INSTITUT NATIONAL DE LA STATISTIQUE ET DES ETUDES ECONOMIQUES
Série des Documents de Travail du CREST
(Centre de Recherche en Economie et Statistique)

n° 2010-44

Using Parallel Computation to Improve Independent Metropolis-Hastings Based Estimation

P. JACOB[1] **− C. P. ROBERT**[2]
M. H. SMITH[3]

1 Université Paris-Dauphine et CREST, Paris.
2 Université Paris-Dauphine, CEREMADE, France et CREST, Paris.
3 National Institute of Water and Atmospheric Research, Wellington

Using parallel computation to improve Independent Metropolis–Hastings based estimation

P. Jacob[1,2], *C.P. Robert*[1,2], *M.H. Smith*[3]
[1] *Université Paris-Dauphine,* [2] *CREST, Paris, and*
[3] *National Institute of Water and Atmospheric Research, Wellington*

Abstract

In this paper, we consider the implications of the fact that parallel raw-power can be exploited by a generic Metropolis–Hastings algorithm if the proposed values are independent. In particular, we present improvements to the independent Metropolis–Hastings algorithm that significantly decrease the variance of any estimator derived from the MCMC output, for a null computing cost since those improvements are based on a fixed number of target density evaluations. Furthermore, the techniques developed in this paper do not jeopardize the Markovian convergence properties of the algorithm, since they are based on the Rao–Blackwell principles of Gelfand and Smith (1990), already exploited in Casella and Robert (1996), Atchadé and Perron (2005) and Douc and Robert (2010). We illustrate those improvement both on a toy normal example and on a classical probit regression model but insist on the fact that they are universally applicable.
Keywords: MCMC algorithm, independent Metropolis–Hastings, parallel computation, Rao–Blackwellization.

1 Introduction

The Metropolis–Hastings (MH) algorithm provides an iterative and converging algorithm to sample from a target distribution π. Each iteration of the algorithm generates a new value of the resulting Markov chain and relies on the result of the previous iteration. The underlying Markovian principle is well understood and leads to a most universal convergence principle as described, e.g., in Robert and Casella (2004). However, due to its Markovian nature, this algorithm is not straightforward to parallelize, which creates difficulties in slower languages like R (R Development Core Team, 2006). Nevertheless, the increasing number of parallel cores available at very low cost drives more and more interest in "parallel-friendly" algorithms, that is, algorithms that can benefit from the number p of available processing units on standard computers (Holmes et al., 2011).

Different techniques have already been used to enhance some degree of parallelism in generic Metropolis–Hastings algorithms, beside the basic scheme of running p MCMC algorithms independently in parallel and merging the results. For instance, a natural entry is to rely on renewal properties of the Markov

chain (Mykland et al., 1995, Robert, 1995, Hobert et al., 2002), waiting for all p chains to exhibit a renewal event and then using the blocks as iid, but the jamming block of Markovianity cannot be removed. Rosenthal (2000) also points out the difficult issue of accounting for the burn-in time: while, for a single MCMC run, the burn-in time is essentially negligible, it does create a bias when running parallel chains (unless perfect sampling can be implemented). Craiu and Meng (2005) mix antithetic coupling and stratification with perfect sampling. Using a different approach, Craiu et al. (2009) exploit p parallel chains to build an adaptive MCMC algorithm, considering in essence that the product of the target densities over the chains is their target, which obviously impacts the convergence properties of the multiple chain. Corander et al. (2006) take advantage of parallelization to build a non-reversible algorithm that can avoid the scaling effect of specific neighborhood structures, hence focussing on a very special type of problem.

A particular case of MH algorithm is the Independent Metropolis–Hastings (IMH) algorithm, where the proposal distribution (and hence the proposed value) does not depend on the current state of the Markov chain. For this very reason, this particular algorithm is easier to parallelize, and can therefore be considered as a good building block toward efficient parallel Markov Chain Monte Carlo algorithms, as will be explained in Section 2. The fundamental idea in the current paper is that one can take advantage of the parallel abilities of arbitrary computing machinery, from cloud computing to graphical cards (GPU), in the case of the generic IMH algorithm, producing an output that corresponds to a much improved Monte Carlo approximation machine at the same computational cost. The techniques presented here are similar to those explained in Perron (1999) and more closely in Atchadé and Perron (2005, Section 3.1), since those authors condition upon the order statistic of the proposed values of the IMH, although in those earlier papers the links with parallel computation were not established and hence the implementation of the Rao-Blackwellization became problematic for long chains.

The plan of the paper is as follows: the standard IMH algorithm is recalled in Section 2, followed by a description of our improvement scheme, called here "block Independent Metropolis–Hastings" (block IMH). This improvement is based on the choice of permutations on $\{1, \ldots, p\}$ that are described in details in Section 3. We demonstrate the connections between block IMH and Rao–Blackwellization in Section 4. Results for a toy example are presented throughout the paper and finally a realistic probit regression example is described in Section 5 as an illustration of the method.

2 Improving the IMH algorithm

2.1 Standard IMH algorithm

We recall here the notations for the (classical) IMH algorithm. Assume the availability of a "proposal" distribution μ from which we can sample, and which probability density is known up to a normalization constant. The independent Metropolis–Hastings algorithm, as proposed in Algorithm 1, generates a Markov chain with invariant distribution π, the target distribution of interest.

Algorithm 1 IMH algorithm

1: Set x_0 to an arbitrary value
2: **for** $t = 1$ to T **do**
3: Generate $y_t \sim \mu$
4: Compute the ratio:

$$\rho(x_{t-1}, y_t) = \min\left\{1, \frac{\pi(y_t)}{\mu(y_t)}\frac{\mu(x_{t-1})}{\pi(x_{t-1})}\right\}$$

5: Set $x_t = y_t$ with probability $\rho(x_{t-1}, y_t)$; otherwise set $x_t = x_{t-1}$
6: **end for**

An important feature of this algorithm with regard to parallelism is that it works only in an iterative manner, since the outcome of step t, namely the value x_t, is required to compute the acceptance ratio at step $t + 1$. This sequential construction is compulsory for the validation of the algorithm given the Markov property at the core of this validation (Robert and Casella, 2004). However since the proposed values (y_t) are generated independently from the current states of the Markov chain (x_t), it is altogether possible to envision the generation of T proposed values first and the computation of the ratios $\omega_t = \pi(y_t)/\mu(y_t)$ at an early stage. Once this computation is concluded, only the acceptance steps need to be considered iteratively. This may make for a huge saving in computing time if the simulation and the derivation of the ω_t's can be done in parallel since the remaining computation of the ratios $\rho(x_{t-1}, y_t)$ given the ω_t's and their subsequent comparison with uniform draws are typically orders of magnitude faster.

In this respect the IMH algorithm strongly differs from the Random Walk Metropolis–Hastings (RWMH) algorithm, for which the acceptance ratios cannot be processed beforehand because the proposed values depend on the value of the Markov chain generated up to the current step. The universal availability of parallel processing schemes may then lead a new surge of popularity for the IMH algorithm. Indeed, when taking advantage of p parallel processing units, an IMH can be run for p times as many iterations as a RWMH, at almost exactly the same computing cost.

In order to describe this increased computing power, we note that, once T successive values of the Markov chain have been produced, the sequence is usually processed as a regular Monte Carlo sample to obtain an approximation of an expectation under the target distribution, $\mathbb{E}_\pi[h(X)]$ say, for some arbitrary functions h. We propose in this paper a technique that improves the precision of the estimation of this expectation by taking advantage of parallel processing units without jeopardizing the validity of the Markov property.

Before presenting our improvement scheme, we introduce the notation $\vee$ (read "or") for the operator that represents a single step of the IMH. Using this notation, given x_t and p proposed values $y_1, \ldots, y_p \sim \mu$, the IMH algorithm goes from step t to step $t + p$ according to the diagram of Figure 1.

$$x_t \longrightarrow x_{t+1} := x_t \vee y_1 \longrightarrow x_{t+2} := x_{t+1} \vee y_2 \longrightarrow \cdots \longrightarrow x_{t+p} := x_{t+p-1} \vee y_p$$

Fig. 1: IMH steps between iteration t and iteration $t + p$.

2.2 Block IMH algorithm

Whether or not parallel processing units are available, we propose to take advantage as much as possible of the simulated proposed values and of the computation of their corresponding ω ratios, since either the simulations or the density derivations may possibly be hard to obtain. To this effect, we propose what we call the *block IMH algorithm*, which is made of successive simulation blocks of size $p \times p$. In this alternative, the number of blocks b is such that the number of desired iterations T is equal to $b * p$, in order to make the comparison with a standard IMH output fair. Usually p needs not be calibrated since it represents the number of parallel processing units that can be exploited by the program. However this number p can as well be set arbitrarily high; in this paper, we take p varying from 4 to 100. We first explain how one block is built, and then how to go from one block to the next.

A $p \times p$ block consists in the generation of p steps of p parallel Markov chains, all starting at time t from the current state x_t and all based on the *same* proposed values $y_1, \ldots, y_p$ but including them in possibly different orders. For instance, these orders may be the p circular permutations of $y_1, \ldots, y_p$, or they may be instead random permutations, as discussed in detail (and compared) in Section 3. The block IMH algorithm is illustrated in Figure 2.

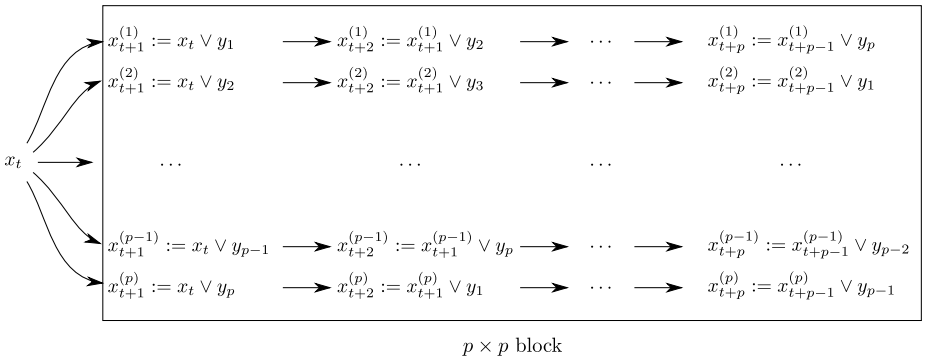


Fig. 2: Block from step $t+1$ to step $t+p$. Here, circular permutations of the proposed values are used for illustration purposes.

Obviously, each of the p parallel chains found in this block is a valid MCMC sequence when taken separately. As such, it can be processed as a regular MCMC output. In particular, if x_t is simulated from the stationary distribution, any of the subsequent $x_{t+i}^{(j)}$ is also simulated from the stationary distribution. However, the point of the p parallel implementations is double:

- it aims at (partly) integrating out the (ancillary) randomness resulting from the order in which the y_k's are chosen, getting close to the conditioning on the order statistics of the y_k's advocated by Perron (1999);
- it also aims at (partly) integrating out the randomness resulting from the generation of uniform variables in the selection process, since the block implementation results in drawing p^2 uniform realizations instead of p uniform realizations for a standard IMH setting.

Both points are essentially implementing a type of Rao–Blackwellization techniques (a more precise connection is drawn in Section 4). In an independent setting, each of the y_k's occurs a number $n_k \geq 0$ of times across the p steps of the p parallel chains, i.e. for a number p^2 of realizations. Therefore, when considering the standard estimator $\hat{\tau}_1$ of $\mathbb{E}_\pi[h(X)]$, based on a *single* MCMC chain,

$$\hat{\tau}_1(x_t, y_{1:p}) = \frac{1}{p}\sum_{k=1}^{p} h(x_{t+k})$$

this estimator necessarily has a larger variance than the double average

$$\hat{\tau}_2(x_t, y_{1:p}) = \frac{1}{p^2}\sum_{j=1}^{p}\sum_{k=1}^{p} h(x_{t+k}^{(j)}) = \frac{1}{p^2}\sum_{k=0}^{p} n_k h(y_k)$$

where $y_0 := x_t$ and n_0 is the number of times x_t is repeated. (The proof for the reduction of the variance from $\hat{\tau}_1$ to $\hat{\tau}_2$ easily follows from a double integration argument.)

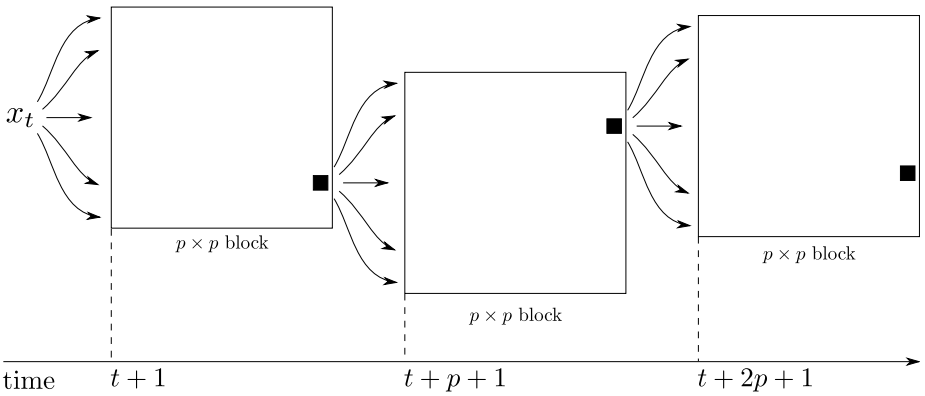


Fig. 3: The block IMH algorithm runs p parallel chains during p steps, then picks one of the final values (represented by the black squares) and iterates. Another transition mechanism is described in Section 2.2.

In order to preserve its Markovian validation, the algorithm must be continued after time $t+p$. An obvious choice is to pick one of the p sequences at random and to take the corresponding $x_{t+p}^{(j)}$ as the value of x_{t+p}, starting point of the next parallel block. This mechanism is represented in Figure 3. While valid from a Markovian perspective, since the sequences are marginally produced by a regular IMH algorithm, this proposal means that the underlying chain deduced from the block IMH algorithm is converging at *exactly* the same speed as the original IMH algorithm. An alternative construction of the sequence of blocks is to take advantage of the weights n_k on the y_k's that are computed via the block structure. Indeed, those weights essentially act as importance weights and they allow for a selection of any of the p^2 $x_{t+i}^{(j)}$'s as the starting point of the incoming block, which corresponds to choosing one of the proposed y_k's with probability proportional to n_k. While this proposal may sound counter-productive because it seems to impose a "backward" step away from convergence by going back in

time, it does not impact the estimation aspects (which involve the p^2 values) and it should on the contrary improve convergence, given that the weighted y_k's behave like a discretized version of a sample from the target distribution π.

The original version of the block IMH algorithm is described in Algorithm 2, the modification stressed above involving a switch of line 12 to the selection of one of the $y_{(i-1)*p+j}$'s with weight $n_{(i-1)*p+j}$.

Algorithm 2 block IMH algorithm

1: Set x_0 to an arbitrary value, compute ω_0
2: Set $x_{\text{start}} = x_0$, $\omega_{\text{start}} = \omega_0$
3: Set a block size p, and a number of blocks b, such that $b * p = T$
4: Generate all proposed values $y_1, \ldots, y_T \sim \mu$
5: Compute all ratios $\omega_1, \ldots, \omega_T$
6: Choose p permutations $\sigma_1, \ldots, \sigma_p$
7: **for** $i = 1$ to b **do**
8: **for** $k = 1$ to p **do**
9: Run p steps of an IMH given:

- $(x_{\text{start}}, \omega_{\text{start}})$
- p proposed values $y_{(i-1)*p+1}, \ldots, y_{i*p}$ shuffled with the permutation σ_k
- the p corresponding ratios ω_j's

10: Save as $x^{(k)}_{(i-1)*p+1}, \ldots, x^{(k)}_{i*p}$ the resulting chain
11: **end for**
12: Draw an index j uniformly in $\{1, \ldots, p\}$, set $x_{\text{start}} = x^{(j)}_{i*p}$, set ω_{start} as the corresponding ratio ω.
13: **end for**

The algorithm is now made of a loop on the b blocks and an inner loop on the p parallel chains of each block. The p steps of this inner loop are actually meant to be computed in parallel. The output of Algorithm 2 is double:

- a standard Markov chain of length T, which is made of b chains of length p, each of which is chosen among p chains at line 12 of Algorithm 2,
- a $p \times T$ array $(x^k_t)^{k=1:p}_{t=1:T}$, on which the estimator $\hat{\tau}_2$ is based.

As mentioned above, the point-wise evaluation of the target density $\pi(y_k)$ is usually the most computer-intensive part of the algorithm. Therefore, the cost of sampling uniform variables is supposed to have a negligible impact here, as are other costs related with the storage of larger vectors than in the original IMH, especially given the fact that the multiple chains do not need to be stored further than during a single block execution time. That is why, although we sample p times more uniforms in the block IMH algorithm, we still consider it to be roughly of the same cost as the original algorithm, since the number of target density evaluations is the same (and can obviously take advantage of the parallel processing abilities of the machine to improve computing time compared with the original IMH algorithm.). Besides pseudo-random generation of uniforms can also benefit from parallel processors, see e.g L'Ecuyer et al. (2001).

The method is presented here for square blocks of dimension (p, p), but blocks could be rectangular as well: the algorithm is equally valid when using $r \neq p$ permutations, leading to (r, p) blocks. We focus here on square blocks because when the machine at hand provides p parallel processing units, then it is most efficient to compute the target densities and the acceptance ratios at the p proposed values in parallel. Once again, the block IMH algorithm with $p \times p$ square blocks has at worst the same cost as the original IMH algorithm, because computing target densities and acceptance ratios does more than compensate for the cost of randomly picking an index at the end of each block. This amounts to say that line 4 of Algorithm 1 and line 5 of Algorithm 2 are (by far) the most computationally demanding ones in the respective algorithms. The proposed algorithm also involves storing r times more values in the memory, so in a context where memory is an issue, choosing $r < p$ can make sense. Alternatively, if the quantities of interest are known in advance, the approximations to the estimators can be updated at each block end and the storage problem would thus be completely eliminated. If a complete Markov chain is needed, it can be randomly selected among the p parallel sequences at the end of each block and stored along with the p weighted proposals, thus requiring only twice as much storage as the regular IMH algorithm.

2.3 Toy example

We now introduce a toy example that we will follow throughout the paper. The target π is the standard $\mathcal{N}(0, 1)$ normal distribution and the proposal μ is the $\mathcal{C}(0, 1)$ Cauchy distribution. Hence

$$\omega(x) = \frac{\pi(x)}{\mu(x)} \propto (1 + x^2) \exp\left(-\frac{1}{2}x^2\right)$$

We only consider the integral $\int x\pi(\mathrm{d}x)$, the expectation of π. The acceptance rate of the IMH algorithm for this example is around 70%.

In all results related to the toy example presented thereafter, $10,000$ independent runs are used to compute the variance of the estimates. The value of p represents the number of parallel processing units that are available, ranging from 4 for a desktop computer to 100 for a cluster or a graphics processing unit (GPU) (this value could even be larger for computers equipped with multiple GPUs).

The results of the simulation experiments are presented as barplots, which indicate the percentage of variance decrease associated with the estimators under comparison, the reference estimator being always plotted as the first bar on the left-hand side. In agreement with the block sampling perspective, the same proposed values and uniform draws were used for all the estimators that are plotted on the same graphs, so that the comparison is not perturbed by an additional noise coming from the drawing.

3 Permutations

The choice of the permutations in line 6 of Algorithm 2 is obviously having an impact on the improvement brought by block sampling. The idea of testing various orders of the proposed values in a IMH algorithm appeared in Atchadé

and Perron (2005) where the permutations were chosen to be circular. We first list some permutations and their justification, and then we compare them on the toy example.

3.1 Five choices of permutations

Let $\mathcal{S}$ be the set of permutations of $\{1, \dots, p\}$. The size of $\mathcal{S}$ is $p!$ and it is therefore too large to allow for an averaging over all permutations, although this solution would be ideal. We consider the simpler problem of finding p efficient permutations in $\mathcal{S}$, denoted by $(\sigma_1, \dots, \sigma_p)$, the goal being a choice favoring a decrease the variance of the estimator $\hat{\tau}_2$ defined in Section 2.

3.1.1 Fixed order

The most basic choice is to pick the same permutation on each of the p chains:

$$\sigma_1 = \sigma_2 = \ldots = \sigma_p$$

This selection sounds useless since it does not actually average on the orders but we actually notice a significant decrease in the variance of $\hat{\tau}_2$ using this set of permutations, compared to $\hat{\tau}_1$. The reason is that p times more uniforms are used in $\hat{\tau}_2$ than in $\hat{\tau}_1$, leading to a natural Rao-Blackwellization phenomenon that is studied in details in Section 4. Nonetheless this simplistic set of permutations is certainly not the best choice since it does not integrate out the (ancillary) randomness resulting from the ordering of the proposed values.

3.1.2 Circular permutations

Another simple choice is to use circular permutations. For $1 \leq i \leq p$m we define

$$\sigma_i(1) = i, \sigma_i(2) = i+1, \dots, \sigma_i(p-i+1) = p, \sigma_i(p-i+2) = 1, \dots, \sigma_i(p) = i-1$$

An appealing property of the circular permutations is that each proposed value y_k is proposed and evaluated at a different step for each chain. However, a disadvantage is that the order is not deeply changed: for instance y_{k-1} will always be proposed one step before y_k except for one of the p chains (the one for which y_k is proposed first).

3.1.3 Random permutations

A third choice is to use random orders, that is random shuffling of the sequence $\{1, \dots, p\}$. We can either draw those random permutations with or without replacement in the set $\mathcal{S}$, but considering the cardinality of the set $\mathcal{S}$ it does not make a large difference. Indeed, it is unlikely to draw twice the same permutation, except for very small values of p.

3.1.4 Half random half reversed permutations

A slightly different choice of permutations consists in drawing $p/2$ permutations at random. Then, denoting the first $p/2$ permutations by $\sigma_1, \dots, \sigma_{p/2}$, we define for $1 \leq k \leq p/2$:

$$\sigma_{k+p/2}(1) = \sigma_k(p), \sigma_{k+p/2}(2) = \sigma_k(p-1), \dots \sigma_{k+p/2}(p) = \sigma_k(1)\,.$$

The motivation for this inversion of the orders is that, in the second half of the permutations, the opposition with the "reversed" first half is maximal. This choice, suggestion of Art Owen (personal communication), aims at minimizing the possible common history among the p parallel chains. Indeed two chains with the same proposed values in reverse order cannot have a common path of length more than 1.

3.1.5 Stratified random permutations

Finally we can try to draw permutations that are far from one another in the set $\mathcal{S}$. For instance we can define the lexicographic order on $\mathcal{S}$, draw indices from a low discrepancy sequence on the set $\{1, \ldots, p!\}$ and select the permutations corresponding to these indices. In a simpler manner, we do use here a stratified sampling scheme: we first draw a random permutation conditionally on its first element being 1, then another permutation beginning with 2, and so forth until the last permutation which begins with p.

3.2 A Monte Carlo comparison

We compare the five described types of permutation on the toy example. Figure 4 shows the results for $p = 4, 10, 50, 100$, displaying the variance of $\hat{\tau}_2$ associated with each of the permutation orders when taking as reference variance the variance of the original IMH estimator τ_1. For each of the $10,000$ independent replications, the block IMH algorithm was launched on one single $p \times p$ block, e.g. with $b = 1$ using the notation of Section 2, since b plays no role in this comparison.

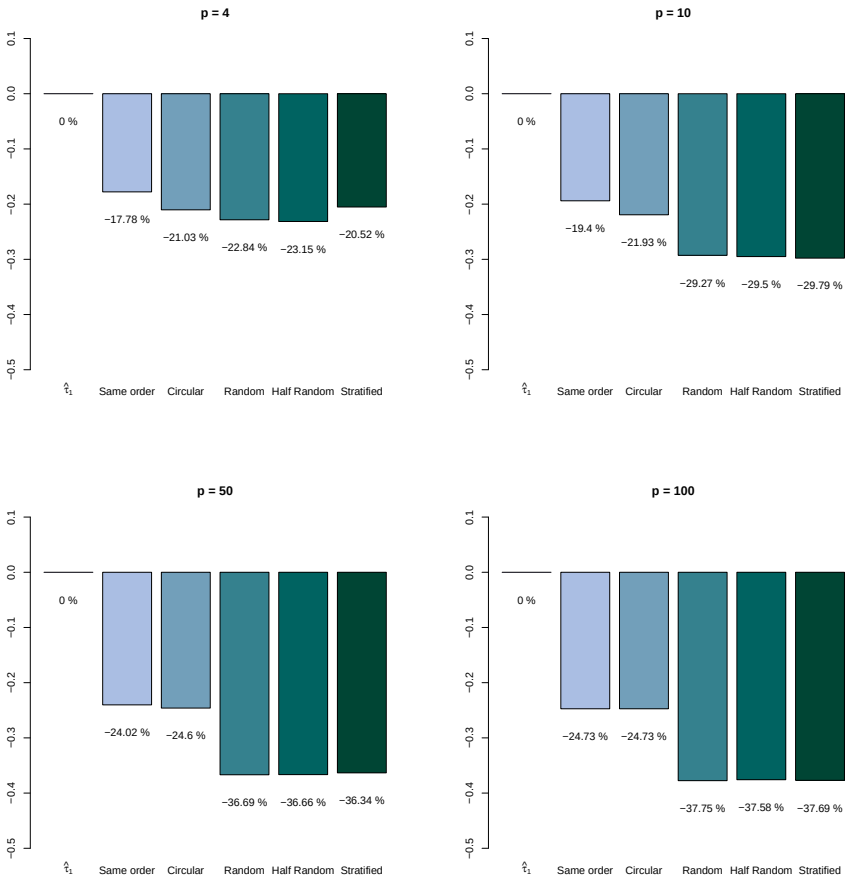


Fig. 4: Variance comparison between the basic estimator $\hat{\tau}_1$ and the various block estimators $\hat{\tau}_2$ associated with each permutation scheme for $p = 4, 10, 50, 100$.

As mentioned above, using the same order in the y_k's for each of the p parallel chains already produces a significant decrease of about 20% in the variance of the estimators. This simulation experiment shows that the three random permutations (random, half-random half-reversed and stratified) are quite equivalent in terms of variance improvement and that they are significantly better than the circular permutation proposal, which only slightly improves over the "same order" scheme. Therefore, in the following Monte Carlo experiments, we will use only the random order solution, given that it is the simplest of the random schemes. The amount of improvement of 35% when $p = 50$ is quite impressive when considering that it is cost-less for a computer with parallel abilities (Holmes et al., 2011).

4 Rao–Blackwellization

Another generic improvement that can be brought over classical MH algorithms is Rao–Blackwellization (Gelfand and Smith, 1990, Casella and Robert, 1996). In this section, two Rao–Blackwellization methods are presented, one that is computationally free and one that, on the contrary, is computationally expensive. We then implement those solutions within the block IMH algorithm and explain why the "same order" scheme already improves the IMH algorithm.

4.1 First Rao–Blackwellization

When we consider the standard IMH algorithm of Section 2.1, a cost-free improvement can be obtained by a straightforward Rao–Blackwellization argument. Given that at step $t+i$, y_i is accepted with probability $\rho(x_{t+i-1}, y_i)$ and rejected with probability $1-\rho(x_{t+i-1}, y_i)$, the weight of y_i can be updated by $\rho(x_{t+i-1}, y_i)$ and the weight of the proposed value y_j corresponding to x_{t+i-1} can be similarly updated by the probability $1-\rho(x_{t+i-1}, y_i)$. Considering instead the block IMH algorithm, at the beginning of each block we can define p weights, denoted by $(w_k)_{k=1}^p$, initialized at 0 and then, denoting by j the index such that $x_{t+i-1} = y_j$, we update these weights at each time $t+i$ as

$$\begin{aligned} w_j &= w_j + 1 - \rho(x_{t+i-1}, y_i) \\ w_i &= w_i + \rho(x_{t+i-1}, y_i) \end{aligned}$$

This is obviously repeated for each of the p parallel chains, thus after the p steps of the block, we have $\sum_k w_k = p^2$. This leads to a third estimator

$$\hat{\tau}_3(x_t, y_{1:p}) = \frac{1}{p^2} \sum_{k=0}^{p} w_k h(y_k)\,.$$

This estimator still depends on all uniform generations created within the block, since those weights w_k depend upon the acceptances and rejections of the proposed values made during the block update. However, along the steps of the block, the w_k's are basically updated by the expectation of the acceptance indicators conditionally upon the results of the previous iterations, whereas the n_k of Section 2 are directly updated according to the acceptance indicators. Hence, the w_k's have a smaller variance than the n_k's by virtue of the Rao–Blackwell theorem, leading to $\hat{\tau}_3$ having a smaller variance than $\hat{\tau}_2$.

We now discuss a more involved Rao-Blackwellization technique in the spirit of Casella and Robert (1996) in the following section.

4.2 Block Rao–Blackwellization

Exploiting the Rao–Blackwellization technique of Casella and Robert (1996) within each parallel chain does provide (by virtue of a conditioning argument) an even more stable approximation of arbitrary posterior quantities. Given the gain brought by parallelization, the additional computing time imposed by the implementation of this Rao–Blackwellization technique is more than compensated by the improvement in efficiency.

As developed in Casella and Robert (1996), for a single Markov chain $(x_1^{(i)}, \ldots, x_p^{(i)})$, a Rao–Blackwell weighting scheme on the proposed values y_t, with weights φ_t, is given by a recursive scheme

$$\varphi_t^{(i)} = \delta_t \sum_{j=t}^{p} \xi_{tj}$$

where $(t > 0)$

$$\delta_0 = 1\,, \qquad \xi_{tt} = 1\,, \qquad \xi_{tj} = \prod_{u=t+1}^{j} (1 - \rho_{tu})$$

and

$$\delta_t = \sum_{j=0}^{t-1} \delta_j \xi_{j(t-1)} \rho_{jt}\,,$$

the ξ_{tj}'s being the occurrence survivals of the proposed value y_t, associated with the Metropolis–Hastings ratio

$$\omega_t = \pi(y_t)/\mu(y_t)\,, \qquad \rho_{tu} = \omega_u/\omega_t \wedge 1\,.$$

The computation of the δ_t's, of the ρ_{tu}'s and of the ξ_{tu}'s requires an $\mathrm{O}(p^2)$ computing time. Given that p is usually not very large, this additional cost is not as redhibitory as in the original proposal of Casella and Robert (1996) who were considering the application of this Rao–Blackwellization technique on the whole chain, with a cost of $\mathrm{O}(T^2)$ (see also Perron, 1999).

Therefore, starting from the estimator $\hat{\tau}_2$, the weight n_k counting the number of occurrences of y_k in the $p \times p$ block can be replaced with the expected number φ_k of times y_k occurs in this block (given the p proposed values), which is the sum of the expected numbers of times y_k occurs in each of the p parallel chain:

$$\varphi_k = \sum_{i=1}^{p} \varphi_k^{(i)}$$

Since the p parallel chains incorporate the proposed values with different orders, the φ's may differ for each chain and must therefore be computed p times. Note that the cost is still in $\mathrm{O}(p^2)$ if this computation can be implemented in parallel. Then, by a Rao-Blackwell argument, $\hat{\tau}_2$ is dominated by $\hat{\tau}_4$ defined as follows:

$$\hat{\tau}_4(x_t, y_{1:p}) = \frac{1}{p^2} \sum_{k=0}^{p} \varphi_k h(y_k)$$

Note that, given this Rao–Blackwellization scheme, the uniform generations are not used at all in the computation of $\hat{\tau}_4$. Hence the randomness coming from these uniforms is completely integrated out.

The four estimators defined up to now can be summarized as follows:

- $\hat{\tau}_1$ is the basic IMH estimator of $\mathbb{E}_\pi\,[h(X)]$,
- $\hat{\tau}_2$ improves $\hat{\tau}_1$ by averaging over permutations of the proposed values, and by using p times more uniforms than $\hat{\tau}_1$,

- $\hat{\tau}_3$ improves upon $\hat{\tau}_2$ by a basic Rao-Blackwell argument,
- $\hat{\tau}_4$ improves upon $\hat{\tau}_2$ by a further Rao-Blackwell argument, integrating out the ancillary uniform variables, but at a cost of $\mathrm{O}(p^2)$.

Note that these four estimators all involve the same number p of target density evaluations.

4.3 A numerical evaluation

Figure 5 gives a comparison between the variances of the three improved estimators and the variance of the basic IMH estimator. The permutations are random in this case. As was already apparent on Figure 4, the block estimator $\hat{\tau}_2$ is significantly better than $\hat{\tau}_1$ for any value of p. Moreover, both Rao-Blackwellization modifications seem to improve only very slightly the estimation when compared with $\hat{\tau}_2$. This is more and more the case when p increases.

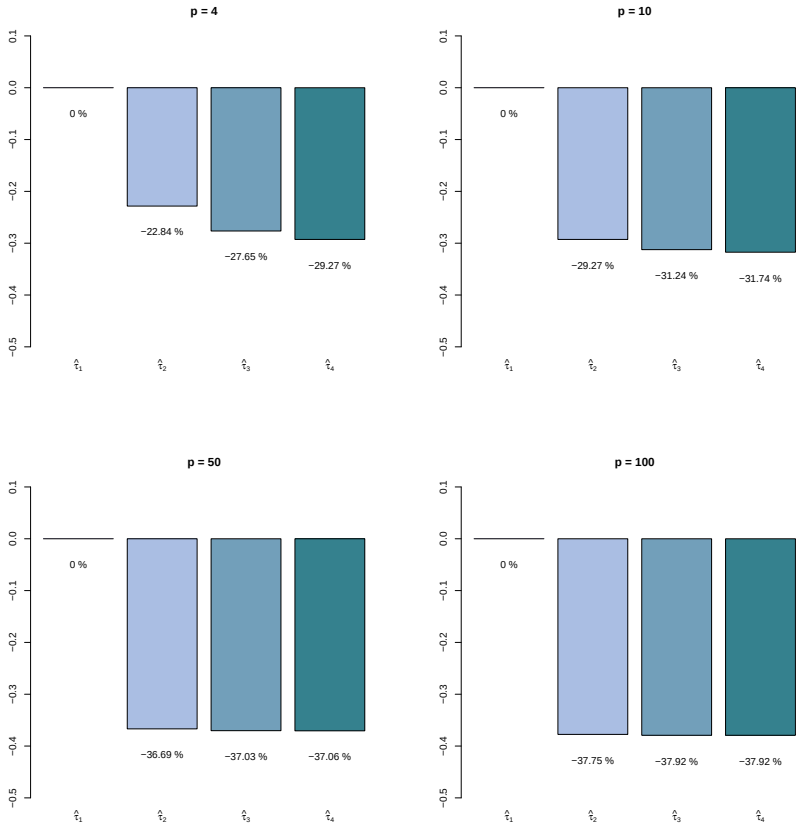


Fig. 5: Variance comparison between the basic estimator $\hat{\tau}_1$ and three improved block IMH estimators.

Recall that the "same order" scheme already provided a significant decrease in the variance of the estimation. In the light of our results, our interpretation

is that using p parallel chains with the same proposed values acts like a "poor man" Rao–Blackwellization technique since p times more uniforms are used. Specifically, each of the p proposed values is proposed p times instead of once, thus reducing the impact of each single uniform draw on the overall estimation.

When we use Rao–Blackwellization on top of the block IMH, in the estimators $\hat{\tau}_3$ and $\hat{\tau}_4$, we try indeed to integrate out a randomness that is already partly gone. This explains why, although Rao–Blackwellization techniques provide a significant improvement over standard IMH, the improvement is much lower and thus rather unappealing when used in the block IMH setting. This outcome was at first frustrating since Rao–Blackwellization is indeed affordable at a cost of only $\mathrm{O}(p^2)$. However, this shows *in fine* that the improvement brought by the block IMH algorithm roughly provides the same improvement as the Rao–Blackwell solution, at a much lower cost.

4.4 Comparison with Importance Sampling

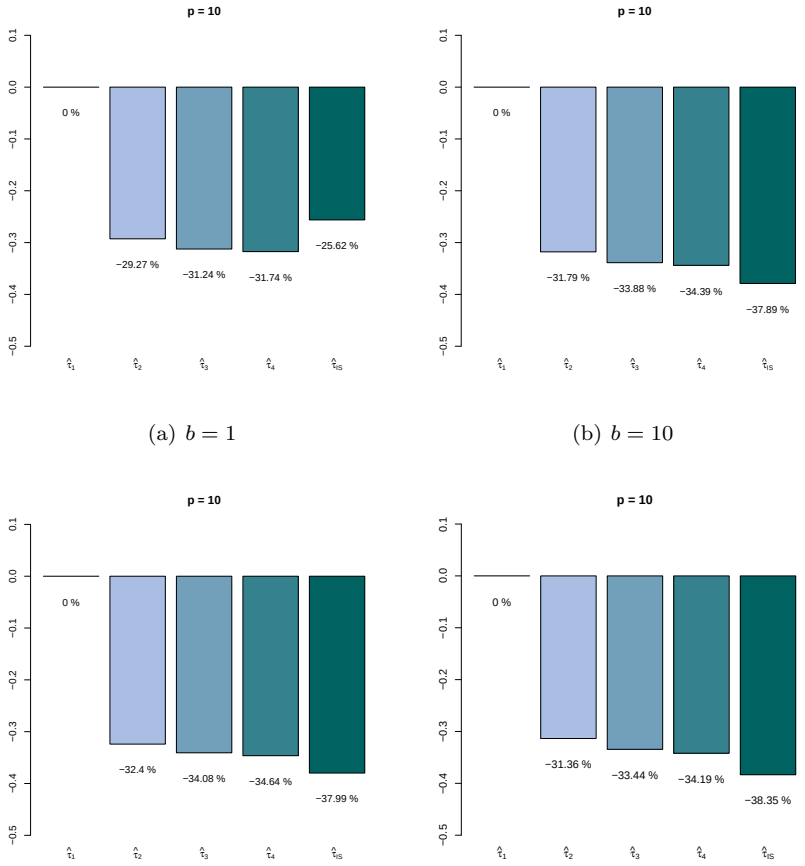
The proposal distribution μ may also be used to construct directly an importance sampling (IS) estimator

$$\hat{\tau}_{IS} = \frac{1}{T}\sum_{t=1}^{T} h(y_t)\frac{\pi(y_t)}{\mu(y_t)}\,,$$

where the values y_t are drawn from the proposal distribution μ. It therefore makes sense to compare the IMH algorithm with an IS approximation because IS is similarly easy to parallelize. Furthermore, since the IS estimator does not involve ancillary uniform variables, it is comparable to the Rao–Blackwellized version of IMH, and hence to the block IMH. Obviously, IS cannot necessarily be used in the settings when IMH algorithms are used, because the latter are also considered for approximating simulations from the target distribution π. In particular, when considering Metropolis-within-Gibbs algorithms, IS cannot be used in a straightforward manner, even for approximating integrals.

Before showing the results of a comparison run on the toy example, we now explain why in this comparison we take the number of blocks to be larger than 1. As mentioned above, the results from the previous sections were computed with $b = 1$, i.e. on a single $p \times p$ block and for a large number of independent runs. The choice of b was then irrelevant since we were comparing methods that were exploiting in different ways the p proposed values generated in each block. When considering the block IMH algorithm as a whole, as explained in Section 2, the end of each block sees a new starting value chosen from the current block. This ensures that the algorithm produces a valid Markov chain. However, our construction also implies that the successive blocks produced by the algorithm are correlated, which should lead to lesser performance than the IS estimator.

In the comparison between IMH and IS, we therefore need to take into account this correlation between successive blocks. To this effect, we produce the variance reductions for several values of b. Those reductions are presented in Figure 6 for $p = 10$ and different values of $b = 1, 10, 100, 1000$. Once again, the permutations in the block IMH algorithm are chosen to be random.



(a) $b = 1$ (b) $b = 10$

(c) $b = 100$ (d) $b = 1000$

Fig. 6: Variance comparison between the basic estimator $\hat{\tau}_1$, three improved block IMH estimators, and the IS estimator $\hat{\tau}_5$.

Figure 6 shows the a priori surprising result that, when selecting $b = 1$ in the experiment, the variance results are in favor of the block IMH solutions over the IS estimator, but, for any realistic application, b is (much) larger than 1. For all $b \geq 10$, the IS estimator has a smaller variance than the three alternative block IMH estimators, if by a small margin. (Note that the variance improvement over the original MCMC estimator is slightly increasing with b despite the correlation between blocks, given that the correlation between the p^2 terms involved in the block IMH estimators is lower than the correlation in the original MCMC chain.) This experiment thus shows that the block IMH solution gets very close to the IS estimator, while keeping the Markovian features of the original IMH algorithm.

5 A probit regression illustration

In order to evaluate the performances of the parallel processing methods presented in this paper on a realistic example, we examine their implementation

the probit model already analyzed in Marin and Robert (2010) for the comparison of model choice techniques because the "plug-in" normal distribution based on MLE estimates of the first two moments works quite well as an independent proposal.

A probit model can be represented as a natural latent variable model in that, if we consider a sample $z_1, \ldots, z_n$ of n independent latent variables associated with a standard regression model, i.e. such that $z_i|\theta \sim \mathcal{N}\left(x_i^{\mathrm{T}}\theta, 1\right)$, where the x_i's are p-dimensional covariates and θ is the vector of regression coefficients, then $y_1, \ldots, y_n$ such that

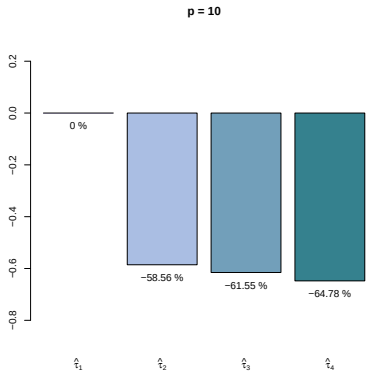
$$y_i = \mathbb{I}_{z_i>0}$$

is a probit sample. Indeed, given θ, the y_i's are independent Bernoulli rv's with $\mathbb{P}(y_i = 1|\theta) = \Phi\left(x_i^{\mathrm{T}}\theta\right)$ where Φ is the standard normal cdf. The choice of a prior distribution for the probit model is open to debate, but the above connection with the latent regression model induced Marin and Robert (2007) to suggest a g-prior model, $\theta \sim \mathcal{N}\left(0_p, n(X^{\mathrm{T}}X)^{-1}\right)$, with n as the g factor and X as the regressor matrix.

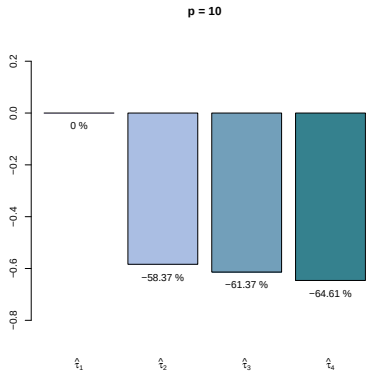
While a Gibbs sampler taking advantage of the latent variable structure is implemented in Marin and Robert (2010) and earlier references (Albert and Chib, 1993), a straightforward Metropolis–Hastings algorithm may be used as well, based on an independent proposal $\mathcal{N}(\hat{\theta}, c\widehat{\Sigma})$, where $\hat{\theta}$ is the MLE estimator, $\widehat{\Sigma}$ its asymptotic variance, and c a scaling factor.

As in Marin and Robert (2010) and Girolami and Calderhead (2010), we use the R Pima Indian benchmark dataset (R Development Core Team, 2006), which contains medical information about 332 Pima Indian women with seven covariates and one explained binary diabetes variable.

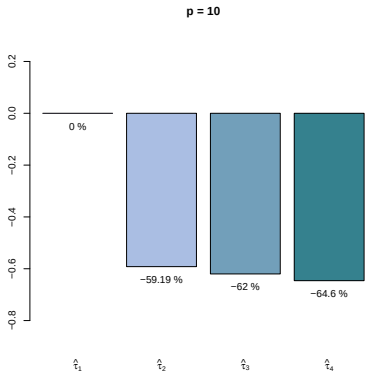
For the purpose of illustrating the implementation of the block IMH algorithm, we only consider here three covariates, namely plasma glucose concentration (with coefficient θ_1), diastolic blood pressure (with coefficient θ_2) and diabetes pedigree function (with coefficient θ_3). We are interested in the posterior mean of those three regression parameters. In our experiment, we ran 10.000 independent replications of our algorithm to produce a reliable evaluation of the variance of the estimators under comparison. We present the variance comparison of the four estimators described in Section 4, for $p = 10$ (Figure 7) and $p = 50$ (Figure 8), for each of the three regression parameters. In the independent proposal, the scale factor is chosen to be 3 since pilot runs showed that it led to an acceptance rate around 37%.



(a) $p = 10$, θ_1

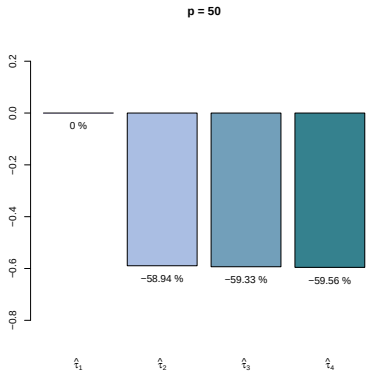


(b) $p = 10$, θ_2

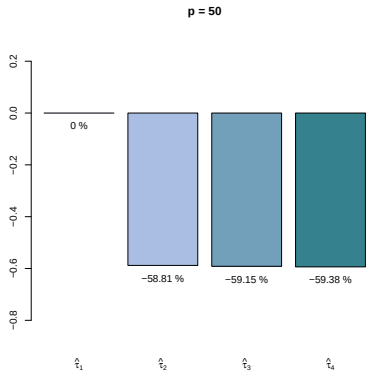


(c) $p = 10$, θ_3

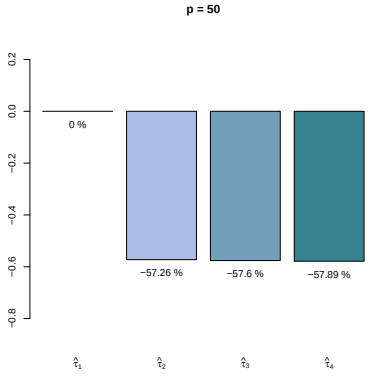
Fig. 7: Variance comparison between the basic estimator $\hat{\tau}_1$ and the three improved block IMH estimators for $p = 10$ and each of the parameters.



(a) $p = 50$, θ_1



(b) $p = 50$, θ_2



(c) $p = 50$, θ_3

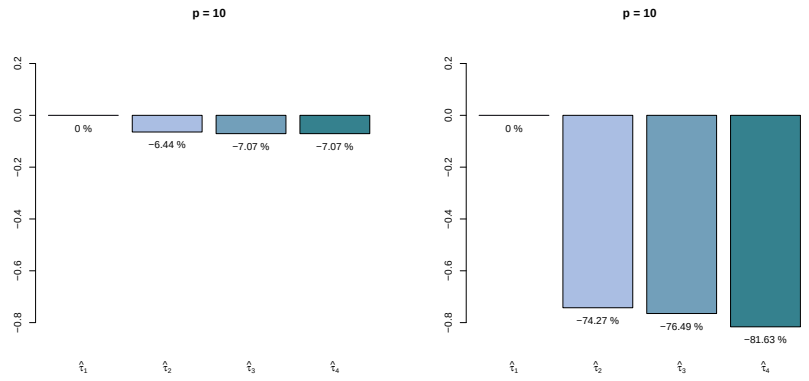
Fig. 8: Variance comparison between the basic estimator $\hat{\tau}_1$ and the three improved block IMH estimators for $p = 50$ and each of the parameters.

The results shown in Figures 7 and 8 confirm the huge decrease in variance previously observed in the toy example. The gains represented in those figures indicate that the block estimator $\hat{\tau}_2$ is nearly as good (in terms of variance improvement) as the Rao–Blackwellized block estimators $\hat{\tau}_3$ and $\hat{\tau}_4$, especially when p moves from 10 to 50. This confirms the previous interpretation given in Section 4 that the block IMH algorithm provides a cost-free Rao–Blackwellization as well as a partial averaging over the order of the proposed values.

The fact that the toy example showed decreases in the variance that were around 35% whereas the probit regression shows decreases around 60% is worth discussing. Indeed the amount of decrease in the variance obviously differs from one example to the other, but it is more importantly depending on the acceptance rate of the Metropolis–Hastings algorithm. In fact, Rao–Blackwellization and permutations of the proposed values are useless steps if the acceptance rate is exactly 1. On the opposite, it may result in a significant improvement when the acceptance rate is low (since the part of the variance due to the uniform

draws would then be much more important).

To illustrate the connection between the observed improvement and the acceptance rate, we propose in Figure 9 a variance comparison for two scaling factors c of the proposal covariance matrix in the probit regression model. In this figure, we have only represented the variances of the estimators of the first parameter θ_1, the results being very similar for the other two estimators. In the previous experiment, we have used $c = 3$, which leads to an acceptance ratio around 37%. Here, if we take $c = 1$, the acceptance ratio rises to 96%, and hence almost all the proposed values are accepted. In this case permuting the proposed values and using Rao–Blackwellization techniques does not bring much of a variance decrease (Figure 9, left). On the other hand, if we take $c = 10$, the acceptance ratio drops down to 8% and the observed decrease in variance is huge. In this second case using all the proposed values gives much better results than relying on the standard IMH estimator, which is only based on 8% of the proposed values that were accepted (Figure 9, right).



(a) Proposal variance $= 1 \times \widehat{\Sigma}$ (b) Proposal variance $= 10 \times \widehat{\Sigma}$

Fig. 9: Variance comparison for two scaling factors: $c = 1$, with an associated acceptance rate of 96% (left) and $c = 10$, with an associated acceptance rate of 8% (right).

The difference observed with this range of scaling factor is thus in agreement with the larger decrease in variance observed for the probit regression model. Note that this is a positive feature of the block IMH method, since in a complex model, the target distribution is most often poorly approximated by the proposal and thus the acceptance rate of the IMH algorithm is quite likely to be low.

6 Conclusion

The Monte Carlo experiments produced in this paper have shown that the proposed method improves significantly the precision of the estimation, when compared with the standard IMH algorithm. We emphasize once again that the block IMH method is close to being 100% parallel (except for the random draw of an index at the end of each block) and that, since parallel computing is

getting more and more easy to use, this free improvement could be available for all implementations of the IMH algorithm. Furthermore, without even considering parallel computing, since we have shown that the method uses the most of each target density evaluation, it brings significant improvement in settings where computing the target density is very costly. In such settings, the cost of drawing p^2 instead of p uniform variates will be negligible, and the block IMH algorithm will therefore run in about the same time as the standard IMH algorithm. We also note that the time required to complete a block in the algorithm is essentially the maximum of the p times required to calculate the density ratios w_i. Therefore, if there is a lot of variation in these times there could be a diminishing saving in computation time through parallel as p increases for both the standard IMH and the block IMH algorithms. But even in those extreme cases, the block IMH algorithm would bring a variance improvement at essentially no additional cost.

We also stress that a straightforward reason for not conducting a comparison with a plain parallel algorithm based on p independent parallel chains is that it does not make sense cost-wise. Indeed, running p parallel chains of the same length T would cost p times more because of the target density evaluations. Obviously, if one wants to run p independent chains, for instance to initialize an MCMC algorithm from several well-dispersed starting points, each of those chains can benefit from our stabilizing method, which will improve the resulting estimation.

Acknowledgements

The work of the second author (CPR) was partly supported by the Agence Nationale de la Recherche (ANR, 212, rue de Bercy 75012 Paris) through the 2009 project ANR-08-BLAN-0218 Big'MC and the 2009 project ANR-09-BLAN-01 EMILE. Pierre Jacob is supported by a PhD fellowship from the AXA Research Fund. Since this research was initiated during the Valencia 9 Bayesian Statistics conference, the paper is dedicated to José Miguel Bernardo for the organisation of this series of unique meetings since 1979. Discussions of the second author with participants during a seminar in Stanford University in August 2010 were quite helpful, in particular the suggestion made by Art Owen to include the "half-inversed half-random" permutations. The authors are also grateful to Julien Cornebise for helpful discussions, in particular those leading to the stratified strategy.

References

Albert, J. and Chib, S. (1993). Bayesian analysis of binary and polychotomous response data. *J. American Statist. Assoc.*, **88** 669–679.

Atchadé, Y. and Perron, F. (2005). Improving on the independent metropolis–hastings algorithm. *Statistica Sinica*, **15** 3–18.

Casella, G. and Robert, C. (1996). Rao-Blackwellisation of sampling schemes. *Biometrika*, **83** 81–94.

Corander, J., Gyllenberg, M. and Koski, T. (2006). Bayesian model learning based on a parallel MCMC strategy. *Statistics and Computing*, **16** 355–362.

Craiu, R., Rosenthal, J. and Yang, C. (2009). Learn from thy neighbour: Parallel-chain and regional adaptive MCMC. *J. American Statist. Assoc.*, **104**.

Craiu, R. V. and Meng, X.-L. (2005). Multiprocess parallel antithetic coupling for backward and forward Markov chain Monte Carlo. *Ann. Statist.*, **33** 661–697.

Douc, R. and Robert, C. (2010). A vanilla variance importance sampling via population Monte Carlo. *Ann. Statist.* To appear.

Gelfand, A. and Smith, A. (1990). Sampling based approaches to calculating marginal densities. *J. American Statist. Assoc.*, **85** 398–409.

Girolami, M. and Calderhead, B. (2010). An object-oriented random-number package with many long streams and substreams. *J. Royal Statist. Society Series B*, **73** 1–37.

Hobert, J., Jones, G., Presnel, B. and Rosenthal, J. (2002). On the applicability of regenerative simulation in Markov chain Monte Carlo. *Biometrika*, **89** 731–743.

Holmes, C., Doucet, A., Lee, A., Giles, M. and Yau, C. (2011). Bayesian computation on graphics cards. In *Bayesian Statistics 9: Proceedings of the Ninth Valencia International Meeting, June 3-8, 2010* (J. Bernardo, M. Bayarri, J. Degroot, A. Dawid, D. Heckerman, A. Smith and M. West, eds.). Oxford University Press.

L'Ecuyer, P., Simard, R., Chen, E. J. and Kelton, W. D. (2001). An object-oriented random-number package with many long streams and substreams. *Operations Research*, **50** 1073–1075.

Marin, J. and Robert, C. (2010). Importance sampling methods for Bayesian discrimination between embedded models. In *Frontiers of Statistical Decision Making and Bayesian Analysis* (M.-H. Chen, D. Dey, P. Müller, D. Sun and K. Ye, eds.). Springer-Verlag, New York, 513–527.

Marin, J.-M. and Robert, C. (2007). *Bayesian Core*. Springer-Verlag, New York.

Mykland, P., Tierney, L. and Yu, B. (1995). Regeneration in Markov chain samplers. *J. American Statist. Assoc.*, **90** 233–241.

Perron, F. (1999). Beyond accept–reject sampling. *Biometrika*, **86** 803–813.

R Development Core Team (2006). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL `http://www.R-project.org`.

Robert, C. (1995). Convergence control techniques for MCMC algorithms. *Statis. Science*, **10** 231–253.

Robert, C. and Casella, G. (2004). *Monte Carlo Statistical Methods*. 2nd ed. Springer-Verlag, New York.

Rosenthal, J. (2000). Parallel computing and Monte Carlo algorithms. *Far East J. Theoretical Statistics*, **4** 207–236.