

# Markov Chain Monte Carlo methods: Implementation and comparison

Charles S. Bos\*

Tinbergen Institute & Vrije Universiteit Amsterdam

June 21, 2004

WORK IN PROGRESS

Updated versions may appear on <http://www.tinbergen.nl/~cbos/>

## Abstract

The paper and presentation will focus on MCMC methods, implemented together in MCMCPack, an ox package which allows you to run a range of sampling algorithm (MH, Gibbs, Griddy Gibbs, Adaptive Polar Importance Sampling, Adaptive Polar Sampling, and Adaptive Rejection Metropolis Sampling) on a given posterior. Computation of the marginal likelihood for the model is also done automatically, allowing for quick and thorough comparison of models and methods.

## 1 Introduction

In a Bayesian analysis, a hurdle for many researchers is the need to implement the sampling method to derive the posterior density of the parameters in a model. For that purpose, this paper describes the MCMCPack package, an add-on for the Ox programming language of Doornik (1999). In this package, a range of sampling methods are implemented, together with algorithms to compute marginal likelihoods and convergence statistics.

The setup of the paper is as follows. First, Section 2 describes the sampling methods. Section 3 presents a small data set, on which a normal mixture regression model is fitted. As the data set is small, a standard classical maximum likelihood procedure is not well able to characterise the parameters. A Bayesian procedure can depict the uncertainty and correlation structure between the parameters with more detail, but needs to be implemented. Using MCMCPack, 6 different sampling procedures are easily ran, and the results shown, and some concluding remarks are made in Section 4

In the appendices, the ox code necessary to implement the model in a Bayesian framework is discussed in Section A, followed by an overview of the functions in the package in part B. All code in this paper is available from the website <http://www.tinbergen.nl/~cbos/>.

## 2 Sampling methods revisited

Most of the scientific or more practical research question posed can be be written in the format “What is the expected value of  $g(\theta)$ ?”, where  $g(\theta)$  may be an inflation figure, precipitation, maximum loss for a large investor, or percentage of votes for a political party, possibly depending on a set of parameters  $\theta$ . In mathematical terms, the object of interest is

$$E(g(\theta)) = \int_{\theta} g(\theta) p_{\theta}(\theta) d\theta. \quad (1)$$

---

\*This paper is largely based on the thesis Bos (2001). Many thanks go to Herman K. van Dijk and Luc Bauwens for fruitful discussion. All remaining mistakes are my own.

The basis for all Monte Carlo integration methods is the approximation of the integral<sup>1</sup> through a sample mean,

$$\mathbb{E}(g(\theta)) \approx \frac{1}{N} \sum_i g(\theta^{(i)}). \quad (3)$$

In this equation  $\theta^{(i)}, i = 1, \dots, N$  is a sample from the distribution  $p_\theta$ . As it is often not possible to sample directly from  $p_\theta$  (coined the target distribution in the following), a range of sampling methods comes into play.

## 2.1 Importance sampling

In Kloek and Van Dijk (1978) the method of importance sampling is introduced. It is aimed at calculating integrals of the form (1) when a sample  $\theta$  from the target density  $p_\theta(\theta)$  is not available. An approximating candidate density  $q_\theta(\theta)$  is used, which relates to the target according to the weight function  $w(\theta) = p_\theta(\theta)/q_\theta(\theta)$ , see figure 1. From the observation that

$$\mathbb{E}(g(\theta)) = \int_\theta g(\theta) \frac{p_\theta(\theta)}{q_\theta(\theta)} q_\theta(\theta) d\theta = \int_\theta g(\theta) w(\theta) q_\theta(\theta) d\theta, \quad (1')$$

the approximation is calculated as

$$\mathbb{E}(g(\theta)) \approx \frac{1}{N} \sum_i g(\theta^{(i)}) w(\theta^{(i)}). \quad (3')$$

The sample  $\theta^{(i)}, i = 1, \dots, N$  is drawn not from the target density  $p_\theta(\theta)$  but from the candidate density  $q_\theta(\theta)$ , adapting for the difference between the two through the use of weights in (3'). The method of integrating through importance sampling works well if the candidate density  $q_\theta(\theta)$  approximates the target closely, i.e. when the weight function  $w(\theta)$  is close to 1. In the tails of the density, this is often not attainable; care should be taken in that case to choose a candidate density with heavier tails than those of the target density, as the weights  $w(\theta)$  could get very large in the opposite case. Also, the expectation of the weight function,  $\mathbb{E}(w(\theta))$ , must be finite.

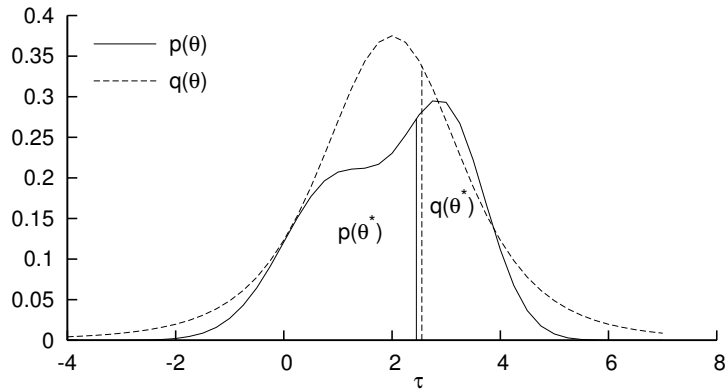


Figure 1: Sampling using the importance sampler, with target and candidate density

<sup>1</sup>When only the kernel of the density  $p_\theta(\theta)$  is known, the integral in equation (1) is divided by the integral over the kernel. This would lead to a formula

$$\mathbb{E}(g(\theta)) = \frac{\int_\theta g(\theta) \kappa_\theta(\theta) d\theta}{\int_\theta \kappa_\theta(\theta) d\theta}, \quad (2)$$

and similar straightforward modifications to subsequent formulas in this section. For simplicity of notation, the integrating constant is assumed known.

## 2.2 The Metropolis-Hastings algorithm

The origins of the Metropolis-Hastings algorithm date back to the 1950s, when the algorithm was introduced by Metropolis, Rosenbluth, Rosenbluth, Teller and Teller (1953). Hastings (1970) reintroduced it to the econometric community, though a clear exposition had to wait until Smith and Roberts (1993) and later Chib and Greenberg (1995).

The algorithm is a true sampling algorithm (like the acceptance-rejection algorithm of the previous section) in the sense that it results in a sample  $\Theta = (\theta^{(1)}, \theta^{(2)}, \dots)$  which may be used for evaluating all kinds of objective functions  $g_1(\theta), g_2(\theta)$  etc. As in the case of both the acceptance-rejection and importance sampling, the target density  $q(\theta|\theta^{(i)})^2$  is compared to a candidate density. The algorithm consists of the following steps:

1. Initialise, start with a drawing  $\theta^{(0)}$ , set  $i = 0$ .
2. Generate a candidate draw  $\theta^* \sim q(\theta|\theta^{(i)})$ .
3. Calculate the acceptance probability

$$\alpha_{\text{MH}}(\theta^{(i)}, \theta^*) = \min \frac{p_{\theta}(\theta^*)q(\theta^{(i)}|\theta^*)}{p_{\theta}(\theta^{(i)})q(\theta^*|\theta^{(i)})}, 1 \quad . \quad (4)$$

4. With probability  $\alpha_{\text{MH}}(\theta^{(i)}, \theta^*)$  set  $\theta^{(i+1)} = \theta^*$ , else retain  $\theta^{(i+1)} = \theta^{(i)}$ .
5. Increase  $i$ .
6. Repeat steps 2-5 until a sufficiently large sample is collected (see also Section 2.6).

Note that this algorithm results in a chain of drawings, where there is correlation between drawing  $\theta^{(i+1)}$  and  $\theta^{(i)}$ .<sup>3</sup> This chain is a Markov chain, depending only on the state  $\theta^{(i)}$  in the previous period.

Without the intention of being anywhere near exhaustive (see e.g. Geweke (1999) for a more elaborate exposition), some remarks can be made. When the candidate density does not depend on the state of the chain, an independence chain results, with transition probability

$$\alpha_{\text{MH}}(\theta^{(i)}, \theta^*) = \min \frac{p_{\theta}(\theta^*)q(\theta^{(i)})}{p_{\theta}(\theta^{(i)})q(\theta^*)}, 1 = \min \frac{w(\theta^*)}{w(\theta^{(i)})}, 1 \quad . \quad (5)$$

Note how in this case the acceptance probability depends on the weights which were also used in the Importance Sampler, Section 2.1. Often such an independence chain, with e.g.  $q(\theta^*) = t(\hat{\theta}, \alpha, \nu)$  a Student- $t$  density with as expectation  $\hat{\theta}$  a preliminary estimate of the mode of the posterior density and variance  $\frac{\nu}{\nu-2}\alpha^2$  works well on unimodal posterior densities.

When the candidate density is symmetric ( $q(\theta^a|\theta^b) = q(\theta^b|\theta^a)$ ), it cancels from the acceptance probability equation, leaving

$$\alpha_{\text{MH}}(\theta^{(i)}, \theta^*) = \min \frac{p_{\theta}(\theta^*)}{p_{\theta}(\theta^{(i)})}, 1 \quad . \quad (6)$$

This is the original algorithm as proposed in Metropolis et al. (1953). A special case is the candidate density which only depends on the distance between  $\theta^*$  and  $\theta$ , i.e.  $q(\theta^*|\theta) = q(\theta^* - \theta)$ . The resulting chain is known under the name of Random Walk Metropolis chain. A simple choice is to use  $\theta^* \sim \mathcal{N}(\theta, \sigma_q^2)$ ; the variance of the candidate density can be calibrated to take steps which are reasonably close to  $\theta$  such that the probability of accepting the candidate is not too low, but with a stepsize large enough to ensure sufficient mixing of the chain.

## 2.3 Gibbs sampling

The Gibbs sampler is possibly the sampling technique which is used most frequently. In the statistical physics literature it was (and still is) known as the heat bath algorithm, but Geman and Geman (1984) christened it in the mainstream statistical literature as the Gibbs sampler. It was popularised by Casella and George (1992) among econometricians, and consists of

<sup>2</sup>The candidate density may depend on the last vector of parameter  $\theta^{(i)}$  drawn in the chain. This last vector of parameters is often called the state of the chain.

<sup>3</sup>There are two sources of correlation. Firstly, when a candidate draw is rejected, the old element  $\theta^{(i)}$  is duplicated. Secondly, the candidate density may depend on the previous drawing  $\theta^{(i)}$ , resulting in correlated draws

splitting up the parameter space into blocks of parameters for which it is possible to specify the full conditionals: Let  $\theta$  be the parameter vector, and let it be subdivided into  $\theta = \{\theta_1, \dots, \theta_k\}$ . Then the algorithm proceeds as follows:

1. Initialise, start with a drawing  $\theta^{(0)}$ , set  $i = 0$ .
2. Given the  $i$ -th drawing  $\theta^{(i)}$ , the next one is found by simulating

$$\begin{aligned} \theta_1^{(i+1)} &\sim \pi(\theta_1 | \theta_2^{(i)}, \dots, \theta_k^{(i)}), \\ \theta_2^{(i+1)} &\sim \pi(\theta_2 | \theta_1^{(i+1)}, \theta_3^{(i)}, \dots, \theta_k^{(i)}), \\ &\vdots \\ \theta_k^{(i+1)} &\sim \pi(\theta_k | \theta_1^{(i+1)}, \dots, \theta_{k-1}^{(i+1)}). \end{aligned}$$

3. Increase  $i$ .
4. Repeat steps 2-3 until convergence (see also Section 2.6).

Given a starting vector of parameters  $\theta^{(0)}$  in the support of the density, the algorithm proceeds by stepwise generating each element of the next  $\theta^{(i+1)}$  from the full conditionals as above. The algorithm can be considered to ‘pass through’ intermediate points  $(\theta_1^{(i+1)}, \theta_2^{(i)}, \dots, \theta_k^{(i)})$ ,  $(\theta_1^{(i+1)}, \theta_2^{(i+1)}, \theta_3^{(i)}, \dots, \theta_k^{(i)})$ ,  $\dots$ ,  $(\theta_1^{(i+1)}, \dots, \theta_{k-1}^{(i+1)}, \theta_k^{(i)})$  towards the new element  $\theta^{(i+1)}$ .

An important concept connected with the Gibbs sampler is the concept of data augmentation, already introduced by Tanner and Wong (1987). In many situations, the posterior density  $p(\theta)$  is hard to sample from, but there is a conditional density  $p(\theta|z)$  which is easily analysed. This often occurs in models with missing or unobserved data, e.g. Tobit or Probit models. If also the distribution of  $z|\theta$  is of a known form, a Gibbs chain is easily built sampling from  $z|\theta^{(i)}$  followed by a draw of  $\theta^{(i+1)} \sim p(\theta|z)$ . Disregarding the values of  $z$  sampled during the process, the  $\theta^{(i)}$  can be shown to have the correct distribution  $p(\theta)$  (see Casella and George (1992) for details).

## 2.4 Griddy Gibbs sampler

For some models the conditional density  $p(\theta_i|\theta_{-i})$  is hard to derive or to sample from. In those cases the Griddy Gibbs sampler (Ritter and Tanner 1992, Bauwens, Lubrano and Richard 1999, Section 3.4.3.2) can offer relief.

The Griddy Gibbs sampler constructs an approximation  $\tilde{p}(\theta_i|\theta_{-i})$  to the conditional density numerically, by evaluating the (joint) posterior density on a grid over the support of values for  $\theta_i|\theta_{-i}$  (that is, keeping the conditioning parameters  $\theta_{-i}$  constant). As

$$p(\theta_i|\theta_{-i}) = \frac{p(\theta_i, \theta_{-i})}{\int_{\theta_i} p(\theta_i, \theta_{-i})}, \quad (7)$$

the conditional density of  $\theta_i|\theta_{-i}$  is proportional to the joint density.

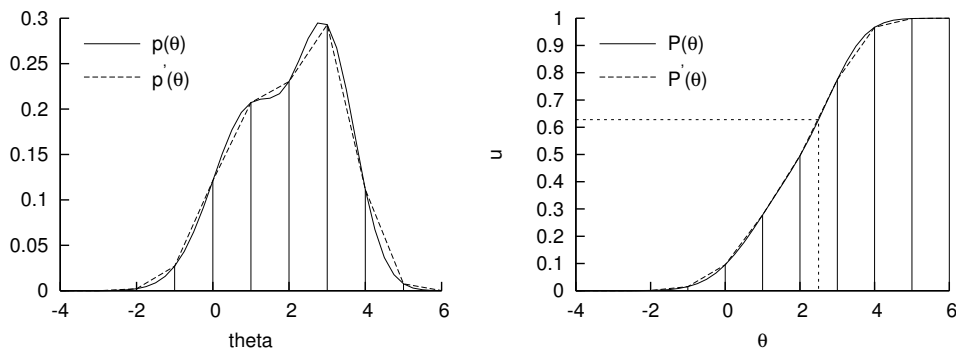


Figure 2: Sampling using the Griddy Gibbs sampler

To sample from a general density function  $p$  with cumulative distribution function  $P$ , we can take a drawing  $u$  from a uniform distribution, and apply the inverse CDF  $P^{-1}$  to arrive at a drawing  $\theta = P^{-1}(u)$  from the original distribution. This method can be used given a numerical approximation  $\tilde{P}(\theta_i|\theta_{-i})$  to  $P(\theta_i|\theta_{-i})$  to sample a new value of  $\theta_i$ . Figure 2 displays a general density function  $p(\theta)$  in the left panel, together with an approximation based on a linear spline connecting a number of support points. Using the linear approximation, the cumulative distribution  $\tilde{P}(\theta_i|\theta_{-i})$  in the right panel was constructed. For a drawing  $u = 0.62$  from the uniform density, the corresponding value of  $\theta$  can be read from the  $x$ -axis.

## 2.5 Adaptive Polar Sampling

The Metropolis-Hastings and Gibbs samplers of Sections 2.2 and 2.3 are flexible enough to sample from a broad range of posterior densities. In some situations, when e.g. no good approximating candidate density is available for the MH sampler, or when correlation in the Gibbs chain is very high, alternative sampling methods can help.

Bauwens, Bos, Van Dijk and Van Oest (2004) devise adaptive radial-based direction samplers, building forth on older integration routines like the MIXIN algorithm (Van Dijk and Kloek 1980, Van Dijk, Kloek and Boender 1985). The idea behind the algorithm can be compared to the ideas behind the Adaptive Direction Samplers (Gilks, Roberts and George 1994). The algorithm is devised to be able to sample effectively also from multimodal posterior densities, even when the number and location of the modes is not known a priori; this is a case where the MH sampler often does not converge. Compared with the Gibbs sampler, APS is not hampered by strong correlation between the parameters of the model. Little information is needed except for the posterior density function itself.

The APS algorithms split the sampling in two: Sampling is done in a transformation of the original parameter space to polar coordinates. The algorithm applies a standard Metropolis algorithm on the directions  $\eta$  of the parameter vectors, and a univariate numerical procedure on the distances  $\rho$ . The distance measure  $\rho$  and the direction  $\eta$  combine into a drawing from the parameter vector  $\theta$ , following the transition

$$(\eta, \rho) = T(\theta | \mu, \Sigma) = T_{y \rightarrow \eta, \rho}(y) \circ T_{\theta \rightarrow y}(\theta | \mu, \Sigma) \quad (8)$$

(the precise definition of the algorithm is given in Bauwens et al. (2004)).

The sampling from directions and distances is based on the observation that the marginal density of directions  $\eta$  is well behaved compared to the density of the parameters in the original space. Problematic heavy tails, multimodality, or oddly shaped support for the parameters translates into an oddly shaped density for the conditional  $\rho|\eta$ , less so for the marginal density of  $\eta$ . Therefore, Bauwens et al. (2004) propose to sample a candidate direction  $\eta^*$  from the transformed normal density (corresponding to the uniform density for  $\eta$  in the bivariate case). A Metropolis-Hastings step is applied on  $\eta$ , accepting the candidate draw  $\eta^*$  instead of the previous  $\eta^{(i)}$  with probability

$$\alpha(\eta^{(i)}, \eta^*) = \min \left[ \frac{p(\eta^*)q(\eta^{(i)})}{p(\eta^{(i)})q(\eta^*)}, 1 \right] \quad (5')$$

$$= \min \left[ \frac{\int_{\rho} p_{\theta} T^{-1}(\eta^*, \rho) |J(\rho)| \partial \rho}{\int_{\rho} p_{\theta} T^{-1}(\eta^{(i)}, \rho) |J(\rho)| \partial \rho}, 1 \right], \quad (9)$$

with  $J(\rho)$  indicating the part of the Jacobian of the transformation depending on the distance parameter  $\rho$ . Alternatively, the candidate direction  $\eta$  can be given a weight as in the importance sampler (Section 2.1). The resulting Adaptive Polar Importance Sampling (APIS) algorithm does not reject any drawings, but may collect a large number of sampled parameter vectors with low weights if the candidate density is not very precise (i.e., especially when the estimates of location and scale are not very accurate).

During the numerical evaluation of the univariate integrals  $\int_{\rho} p_{\theta} T^{-1}(\eta^*, \rho) |J(\rho)| \partial \rho$  and  $\int_{\rho} p_{\theta} T^{-1}(\eta^{(i)}, \rho) |J(\rho)| \partial \rho$ , information is collected to construct the conditional density  $p(\rho|\eta^{(i+1)})$ .<sup>4</sup> From the conditional distribution (see the method explained in Section 2.4, on

<sup>4</sup>Note that  $\eta^{(i+1)}$  is either  $\eta^*$  or  $\eta^{(i)}$ , such that the conditional density of  $\rho|\eta^{(i+1)}$  is proportional to  $p(\eta^{(i+1)}, \rho)$  which is merely a transformation from  $p_{\theta} T^{-1}(\eta^*, \rho)$ .

the Griddy Gibbs sampler) we draw a value of  $\rho$ , which together with  $\eta^{(i+1)}$  defines a point  $\theta = T^{-1}(\eta^{(i+1)}, \rho^{(i+1)})$  in the original parameter space.

Sampling new values of  $\eta$  and  $\rho$  (or, equivalently, of  $\theta$ ) continues until an improved estimate of the location and scale parameters  $\mu$  and  $\Sigma$  can be derived from the drawings, or until the sample is large enough for other purposes. As the sampling of  $\eta$  is computationally intensive due to the numerical integral over  $\rho$  that has to be evaluated, it is advisable to sample multiple values of  $\rho$  for the same direction  $\eta$ . As long as a sufficient number of different directions  $\eta$  have been sampled to cover the parameter space of the directions, this does not hamper convergence.

Practical experience tells us that good initial estimates of location and scale are not needed: Using one or more short initial sets of drawings, the estimates can easily be updated. During initial rounds, it is advisable to force acceptance in the Metropolis-Hastings step after a low (e.g. 5) number of rejections, to keep the chain moving. This leads to a sample which is not from the correct posterior distribution, but which serves well for improving the estimate of the location and scale parameters. In later rounds, forced acceptance should not (or only after a large number of rejections) occur. The structure of the APS algorithm ensures that the final sample converges to a sample from the posterior density function, irrespective of initial starting values for location and scale parameters. For a proof, see Bauwens et al. (2004).

## 2.6 On convergence

When the sampling from the Markov chain continues for a long period, eventually the drawing can be considered to be a drawing from the invariant distribution. Each subsequent drawing is also a drawing from the invariant distribution, but a dependent one: There can be a strong correlation between successive drawings. The following measures can be taken to counter this correlation:

1. **Single chain:** In some cases, correlation is not much of a practical problem. As long as the researcher realises that the sample of size  $N$  is a dependent sample, and therefore only corresponds in information content to a sample of  $N_1 < N$  drawings, results can be sufficiently good. With strong correlation, a larger sample is needed than in the case when the correlation is relatively weak.
2. **Multiple chains:** A range of  $N$  different Markov chains of length  $N_2$  can be run, saving only the last drawing of the chain. If the length  $N_2$  is large enough for the chain such that the drawing has converged to the target distribution, the  $N$  parameter vectors at the end of each of the chains are independent.
3. **Interspersed chain:** An intermediate position is obtained when one long single chain of length  $N \times N_3$  is run, saving only one drawing out of every  $N_3$ . The  $N$  resulting drawings have lower correlation than the sample from the single chain itself, and the computational overhead is less than with the multiple-chain method, as usually  $N_3 \ll N_2$  is sufficient to get a considerable reduction in correlation.

In the implementation in MCMCPack, the solutions of the single, possibly interspersed, chain is available. In most situations running multiple chains using only the last element is overdone.

For assessing convergence, several statistics have been devised (Geweke 1992, Kim, Shephard and Chib 1998, Yu and Mykland 1998). The basic idea behind most of them is to compare moments of the sampled parameters at different parts of the chain. The implementation of Yu and Mykland (1998) is most practical. They propose to consider a plot of the CUSUM path with

$$\text{CUSUM}_t = \frac{1}{t\sigma_\theta} \sum_{i=1}^t (\theta^{(i)} - \mu_\theta), \quad t = 1, \dots, N. \quad (10)$$

The cumulative sum of the drawings is adapted for the empirical mean and standard deviation of the complete sample. This is not necessary, but can be convenient to compare the CUSUM plots for different parameters. A plot of  $\text{CUSUM}_t$  against time which diverges from zero for a prolonged period of time is an indication of bad convergence.

Apart from the CUSUM plot it is often convenient to check convergence by visual means from a plot of the drawings themselves, of a moving average of e.g. the last 100 drawings, and of the cumulative or running mean of drawings  $1, \dots, t$ .

Geweke (1992) promotes the use of the Relative Numerical Efficiency (RNE) as a measure for the quality of a correlated sample. It compares the empirical variance of the sample with a correlation-consistent variance estimator,

$$\text{RNE} = \frac{\sigma_{\hat{\theta}}^2}{\sigma_{\text{NW},q}^2}, \quad (11)$$

where  $\sigma_{\hat{\theta}}^2$  is a direct estimator of the variance, and  $\sigma_{\text{NW},q}^2$  is the Newey-West (Newey and West 1987) variance estimator taking the correlation up to lags of  $q\%$  of the size of the sample into account. Practical values for  $q$  can be 4, 8 or even 15%. The implementation of Kim et al. (1998) computes the RNE as

$$\hat{R}_{B_m} = 1 + \frac{2B_m}{B_m - 1} \sum_{i=1}^{B_m} K(i/B_m) \hat{\rho}(i)$$

with  $K(j)$  the Parzen kernel,  $B_m$  the bandwidth and  $\hat{\rho}(i)$  the  $i$ th order autocorrelation. This is the version implemented in the MCMCPack package.

Another practical method of assessing convergence is to monitor the estimates of location and scale. Assume an initial estimate of the location  $\mu_0$  is given, and that the sample run leads to estimates  $\hat{\mu}, \hat{\Sigma}$  of location and scale. The Mahalanobis distance between the previous and present location is

$$d_{\text{Mah}} = (\hat{\mu} - \mu_0) \hat{\Sigma}^{-1} (\hat{\mu} - \mu_0) \quad (12)$$

If the location estimate  $\mu_0$  is updated between consecutive runs,  $d_{\text{Mah}}$  eventually drops to zero, as the location should no longer change. When  $d_{\text{Mah}}$  is close to zero, or, as it is a random quantity, when it no longer drops considerably between consecutive runs, the algorithm has probably found the stable density.

Note that there is never a true guarantee that full convergence has taken place; in practice however the rule to stop sampling new runs (with the Metropolis-Hastings, Importance, Adaptive Polar, and Adaptive Polar Importance sampling, as they depend strongly on initial location and scale estimates) whenever the Mahalanobis distance between the previous and present location is not changing more than a fixed fraction between runs, is a good rule of thumb.

### 3 Example: Contaminated normal density estimation

As an example of the possibilities of the package, reconsider the so-called stack loss data of Brownlee (1965). The data set was previously analysed (a Atkinson 1985, Rousseeuw and Van Zomeren 1990, Justel and Peña 1996, e.g.) and underlies a marginal likelihood calculation comparison in Bos (2002). It consists of 21 observations on three explanatory variables and a response variable relating the fabrication of nitric acid from ammonia, by oxidation. The Among the observations, several seem to be outliers, or might stem from a density with higher variance than the others. Hence, one of the models proposed is a normal regression model with a normal mixture distribution for the disturbances. Following notation in Justel and Peña (1996),

$$y_i = X_i \beta + u_i \quad (13)$$

$$u_i \sim (1 - \alpha) \mathcal{N}(0, \sigma^2) + \alpha \mathcal{N}(0, k^2 \sigma^2) \quad (14)$$

The corresponding likelihood is can be multimodal (in  $\alpha$  and  $k$  especially, when the regions for these parameters are not restricted) and the displays strong correlations between the parameters. Hence, standard optimisation routines for finding maximum likelihood estimates of the parameters might not be sufficient, and a Bayesian sampling procedure to derive a full distribution of the parameters is warranted.

A direct approach combines the likelihood function from (14) with a prior function  $\pi(\beta, \sigma, k, \alpha)$ , to obtain a posterior  $P(\beta, \sigma, k, \alpha | Y) \propto \mathcal{L}(\beta, \sigma, k, \alpha; Y) \pi(\beta, \sigma, k, \alpha)$ . Sampling can be done directly from this posterior using the MH, IS, APS, APS or (Griddy) Gibbs algorithms described before.

Alternatively, the model can be augmented with a set of indicators  $\delta_i = 1$  if observation  $i$  stems from the high-variance disturbance density, or  $\delta_i = 0$  otherwise. This introduces  $n$  parameters  $\delta_i$  which have to be sampled as well, but simplifies the conditional disturbance densities  $u_i|\delta_i$  in the model to a great extent. In a Gibbs sampling scheme, the parameters  $\delta_i|\beta, \sigma, \alpha, k, Y$  are easily drawn from the Bernoulli density with

$$P(\delta_i = 1|\beta, \sigma, \alpha, k, Y) = \frac{\alpha \exp -\frac{u_i^2}{2k^2\sigma^2}}{\alpha \exp -\frac{u_i^2}{2k^2\sigma^2} + (1 - \alpha)k \exp -\frac{u_i^2}{2\sigma^2}} \quad (15)$$

Conditional on the indices, also the posterior of  $\beta|\sigma, \alpha, k, \delta, Y$  is a normal density, assuming a conjugate normal prior density  $\pi(\beta) \sim \mathcal{N}(\beta_0, \Sigma_0)$  is used. It is

$$\begin{aligned} P(\beta|\sigma, \alpha, k, \delta, Y) &\sim \mathcal{N}(\tilde{\beta}, \tilde{\Sigma}) \\ \tilde{\beta} &= \tilde{\Sigma}(\tilde{\Sigma}^{-1}\hat{\beta} + \Sigma_0^{-1}\beta_0) & \tilde{\Sigma} &= (\tilde{\Sigma}^{-1} + \Sigma_0^{-1})^{-1} \\ \hat{\beta} &= (X'VX)^{-1}X'VY & \tilde{\Sigma} &= \sigma^2(X'VX)^{-1} \end{aligned}$$

with  $V$  a diagonal matrix with elements  $v_{ii} = k^{-2\delta_i}$ .

For  $\sigma$  and  $\alpha$ , the conditional densities are not as easily derived, but the implementation is able to sample these using Griddy Gibbs steps, given the (augmented) likelihood function and the prior.

Table 1: Prior densities

Parameter	Density	$\mu$	$\sigma$	L	U
$\beta$	$\mathcal{N}$	0	1	-30	30
$\sigma$	$\text{IG}(\alpha = 3, \beta = .05)$	3	1.1	.5	10
$k$	U	10.5	$\frac{19}{12}$	1	20
$\alpha$	$\text{Beta}(0.5, 1)$	0.33	0.3	.02	1

In the application, a set of prior densities as specified in Table 1 are used. The upper and lower bounds mentioned serve to guarantee existence of the (conditional) posterior densities, and as integration bound for the APS, APIS and Griddy Gibbs algorithms. For the parameter  $k$ , the prior is uniform on a bounded region. The lower bound of 1 ensures that we model a variance *increase*, whereas the upper bound limits the increase in variance to acceptable values. The bound on parameter  $\alpha$  is implemented to keep the model observable: If  $\alpha \equiv 0$ ,  $k$  would no longer be defined. The other bounds are in practice not limiting.

Simulating from the densities is done using the full set of implemented method, i.e. Metropolis-Hastings sampling, importance sampling, adaptive polar sampling, adaptive polar importance sampling, griddy Gibbs sampling and partially griddy Gibbs sampling on the augmented model. Where applicable, a Student- $t$  density with 4 degrees of freedom is used as candidate/importance density. Initially, the candidate density is transformed to have as a location the least squares estimates of  $\beta$  and  $\sigma$ , with  $k = 1.5$  and  $\alpha = 0.1$ , and scale matrix equal to the unit matrix. Sampling is continued until 10.000 accepted parameter vectors have been collected, after a burn-in period of 1.000 accepted parameter vectors. If the location and scale of the sample has changed more than 10% according to the Mahalanobis-distance, the sampling is repeated with updated location and scale, for the MH, IS, APS and APIS algorithms. Note that this setup is chosen solely to give a quick comparison between the algorithms; for more serious modelling the sample size would have to increase considerably, and more care be given to the convergence of each of the algorithms.

Figure 3 displays the marginal posteriors, for each of the sampling algorithms. It is seen that in this case the augmented Gibbs sampler, but also the direct griddy Gibbs sampler, display the smoothest posterior densities, and the adaptive polar sampler comes close. The importance sampling algorithm has more problems coping with the longer tails in the density of especially parameter  $k$ , but also other parameters are inflicted with a few outlying draws obtaining higher weights, and hence disturbing the estimated density.

Table 2 reports a set of posterior statistics. First, for the augmented Gibbs sampler, the mode, mean and standard deviation of the posterior are given. These statistics are

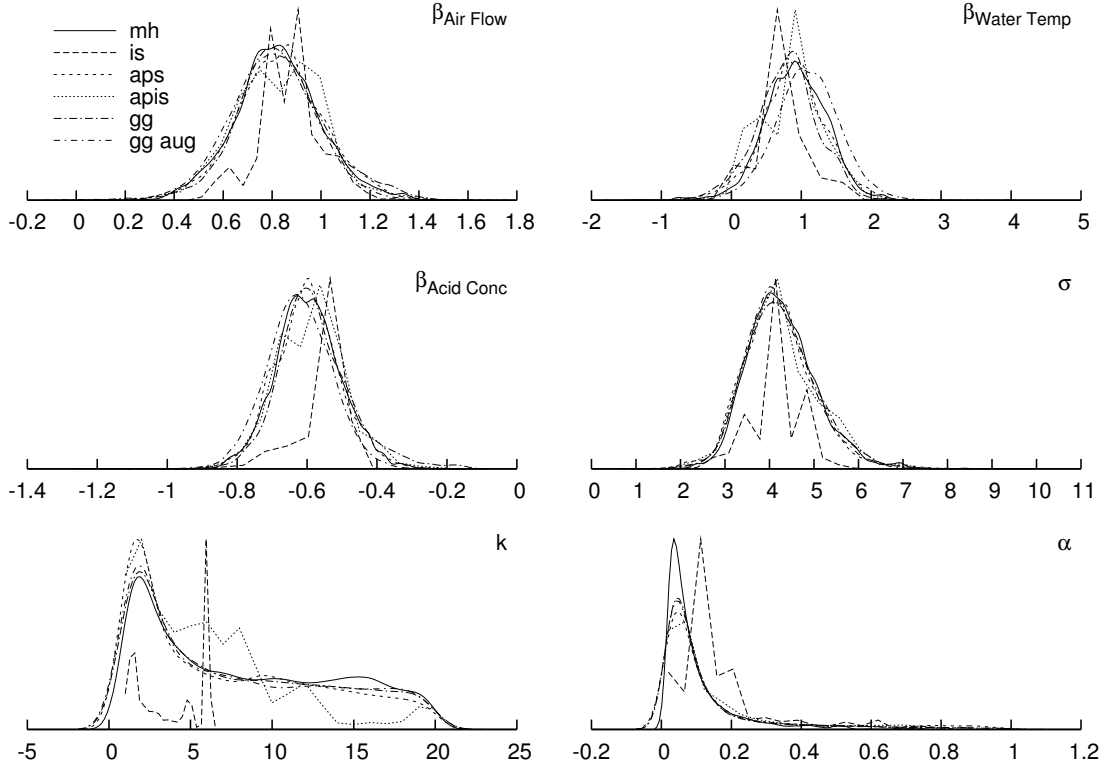


Figure 3: Posterior densities of the stack-loss parameters. Note that the densities of the importance sampling algorithm are put on a different scale, for matters of presentation.

Parameter	AugGG			MH	IS	APS	APIS	GG	AugGG	
	Mode	Mean	st.dev							
$\beta_{Air\ flow}$	0.794	0.801	0.174	14.42	0.78	25.98	11.55	65.06	0.82	
$\beta_{Water\ temp}$	0.990	1.092	0.456	19.46	1.23	32.01	5.68	112.13	0.94	
$\beta_{Acid\ conc}$	-0.631	-0.623	0.092	20.90	1.22	11.23	9.12	50.41	0.93	
$\sigma$	4.011	4.210	0.761	19.06	0.87	28.25	5.00	8.42	5.03	
$k$	1.817	7.471	5.842	18.79	0.90	24.85	8.60	0.84	11.04	
$\alpha$	0.042	0.132	0.176	83.75	0.90	4.66	13.23	2.80	39.54	
Acc. rate				0.27	0.54					
ML				-66.02	-55.73	-66.17	-64.23	-65.56		
Time				4:53.49	16.11	7:52.66	2:17.10	9:55.62	3:07.55	

roughly similar for the other samples. The estimates indicate that there is some evidence of a small fraction (0.04) of the sample stemming from a density with a twofold higher standard deviation, but with large uncertainty especially over the parameter indicating the increase in variance. The uncertainty is quite in line with the expectation, as the sample size of 21 does not allow for very strong conclusions.

Of more interest than the parameter estimates themselves are the relative numerical efficiencies (RNE, see e.g. Kim et al. 1998), which indicate the loss in efficiency of the sample as compared with a truly independent sample. As the importance sample is independent by definition, its RNE-values are close to 1, but in this case this is not a sign of a high quality posterior sample, as is seen from Figure 3.

The augmented (grid) Gibbs sampler uses most information on the posterior density, in the form of the prespecified conditional densities of  $\delta, \beta$ . The result is a swiftly running sampling algorithm, with very little correlation in  $\beta^5$  and reasonable behaviour in the other parameters. Of this density, the posterior density plot is smoothest, indicating good mixing of the sample over the full parameter space.

The MH algorithm is able to sample from this density, but with a relatively low acceptance rate; other candidate densities could be specified to further improve the sample. Especially in  $\alpha$ , correlations are higher than in any other sampling algorithm.

The APS algorithm has a smoothness of the posterior density estimates which is comparable to the Gibbs algorithms. Its efficiency is not very high, though especially on  $\alpha$  it performs well. This is a consequence of the APS algorithm being devised to handle parameters with odd-shaped posterior densities well; on other parameters it behaves roughly like a Metropolis-Hastings algorithm.

The APIS algorithm is again of the family of importance sampling algorithms, and has the same drawback that some drawings receive considerably higher weights.

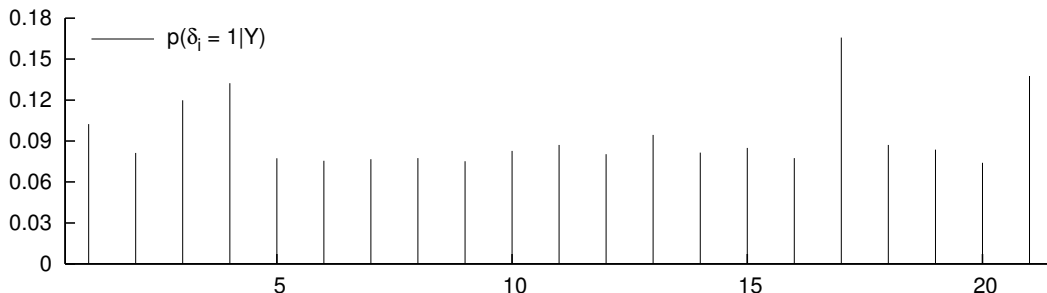


Figure 4: Posterior probabilities  $p(\delta_i = 1 | Y)$

An advantage of the augmented Gibbs sampler is that it also collects information on the posterior probabilities of each of the observations to stem from the alternative, high variance disturbance density. Figure 4 displays the probabilities, indicating that observations 3, 4, 17 and 21 might well derive from a density with higher variance. On the other hand, given the small data set, this information is not very decisive, with probabilities jumping to 0.12-0.17 from the overall background probability of around 0.08. These results are largely in correspondence with earlier findings. Depending on the method, sometimes observation 1 is labelled outlier instead of observation 17.

Table 2 also reports estimates of the marginal likelihood for each of the drawn samples, using a kernel approximation to the posterior density function. See Bos (2002) for an overview of possible methods of computing marginal likelihood statistics, and the method with which they are implemented in the package. As the statistics have been computed at the posterior mode, which is considerably different for the importance sample, some difference in marginal likelihood values is found as well. Note that the marginal likelihood is not calculated with the augmented Gibbs sampler; for this purpose, all the augmentation parameters  $\delta_i$  would have

<sup>5</sup>This is mostly due to the joint sampling of the three parameters of  $\beta$ , instead of sampling each  $\beta_i$  separately from its conditional density.

to be taken up into the parameter vector, leading to large memory requirements to store the sample and to construct the posterior density.

In Appendix A an outline of the needed ox-code to implement these computations is discussed.

## 4 Conclusions

This paper swiftly discussed a range of sampling methods, together with algorithms to compute the marginal likelihoods of models, useful for a Bayesian analysis of a model at hand.

Using the Ox package `MCMCPack` the implementation, which used to be a hurdle for many researchers, is simplified to a matter of specifying the prior, likelihood and posterior routines. A range of methods is available, both standard Metropolis-Hastings samplers, brute force Griddy Gibbs samplers, or more advanced Polar Sampling algorithms.

With the package a mixture model on the stack-loss data set was estimated, in order to find the fraction of observations which might stem from a high-variance density. In order to get an idea of the outlying observations the data set can be augmented with parameters indicating which element could stem from the higher-variance disturbance density. In this case, Bayesian sampling using the augmented Gibbs method leads to estimates for the densities of the augmentation parameters, and to a judgement for each of the observations if it is in line with the others. In `MCMCPack`, such augmented sampling is easily implemented.

## A Implementing MCMC on the stackloss dataset

This appendix explains how to implement the model described in Section 3. The full program and data file can be downloaded from <http://www.tinbergen.nl/~cbos/>.

To implement the model, several steps have to be taken. First, for all sampling routines the prior and likelihood, together forming the posterior, have to be defined. Listing 3 starts off with implementing the prior densities as defined in Table 1. These functions are specified conforming to the standard of the Ox `MaxBFGS` target function, even though the score and Hessian arguments are never used. The function in our case extracts the parameters from the parameter vector, and successively adds the logarithm of the normal, the inverted gamma, the uniform and the beta priors together. For the inverted gamma density, use is made of a separate routine from the file `oxprobig.ox`, it is not part of the standard version of Ox.

Table 3: Defining the prior

```
#include "include/oxprobig.ox"          // Include the inv gamma density
...
AvgLnPriorStack(const vTheta, const adLnAvgPr, const avS, const amH)
{
  decl iP, vBeta, dS, dK, dAlpha, dLnPr;

  iP= rows(s_MX_mX);
  vBeta= vTheta[:iP-1]; dS= fabs(vTheta[iP]);
  dK= vTheta[iP+1]; dAlpha= vTheta[iP+2];

  dLnPr= -0.5*vBeta'vBeta - 0.5*iP*log(M_2PI); // N(0, 1) on Beta
  dLnPr+= lndensigamma(dS, 3, .05);          // IG(3, .05) on sigma
  dLnPr+= 0;                                // U(a, b) on k
  dLnPr+= log(densbeta(dAlpha, 0.5, 1));     // Beta(0.5, 1) on alpha

  adLnAvgPr[0]= dLnPr/columns(s_MX_mX);

  return !ismissing(adLnAvgPr[0]);
}
```

For the likelihood, in Listing 4, again the parameters are extracted. The final formula follows exactly the specification in (14). Again, the average loglikelihood is returned. The routine for the posterior adds the log-prior and log-likelihood together, and returns the result.

In the main program, in Listing 5, first the data is prepared, and an initial location for the parameters is derived from the OLS estimates, in `vMu`. Then, a set of lines initialises the `MCMCPack` class, setting the method to Metropolis-Hastings sampling, preparing the parameter names and choosing a candidate density. Here, a Student- $t$  density is chosen with a scale matrix equal ( $c_t = 1$ ) to the overall estimate of the covariance matrix, and  $\nu = 4$  degrees of freedom.

The settings for the integration method are not used for the MH sampler, but only if the method is changed to `MC_APS`, `MC_APIIS` or `MC_GG`. Default settings can be used, here (see the manual in Appendix B) an adaptive QAG routine is chosen, changing the bounds of the integration region for greater efficiency, searching for a maximum before integrating in order to better scale the calculations, and with adaptive polar sampling separating the integral over the negative and positive beam of  $\rho$  (see Section 2.5).

Then, the bounds for the parameters are given, followed by the sample size. Two runs are specified, an initial run of size 1000 and a second one of size `iSize`, with 10% of a burn-in sample, not interspersing the sample at all, forcedly moving on to a new parameter vector after 100 rejections in a Metropolis-Hastings step, and sampling `iSize/10` directions when the adaptive polar (importance) samplers are used.

Each iteration is repeated if the Mahalanobis distance between the previous and present location drops by more than 10%, and a location for the output files is specified.

Table 4: Defining the likelihood and posterior

```

AvgLnLiklStack(const vTheta, const adLnAvgP, const avS, const amH)
{
  decl iP, vBeta, dS, dK, dAlpha, vU, vL, vU2s;

  iP= rows(s_MX_mX);
  vBeta= vTheta[:iP-1]; dS= fabs(vTheta[iP]);
  dK= vTheta[iP+1]; dAlpha= vTheta[iP+2];

  vU= s_MX_vY - vBeta'*s_MX_mX;
  vU2s= -0.5*sqr(vU/dS);
  vL= (1-dAlpha)*exp(vU2s)
      +dAlpha*exp(vU2s/sqr(dK))/fabs(dK);
  adLnAvgP[0]= meanr(log(vL))-0.5*log(M_2PI)-log(dS);

  return !missing(adLnAvgP[0]);
}

AvgLnPostStack(const vTheta, const adLnAvgP, const avS, const amH)
{
  decl ir, dLnPr, dLnLikl;

  dLnPr= dLnLikl= M_NAN;
  ir= AvgLnPriorStack(vTheta, &dLnPr, 0, 0) &&
      AvgLnLiklStack(vTheta, &dLnLikl, 0, 0);
  adLnAvgP[0]= dLnPr + dLnLikl;

  return ir;
}

```

All that remains is the specification of our posterior density. After this, the main simulating routine can be called, which automatically prepares output in the specified location. If requested, the log-marginal likelihood is computed easily enough using the `Marglik` routine.

Note that this specification did not yet use the data augmentation, and the prespecified conditional densities. On the website of MCMCPack, at <http://www.tinbergen.nl/~cbos/>, the full programs can be found to also implement these conditional densities.

Table 5: The main program

```
#include <oxstd.h>           // Include the Ox standard library header
#include "include/oxprobig.ox" // Include the inv gamma density
#include <packages/gnudraw/gnudraw.h> // Include graphics capabilities
#include <packages/mcmcpack2/mcmcpack.h> // Include the sampling package

// Static declaration
static decl s_MX_mX, s_MX_vY;

main()
{
    decl iMethod, iSize, vBeta, dS, vMu, mS2, mTheta, vW,
        mcmc, mX, mY, mRNE, dML;

    // Initialise
    iSize= 10000;
    mX= loadmat("data/stackloss.mat");
    s_MX_mX= mX[:2] []; s_MX_vY= mX[3] [];
    olsr(s_MX_vY, s_MX_mX, &vBeta);
    dS= sqrt(varr(s_MX_vY - vBeta*s_MX_mX));
    vMu= vBeta' |dS| 1.5|.1;
    mS2= unit(sizerc(vMu));

    // Use package
    mcmc= new MCMCPack();
    mcmc.SetMethod(MC_MH);
    mcmc.SetParNames({"Air Flow", "Water Temp", "Acid Conc", "Sigma", "k", "Alpha"});
    mcmc.SetCandidate(MC_CSTUD, {1, 4});
    mcmc.SetIntegration(1, 1, TRUE, TRUE);
    mcmc.SetLimits(<-30, 30; -30, 30; -30, 30; // Bounds on Beta
                  0.5, 10; // Sigma
                  1, 20; // k
                  0.02, 1>); // Alpha

    mcmc.SetSample(1000~iSize, .1*iSize, 0, 100, .1*iSize);
    mcmc.SetMahalanobisFraction(0.9);
    mcmc.SetLocationScale(&vMu, &mS2, sizerc(s_MX_vY), FALSE);
    mcmc.SetOutput("excl/mcstack");

    mcmc.SetPosterior(AvgLnPostStack);
    mcmc.Simulate();

    // Perform marginal likelihood computation
    dML= mcmc.Marglik(MC_MLKERN);
    print ("Marginal likelihood according to the kernel method", dML);
}
```

## B Ox function calls

Table 6: Sampling methods

MC_MH	Metropolis Hastings
MC_IS	Importance Sampling
MC_APS	Adaptive Polar Sampling
MC_APIS	Adaptive Polar Importance Sampling
MC_GG	Griddy Gibbs Sampling

Table 7: Candidate densities

MC_CNORM	Normal candidate density (with location equal to preset value, scale a multiplication of last scale)
MC_CSTUD	Student-t candidate density with <code>df</code> degrees of freedom, (with location equal to preset value, scale a multiplication of last scale)
MC_CRW	Random walk candidate density with covariance matrix proportional to the last estimate of the covariance matrix
MC_CUSER	User specified candidate density

Table 8: Marginal likelihood computation methods

MC_MLKERN	Use a kernel approximation to the posterior density, to compute the marginal likelihood as the difference between the log-density and log-posterior kernel at a specific location
MC_MLLP	Use a LaPlace approximation to the posterior density, to compute the marginal likelihood as the difference between the log-density and log-posterior kernel at a specific location
MC_MLHM	Compute the likelihood over all sampled data points, and compute the marginal likelihood as the harmonic mean of these densities.
MC_MLGG	Use a series of Gibbs chains, and derive the the marginal likelihood from the conditional densities.
MC_MLINT	Use a pure multidimensional numerical integration to compute the the marginal likelihood.
MC_MLPR	Sample from the prior and average over the corresponding values of the likelihood to compute the marginal likelihood.

## Cusum

```
MCMCPack::Cusum(const bAdapt)
```

```
MCMCPack::Cusum(const bAdapt, const viSum)
```

`bAdapt`

in: Boolean, indicating if Cusum plots should be adapted to mean and variance of the sample

`viSum`

in: integer, or vector of integers, with length of past to account for in Cusum plots. Default= 100.

*No return value*

Prepares and shows a Cusum plot according to Yu and Mykland (1998).

## GetDraws

`MCMCPack::GetDraws(const amTheta, const avW)`

`amTheta`

out: Pointer to matrix of size `k x nTheta` with sampled parameter values.

`avW`

out: if !0 on input, `1 x nTheta` vector with weights, for samplers using importance sampling, scaled such that the weights sum to 1. For non-importance samplers, equal weights are returned.

*Return value*

1 if a sample was recently run, 0 otherwise.

Return the draws of the last run of the sampler

## GetLocationScale

`MCMCPack::GetLocationScale()`

*Return value*

Returns two arrays `avMu` and `amS2` with both the first, initial, and the updated versions of locations and scale matrices of intermediate rotations.

Extracts the location and scale matrices of successive rotations.

## GetPackageName

`MCMCPack::GetPackageName()`

*Return value*

String with name of package, in this case "MCMCPack"

## GetPackageVersion

`MCMCPack::GetPackageVersion()`

*Return value*

Integer, version number of "MCMCPack" times 100

## GetParNames

`MCMCPack::GetParNames()`

*Return value*

`asNames`: array of strings, parameter names

Return the names of the parameters being sampled

## GetRNE

`MCMCPack::GetRNE()`

*Return value*

`mRNE`: `iK x nTap` matrix with relative numerical efficiencies, or 0 when no sample is available

Return the relative numerical efficiency according to Kim et al. (1998), with

$$\hat{R}_{B_m} = 1 + \frac{2B_m}{B_m - 1} \sum_{i=1}^{B_m} K(i/B_m) \hat{\rho}(i)$$

with  $K(j)$  the Parzen kernel,  $B_m$  the bandwidth and  $\hat{\rho}(i)$  the  $i$ th order autocorrelation. The bandwidth, i.e. the number of drawings used in the computations, is set with `SetTaper`.

## GetTaper

`MCMCPack::GetTaper()`

*Return value*

`vTaper`: 1 x `nTap` vector with tapering perunages

Return the values used in tapering, in computing the relative numerical efficiencies

## SetCandidate

`MCMCPack::SetCandidate(const iCand, const xCand);`

`iCand`

in: Either `MC_CNORM`, `MC_CSTUD`, `MC_CRW` or `MC_CUSER`, see also table  
Candidate densities

`xCand`

in: Meaning depends on `cForm`. If `cForm` is  
`MC_CNORM`

double `dFact`

`MC_CSTUD`

vector with `dFact` and degrees of freedom of Student-t density

`MC_CRW`

double `dFact`

`MC_CUSER`

function of format

`fnRndCand(const iR, const vTheta, const vMu, const mCS2,  
const amCand, const avLnDens)`

with inputs

`iR`

in: integer, number of vectors to be drawn

`vTheta`

in: `iK` x 1 vector of last draw

`vMu`

in: `iK` x 1 vector of location parameter

`mCS2`

in: `iK` x `iK` matrix with choleski decomposition of scale matrix

and outputs

`amCand`

out: `iK` x `iR` matrix of random vectors drawn from candidate density

`avLnDens`

out: 1 x `iR` vector with logarithm of density in `amCand`.

*No return value*

Select the functional form of the candidate density to be used. The value `dFact` specifies the multiplication factor for the choleski decomposition of the last covariance matrix. For `MC_CNORM`, `MC_CSTUD` a value slightly larger than 1 would be logical, for `MC_CRW` a small value e.g. 0.05 is better.

Note that the matrix `mCS2` may be singular, when only a subset of parameters should be sampled as indicated through the `SetConditional(vInd, -1)` function.

## SetConditional

`MCMCPack::SetConditional(const vInd, const fnRanCondGG)`

`MCMCPack::SetConditional(const vInd, const fnRanCondGG, const fnDensCondGG)`

**vInd**

in: **iI** vector with indices into the parameter vector, indicating which parameters are sampled from this conditional density.

**fnRanCondGG**

in: function which samples from the conditional density for elements **vInd**. Function should be of the format

**ir= fnRanCondGG(const vInd, const avP)**

The function should replace the elements **avP[0][vInd]** with newly sampled elements. If, instead of a function, a number (e.g. -1) is passed along to **SetConditional**, the parameter is skipped in the simulation.

**fnLnDensCondGG**

in: (optional) function which computes the logarithm of the conditional density for elements **vInd**, conditional on the other elements. The function should be of the format

**dLnDens= fnLnDensCondGG(const vInd, const vP)**

The function should compute the logarithm of the conditional density of the elements **vP[vInd]**, conditional on the other elements. Note that the conditional density should contain all integrating constants.

The function is only used for computing the marginal likelihood using the Gibbs method, see **MargLik(MC\_MLGG, ...)**.

*Return value*

1 if everything went correctly, 0 in case of problems

Indicate the sampling function from the analytic conditional density for the (Griddy) Gibbs sampler.

## SetDebug

**MCMCPack::SetDebug(const iDebug)**

**iDebug**

in: integer, level of debug information.

*No return value*

Set the level of debugging information to be generated. Default is 0, no extra information.

## SetEps

**MCMCPack::SetEps(const dEpsAbs, const dEpsRel)**

**dEpsAbs**

in: double, absolute precision used by **QuadPack** in **MCMCPack**, default 0

**dEpsRel**

in: double, relative precision used by **QuadPack** in **MCMCPack**, default 1e-2

*No return value*

Set the level of precision during the integration routines.

## SetGraphs

**MCMCPack::SetGraphs(const bShowGraphs)**

**bShowGraphs**

in: boolean, indicating if graphs should be shown (default=TRUE)

*No return value*

Indicate if intermediate graphics should be shown on the screen

## SetInfo

`MCMCPack::SetInfo(const iInfoRep)`

`iInfoRep`

in: integer, number of iterations between printing info. Default= 1000 (accepted)

`drawings`

*No return value*

Change the setting of the number of iterations between printing a report on the expected time until termination.

## SetIntegration

`MCMCPack::SetIntegration(const iAdaptive, ...)`

`MCMCPack::SetIntegration(const iAdaptive, const iBounds, const bSMax,  
const bSeparate)`

`iAdaptive`

in: integer, indicating the method of integration

0: non-adaptive QNG (default)

1: adaptive QAG

2: iterative Simpson

`iBounds`

in: integer, indicating change of bounds, with

0: No change

1: Change both bounds (default)

2: Use integration from  $-\infty$  to  $+\infty$

`bSMax`

in: boolean, search maximum? (default= TRUE)

`bSeparate`

in: boolean, indicating if integration should be separated (fixing  $p(\eta|\rho = 0)$  at 0) or not. Default is true, only used for APS/APIS sampling.

*No return value*

Specify the method of integration.

## SetLikelihood

`MCMCPack::SetLikelihood(const fnLikelihood)`

`fnLikelihood`

in: function, of same format as in MaxBFGS

*No return value*

Specify the function which returns the *average* loglikelihood, where the averaging is done with respect to the number of observations specified in `SetLocationScale`.

## SetLimits

`MCMCPack::SetLimits(const mLUBounds)`

`MCMCPack::SetLimits(const mLUBounds, const mBC)`

`MCMCPack::SetLimits(const mLUBounds, const mBC, const bChangeLU)`

`mLUBounds`

in:  $iK \times 2$  matrix with bounds on the parameters.

`mBC`

in:  $iR \times iK+1$  matrix, consisting of  $iR \times iK$  matrix `mB` with a  $iR \times 1$  vector `mC` added, indicating the linear restraints `mB*vTheta <= mC`

`bChange`

in: Boolean, if `TRUE` the bounds `mLUBounds` are adapted between rotations to the average between the (empirical bounds  $\pm 10$ bounds, in such a way that the original bounds are never enlarged. Default= `FALSE`).

*No return value*

Set the limits for the sampling using APS/APIS/Griddy Gibbs

## SetLocationScale

```
MCMCPack::SetLocationScale(const avMu, const amS2, const iN,  
                           const bOptimize, ...)
```

```
MCMCPack::SetLocationScale(const avMu, const amS2, const iN,  
                           const bOptimize, const asNames)
```

`avMu`

in: (column) vector, initial value of mean

`amS2`

in: matrix, initial value of covariance

`iN`

in: Integer, number of observations contained in the posterior.

`bOptim`

in: boolean, indicating if initial values should be used as a starting point for an optimization to find the posterior mode.

`asNames`

in: (optional) array of strings with names of parameters

`avMu`

out: vector, initial value of mean used for sampling (unchanged if `bOptim` was false)

`amS2`

out: matrix, initial value of covariance used for sampling (unchanged if `bOptim` was false)

*Return value.*

Value according to `MaxBFGS`, or `MAX_CONV` if `bOptim` was false

Provide a starting point for the sampler

## SetMahalanobisFraction

```
MCMCPack::SetMahalanobisFraction(const dFrac)
```

`dFrac`

in: double, the fraction of improvement in the Mahalanobis distance that warrants repeating a rotation, default= 0.5

*No return value*

## SetMethod

```
MCMCPack::SetMethod(const iMethod)
```

`iMethod`

in: Method, one of `MC_MH`, `MC_IS`, `MC_APS`, `MC_APIS`, `MC_GG`, see table MCMC sampling Methods

*No return value*

Choose the method of simulation

## SetOutput

`MCMCPack::SetOutput(const sResbase)`

`MCMCPack::SetOutput(const sResbase, const bRewrite)`

`sResbase`

in: string, base-name of output files

`bRewrite`

in: boolean, optional argument, indicates if output file is rewritten (default is FALSE)

*No return value*

Indicate the base-name of the output files. By default, output is written to files with name `mcmc<method>`. If `bRewrite=TRUE` (default), textual output file is rewritten.

## SetParNames

`MCMCPack::SetParNames(const asNames)`

`asNames`

in: array of strings, parameter names

*No return value*

Set the names of the parameters being sampled

## SetPosterior

`MCMCPack::SetPosterior(const fnPosterior)`

`fnPosterior`

in: function, of same format as in MaxBFGS

*No return value*

Specify the function which returns the *average* log-posterior, where the averaging is done with respect to the number of observations specified in `SetLocationScale`.

## SetPrior

`MCMCPack::SetPrior(const fnPrior)`

`fnPrior`

in: function, of same format as in MaxBFGS

*No return value*

Specify the function which returns the *average* log-prior, where the averaging is done with respect to the number of observations specified in `SetLocationScale`.

## SetRanPrior

`MCMCPack::SetRanPrior(const fnRanPrior)`

`fnRanPrior`

in: function, of format as in

`mU= fnRanPrior(iR);`

*Return value*

`iK x iR` matrix with `iR` draws from the prior.

Specify the function which returns `iR` draws from the prior

## SetSample

`MCMCPack::SetSample(const vRep, ...)`

`vRep`

in: vector of size `iRot`, or scalar, number of repetitions of drawing sampled parameter values in each of the rotations

`vBurn`

in: vector of size `iRot`, or scalar, number of drawings which are disregarded for a burn-in period

`vSkip`

in: vector of size `iRot`, or scalar, number of drawings from which one drawing is saved (ie, if `vSkip = 1`, all drawings are saved)

`vSerRej`

in: vector of size `iRot`, or scalar, number of iterated rejections of the Metropolis-Hastings step, before an acceptance is forced.

`vDir`

in: vector of size `iRot`, or scalar, number of directions which are sampled in the case of APS/APIS sampling

*No return value*

Set the size of the sample

## SetTaper

`MCMCPack::SetTaper(const vTaper)`

`vTaper`

1 x `nTap` vector with tapering perunages, or number of parameter vectors to use. Default= <0.05, 0.15>

*No return value*

Set the values used in tapering, in computing the relative numerical efficiencies

## Marglik

`Marglik(const iMethod, ...)`

The actual inputs depend on the method which is chosen for computing the marginal likelihood. Possible inputs are:

`iMethod`

in: One of `MC_MLKERN`, `MC_MLLP`, `MC_MLHM`, `MC_MLGG`, `ML_MLINT`, `MC_MLPR`, `MC_MLIS` (see also table 3), indicating the method to be used.

`vMu`

in: `iK` x `iP` matrix with `iP` location vectors for computation. If not provided, the mean of the sampled drawings is used.

`mS2`

in: `k` x `k` covariance matrix. If not provided (or `M_NAN`) the numerical approximation to the covariance is used.

`nRep`

in: Number of repetitions to be used in the Gibbs chain, in the prior sampling, or in the importance sampling. If not provided, the number of repetitions in the last iteration is used. Do not put this number too high, as computing the marginal likelihood using the Gibbs method may be a lengthy operation.

`nInt`

in: Number of intervals using in numerically integrating the posterior density, when using the method `MC_MLNUM`. Default= 20. Put this number very low, as the numerical integral in `iK` dimension may take a tremendously long time.

**dFrac**

in: Double, fraction of sample to use in Harmonic Mean computations, or number of parameter vectors (if > 1). Default= 1.

The following table gives the arguments for each of the computation methods, and functions which have to be set before calling Marglik:

Method	Arguments	Functions used
MC_MLKERN	vMu	Posterior
MC_MLLP	vMu, mS2	Posterior
MC_MLHM	dFrac	Likelihood
MC_MLGG	vMu, nRep	Posterior, Conditionals (if available)
MC_MLINT	nInt	Posterior
MC_MLPR	nRep	Likelihood, RanPrior
MC_MLIS	vMu, mS2, nRep	Prior, Likelihood

*Return value*

1 x iP vector with logarithm of marginal likelihoods at locations vMu.

Compute the marginal likelihood of a model, using either a kernel approximation, a LaPlace approximation, the harmonic mean method, the Gibbs approximation, brute force numerical integration or sampling from the prior density. See Bos (2002) for an overview and further references.

## Simulate

`MCMCPack::Simulate()`

*Return value*

Array with components vMu, mS2, the mean and covariance of the final sample.

Initiates the sampling routines. Output is in the text file as indicated by SetOutput with extension .out, and in a data file with extension .fmt. The data file is of size nDraws x k or nDraws x k+1, with the extra column containing the log-weights of the importance sampling algorithms.

## References

- Atkinson, A. C. (1985), *Plots, Transformations, and Regression: An Introduction to Graphical Methods of Diagnostic Regression Analysis*, Oxford statistical science series, Clarendon, Oxford.
- Bauwens, L., Bos, C. S., Van Dijk, H. K. and Van Oest, R. D. (2004), ‘Adaptive radial-based direction sampling: Some flexible and robust Monte Carlo integration methods’, *Journal of Econometrics* p. #25. Forthcoming.
- Bauwens, L., Lubrano, M. and Richard, J.-F. (1999), *Bayesian Inference in Dynamic Econometric Models*, Advanced Texts in Econometrics, Oxford University Press, Oxford.
- Bos, C. S. (2001), Time Varying Parameter Models for Inflation and Exchange Rates, PhD thesis, Tinbergen Institute, Erasmus University Rotterdam. TI 256.
- Bos, C. S. (2002), A comparison of marginal likelihood computation methods, in W. Härdle and B. Ronz, eds, ‘COMPSTAT 2002 – Proceedings of Computational Statistics’, pp. 111–117.
- Brownlee, K. A. (1965), *Statistical Theory and Methodology in Science and Engineering*, 2 edn, Wiley, New York.

- Casella, G. and George, E. (1992), ‘Explaining the Gibbs sampler’, *The American Statistician* **46**(3), 167–174.
- Chib, S. and Greenberg, E. (1995), ‘Understanding the Metropolis-Hastings algorithm’, *The American Statistician* **49**(4), 327–335.
- Doornik, J. A. (1999), *Object-Oriented Matrix Programming using Ox*, 3rd edn, Timberlake Consultants Ltd, London. See <http://www.nuff.ox.ac.uk/Users/Doornik>.
- Geman, S. and Geman, D. (1984), ‘Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images’, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-6**, 721–741.
- Geweke, J. (1992), Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments, in J.-M. Bernardo, J. O. Berger, A. P. Dawid and A. F. M. Smith, eds, ‘Bayesian Statistics 4: Proceedings of the Fourth Valencia International Meeting’, Oxford: Clarendon Press, pp. 169–193.
- Geweke, J. (1999), ‘Using simulation methods for Bayesian econometric models: Inference, development, and communication’, *Econometric Reviews* **18**(1), 1–73.
- Gilks, W. R., Roberts, G. O. and George, E. I. (1994), ‘Adaptive direction sampling’, *The Statistician* **43**, 179–189.
- Hastings, W. K. (1970), ‘Monte Carlo sampling methods using Markov chains and their applications’, *Biometrika* **57**, 97–109.
- Justel, A. and Peña, D. (1996), ‘Gibbs sampling will fail in outlier problems with strong masking’, *Journal of Computational & Graphical Statistics* **5**(2), 176–189.
- Kim, S., Shephard, N. and Chib, S. (1998), ‘Stochastic volatility: Likelihood inference and comparison with ARCH models’, *Review of Economic Studies* **64**, 361–393.
- Kloek, T. and Van Dijk, H. K. (1978), ‘Bayesian estimates of equation system parameters: An application of integration by Monte Carlo’, *Econometrica* **46**, 1–20.
- Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. and Teller, E. (1953), ‘Equations of state calculations by fast computing machines’, *Journal of Chemical Physics* **21**, 1087–1091.
- Newey, W. K. and West, K. D. (1987), ‘A simple, positive semi-definite, heteroskedasticity and autocorrelation consistent covariance matrix’, *Econometrica* **55**, 703–708.
- Ritter, C. and Tanner, M. A. (1992), ‘Facilitating the Gibbs sampler: The Gibbs stopper and the Griddy-Gibbs sampler’, *Journal of the American Statistical Association* **87**(419), 861–868.
- Rousseeuw, P. J. and Van Zomeren, B. C. (1990), ‘Unmasking multivariate outliers and leverage points’, *Journal of the American Statistical Association* **85**, 633–651.
- Smith, A. F. M. and Roberts, G. O. (1993), ‘Bayesian computation via the Gibbs sampler and related Markov Chain Monte Carlo methods’, *Journal of the Royal Statistical Society, Series B* **55**(1), 3–24.
- Tanner, M. A. and Wong, W. H. (1987), ‘The calculation of posterior distributions by data augmentation’, *Journal of the American Statistical Association* **82**, 528–550.
- Van Dijk, H. K. and Kloek, T. (1980), ‘Further experience in Bayesian analysis using Monte Carlo integration’, *Journal of Econometrics* **14**, 307–328.
- Van Dijk, H. K., Kloek, T. and Boender, C. G. E. (1985), ‘Posterior moments computed by mixed integration’, *Journal of Econometrics* **29**, 3–18.
- Yu, B. and Mykland, P. (1998), ‘Looking at Markov samplers through cusum path plots: A simple diagnostic idea’, *Statistics and Computing* **8**(3), 275–286.