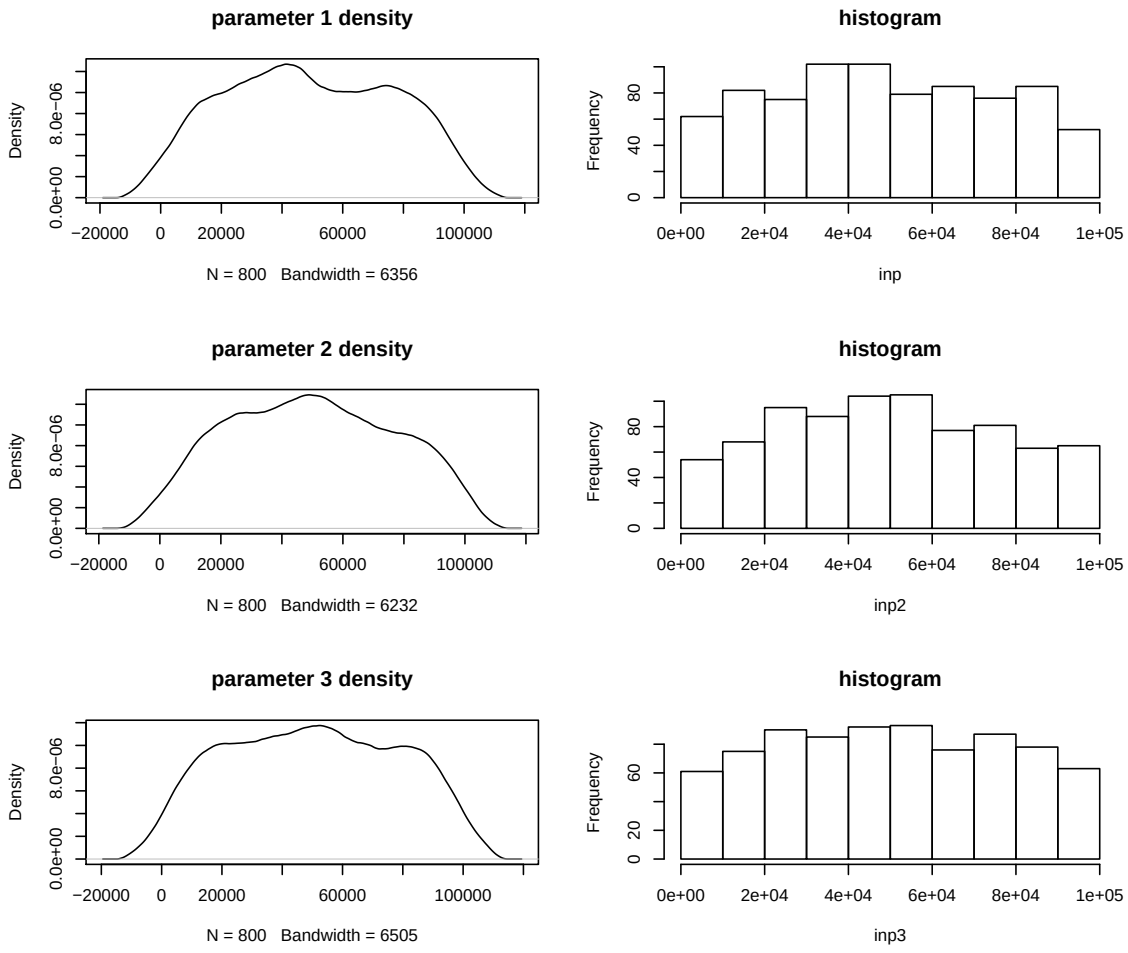

Figure 2: SuffStat MCMC, Density and histogram of the 'threshold count' of 800 chains of length 100K using 1000 calibration simulations sampled from 3 uniform priors between -1 and +1. Tolerance 10%

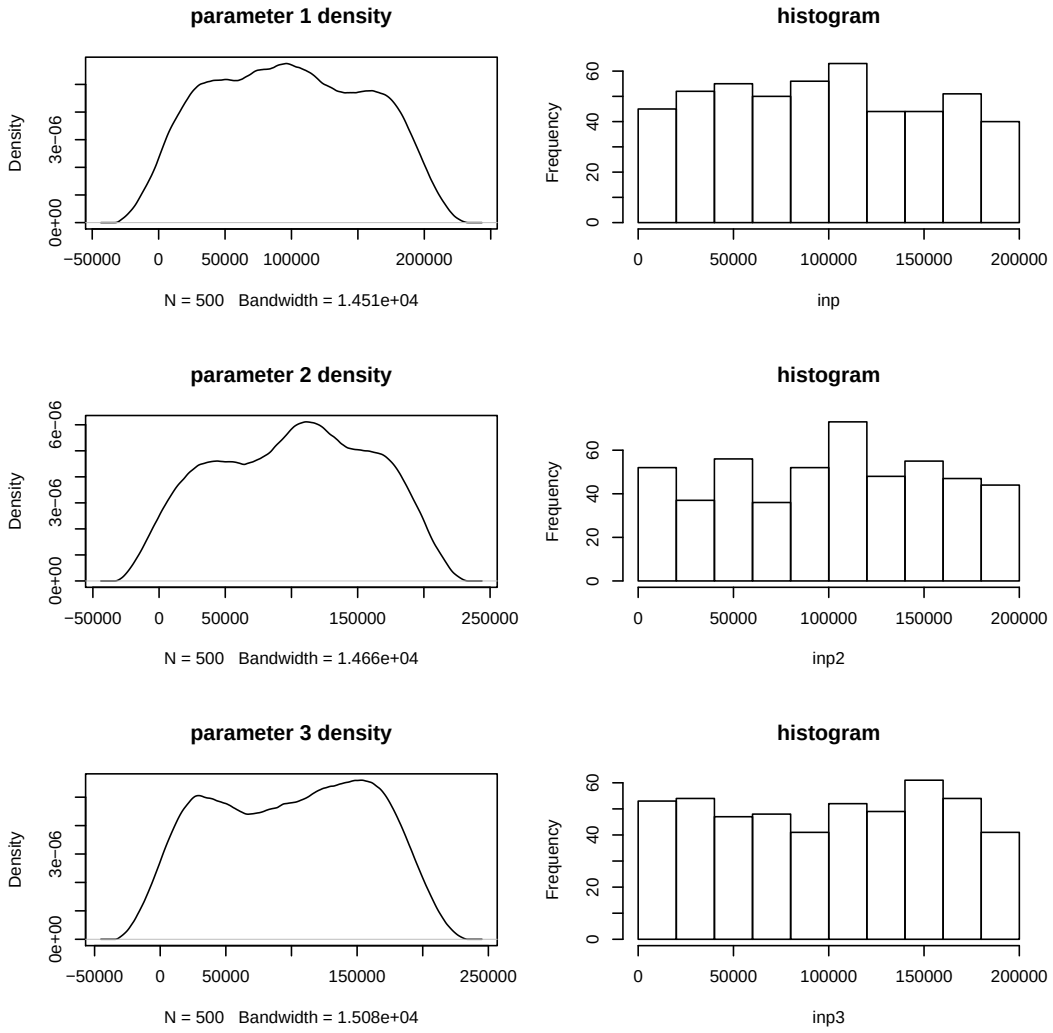

Figure 3: SuffStat MCMC, Density and histogram of the 'threshold count' of 500 chains of length 200K using 10000 calibration simulations, uniform priors in [-1, +1]. Tolerance 10%

After numerous attempts with other lengths and tolerances, we ran 500 trials with a length of 200K and a tolerance of 5%. These runs are depicted in figure 3. The p-values were all reasonably high: 0.3295684, 0.08976615 and 0.8292563.

4.3 Comparison with MCMC

The result of running an MCMC with the same parameters as in the last comparison (chain length 200K, tolerance of 5% and RangeProp of 0.5) is shown in figure 4. The fact that it doesn't consider parameters independently leads to a distorted posterior distribution. The p-values of the

Kolmogorov-Smirnov test were smaller than 10^{-10} for all three parameters. To make sure that this was not a mistake in the MCMC implementation, we ran the results through the ABCestimator to do a post-sampling adjustment. After the post-sampling adjustment, the distribution was indeed close to uniform (cf. figure 5). This can be explained by the fact that while the three parameters don't vary independently during the MCMC, the regression is calculated separately for each parameter and can therefore make individual corrections.

Figure 5 shows that the distribution is much closer to uniform after the adjustment step with ABCestimator. The pointed shape of the curve is due to the fact that standard MCMC without likelihood leads to posteriors which are too broad. It cannot take into account the independence of certain parameters and summary statistics, which can give rise to a significant distortion. The result makes a convincing case for SuffStat MCMC, which does not suffer from this disadvantage and gives a posterior very close to the expected one.

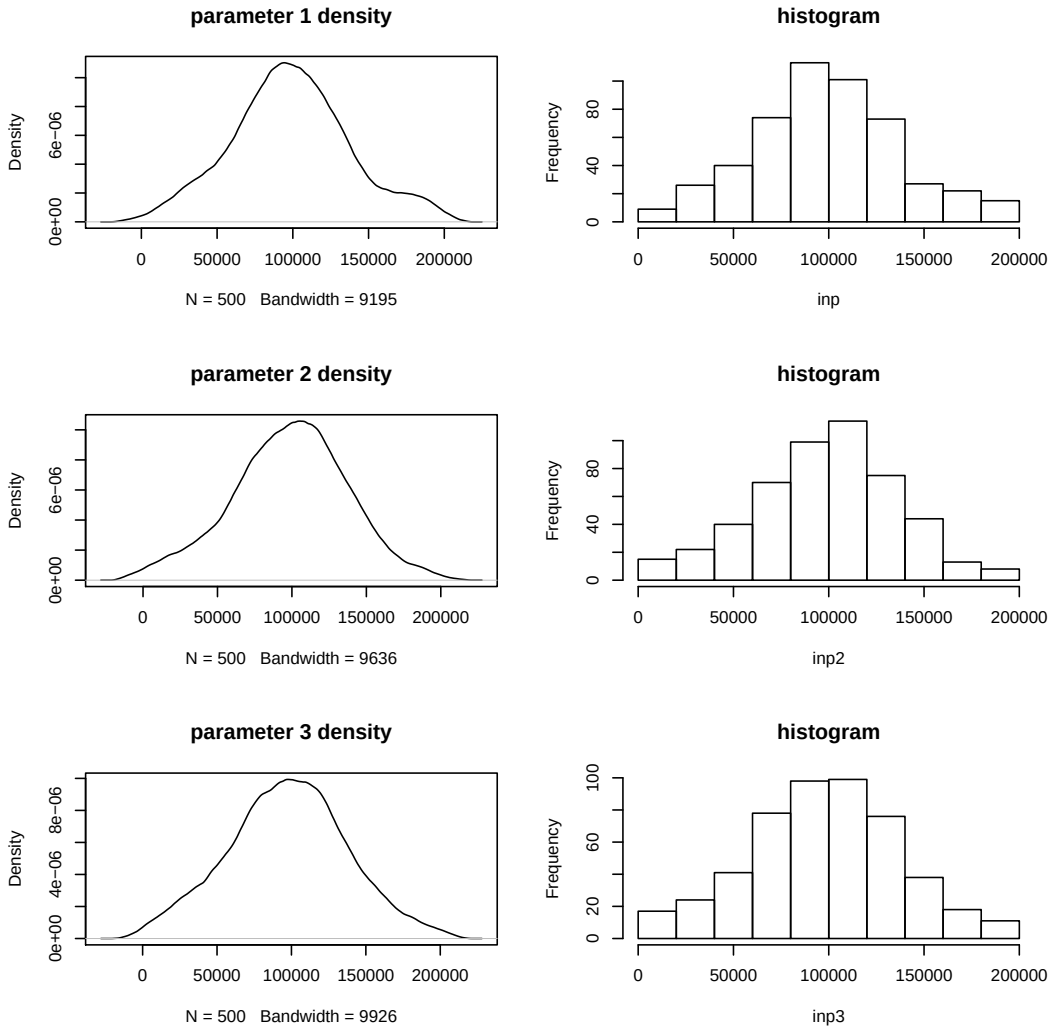

Figure 4: MCMC, Density and histogram of the 'threshold count' of 500 chains of length 200K using 10000 calibration simulations, uniform priors [-1, +1]. Tolerance 5%

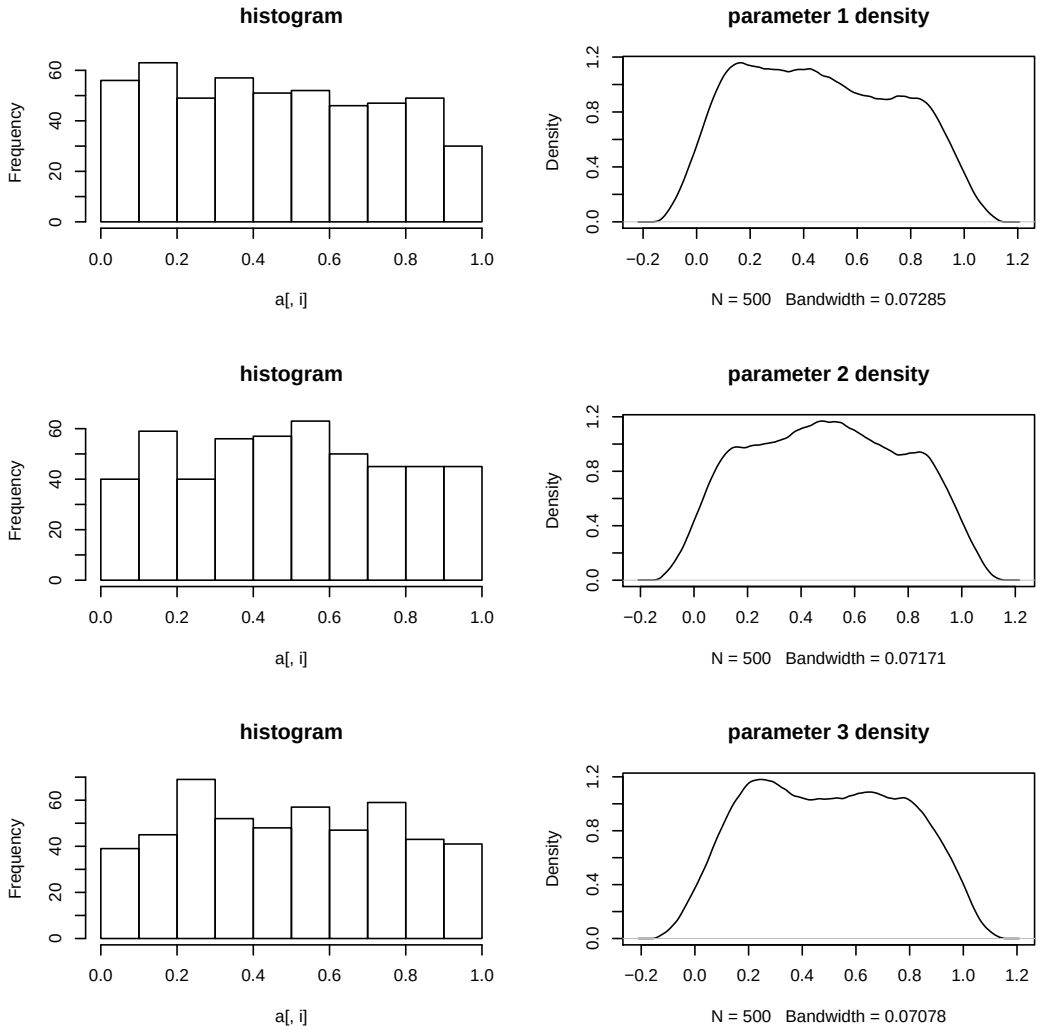

Figure 5: Density and histogram after post sampling adjustment of 500 chains of length 200K using 10000 calibration simulations, uniform priors [-1, +1]. Tolerance 5%

5 Discussion

5.1 Achievements

We have successfully implemented the SuffStat MCMC as a part of ABCsampler in the ABCtoolbox. We have verified that it converges after a reasonable amount of iterations and have shown that it works correctly by checking the posterior distribution of several thousand runs of ABCsampler. We gathered evidence that SuffStat MCMC is more accurate than ABC-MCMC when used on our toy model. It works without a post-sampling regression adjustment. We illustrated that SuffStat MCMC performs better than standard MCMC because it can take into account that summary statistics may only be influenced by a part of the parameters. We showed that by ignoring this fact standard MCMC without likelihood results in an overly broad posterior, which can be corrected with post sampling adjustment.

5.2 Difficulties encountered

The debugging of our implementation was very challenging but forced us to better understand the existing parts of ABCsampler and its implentations of rejection sampling and MCMC without likelihood. Running the many thousand trials was a very slow and time-consuming process, but it led to progressively better bash and R-scripts and a very neat approach to file-handling. The results of the posterior tests were quite unexpected at first, but they challenged us to better understand the statistics involved and gave us a better understanding of the advantages of SuffStat MCMC over MCMC without likelihood.

5.3 Outlook on the remainder of the project

1. make an accurate comparison to MCMC, taking into account the acceptance rate.

2. Apply SuffStat MCMC to a real-life problem in which we find the SuffStat blocks with PLS.

3. Complete the ABCtoolbox manual with the information on how to use our implementation of SuffStat MCMC.

References

[1] M. A. Beaumont, W. Zhang, and D. J. Balding. Approximate bayesian computation in population genetics. *Genetics*, 162(4):2025–35, 2002. 22412157 0016-6731 Journal Article.

[2] Christoph Leuenberger and Daniel Wegmann. Bayesian Computation and Model Selection in Population Genetics. *genetics*, 2009.

[3] P. Marjoram, J. Molitor, V. Plagnol, and S. Tavare. Markov chain monte carlo without likelihoods. *Proceedings Of The National Academy Of Sciences Of The United States Of America*, 100(26):15324–15328, 2003.

[4] S. Tavare, D. J. Balding, R. C. Griffiths, and P. Donnelly. Inferring coalescence times from dna sequence data. *Genetics*, 145(2):505–18, 1997. 0016-6731 Journal Article.

[5] D. Wegmann and L. Excoffier. Efficient Approximate Bayesian Computation coupled with Markov Chain Monte Carlo without likelihood. *Genetics*, 2008.

[6] Daniel Wegmann, Christoph Leuenberger, Samuel Neuenschwander, and Laurent Excoffier. Abctoolbox: a versatile toolkit for approximate bayesian computations. *BMC Bioinformatics*, 11(1):116, 2010.

6 Appendix

6.1 example_glm.input

```
//Inputfile for the program ABCsampler

//------------------------------------------------------------------------

samplerType xxx

//------------------------------------------------------------------------

tolerance 0.1 // accept 1% of simulations

rangeProp 0.5 // 0.5* SD of accepted simulations during calibrations

mcmcSampling 1

numCaliSims 1000

suffStatBlocksFile suffStatBlocks.input

estName example_glm.est

obsName example_glm.obs

outName example_glm_output

separateOutputFiles 0

simDataName sumstats.txt

sumStatFile sumstats.txt

nbSims 10000

writeHeader 1

simulationProgram INTERNALGLM

simInputName matrix.txt

simParam matrix.txt#sumstats.txt#PARAM_A#PARAM_B#PARAM_C

//sumStatProgram arlsumstat

//sumStatParam SIMDATANAME#SSFILENAME#0#1
```

6.2 paramA_file.pls

```
//Name        mean        SD        STAT_A

Stat_1        0        1        1

Stat_2        0        1        0

Stat_3        0        1        0
```

6.3 matrix.txt

```
C

1 0 0

0 1 0

0 0 1

c0

10 10 10

Sigma

0.01 0 0

0 0.01 0

0 0 0.01
```

6.4 run_mcmc_simulations.sh

```
#! /bin/bash


echo $1
./ABC_MCMC_FAST example_glm.input nbSims=1 samplerType=MCMC numCaliSims=1000


mv calibration_file.txt samples.txt


c=1
tail -n$1 samples.txt | while read line
do


        echo "--------------------------------------------------------------------------------"
        echo running simulation nr. $c of $1
        echo


        echo Stat_1        Stat_2        Stat_3 > example_glm.obs
        #echo $line | cut -f4,5,6  -d ' '
        echo $line | cut -f4,5,6  -d ' '>> example_glm.obs
        echo $line | cut -f1,2,3 -d ' '



        ./ABC_MCMC_FAST example_glm.input obsName=example_glm.obs outName=mcmc_example_glm$c.out %
                numCaliSims=10000 nbSims=200000 samplerType=MCMC
        mv ABCsampler.log mcmc_ABCsampler$c.log


        ./ABC_MCMC_FAST example_glm.input obsName=example_glm.obs outName=example_glm$c.out %
        numCaliSims=10000 nbSims=200000 samplerType=SuffStatMCMC
        mv ABCsampler.log ABCsampler$c.log


        for p in 1 2 3
        do
```

```
        obs=$(echo $line | cut -f$p  -d ' ')
        echo threshold = $obs , $c . $p
        pos=`expr $p + 1`
        #tail -n2 example_glm$c.out_sampling1.txt | cut -f$pos | gawk '{print $1}'
        tail -n+2 example_glm$c.out_sampling1.txt | cut -f$pos | %
        gawk -v threshold=$obs '{if($1<threshold) print $1}' | wc | gawk '{print $1}' >> param$p.result
        tail -n+2 mcmc_example_glm$c.out_sampling1.txt | cut -f$pos | %
        gawk -v threshold=$obs '{if($1<threshold) print $1}' | wc | gawk '{print $1}' >> mcmc_param$p.result

        done

        c=`expr $c + 1`

done

tar -zcf example_mcmc_sampling.tar *out_sampling1.txt samples.txt
tar -zcf example_mcmc_logs.tar *.log
tar -zcf results_mcmc.tar *.result
rm *sampling1.txt
rm *.log
rm matrix-temp.txt
rm calibration_file.txt
rm *.result
```

6.5 plot_density.r

```
str <- "~/Desktop/ONE_TRUE/sufstat/compare_again/mcmc_"

count<-200000


par(mfrow=c(3,2))

inp <- scan(paste(str,"param1.result",sep=''),0)

#inp <- c(inp,scan("param1.result2",0)

d <- density(inp,kernel="epanechnikov")

#print(d)

ks.test(inp/count,"punif")$p.value

plot(d, main="parameter 1 density")

hist(inp, main="histogram")

#plot(inp)


inp2 <- scan(paste(str,"param2.result",sep=''),0)

#inp <- c(inp,scan("param1.result2",0)

d <- density(inp2,kernel="epanechnikov")

#print(d)

ks.test(inp2/count,"punif")$p.value

plot(d,main="parameter 2 density")

hist(inp2, main="histogram")

#plot(inp2)


inp3 <- scan(paste(str,"param3.result",sep=''),0)

#inp <- c(inp,scan("param1.result2",0)

d <- density(inp3,kernel="epanechnikov")

#print(d)

ks.test(inp3/count,"punif")$p.value

plot(d,main="parameter 3 density")

hist(inp3,main="histogram")

#plot(inp3)
```

6.6 plot_quantiles.r

```
#quantile blabla plot


a<-read.table("example_glm_sampling/example_glm5.out_sampling1.txt",header=T,nrows=1000000)

par(mfrow=c(1,3))

for(j in 2:4){


        my_q<-quantile(a[1:10000,j],c(0.01,0.05,0.25,0.5,0.75,0.95,0.99))

        for(i in seq(20000,length(a[,1]),by=10000)){

                my_q<-rbind(my_q,quantile(a[1:i,j],c(0.01,0.05,0.25,0.5,0.75,0.95,0.99)))

        }


        plot(my_q[,1],type="l",ylim=c(-1,1), main=paste("parameter ", j-1,sep=''),xlab="len",ylab="quantiles")

        for(i in 2:7){

                lines(my_q[,i])

        }



}
```