# Bayesian Calibration (BC) and Bayesian Model Comparison (BMC) of process-based models: Theory, implementation and guidelines

Marcel van Oijen, 10 March 2010

[*An earlier version of this document is freely available from the NERC Open Research Archive, http://nora.nerc.ac.uk/6087/. Only minor corrections were made.*]

## 1. Introduction and acknowledgments

In the environmental sciences, process-based models are used to simulate biogeochemical fluxes in ecosystems at spatial scales that range from plots to regions. This document describes theory and application of Bayesian methods for model parameterisation and model evaluation. The emphasis is on methods that are easy to understand and implement, rather than on efficiency and speed. This may seem at odds with the stated aim of providing methods that work with process-based models, which can be notoriously slow. However, the methods presented here are correct for all models and practical experience should dictate when more complicated and faster methods are needed. If one has attempted the easy methods described here, the complex methods will be easier to understand. Some pointers to advanced literature are given.

The examples that are given in this document use extremely simple models, like the three- and four-parameter versions of the expolinear equation and even the two-parameter straight line. This is for didactic purposes only: all methods are generic and have been tested on process-based models with at least 40 parameters.

A large and increasing number of colleagues have provided useful feedback on the text. The text is expected to keep on changing in the coming years, while we learn from more practical experience. Appendix 4, 'MCMC code in R: the Metropolis algorithm' was written by Marie Guillot, Nathalie Yauschew-Raguenes, Annabel Porté and Simon Lehuger (INRA, France).

## 2. Theory of BC and BMC

The output from process-based models is subject to uncertainty about parameter values and model structure. Bayesian Calibration (BC) is a method for reducing uncertainty about parameter values for a given model, while Bayesian Model Comparison (BMC) examines multiple models to quantify their relative likelihoods of having the correct structure. Both methods rely on data and can be applied to all types of models, from empirical formulas to parameter-rich process-based models. In this section on the theory underlying BC and BMC, we demonstrate the methods using two simple process-based models: the 3- and 4-parameter expolinear models (EL3, EL4; Goudriaan 1994).

In both models, biomass growth rate is proportional to light interception by leaf area, calculated using Beer's Law. In EL3, LAI is proportional to biomass but in EL4, LAI gradually reaches an asymptote. Both models produce expolinear biomass growth curves as output. Initial biomass ($W_0$, kg C $m^{-2}$), Light-Use Efficiency ($LUE$ , kg C $MJ^{-1}$) and Leaf Area Ratio ($LAR$, $m^2$ $kg^{-}$) are the three free parameters of EL3 with $LAI_{max}$ ($m^2$ $m^{-2}$) as a fourth for EL4.

We use biomass and LAI data from a Norway spruce site at Skogaby (Sweden; Schulze 2000) for the BC and BMC.

BC operates on a single model, whose structure is assumed to be correct, but whose parameter values are not exactly known. The first step in BC is expressing our initial "prior" uncertainty about the parameter values in the form of a joint probability density function (pdf), symbolized $P(\vartheta)$. Typical choices for $P(\vartheta)$ are truncated multivariate normal or beta distributions – here we use a uniform distribution bounded at 0.1 and 10 kg C $m^{-2}$ ($W_0$), 0.0002 and 0.002 kg C

$MJ^{-1}$ (*LUE*), 0.5 and 2 $m^2\ kg^{-1}$ C (*LAR*) and 1 and 5 $m^2\ m^{-2}$ ($LAI_{max}$, only for EL4).

BC uses data on model outputs – here biomass and LAI – to update the pdf for the parameters. This is done by applying Bayes' Theorem: $P(\vartheta|D)=P(\vartheta)P(D|\vartheta)/P(D)$, where $P(\vartheta|D)$ is the posterior distribution for $\vartheta$ given the data D, $P(D|\vartheta)$ is the likelihood of the data given model output using parameters $\vartheta$ and P(D) is a normalization constant (e.g. Van Oijen et al, 2005). The data are only used in the calculation of the likelihood, with measurement error determining how likely any given model-data mismatch is. If the data are informative, i.e. are plenty and have low measurement error, the posterior pdf $P(\vartheta|D)$ will be narrower, more sharply peaked than the prior $P(\vartheta)$, indicating that parameter uncertainty is reduced.

There is no closed-form solution for Bayes' Theorem applied to process-based models, but it is always possible to generate a representative sample from the posterior by Markov Chain Monte Carlo (MCMC) methods, e.g. the Metropolis algorithm. Fig. 1 shows histograms of MCMC-generated samples ($n = 10^5$) from $P(\vartheta|D)$ for models EL3 and EL4, where the data from Skogaby were used to reduce our initial uncertainty embodied in the uniform prior. The histograms do not show the correlations between parameters, but for EL3 these were $r_{(W0,\ LUE)} = -0.54$, $r_{(W0,\ LAR)} = -0.35$, $r_{(LAR,\ LUE)} = -0.35$. For EL4, the same correlations were -0.46, -0.13 and -0.30, respectively and further $r_{(W0,\ LAImax)} = -0.15$, $r_{(LUE,\ LAImax)} = -0.32$, $r_{(LAR,\ LAImax)} = -0.02$.

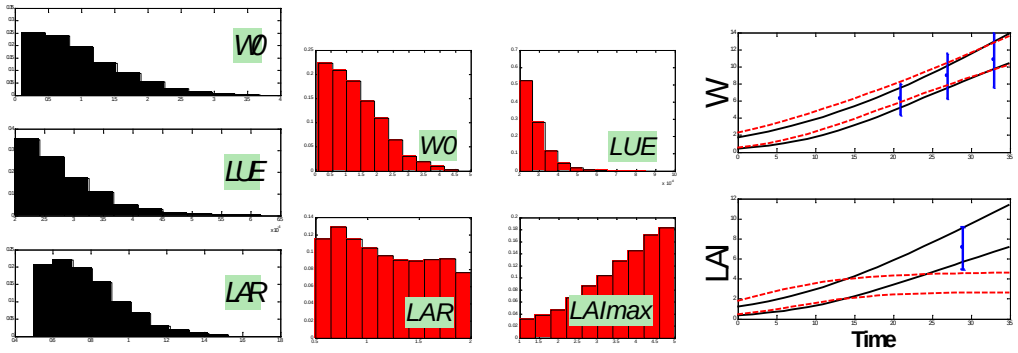


Fig. 1. Results from Bayesian Calibration of models EL3 and EL4, using data from Skogaby (Sweden). Left: Relative frequencies of parameter values in the MCMC sample from the posterior pdf for EL3. Centre: Idem for EL4. Right. Posterior uncertainty of outputs (biomass, LAI) for the Skogaby site for EL3 (black continuous lines) and EL4 (red broken lines). Blue dots with error bars: Skogaby data.

BMC is a simple but powerful extension of BC, in which the prior and posterior are probability distributions over a set of models, M, rather than over the parameter space of an individual model. BMC thus relaxes the assumption that we know the correct model structure. In our example, M consists of two models ($M_1$ = EL3, $M_2$ = EL4) and if we assume no initial preference for either, the prior P(M) is $P(M_1) = P(M_2) = 0.5$. We do not try to directly quantify the posterior P(M|D), but calculate the posterior ratio $P(M_2|D)/P(M_1|D)$. Expanding both terms using Bayes' Theorem and setting $P(M_1)=P(M_2)$, the posterior ratio simplifies to the so-called "Bayes Factor" $P(D|M_2)/P(D|M_1)$. The Bayes Factor tells us how much the odds of one model over the other change because of the information in the data. The "integrated likelihoods" $P(D|M_i)$ must be defined over the whole parameter space of $M_i$, i.e. $P(D|M_i) = \int P(D|\vartheta_i)P(\vartheta_i)d\vartheta_i$. This formula contains the same terms $P(\vartheta)$ and $P(D|\vartheta)$ as in BC, so we can use the MCMC-sample from $P(\vartheta_i|D)$ - see above - to estimate the integrated likelihood: $P(D|M_i) = n / \Sigma P^{-1}(D|\vartheta)$ where the summation is over the *n* likelihoods in the MCMC-sample (Kass & Raftery 1995). Note that an integrated likelihood estimated in this way, by means of the Kass & Raftery formula applied to the output of a BC using MCMC, is the likelihood integrated over the prior pdf for the parameters, and not the posterior. In short, the data are used to calculate two results in a single MCMC: (1) The integrated likelihood under the prior; (2) A posterior pdf.

Applied to the Skogaby data, the integrated likelihoods for EL3 and EL4 were 1.7E-4 and 0.35E-4 respectively, so the Bayes factor was 0.21. The Skogaby data thus supported the simpler model.

Both BC and BMC can drive a learning process: each time we receive new data we perform

BC and BMC using the posterior pdf's from the previous cycle as the prior of the current one – thus ever improving our knowledge of parameters, models and, by implication, the real system.

## 3. Methods for doing BC: MCMC and Accept-Reject

In the previous section, we already mentioned MCMC as a method for doing BC. The Metropolis algorithm is the simplest example of an MCMC algorithm. However, other MCMC algorithms exist and it is even possible to do BC using non-MCMC methods. In these guidelines we restrict ourselves to three simple but effective methods and give example code for each of them:
1. Standard Metropolis (code in App. 1 and 4)
2. Metropolis with a modified proposal generating mechanism ("Reflection method", App. 2)
3. Accept-Reject (code in App. 3)
All three methods do the same thing: they take a model with a prior parameter pdf, feed in new data, and generate a sample from the posterior parameter pdf. The methods do differ in how easy it is to implement and use them, and their scope, i.e. when they are likely to work well.

### 3.1 Standard Metropolis algorithm

Both Metropolis algorithms are sequential in the sense that they create a "walk through parameter space" in such a way that the chain of visited points is the sought-after sample from the posterior. This is called a random walk, because each new point in the chain is found by randomly generating a candidate parameter vector, which can be accepted or rejected. The candidate parameter vector is usually generated by taking a multivariate normal step away from the current vector. That means that we need to have a variance matrix to define this multivariate normal "proposal distribution". Usually the proposal matrix is a simple diagonal matrix with values that are related to how uncertain we are about the different parameters. Whether a proposed candidate vector is accepted or not depends on the *Metropolis ratio*, which is the ratio of two products: likelihood times prior for the candidate and likelihood times prior for the current point. If the Metropolis ratio is larger than 1 (i.e. the candidate point has a higher posterior probability then the current point), it is always accepted. If the Metropolis ratio is less than 1 (i.e. the candidate is "less probable" than the current vector), the candidate can still be accepted but only with probability equal to the Metropolis ratio. We stop the chain when it has "converged", i.e. it has explored the posterior parameter space adequately. Convergence can be confirmed visually using the trace plots of the different parameters, i.e. plots that show how the chain moves through parameter space for each individual parameter (Fig. 2).

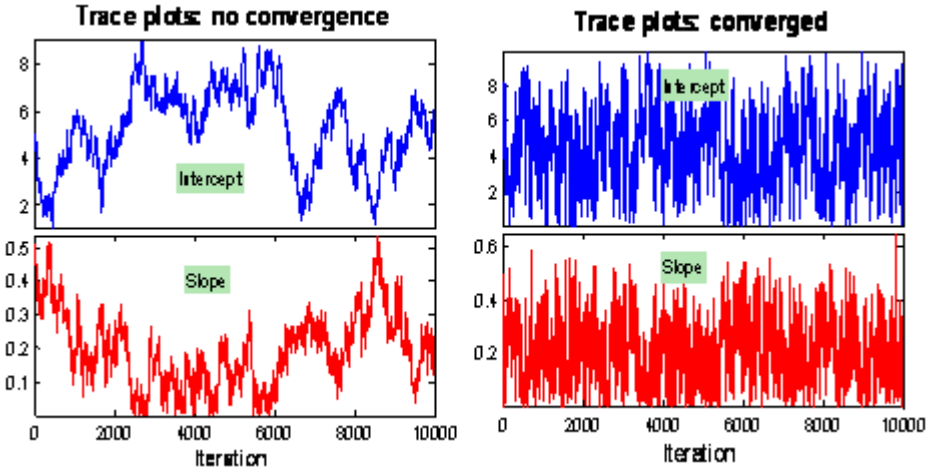


Fig. 2. Two pairs of trace-plots, showing chains of parameter values generated by MCMC. Left-column: convergence not yet reached. Right column: convergence reached.

If one or more of the trace plots are still showing drift towards unexplored parts or parameter space, the chain has not yet converged. There exist more formal methods of establishing convergence, which usually consist of running multiple chains in parallel and determining whether the inter-chain

variance has dropped down to the level of average intra-chain variance.

A semi-formal summary of the Metropolis algorithm is provided in Box 1.

**Box 1. The Metropolis algorithm: prerequisites, pseudocode, posterior.**

We have:
(1) A model: M
(2) Data: D
(3) A prior probability distribution for the parameters: $P(\theta)$
(4) A likelihood function: $P(D|\theta)$ which is a function of the difference $M(\theta)$-D

We do:
(1) Select any starting point in parameter space, $\theta^{(0)}$
(2) Calculate the product of prior and likelihood: $\pi^{(0)}=P(\theta^{(0)})\bullet P(D|\theta^{(0)})$
(3) Set i=0
(4) Run the following loop until we have enough sample points:
  (i) Generate a random new point, $\theta^{(candidate)}$
  (ii) Calculate $\pi^{(candidate)} = P(\theta^{(candidate)})\bullet P(D|\theta^{(candidate)})$
  (iii) Calculate the Metropolis ratio: $\pi^{(candidate)} / \pi^{(i)}$
  (iv) Accept the candidate with probability equal to *min*(Metropolis ratio,1)
  (v) If the candidate is accepted: $\theta^{(i+1)} = \theta^{(candidate)}$, else: $\theta^{(i+1)} = \theta^{(i)}$
  (vi) Set i=i+1

We get:
(1) A representative sample $\{\theta^{(0)} \ldots \theta^{(n)}\}$ from the posterior pdf, $P(\theta|D)$

An example implementation of the standard Metropolis algorithm, with brief explanation, is given in App. 1. The algorithm itself was originally described by Metropolis et al. (1953). A very readable introduction to the algorithm, and the underlying Bayesian methodology, is given by Gilks et al. (1996). Also quite readable, and with practical advice for MCMC, are chapter 11 of the book by Gelman et al. (2004) and chapter 15.8 of the latest edition of the Numerical Recipes (Press et al., 2007). Some of that practical advice is discussed in the FAQ section of these guidelines. A more technical and thorough analysis of the algorithm is provided by Robert and Casella (2004). Application of the Metropolis algorithm to parameter-rich process-based ecosystem models is described by Van Oijen et al. (2005).

### 3.2 Metropolis with a modified proposal generating mechanism ("Reflection method")

This version of the Metropolis algorithm is the same as the standard version described in 3.1, except for the proposal generating mechanism. If the prior parameter pdf includes hard lower and/or upper limits, the multivariate normal proposal step may generate a candidate outside those bounds. That means the prior probability for that parameter vector is zero, and the candidate will be rejected. If the rejection rate is very high, the chain will take long to explore the parameter space. It may thus be useful to modify the Metropolis algorithm slightly and "reflect back" any proposed outside jumps to end up within the bounds (C.J.F. Ter Braak, pers. comm. 2006). Such reflection is permissible if it does not interfere with the criteria for a correct Metropolis algorithm (i.e. one that converges to the posterior pdf). This requires that the proposal mechanism is symmetric, which means that the probability for jumping from any point *a* to *b* should be equal to the probability for jumping from *b* to *a*. The reflection method is symmetric only if the proposal distribution itself is radially symmetric, i.e. if there is no preferred "direction" of our proposal steps. Therefore, when using a multivariate normal proposal, the Reflection method only allows a diagonal matrix, whereas the Standard Metropolis method of 3.1 does allow correlations.

An example implementation of the Metropolis-with-reflection algorithm is given in App. 2.

### 3.3 Accept-Reject algorithm

The Accept-Reject method (A-R), also called *rejection sampling*, is not an MCMC method: it is not

sequential and there is no walk through parameter space. For BC, the A-R method works as follows. First, take a large sample from the prior parameter pdf, e.g. $10^4$ parameter vectors. Then, for each of the parameter vectors in the sample, calculate model output and associated data-likelihood. Finally, for each of the vectors in the sample, there is an accept-reject decision that resembles the one in the Metropolis algorithm except that the probability for acceptance is not the Metropolis ratio but the ratio of the vector's likelihood and the maximum likelihood (ML). The accepted vectors together are then our sample from the posterior parameter pdf.

If the sample size is large enough, the ML can simply be approximated as the largest likelihood among the vectors in the sample. Formally, the ML is the highest possible likelihood in all of parameter space, not just in the sample, and the A-R method does not allow underestimating the ML. However, in practice the maximum in the sample may be close enough if the sample size is not too small. It is possible to build in a safety factor by estimating the ML to be greater than the maximum in the sample, but that may lead to the problem that too many vectors are rejected to provide a proper sample from the posterior.

An example implementation of the A-R method, with brief explanation, is given in App. 3. A good description, but slightly formal, of the method is provided by Gelman et al. (2004). Even more technical, and more thorough, is the description provided by Robert and Casella (2004).

## 4. FAQ – Bayesian Calibration & Bayesian Model Comparison

[NOTE: The answers given below to frequently asked questions can not be guaranteed to apply exactly in each case. It is always possible that a specific combination of data and model behaves differently – in the statistical sense – from the majority of cases. Some degree of trial-and-error remains required.]

### 4.1 BC ALGORITHM

***- Is there any point in using more than one chain in MCMC?***
Yes. If you run more than one chain, each starting from a different point in parameter space, it is much easier to see when the chains have converged. That is the case when the different chains have moved to the same part of parameter space – and we can establish this formally by testing whether the interchain variance has come down to the level of intrachain variance using, for example, the Gelman-Rubin statistic (Robert & Casella 2004, Ch. 12). However, full-proof convergence diagnostics do not exist and visual inspection of the trace plots of a single-chain MCMC remains an acceptable way of working.

***- My model is not a process-based model: does that affect how I should do the BC?***
No. The method is completely generic and works with all models. The emphasis on process-based models in this document is only because such models are typical in environmental science and BC has not been used much with them. Note that the code-examples in the Appendices are all for an empirical model (straight line) rather than a process-based model. [Appendix 1 does explain where the call to the empirical model should be replaced with a call to a process-based model: line 7]

***- My model is perhaps not very good: does that affect how I should do the BC?***
No. The BC takes your model as given. The posterior distribution for the parameters is always conditional on both the data and the model that are being used. If we wanted, we could express this by writing P($\vartheta$|D,Model) for our posterior pdf, rather than just P($\vartheta$|D). However, we write the shorter version and treat the conditionality on the model as understood. Note that the use of a poor model together with informative data may lead to a posterior pdf for the parameters that will seem unrealistic if considered in isolation – even though conditional on the model that posterior is the correct one. The BC will thus help us identify models of poor quality – and in such cases it is

advisable to try different models and carry out a Bayesian Model Comparison (see Section 2).
[There are methods for BC which do not assume that the original model is correct. Instead they add a stochastic term to the model, called the 'discrepancy', to quantify the mismatch between model behaviour and reality. Often the discrepancy is modelled as a multivariate Gaussian that quantifies the mismatch for each data point. In short, these methods replace the assumption of the original deterministic model being correct with that of the extended (deterministic + stochastic) model being correct. There are two major problems with that approach. First the extended model will generate time series which are physically impossible because of the stochastic jumps that are allowed. Secondly, the stochastic discrepancy term is only defined for data points and does not help us with predictions.]

***- The example codes are in Matlab and R: what if I work in a different programming language?***
That should be no problem. The appendices show how short the BC algorithms are and it is easy to convert them to any other language. There are various code libraries available that may make the programming easy (Press et al., 2007), for example for manipulating multivariate normal pdf's. In the NitroEurope community alone, BC has been implemented in six different languages.

There is also the possibility of keeping your model in one language, but doing the Bayesian bits in another. Simon Lehuger (INRA-Grignon) has written R-code for MCMC that calls the crop model CERES-EGC which is in Fortran (Lehuger et al., 2008). Working in this way allows the user to benefit from the speed of Fortran, and has the added advantage that there are plenty of R procedures related to Bayesian inference that are freely available on the web, like the CODA-software for analysis and diagnostics for MCMC output (http://cran.r-project.org/web/packages/coda/index.html).

***- Where can I read more about the BC methods?***
There is a vast literature on BC methods, but only a very small fraction of that deals with process-based models. Some introductory texts are mentioned at the end of §§ 3.1 and 3.3.

***- Which BC algorithm should I use?***
The algorithms we present here (Metropolis, Metropolis-with-Reflection and A-R) are completely general: they will give the right answer for any combination of model and data – provided the sample from the posterior is large enough. There are other methods which may work better in specific circumstances, like Gibbs Sampling (Gilks et al. 1996) or adaptive MCMC algorithms in which the proposal distribution changes during the random walk. However, these methods are more complicated, less general and for the adaptive algorithms there is the risk that they do not converge to the correct posterior pdf (Robert & Casella, 2004). Having said that, the future is likely to see increasing use of adaptive algorithms. Promising methods include DRAM (Haario, 2006; see also Calanca, 2008) and $DEMC_{zs}$ (Ter Braak & Vrugt, 2008).

Among the three algorithms inspected here, the Metropolis is the one that works best for parameter-rich models, say more than 10 parameters to be calibrated. Metropolis-with-Reflection can be more appropriate than standard Metropolis if one or more of the parameters have fixed lower or upper bounds. However, Metropolis-with-Reflection can not take advantage of any correlations between parameters, as it requires a radially symmetric proposal distribution. A-R should not be used with parameter-rich models, unless a pre-screening step is added in which the number of to-be-calibrated parameters is reduced to less than 10 first. A-R has problems with parameter-rich models because A-R begins by sampling from the prior across the whole of parameter space. This cannot be done exhaustively if the space is high-dimensional. In contrast, Metropolis samples directly from the posterior pdf and is efficient in that it spends most time in the area of high probability. However, A-R has advantages too: it is simpler (compare App. 3 with Apps 1 and 2) and it is easier to apply as there is no proposal distribution to optimise.

## 4.2 DATA AND LIKELIHOOD

***- Does it matter whether we use all data at the same time or in steps?***
No. Assume you have two datasets D1 and D2. You could use all the data at once to calculate the posterior $P(\vartheta|D1,D2)$. Alternatively, you could first calculate $P(\vartheta|D1)$ and then use that as the prior for a BC on D2. In both cases you get the same final posterior pdf $P(\vartheta|D1,D2)$. This shows one of the major strengths of BC: it is firmly based in probability theory and always gives the same answer if the same information (in this case D1 plus D2) is used.

***- How do I handle systematic measurement error?***
Systematic measurement error, where all measurements are off by a constant but unknown amount or proportion, can be handled like any other unknown constant: as a parameter that we calibrate. This only tends to be necessary for some types of measurements, e.g. poorly replicated ones. An example may be the measurement of soil emissions of $N_2O$, where we are always in danger of having used a not representative placement of soil chambers. For such emission data we could use a multiplicative systematic error parameter $k$ with prior $k \sim U[0.5,2]$. The easiest way of calibrating that parameter is by including it directly into the likelihood. The model calculations are then not changed but the likelihood changes from, for example, $L \sim N(y-D,\sigma^2)$ to $L \sim N(y-D*k,\sigma^2)$.

***- I have a lot of data: do I need to use them all?***
In principle, yes, as long as the uncertainty about random as well as systematic measurement errors is properly quantified in the likelihood. If the model is very good, then every bit of empirical information will tend to give us a posterior pdf for the parameters which will improve model performance for any application. However, with our imperfect models it may be prudent to aim for balance between different kinds of measurement information. We can do this by not supplying many more data for one variable, one site or one time period, than for others.

***- What do I do if measurement errors are not independent?***
This is important for calculating the likelihood. We often assume that measurement errors are independent and calculate the overall likelihood of a dataset $P(D|\vartheta)$ as the product of likelihoods for the individual data points $P(D_i|\vartheta)$ [See Section 2 for this notation]. This is usually done as a straightforward multiplication of Gaussian probability densities, with the standard deviations $\sigma_i$ representing each data point's uncertainty. If the measurement errors are correlated we can take that into account by calculating the likelihood differently, e.g. as a multivariate normal distribution where the off-diagonal elements of the variance-covariance are not all zero. For technical details on this and other methods for modifying the likelihood function, see for example Chapter 8 of Sivia (2006).

***- What do I do if my measurements have uncertainties in the x- as well the y-direction?***
This can happen if it is unclear at what exact time the measurements were taken. This is easily incorporated in the BC, by adjusting the calculation of the likelihood. Say there is 50% chance that a certain data point D was indeed measured at the reported day and 25% chance each for the days before and after. Then the likelihood for that point is calculated as the weighted average of the likelihoods calculated for each of the three days, with weights 0.25, 0.5 and 0.25.

## 4.3 PARAMETERS

***- Can BC handle correlations between parameters?***
Yes. Correlations appear whenever we do not know the exact values of two parameters, but do know something about their combined values. For example, we may know that if parameter *a* has a high value, parameter *b* is likely to have a low value. In fact, each time we speak of "the" parameter pdf, we speak of one single probability distribution for all possible combinations of parameter values, and that pdf typically includes correlations. In other words, the parameters pdf is a joint

probability distribution, not a collection of independent distributions for individual parameters. Furthermore, even if the prior pdf has no correlations between parameters, the posterior pdf almost always will.

***- Should I include all my model's parameters in the BC or just a subset?***

Yes. Using a subset of the parameters will inevitably lead to underestimation of the contribution of the parameters to model uncertainty. MCMC can handle high-dimensional parameter sets, but the Accept-Reject method will struggle (see *'Which BC algorithm should I use?'*). However, if it is possible to identify a subset of parameters that together account for most of the model behaviour, then there is little harm in using the subset only.

***- What distribution should I use for the prior pdf?***

The prior pdf should represent what we know about our model's parameters. That means representing what parameter values are considered impossible, which ones are possible but not likely, which ones are quite plausible etc. If we have no idea about parameter correlations, we can use the product of different distributions for each individual parameter. Using the uniform distribution is generally not a good idea: we typically do not consider all values between some lower and upper limits equally plausible. A triangular distribution is better but still has strange discontinuities. Good choices are the (truncated) normal distribution, the lognormal distribution and the beta distribution. However, there is no golden rule: it all depends on what we can say about our parameter values. Note that there may exist all kinds of complicated constraints on the parameter values that we want to represent in the prior. For example, we may have two parameters $p_{min}$ and $p_{max}$ that are both highly uncertain but we know that $p_{max} > p_{min}$. The easiest way to include such information in the prior pdf for the parameters is to reparameterise, e.g. by replacing $p_{min}$ with a new parameter $f_{pmin}$, defined as $p_{min}/p_{max}$. Reparameterisation may in some cases lead to parameter vectors that are easier to calibrate (Gilks & Roberts, 1996).

## 4.4 CODE EFFICIENCY AND COMPUTATIONAL ISSUES

***- How does the strange-looking Matlab code for the proposal algorithm work, i.e. where a candidate vector is generated using "transpose" and "chol" functions (e.g. line 13-14 of Appendix 1)?***

That code generates a random vector from the multivariate normal distribution, and adds that vector to the current position of the chain ("pValues"). It does this using a common and simple matrix algebraic method. Basically, the random vector is generated by pre-multiplying a vector of random univariate normal numbers (the "randn" bit) with the so-called Cholesky decomposition of the variance-covariance matrix from our proposal algorithm. The various "transpose"-functions are there to make sure that the random vector and the decomposed matrix are aligned up properly.

***- How do I make the proposal algorithm of my MCMC method more efficient?***

Very often, if acceptance rate is low (<<23%), the average proposal step length is too high and vice versa. It may also be that the step lengths are OK but that we are making too many steps in the wrong direction. This could happen, for example, if we use a radially symmetric proposal distribution (as in our examples in the Appendices) while the posterior that we are aiming for has strong correlations between parameters, i.e. is ridge-shaped. There are two general ways of improving the proposal in such circumstances: (1) perform some trial-and-error calibration optimisation to find an efficient proposal distribution, (2) automate the process of optimising the proposal by using so-called adaptive proposal algorithms. The second method is currently subject of intensive research and researchers like Haario (2006; see also Calanca, 2008), and ter Braak have published promising algorithms (Ter Braak & Vrugt, 2008). These algorithms tend to be more complex than the ones treated in this document, and we refer to the original literature for details. Some trial-and-error proposal optimisation can however be fully adequate. Gelman et al. (2004, Ch.

11) give some suggestions for optimising the proposal in algorithms like Metropolis that use a multivariate normal distribution to generate the random steps. They advise to make the proposal distribution proportional to the posterior pdf that we are going to generate. However, we do not know this distribution beforehand – that is why we do the MCMC in the first place - so we have to work with the prior pdf instead. The recommendation is to use a proposal distribution where the entries in the variance-covariance matrix are $2.38^2/n$ ( $\approx 5.66/n$) larger than the posterior (in our case the prior), with n being the number of parameters to be calibrated (Gelman et al. 2004). So with 6 parameters our proposal distribution could be about the same as our prior. However, if we have many parameters that have little impact on the model results, this recommendation may lead to too small steps. In general, convergence is fastest, so required chain length smallest, when the proportion of accepted proposals is about 23% for models that have more than 5 parameters (Gelman et al., 2004). Again, this may not apply to every combination of model and data, so some degree of trial-and-error remains necessary. The most general recommendation that can be made is to try out various proposal distributions in short chains and then select the best for the definitive BC.

***- What do I do if the MCMC crashes?***

Obviously, this is not supposed to happen! Fortunately it is also very unlikely given the simplicity and robustness of the Metropolis algorithm. However, there is one part in the algorithm that occasionally gives problems: calculation of the prior probability for a new candidate parameter vector. If the prior pdf is very simple, e.g. multivariate uniform or triangular, calculation of the prior probability density for any new parameter vector poses no problems. However, if we use a distribution like the multivariate normal for the prior, then calculation of parameter vector probability includes a matrix inversion step, where the variance-covariance matrix needs to be inverted. This may run into limitations with the numerical precision of computers. Matrix inversion is a widely-studied issue in applied mathematics and there are robust but complicated algorithms available. However, we can generally avoid all problems if we don't have too many orders of magnitude difference between the smallest and largest parameters. A large variance-covariance matrix where some entries are $\sim 10^{-6}$ and others $\sim 10^{8}$ is difficult to invert. We can deal with this by rescaling the model's parameters to be all close to unity. Of course, that requires undoing the rescaling each time we need to run our model.

***- What do I do if the MCMC does not converge?***

The first thing to do is make the chain longer. If that takes too much computing time, consider changing the proposal algorithm, e.g. by making the proposal distribution narrower (i.e. shortening the average proposal step) or by including reflection (App. 2). For more suggestions, see "*How do I make the proposal algorithm of my MCMC method more efficient?*" and Gilks & Roberts (1996).

## 4.5 RESULTS FROM THE BC

***- If my chain length is, say, 10000 and acceptance rate is 25%, does that mean my sample from the posterior only consists of 2500 parameter vectors?***

No, the sample consists of all 10000 parameter vectors, but many vectors will appear two or more times in the sample. In other words, if during the MCMC a candidate parameter vector is rejected, the current vector will receive extra weight. So if we want to calculate, for example, the mean posterior parameter vector, we need to sum all 10000 vectors (including the repeats) and divide by 10000.

***- What do I do if the sample from the posterior parameter pdf looks wrong, or if the posterior predictions of model output variables look wrong?***

Not necessarily anything. All the BC does, is combine the information that has been supplied – via the prior, the model and the data-likelihood – and show what that means for what we can say about the parameters. The results may be surprising but nevertheless correct. However, if something really

seems amiss, e.g. posterior predictions for some variables seem impossible, there may be an error somewhere. First check the BC-code (perhaps check how your implementation differs from those in Apps 1-3). Then check if the prior was not incorrectly set, e.g. by disallowing perfectly reasonable parameter values. Then check whether the likelihood was correctly defined: perhaps there was less information in the data than you assumed, e.g. if you neglected representation error (where the data were accurate but not representative of what the model actually tries to simulate) or if you overlooked systematic errors in the data. It may also be that you have lots of unused information: if you have good reason to disbelieve certain model results, then that suggests you have information that could be included in the likelihood function. What's left after all those checks is the most common cause of seemingly poor BC-results: the model itself may not be very good and the calibration just makes that very apparent. In that case there are only three options left: (1) Accept your results but consider them to be conditional on a model that you think is flawed; (2) Improve your model; (3) Re-interpret your model as not being intended to simulate all the characteristics of the data, e.g. by treating your model outputs as being correct for a specific spatiotemporal scale, which may have lower resolution than that of the data.

***- What do I do with my results, i.e. how do I use my sample from the posterior?***

Some key characteristics of the posterior to be reported are its mode, mean and variance-covariance matrix. The mode is the vector in the posterior sample with the highest probability, i.e. the highest product of prior and likelihood. The mode will be equal to the mean if the posterior is symmetric. In most applications of BC, the variance-covariance matrix shows many pair-wise parameter correlations. In many cases the posterior can be well approximated by a multivariate normal distribution, fully defined by the sample mean and variance-covariance matrix. Approximating the posterior sample by a parameterised pdf like the multivariate normal is essential in case new data become available with which we want to do a further BC. The posterior from the first BC then becomes the prior for the second BC and we always need a parameterised pdf for the prior to allow us to calculate the prior probability density at any candidate vector in parameter space.

***- What is burn-in and do I need it?***

Burn-in is the first phase of a MCMC, when the chain is still "searching" for the area of high posterior probability, i.e. before convergence. The parameter vectors sampled during the burn-in phase should be removed from the posterior sample because they are not representative. Generally removing the first 10% of a chain is adequate.

**4.6 BMC**

***- How do I calculate the 'Integrated Likelihood' $P(D|M_i)$?***

The Integrated Likelihood (*IL*) for a given model is defined as $\int P(D|\vartheta)P(\vartheta)d\vartheta$ (see Section 2), where $\vartheta$ represents the parameter vector for that model. The definition states that *IL* is the integral, over the parameter space of the model, of the product of likelihood and prior. Another way of stating the same thing is: *IL* is the expected value of $P(D|\vartheta)$ under the probability distribution $P(\vartheta)$. This last restatement immediately suggests a simple way to calculate *IL*: just take a representative sample of *n* parameter vectors from the prior $P(\vartheta)$, run the model for the *n* vectors, calculate the likelihood $L = P(\vartheta|D)$ for each run and calculate *IL* as $\Sigma L/n$. That simple method for calculating *IL* is fine, but it would mean doing extra work if we had planned to use the same dataset D not only to calculate *IL*, but also to calibrate the model. In that case, after having calculated *IL*, we would still have to run an MCMC for the calibration. Unfortunately we can not simply calculate *IL* as the mean of the likelihoods for all the runs in the MCMC because the MCMC gives a sample from the *posterior* and *IL* is the mean likelihood under the *prior*. However, it turns out that we can still use the likelihoods from the MCMC to calculate *IL*, by reweighing the posterior sample, which leads to the formula $IL = n / \Sigma P^{-1}(D|\vartheta)$ (Kass & Raftery 1995). The term $n / \Sigma P^{-1}(D|\vartheta)$ can be recognised as the harmonic mean of the likelihoods in our sample from the posterior. See elsewhere in this FAQ for the

derivation of the harmonic mean formula, and for a method to calculate it.

***- How is the formula for calculating the 'Integrated Likelihood' from MCMC-output, $P(D|M_i) = n / \Sigma P^{-1}(D|\vartheta)$, derived?***

Perhaps surprisingly, the Integrated Likelihood (*IL*) *under the prior distribution* for a given model can be calculated as the harmonic mean of a sample *from the posterior*: $IL = P(D|M_i) = n / \Sigma P^{-1}(D|\vartheta)$. In other words, if we have MCMC output, i.e. a sample from the posterior, the formula tells us that we can calculate *IL* as the chain length (*n*) divided by the sum of the inverse likelihoods: $\Sigma\ 1 / P(D|\vartheta)$. To derive this formula, we use the following helpful theorem from probability theory: "Given two pdf's for variable x, $\pi(x)$ and q(x), the expected value of any function of x under distribution $\pi(x)$ can be calculated as $E_\pi[f(x] = E_q[f(x)\ \pi(x)/q(x)] / E[\pi(x)/q(x)]$". The theorem says that an expectation under q can be interpreted as an expectation under $\pi$ if we weigh consistently with $\pi/q$. This theorem is useful if we already have a representative sample of x under q(x) and would like to calculate the expected value of f(x) under the other distribution $\pi(x)$, without having to sample from it. How does this help us to calculate the Integrated Likelihood *IL* from MCMC output? Well, in that case: (1) x is the parameter vector $\vartheta$, (2) f(x) is the likelihood $P(D|\vartheta)$, (3) $\pi(x)$ is the prior $P(\vartheta)$, (4) q(x) is the posterior $P(\vartheta|D)$ and therefore (5) $E_\pi[f(x] = IL$. That gives:

$$
\begin{aligned}
IL &= E_q[f(x)\ \pi(x)/q(x)] / E[\pi(x)/q(x)] \\
&= E_{posterior}[\text{likelihood} * \text{prior} / \text{posterior}] / E_{posterior}[\text{prior} / \text{posterior}] \\
&= E_{posterior}[P(D|\vartheta) * P(\vartheta) / \{P(D|\vartheta)*P(\vartheta)/P(D)\}] / \\
&\qquad\qquad E_{posterior}[P(\vartheta) / \{P(D|\vartheta)*P(\vartheta)/P(D)\}] \\
&= 1 / E_{posterior}[1 / P(D|\vartheta)]
\end{aligned}
$$

And the last expression can be estimated from the MCMC-output as the inverse of the mean inverse likelihood, i.e. $n / \Sigma P^{-1}(D|\vartheta)$.

***- Is the harmonic mean formula for calculating the 'Integrated Likelihood' $P(D|M_i)$ reliable?***

Practical experience with process-based models suggests that the harmonic mean formula works well (e.g. report Stephen Pearson). However, in the statistical literature there are demonstrations of pathological cases where the sample needs to be unfeasibly large for the harmonic mean to work.

***- My program gives nonsense results, or even crashes, when I try to use MCMC-output to calculate the 'Integrated Likelihood': Why is that and what do I do about it?***

The problem is in the calculation of the following formula: Integrated Likelihood (*IL*) = $n / \Sigma P^{-1}(D|\vartheta)$. That formula is derived and discussed elsewhere in this document. When we try to calculate the formula using our MCMC-output, it may seem that we need to know the (inverted) likelihood for every parameter vector in the chain. However, most of the likelihoods in the chain will be extremely small – so small in fact that they can not be accurately represented on a digital computer. For that reason, all our MCMC-algorithms work with logarithms of likelihoods rather than the likelihoods themselves (see Appendices). That means that, after an MCMC, we have available a chain of log-likelihoods rather than likelihoods. Now, to calculate *IL*, we should *not* try to back-transform all those individual likelihoods, because of the numerical problems. Instead, we should first calculate the logarithm of the *IL*, before back-transforming that to give us our answer. We can use the following formula to calculate the logarithm of *IL*:

$$
\begin{aligned}
\log(IL) &= \log(1 / \overline{L^{-1}}) \\
&= \overline{\log(L)} - \log(\overline{L^{-1}} \exp[\overline{\log(L)}]) \\
&= \overline{\log(L)} - \log(\exp[\overline{\log(L)} - \log(L)])
\end{aligned}
$$

where L stands for the likelihoods $P(D|\vartheta)$ and the overbars indicate averages over the whole chain. Note that the formula does include back-transformation (using 'exp'). This does not cause any numerical problems because we do not back-transform the individual log-likelihoods, log(L), but the differences between them and the mean log-likelihood. The differences tend to be more

manageable numbers. In Matlab, the calculation of log(*IL*) might look as follows:

```
logpData = mean(logLChain) - log(mean(exp(mean(logLChain)-logLChain)))
```

***- Where can I find out more about the use of Bayes Factors?***

The key reference remains Kass & Raftery (1995). See Chapter 9 of Gilks et al. (1996) for some caveats regarding the calculation of the Bayes Factors.

## References


Calanca, P. (2008). Testing an adaptive Metropolis algorithm for BC of NEU models. Internal report NitroEurope: 17 pp.

Gelman, A., Carlin, J.B., Stern, H.S. & Rubin, D.B. (2004). Bayesian Data Analysis. Second Edition. Chapman & Hall/CRC, Boca Raton: 668 pp.

Gilks, W.R., S. Richardson and D.J. Spiegelhalter (1996). Introducing Markov chain Monte Carlo. In Markov Chain Monte Carlo in Practice Eds. W.R. Gilks, S. Richardson and D.J. Spiegelhalter. Chapman & Hall, London, pp. 1-19.

Gilks, W.R. & Roberts, G.O. (1996). Strategies for improving MCMC. In Markov Chain Monte Carlo in Practice Eds. W.R. Gilks, S. Richardson and D.J. Spiegelhalter. Chapman & Hall, London, pp. 89-114.

Goudriaan (1994). In: Monteith et al. (Eds), Resource Capture by Crops. Nottingham Univ. Press: 99-110.

Haario, H., Laine M., Mira A., and Saksman E. (2006). DRAM: efficient adaptive MCMC. Stat. Comput. 16: 339-354

Lehuger, S., Gabrielle, B. & Van Oijen, M. (2008). Bayesian calibration of CERES-EGC: BC in practice using the R software. Internal report: 9 pp.

Kass, R.E. & Raftery, A.E. (1995). Bayes Factors. Journal of the American Statistical Association. 90:773-795.

Metropolis, N., A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller and E. Teller (1953). Equations of state calculations by fast computing machines. Journal of Chemical Physics. 21:1087-1092.

Press, Teukolsky, Vetterling & Flannery (2007). Numerical Recipes. Third Edition. Cambridge Univ. Press, 1235 pp.

Schulze, E-D (Ed.) (2000). Ecol. Stud. 142. Springer, Berlin.

Ter Braak, C.J.F. & Vrugt, J.A. (2008). Differential Evolution Markov Chain with snooker updater and fewer chains. Stat. Comp.

Van Oijen, M., Rougier, J. & Smith, R. (2005). Bayesian calibration of process-based forest models: bridging the gap between models and data. Tree Physiology 25: 915-927.

Robert, C.P. & Casella, G. (2004). Monte Carlo Statistical Methods. Second Edition. New York. 645 pp.

Sivia, D.S. (2006). Data Analysis: A Bayesian Tutorial. Second Edition. Oxford University Press, Oxford: 264 pp.

# Appendix 1
# MCMC code in MATLAB: the Metropolis algorithm

```
1   chainLength    = 10000                                                          ;
2   data           = [10, 6.09, 1.83 ; …
3                     20, 8.81, 2.64 ; …
4                     30, 10.66, 3.27 ]                                            ;
5   pMinima        = [0, 0] ; pMaxima = [10, 1] ; pValues = [5, 0.5]                ;
6   vcovProposal   = diag( (0.3*(pMaxima-pMinima)) .^2 )                           ;
7   pChain         = zeros( chainLength, length(pValues) ) ; pChain(1,:) = pValues ;
8   Prior0         = prod((pMaxima-pMinima).^-1) ; logPrior0 = log(Prior0)          ;
9   model          = inline('intercept+slope*t')                                    ;
10  y              = model(pValues(1),pValues(2),1:30)                              ;
11  for i = 1:length(data),
12      logLi(i) = -0.5*((y(data(i,1))-data(i,2))/data(i,3))^2 -log(data(i,3))     ;
13  end ; logL0 = sum( logLi )                                                      ;
14  for c = 2 : chainLength,
15  candidatepValues = pValues + ...
16      transpose( transpose(chol(vcovProposal))*transpose(randn(size(pValues))) ) ;
17      if all(candidatepValues>pMinima) & all(candidatepValues<pMaxima),
18          Prior1 = prod((pMaxima-pMinima).^-1) ;
19      else
20          Prior1 = 0 ;
21      end;
22      if (Prior1 > 0),
23         y = model(candidatepValues(1),candidatepValues(2),1:30)                  ;
24         for i = 1:length(data),
25             logLi(i) = -0.5*((y(data(i,1))-data(i,2))/data(i,3))^2 -log(data(i,3));
26         end ; logL1 = sum( logLi )                                               ;
27         logalpha = (log(Prior1)+logL1) - (logPrior0+logL0)                       ;
28         if ( log(rand) < logalpha ),
29            pValues = candidatepValues ; logPrior0 = log(Prior1) ; logL0 = logL1 ;
30         end                                                                      ;
31      end                                                                         ;
32      pChain(c,:) = pValues                                                       ;
33  end                                                                             ;
34  nAccepted = length(unique(pChain(:,1)))                                         ;
35  disp(['MCMC: ' num2str(nAccepted) ' candidate parameter vectors accepted (=' ...
36        num2str(round(100*nAccepted/chainLength)) '%)'])                          ;
37  disp(mean(pChain)) ; disp(cov(pChain))                                          ;
```

This code carries out a Bayesian calibration for the 2 parameters of a straight line, by means of MCMC. The simplest algorithm is used: Metropolis sampling. No external files are called: the data used for calibration are incorporated in the code. Most of the code can be used for calibration of any model. The following changes are necessary if another model-data combination is targeted:

1. Change the data [1ines 2-4]
2. Change the bounds of the prior pdf for the parameters and the start values of the parameters in the MCMC [1ine 5]
3. Change the model [line 9]

Other changes are not essential but may be considered for better results:

4. Change the prior pdf for the parameters to be a non-uniform distribution [1ines 5, 8, 18]
5. Improve the MCMC by: (i) changing the variance-covariance matrix for proposal generation [1ine 6], (ii) accounting for an initial burn-in period, e.g. by removing the first 10% of the posterior chain, (iii) running multiple chains in parallel, each with different start points and performing convergence tests to decide whether the chains are long enough

The code consists of three parts: INITIALISATION (lines 1-13), MCMC (lines 14-33), REPORTING (lines 34-37).

Line 1: Setting the length of the Markov Chain to be generated.
Line 2-4: Providing the data, in three columns: time of measurement, value, uncertainty (standard deviation.)
Line 5: Setting lower and upper bounds of the prior parameter pdf, and start point of the chain.
Line 6: Defining the variance-covariance matrix for proposal generation.
Line 7: Initialising the chain.
Line 8: Calculating the prior probability density at the starting point of the chain, and log-transforming it.
Line 9: Defining the model, here defined as a MATLAB inline function with 2 parameters: intercept & slope.
Line 10: Calculating model outputs for the start point of the chain, i.e. the first parameter vector to be examined
Line 11-13: Calculating the log-likelihood of the data for the start point of the chain.
Line 15-16: Calculating the next candidate parameter vector, as a multivariate normal jump away from the current point.
Line 17-21: Calculating the prior probability density for the candidate parameter vector.
Line 23: Calculating the outputs for the candidate parameter-vector.
Line 24-26: Calculating the log-likelihood of the data given the outputs for the candidate parameter-vector.
Line 27: Calculating the logarithm of the Metropolis ratio (i.e. the ratio of posterior probability of candidate and current vector)
Line 28-29: Accepting or rejecting the candidate vector.

# Appendix 2
# MCMC code in MATLAB: Metropolis-with-Reflection

```
 1  chainLength    = 10000                                                          ;
 2  data           = [10, 6.09, 1.83 ; 20, 8.81, 2.64 ; 30, 10.66, 3.27 ]          ;
 3  pMinima        = [0, 0] ; pMaxima = [10, 1] ; pValues = [5, 0.5]                ;
 4  vcovProposal   = diag( (0.1*(pMaxima-pMinima)) .^2 )                           ;
 5  pChain         = zeros( chainLength, length(pValues) ) ; pChain(1,:) = pValues ;
 6  Prior0         = prod((pMaxima-pMinima).^-1) ; logPrior0 = log(Prior0)          ;
 7  model          = inline('intercept+slope*t')                                    ;
 8  y              = model(pValues(1),pValues(2),1:30)                              ;
 9  for i = 1:length(data),
10      logLi(i) = -0.5*((y(data(i,1))-data(i,2))/data(i,3))^2 -log(data(i,3))     ;
11  end ; logL0 = sum( logLi )                                                      ;
12  for c = 2 : chainLength,
13  candidatepValues = pValues + ...
14      transpose( transpose(chol(vcovProposal))*transpose(randn(size(pValues))) ) ;
15          reflectionFromMin = min( 0, candidatepValues-pMinima ) ;
16          reflectionFromMax = max( 0, candidatepValues-pMaxima ) ;
17          candidatepValues   = candidatepValues - 2 * reflectionFromMin - ...
18                                                  2 * reflectionFromMax        ;
19      if all(candidatepValues>pMinima) & all(candidatepValues<pMaxima),
20          Prior1 = prod((pMaxima-pMinima).^-1) ;
21      else
22          Prior1 = 0 ;
23      end;
24      if (Prior1 > 0),
25         y = model(candidatepValues(1),candidatepValues(2),1:30)                  ;
26         for i = 1:length(data),
27             logLi(i) = -0.5*((y(data(i,1))-data(i,2))/data(i,3))^2 -log(data(i,3));
28         end ; logL1 = sum( logLi )                                               ;
29         logalpha = (log(Prior1)+logL1) - (logPrior0+logL0)                       ;
30         if ( log(rand) < logalpha ),
31            pValues = candidatepValues ; logPrior0 = log(Prior1) ; logL0 = logL1 ;
32         end                                                                      ;
33      end                                                                         ;
34      pChain(c,:) = pValues                                                       ;
35  end                                                                             ;
36  nAccepted = length(unique(pChain(:,1))) ;
37  disp(['MCMC: ' num2str(nAccepted) ' candidate parameter vectors accepted (=' ...
38        num2str(round(100*nAccepted/chainLength)) '%)'])                          ;
39  disp(mean(pChain)) ; disp(cov(pChain))
```

This code, like the code in Appendix 1, carries out a Bayesian calibration for the 2 parameters of a straight line, by means of MCMC. The simplest algorithm is again used: Metropolis sampling. However, there is a small change to the proposal-generating method, i.e. the way new candidate vectors are generated. That is done here "with reflection" (Ter Braak, pers. comm.. 2006). This means that candidate parameter vectors that lie outside the bounds for any of the parameters are "reflected back" to generate an alternative candidate vector that is not outside the bounds and thus not automatically rejected. Including reflection can considerably reduce the number of candidate vectors that are rejected (e.g. the default Metropolis example in Appendix 1 has about 80% rejection rate, and the example with reflection in Appendix 2 about 40%).

Note that the proposal matrix, defined in line 4, has to be diagonal for this algorithm. We use a diagonal matrix also in the code of Appendix 1, for Standard Metropolis, but there we have the option of choosing otherwise.

Line 15-18: Reflection of a candidate vector if it lies outside the prior bounds.

# Appendix 3
# ACCEPT-REJECT code in MATLAB

```
 1  nPrior          = 10000                                                    ;
 2  data            = [10, 6.09, 1.83 ; 20, 8.81, 2.64 ; 30, 10.66, 3.27 ]    ;
 3  pMinima         = [0, 0] ; pMaxima = [10, 1] ;
 4  pCollectionPrior      = pMinima(1) + (pMaxima(1)-pMinima(1)) * rand(nPrior,1)  ;
 5  pCollectionPrior(:,2) = pMinima(2) + (pMaxima(2)-pMinima(2)) * rand(nPrior,1)  ;
 6  pCollectionPosterior  = []                                                   ;
 7  model           = inline('intercept+slope*t')                              ;
 8     for i = 1:length(data), logLmaxi(i) = - log(data(i,3)) ; end            ;
 9  logLmax         = sum( logLmaxi )                                          ;
10  for c = 1 : nPrior,
11      y = model(pCollectionPrior(c,1),pCollectionPrior(c,2),1:30)            ;
12      for i = 1:length(data),
13          logLi(i) = -0.5*((y(data(i,1))-data(i,2))/data(i,3))^2 - log(data(i,3));
14      end ; logL = sum( logLi )                                              ;
15      logalpha           = logL - logLmax                                    ;
16      if ( log(rand) < logalpha ),
17         pCollectionPosterior = [ pCollectionPosterior ; pCollectionPrior(c,:) ] ;
18      end                                                                    ;
19  end                                                                        ;
20  nAccepted = length(pCollectionPosterior) ;
21  disp(['A-R: ' num2str(nAccepted) ' parameter vectors sampled from prior accepted (=' ...
22        num2str(round(100*nAccepted/chainLength)) '%)'])                     ;
23  disp(  mean(pCollectionPosterior)) ; disp(cov(pCollectionPosterior))       ;
```

This code, like the code in Appendices 1 and 2, carries out a Bayesian calibration for the 2 parameters of a straight line, but by means of the Accept-Reject (or Rejection sampling) method instead of MCMC.

Line 1: Set the size of the sample that will be taken from the prior parameter pdf.
Line 2: Define the data.
Line 3: Define the prior parameter distribution, in this case by setting the bounds of the uniform pdf.
Line 4-5: Sample from the prior.
Line 6: Reserve memory space for the sample from the posterior.
Line 7: Define the model.
Line 8-9: Estimate the Maximum Likelihood. In this example we assume that the maximum is reached when the model goes exactly through each data point. That assumption is not in fact correct for this combination of data and model (the three points do not lie on a straight line, so there is always a model-data mismatch), so we are overestimating ML. However, A-R only may fail if we underestimate ML.
Line 10-19: For each point in the generated sample from the prior, calculate model output (Line 11), calculate the log-likelihood (Lines 12-14), calculate the (log of) the ratio of likelihood and ML (Line 15), and accept or reject (Line 16-18).

# Appendix 4
# MCMC code in R: the Metropolis algorithm

Marie Guillot, Nathalie Yauschew-Raguenes, Annabel Porté, Simon Lehuger

```
1   install.packages("mvtnorm")
2   require(mvtnorm)
3   chainLength    = 10000
4   data  <- matrix(c(10,6.09,1.83, 20,8.81,2.64, 30,10.66,3.27), nrow=3, ncol=3, byrow=T)
5   param <- matrix(c(0,5,10, 0,0.5,1)                          , nrow=2, ncol=3, byrow=T)
6   pMinima <- c(param[1,1], param[2,1])
7   pMaxima <- c(param[1,3], param[2,3])
8   logli <- matrix(, nrow=3, ncol=1)
9   vcovProposal   = diag( (0.05*(pMaxima-pMinima)) ^2 )
10  pValues <- c(param[1,2], param[2,2])
11  pChain <- matrix(0, nrow=chainLength, ncol = length(pValues)+1)
12  logPrior0 <- sum(log(dunif(pValues, min=pMinima, max=pMaxima)))
13  model <- function (times,intercept,slope) {y <- intercept+slope*times
14                                              return(y)}
15  for (i in 1:3) {logli[i] <- -0.5*((model(data[i,1],pValues[1],pValues[2])-
16                                   data[i,2])/data[i,3])^2 - log(data[i,3])}
17  logL0 <- sum(logli)
18  pChain[1,] <- c(pValues, logL0) # Keep first values
19  for (c in (2 : chainLength)){ candidatepValues <- rmvnorm(n=1, mean=pValues,
20  sigma=vcovProposal)
21  if (all(candidatepValues>pMinima) && all(candidatepValues<pMaxima))
22      {Prior1 <- prod(dunif(candidatepValues, pMinima, pMaxima))}
23  else {Prior1 <- 0}
24  if (Prior1 > 0)
25      { for (i in 1:3){logli[i] <-
26            -0.5*((model(data[i,1],candidatepValues[1],candidatepValues[2])-
27                   data[i,2])/data[i,3])^2 - log(data[i,3])}
28        logL1 <- sum(logli)
29  logalpha <- (log(Prior1)+logL1) - (logPrior0+logL0)
30  if ( log(runif(1, min = 0, max =1)) < logalpha )
31       { pValues <- candidatepValues
32         logPrior0 <- log(Prior1)
33         logL0 <- logL1}}
34         pChain[c,1:2] <- pValues
35         pChain[c,3] <- logL0     }
36  nAccepted = length(unique(pChain[,1]))
37  acceptance = (paste(nAccepted, "out of ", chainLength, "candidates accepted ( = ",
38                 round(100*nAccepted/chainLength),  "%)"))
39  print(acceptance)
40  mp <- apply(pChain, 2, mean)
41  print(mp)
42  pCovMatrix <- cov(pChain)
43  print(pCovMatrix)
```

This code carries out a Bayesian calibration for the 2 parameters of a straight line, by means of MCMC. Essentially, the code is the same as in Appendix 1, but here we use R instead of MATLAB. The point of departure was the code developed by Simon Lehuger (INRA-Grignon) for Bayesian calibration of the crop model CERES-EGC and this code was simplified by M. Guillot, N. Yauschew-Raguenes and A. Porté (INRA-Bordeaux). Most of the code is similar to the MATLAB version, except for the following:

Line 1&2: Install and call the library that contains the "rmvnorm" random number generator (multivariate Gaussian) that is used in line 19.