Environmental Modelling & Software 75 (2016) 273–316

Contents lists available at ScienceDirect

Environmental Modelling & Software

journal homepage: www.elsevier.com/locate/envsoft

# Markov chain Monte Carlo simulation using the DREAM software package: Theory, concepts, and MATLAB implementation

Jasper A. Vrugt [a, b, c, *]

[a] Department of Civil and Environmental Engineering, University of California Irvine, 4130 Engineering Gateway, Irvine, CA, 92697-2175, USA
[b] Department of Earth System Science, University of California Irvine, Irvine, CA, USA
[c] Institute for Biodiversity and Ecosystem Dynamics, University of Amsterdam, Amsterdam, The Netherlands

ARTICLE INFO

*Article history:*
Received 17 February 2015
Received in revised form
13 August 2015
Accepted 13 August 2015
Available online 14 November 2015

*Keywords:*
Bayesian inference
Markov chain Monte Carlo (MCMC) simulation
Random walk metropolis (RWM)
Adaptive metropolis (AM)
Differential evolution Markov chain (DE-MC)
Prior distribution
Likelihood function
Posterior distribution
Approximate Bayesian computation (ABC)
Diagnostic model evaluation
Residual analysis
Environmental modeling
Bayesian model averaging (BMA)
Generalized likelihood uncertainty estimation (GLUE)
Multi-processor computing
Extended metropolis algorithm (EMA)

ABSTRACT

Bayesian inference has found widespread application and use in science and engineering to reconcile Earth system models with data, including prediction in space (interpolation), prediction in time (forecasting), assimilation of observations and deterministic/stochastic model output, and inference of the model parameters. Bayes theorem states that the posterior probability, $p(H|\tilde{\mathbf{Y}})$ of a hypothesis, $H$ is proportional to the product of the prior probability, $p(H)$ of this hypothesis and the likelihood, $L(H|\tilde{\mathbf{Y}})$ of the same hypothesis given the new observations, $\tilde{\mathbf{Y}}$, or $p(H|\tilde{\mathbf{Y}}) \propto p(H)L(H|\tilde{\mathbf{Y}})$. In science and engineering, $H$ often constitutes some numerical model, $\mathscr{F}(\mathrm{x})$ which summarizes, in algebraic and differential equations, state variables and fluxes, all knowledge of the system of interest, and the unknown parameter values, $\mathbf{x}$ are subject to inference using the data $\tilde{\mathbf{Y}}$. Unfortunately, for complex system models the posterior distribution is often high dimensional and analytically intractable, and sampling methods are required to approximate the target. In this paper I review the basic theory of Markov chain Monte Carlo (MCMC) simulation and introduce a MATLAB toolbox of the DiffeRential Evolution Adaptive Metropolis (DREAM) algorithm developed by Vrugt et al. (2008a, 2009a) and used for Bayesian inference in fields ranging from physics, chemistry and engineering, to ecology, hydrology, and geophysics. This MATLAB toolbox provides scientists and engineers with an arsenal of options and utilities to solve posterior sampling problems involving (among others) bimodality, high-dimensionality, summary statistics, bounded parameter spaces, dynamic simulation models, formal/informal likelihood functions (GLUE), diagnostic model evaluation, data assimilation, Bayesian model averaging, distributed computation, and informative/noninformative prior distributions. The DREAM toolbox supports parallel computing and includes tools for convergence analysis of the sampled chain trajectories and post-processing of the results. Seven different case studies illustrate the main capabilities and functionalities of the MATLAB toolbox.



## 1. Introduction and scope

Continued advances in direct and indirect (e.g. geophysical, pumping test, remote sensing) measurement technologies and improvements in computational technology and process knowledge have stimulated the development of increasingly complex environmental models that use algebraic and (stochastic) ordinary (partial) differential equations (PDEs) to simulate the behavior of a myriad of highly interrelated ecological, hydrological, and biogeochemical processes at different spatial and temporal scales. These water, energy, nutrient, and vegetation processes are often non-separable, non-stationary with very complicated and highly-nonlinear spatio-temporal interactions (Wikle and Hooten, 2010) which gives rise to complex system behavior. This complexity poses significant measurement and modeling challenges, in particular how to adequately characterize the spatio-temporal processes of the dynamic system of interest, in the presence of (often)

* Department of Civil and Environmental Engineering, University of California Irvine, 4130 Engineering Gateway, Irvine, CA, 92697-2175, USA.
*E-mail address:* jasper@uci.edu.
*URL:* http://faculty.sites.uci.edu/jasper

http://dx.doi.org/10.1016/j.envsoft.2015.08.013
1364-8152/

incomplete and insufficient observations, process knowledge and system characterization. This includes prediction in space (interpolation/extrapolation), prediction in time (forecasting), assimilation of observations and deterministic/stochastic model output, and inference of the model parameters.

The use of differential equations might be more appropriate than purely empirical relationships among variables, but does not guard against epistemic errors due to incomplete and/or inexact process knowledge. Fig. 1 provides a schematic overview of most important sources of uncertainty that affect our ability to describe as closely and consistently as possible the observed system behavior. These sources of uncertainty have been discussed extensively in the literature, and much work has focused on the characterization of parameter, model output and state variable uncertainty. Explicit knowledge of each individual error source would provide strategic guidance for investments in data collection and/or model improvement. For instance, if input (forcing/boundary condition) data uncertainty dominates total simulation uncertainty, then it would not be productive to increase model complexity, but rather to prioritize data collection instead. On the contrary, it would be naive to spend a large portion of the available monetary budget on system characterization if this constitutes only a minor portion of total prediction uncertainty.

Note that model structural error (label 4) (also called epistemic error) has received relatively little attention, but is key to learning and scientific discovery (Vrugt et al., 2005; Vrugt and Sadegh, 2013).

The focus of this paper is on spatio-temporal models that may be discrete in time and/or space, but with processes that are continuous in both. A MATLAB toolbox is described which can be used to derive the posterior parameter (and state) distribution, conditioned on measurements of observed system behavior. At least some level of calibration of these models is required to make sure that the simulated state variables, internal fluxes, and output variables match the observed system behavior as closely and consistently as possible. Bayesian methods have found widespread application and use to do so, in particular because of their innate ability to handle, in a consistent and coherent manner parameter, state variable, and model output (simulation) uncertainty.

If $\tilde{\mathbf{Y}} = \{\tilde{y}_1, \ldots, \tilde{y}_n\}$ signifies a discrete vector of measurements at times $t = \{1,\ldots,n\}$ which summarizes the response of some environmental system $\Im$ to forcing variables $\mathbf{U} = \{\mathbf{u}_1,\ldots,\mathbf{u}_n\}$. The observations or data are linked to the physical system.

$$\tilde{\mathbf{Y}} \leftarrow \Im(\mathbf{x}^*) + \boldsymbol{\varepsilon}, \tag{1}$$

where $\mathbf{x}^* = \{x_1^*, \ldots, x_d^*\}$ are the unknown parameters, and $\boldsymbol{\varepsilon} = \{\varepsilon_1,\ldots,\varepsilon_n\}$ is a $n$-vector of measurement errors. When a hypothesis, or simulator, $\mathbf{Y} \leftarrow \mathscr{F}(\mathbf{x}^*, \tilde{\mathbf{u}}, \tilde{\psi}_0)$ of the physical process is available, then the data can be modeled using

$$\tilde{\mathbf{Y}} \leftarrow \mathscr{F}\left(\mathbf{x}^*, \tilde{\mathbf{U}}, \tilde{\psi}_0\right) + \mathbf{E}, \tag{2}$$

where $\tilde{\psi}_0 \in \boldsymbol{\Psi} \in \mathbb{R}^\tau$ signify the $\tau$ initial states, and $\mathbf{E} = \{e_1,\ldots,e_n\}$ includes observation error (forcing and output data) as well as error due to the fact that the simulator, $\mathscr{F}(\cdot)$ may be systematically different from reality, $\Im(\mathbf{x}^*)$ for the parameters $\mathbf{x}^*$. The latter may arise from numerical errors (inadequate solver and discretization), and improper model formulation and/or parameterization.

By adopting a Bayesian formalism the posterior distribution of the parameters of the model can be derived by conditioning the spatio-temporal behavior of the model on measurements of the observed system response

$$p\left(\mathbf{x}\middle|\tilde{\mathbf{Y}}\right) = \frac{p(\mathbf{x})p\left(\tilde{\mathbf{Y}}\middle|\mathbf{x}\right)}{p\left(\tilde{\mathbf{Y}}\right)}, \tag{3}$$

where $p(\mathbf{x})$ and $p\left(\mathbf{x}\middle|\tilde{\mathbf{Y}}\right)$ signify the prior and posterior parameter distribution, respectively, and $L\left(\mathbf{x}\middle|\tilde{\mathbf{Y}}\right) \equiv p\left(\tilde{\mathbf{Y}}\middle|\mathbf{x}\right)$ denotes the likelihood function. The evidence, $p(\tilde{\mathbf{Y}})$ acts as a normalization constant (scalar) so that the posterior distribution integrates to unity

$$p\left(\tilde{\mathbf{Y}}\right) = \int_{\chi} p(\mathbf{x})p\left(\tilde{\mathbf{Y}}\middle|\mathbf{x}\right)\mathrm{d}\mathbf{x} = \int_{\chi} p\left(\mathbf{x},\tilde{\mathbf{Y}}\right)\mathrm{d}\mathbf{x}, \tag{4}$$

over the parameter space, $\mathbf{x} \in \chi \in \mathbb{R}^d$. In practice, $p(\tilde{\mathbf{Y}})$ is not required for posterior estimation as all statistical inferences about $p\left(\mathbf{x}\middle|\tilde{\mathbf{Y}}\right)$ can be made from the unnormalized density

$$p\left(\mathbf{x}\middle|\tilde{\mathbf{Y}}\right) \propto p(\mathbf{x})L\left(\mathbf{x}\middle|\tilde{\mathbf{Y}}\right) \tag{5}$$

If we assume, for the time being, that the prior distribution, $p(\mathbf{x})$

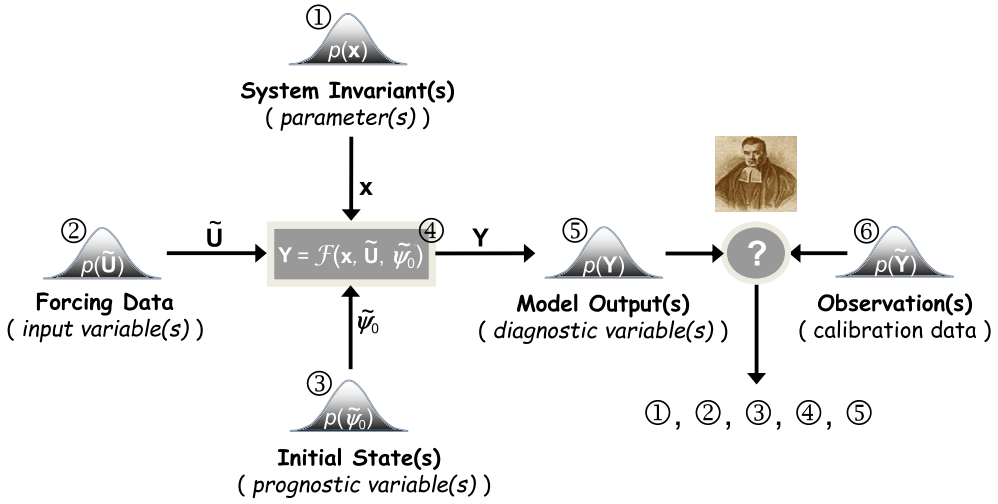


**Fig. 1.** Schematic illustration of the most important sources of uncertainty in environmental systems modeling, including (1) parameter, (2) input data (also called forcing or boundary conditions), (3), initial state, (4) model structural, (5) output, and (6) calibration data uncertainty. The measurement data error is often conveniently assumed to be known, a rather optimistic approach in most practical situations. Question remains how to describe/infer properly all sources of uncertainty in a coherent and statistically adequate manner.

is well defined, then the main culprit resides in the definition of the likelihood function, $L(\mathbf{x}|\tilde{\mathbf{Y}})$ used to summarize the distance between the model simulations and corresponding observations. If the error residuals are assumed to be uncorrelated then the likelihood of the $n$-vector of error residuals can be written as follows

$$L\left(\mathbf{x}\middle|\tilde{\mathbf{Y}}\right) = f_{\tilde{y}_1}(y_1(\mathbf{x})) \times f_{\tilde{y}_2}(y_2(\mathbf{x})) \times \ldots \times f_{\tilde{y}_n}(y_n(\mathbf{x})) = \prod_{t=1}^{n} f_{\tilde{y}_t}(y_t(\mathbf{x})), \tag{6}$$

where $f_a(b)$ signifies the probability density function of $a$ evaluated at $b$. If we further assume the error residuals to be normally distributed, $e_t(\mathbf{x}) \overset{D}{\sim} \mathcal{N}(0, \hat{\sigma}_t^2)$ then Equation (6) becomes

$$L\left(\mathbf{x}\middle|\tilde{\mathbf{Y}}, \hat{\boldsymbol{\sigma}}^2\right) = \prod_{t=1}^{n} \frac{1}{\sqrt{2\pi\hat{\sigma}_t^2}} \exp\left[-\frac{1}{2}\left(\frac{\tilde{y}_t - y_t(\mathbf{x})}{\hat{\sigma}_t}\right)^2\right], \tag{7}$$

where $\hat{\boldsymbol{\sigma}} = \{\hat{\sigma}_1, \ldots, \hat{\sigma}_n\}$ is a $n$-vector with standard deviations of the measurement error of the observations. This formulation allows for homoscedastic (constant variance) and heteroscedastic measurement errors (variance dependent on magnitude of data).[1] For reasons of numerical stability and algebraic simplicity it is often convenient to work with the log-likelihood, $\mathscr{L}(\mathbf{x}|\tilde{\mathbf{Y}}, \hat{\boldsymbol{\sigma}}^2)$ instead

$$\mathscr{L}\left(\mathbf{x}\middle|\tilde{\mathbf{Y}}, \hat{\boldsymbol{\sigma}}^2\right) = -\frac{n}{2}\log(2\pi) - \sum_{t=1}^{n}\{\log(\hat{\sigma}_t)\} - \frac{1}{2}\sum_{t=1}^{n}\left(\frac{\tilde{y}_t - y_t(\mathbf{x})}{\hat{\sigma}_t}\right)^2. \tag{8}$$

If the error residuals, $\mathbf{E}(\mathbf{x}) = \tilde{\mathbf{Y}} - \mathbf{Y}(\mathbf{x}) = \{e_1(\mathbf{x}), \ldots, e_n(\mathbf{x})\}$ exhibit temporal (or spatial) correlation then one can try to take explicit account of this in the derivation of the log-likelihood function. For instance, suppose the error residuals assume an AR(1)-process

$$e_t(\mathbf{x}) = c + \phi e_{t-1}(\mathbf{x}) + \eta_t\,, \tag{9}$$

with $\eta_t \overset{D}{\sim} \mathcal{N}(0, \hat{\sigma}_t^2)$, expectation $\mathbb{E}[e_t(\mathbf{x})] = c/(1-\phi)$, and variance $\mathrm{Var}[e_t(\mathbf{x})] = \hat{\sigma}^2/(1-\phi^2)$. This then leads to the following formulation of the log-likelihood (derivation in statistics textbooks)

$$\begin{aligned}\mathscr{L}\left(\mathbf{x}\middle|\tilde{\mathbf{Y}}, c, \phi, \hat{\boldsymbol{\sigma}}^2\right) = &-\frac{n}{2}\log(2\pi) - \frac{1}{2}\log\left[\hat{\sigma}_1^2\middle/\left(1-\phi^2\right)\right] \\ &- \frac{(e_1(\mathbf{x}) - [c/(1-\phi)])^2}{2\hat{\sigma}_1^2/(1-\phi^2)} - \sum_{t=2}^{n}\{\log(\hat{\sigma}_t)\} \\ &- \frac{1}{2}\sum_{t=2}^{n}\left(\frac{(e_t(\mathbf{x}) - c - \phi e_{t-1}(\mathbf{x}))}{\hat{\sigma}_t}\right)^2\end{aligned} \tag{10}$$

where $|\phi|<1$ signifies the first-order autoregressive coefficient. If we assume $c$ to be zero (absence of long-term trend) then Equation (10) reduces, after some rearrangement, to

$$\begin{aligned}\mathscr{L}\left(\mathbf{x}\middle|\tilde{\mathbf{Y}}, \phi, \hat{\boldsymbol{\sigma}}^2\right) = &-\frac{n}{2}\log(2\pi) + \frac{1}{2}\log\left(1-\phi^2\right) \\ &- \frac{1}{2}\left(1-\phi^2\right)\hat{\sigma}_1^{-2}e_1(\mathbf{x})^2 - \sum_{t=2}^{n}\{\log(\hat{\sigma}_t)\} \\ &- \frac{1}{2}\sum_{t=2}^{n}\left(\frac{(e_t(\mathbf{x}) - \phi e_{t-1}(\mathbf{x}))}{\hat{\sigma}_t}\right)^2,\end{aligned} \tag{11}$$

and the nuisance variables $\{\phi, \hat{\sigma}\}$ are subject to inference with the model parameters, $\mathbf{x}$ using the observed data, $\tilde{\mathbf{Y}}$[2].

Equation (11) is rather simplistic in that it assumes a-priori that the error residuals follow a stationary AR(1) process. This assumption might not be particularly realistic for real-world studies. Various authors have therefore proposed alternative formulations of the likelihood function to extend applicability to situations where the error residuals are non-Gaussian with varying degrees of kurtosis and skewness (Schoups and Vrugt, 2010; Smith et al., 2010; Evin et al., 2013; Scharnagl et al., 2015). Latent variables can also be used to augment likelihood functions and take better consideration of forcing data and model structural error (Kavetski et al., 2006a; Vrugt et al., 2008a; Renard et al., 2011). For systems with generative (negative) feedbacks, the error in the initial states poses no harm as its effect on system simulation rapidly diminishes when time advances. One can therefore take advantage of a spin-up period to remove sensitivity of the modeling results (and error residuals) to state value initialization.

The process of investigating phenomena, acquiring new information through experimentation and data collection, and refining existing theory and knowledge through Bayesian analysis has many elements in common with the scientific method. This framework, graphically illustrated in Fig. 2 is adopted in many branches of the earth sciences, and seeks to elucidate the rules that govern the natural world.

Once the prior distribution and likelihood function have been defined, what is left in Bayesian analysis is to summarize the posterior distribution, for example by the mean, the covariance or percentiles of individual parameters and/or nuisance variables. Unfortunately, most dynamic system models are highly nonlinear, and this task cannot be carried out by analytical means nor by analytical approximation. Confidence intervals construed from a classical first-order approximation can then only provide an approximate estimate of the posterior distribution. What is more, the target is assumed to be multivariate Gaussian ($\ell^2$-norm type likelihood function), a restrictive assumption. I therefore resort to Monte Carlo (MC) simulation methods to generate a sample of the posterior distribution.

In a previous paper, we have introduced the **D**iffe**R**ential **E**volution **A**daptive **M**etropolis (DREAM) algorithm (Vrugt et al., 2008a, 2009a). This multi-chain Markov chain Monte Carlo (MCMC) simulation algorithm automatically tunes the scale and orientation of the proposal distribution en route to the target distribution, and exhibits excellent sampling efficiencies on complex, high-dimensional, and multi-modal target distributions. DREAM is an adaptation of the Shuffled Complex Evolution Metropolis (Vrugt et al., 2003) algorithm and has the advantage of maintaining detailed balance and ergodicity. Benchmark experiments [e.g. (Vrugt et al., 2008a, 2009a; Laloy and Vrugt, 2012; Laloy et al., 2013; Linde and Vrugt, 2013; Lochbühler et al., 2014; Laloy et al., 2015)] have shown that DREAM is superior to other adaptive MCMC sampling approaches, and in high-dimensional search/variable

[1] If homoscedasticity is expected and the variance of the error residuals, $s^2 = \frac{1}{n-1}\sum_{t=1}^{n}(e_t(\mathbf{x}))^2$ is taken as sufficient statistic for $\hat{\sigma}^2$, then one can show that the likelihood function simplifies to $L\left(\mathbf{x}\middle|\tilde{\mathbf{Y}}\right) \propto \sum_{t=1}^{n}|e_t(\mathbf{x})|^{-n}$.

[2] A nuisance variable is a random variable that is fundamental to the probabilistic model, but that is not of particular interest itself.

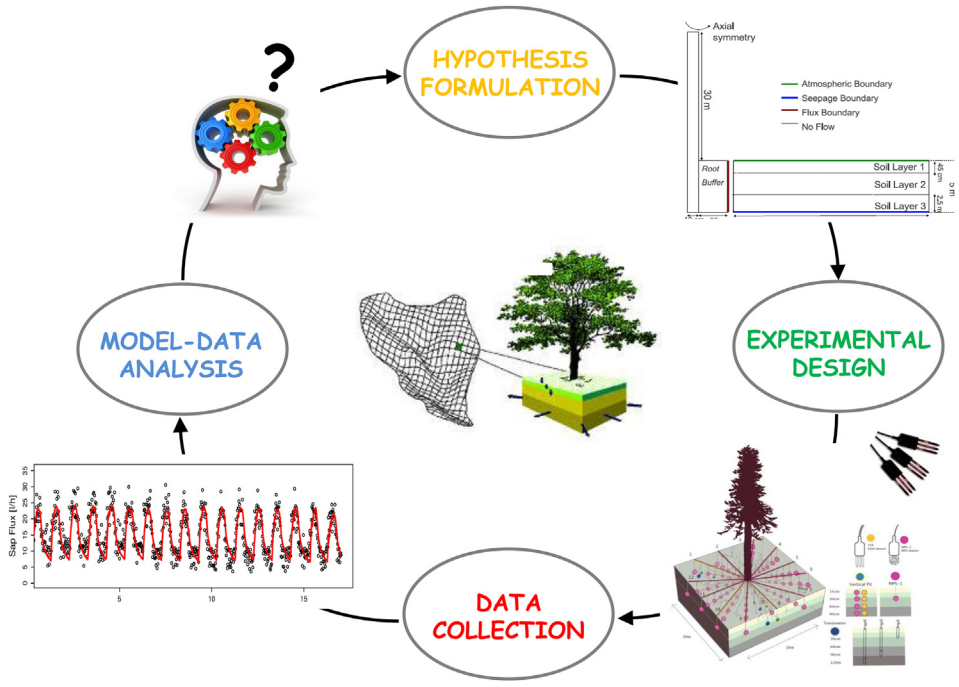


**Fig. 2.** The iterative research cycle for a soil-tree-atmosphere-continuum (STAC). The initial hypothesis is that this system can be described accurately with a coupled soil-tree porous media model which simulates, using PDEs, processes such as infiltration, soil evaporation, variably saturated soil water flow and storage, root water uptake, xylem water storage and sapflux, and leaf transpiration. Measurements of spatially distributed soil moisture and matric head, sapflux, and tree trunk potential are used for model calibration and evaluation. The model-data comparison step reveals a systematic deviation in the early afternoon and night time hours between the observed (black circles) and simulated (solid red line) sapflux data. It has proven to be very difficult to pinpoint this epistemic error to a specific component of the model. Ad-hoc decisions on model improvement therefore usually prevail. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

spaces even provides better solutions than commonly used optimization algorithms.

In just a few years, the DREAM algorithm has found widespread application and use in numerous different fields, including (among others) atmospheric chemistry (Partridge et al., 2011, 2012), biogeosciences (Scharnagl et al., 2010; Braakhekke et al., 2013; Ahrens and Reichstein, 2014; Dumont et al., 2014; Starrfelt and Kaste, 2014), biology (Coelho et al., 2011; Zaoli et al., 2014), chemistry (Owejan et al., 2012; Tarasevich et al., 2013; DeCaluwe et al., 2014; Gentsch et al., 2014), ecohydrology (Dekker et al., 2010), ecology (Barthel et al., 2011; Gentsch et al., 2014; Iizumi et al., 2014; Zilliox and Goselin, 2014), economics and quantitative finance (Bauwens et al., 2011; Lise et al., 2012; Lise, 2013), epidemiology (Mari et al., 2011; Rinaldo et al., 2012; Leventhal et al., 2013), geophysics (Bikowski et al., 2012; Linde and Vrugt, 2013; Laloy et al., 2012; Carbajal et al., 2014; Lochbühler et al., 2014, 2015), geostatistics (Minasny et al., 2011; Sun et al., 2013), hydrogeophysics (Hinnell et al., 2011), hydrologeology (Keating et al., 2010; Laloy et al., 2013; Malama et al., 2013), hydrology (Vrugt et al., 2008a, 2009a; Shafii et al., 2014), physics (Dura et al., 2011; Horowitz et al., 2012; Toyli et al., 2012; Kirby et al., 2013; Yale et al., 2013; Krayer et al., 2014), psychology (Turner and Sederberg, 2012), soil hydrology (Wöhling and Vrugt, 2011), and transportation engineering (Kow et al., 2012). Many of these publications have used the MATLAB toolbox of DREAM, which has been developed and written by the author of this paper, and shared with many individuals worldwide. Yet, the toolbox of DREAM has never been released formally through a software publication documenting how to use the code for Bayesian inference and posterior exploration.

In this paper, I review the basic theory of Markov chain Monte Carlo (MCMC) simulation, provide MATLAB scripts of some commonly used posterior sampling methods, and introduce a MATLAB toolbox of the DREAM algorithm. This MATLAB toolbox provides scientists and engineers with a comprehensive set of capabilities for application of the DREAM algorithm to Bayesian inference and posterior exploration. The DREAM toolbox implements multi-core computing (if user desires) and includes tools for convergence analysis of the sampled chain trajectories and post-processing of the results. Recent extensions of the toolbox are described as well, and include (among others) built-in functionalities that enable use of informal likelihood functions (Beven and Binley, 1992; Beven and Freer, 2001), summary statistics (Gupta et al., 2008), approximate Bayesian computation (Nott et al., 2012; Sadegh and Vrugt, 2013, 2014), diagnostic model evaluation (Vrugt and Sadegh, 2013), and the limits of acceptability framework (Beven, 2006; Beven and Binley, 2014). These developments are in part a response to the emerging paradigm of model diagnostics using summary statistics of system behavior. Recent work has shown that such approach provides better guidance on model malfunctioning and related issues than the conventional residual-based paradigm (Sadegh et al., 2015b; Vrugt, submitted for publication). The main capabilities of the DREAM toolbox are demonstrated using seven different case studies involving (for instance) bimodality, high-dimensionality, summary statistics, bounded parameter spaces, dynamic simulation models, formal/informal likelihood functions, diagnostic model evaluation, data assimilation, Bayesian model averaging, distributed computation, informative/non-informative prior distributions, and limits of acceptability. These example studies are easy to run and adapt and serve as

templates for other inference problems.

The present contribution follows papers by others in the same journal on the implementation of DREAM in high-level statistical languages such as R (Joseph and Guillaume, 2014) as well as general-purpose languages such as Fortran (Lu et al., 2014). Other unpublished versions of DREAM include codes in C (http://people.sc.fsu.edu/~jburkardt/c_src/dream/dream.html) and Python (https://pypi.python.org/pypi/multichain_mcmc/0.2.2). These different codes give potential users the option to choose their preferred language, yet these translations are based on source code supplied by the author several years ago and have limited functionalities compared to the MATLAB package described herein. The present code differs from its earlier versions in that it contains a suite of new options and new methodological developments (Vrugt and Sadegh, 2013; Sadegh and Vrugt, 2014; Vrugt, 2015a,submitted for publication).

The remainder of this paper is organized as follows. Section 2 reviews the basic theory of Monte Carlo sampling and MCMC simulation, and provides a MATLAB code of the Random Walk Metropolis algorithm. This is followed in Section 3 with a brief discussion of adaptive single and multi-chain MCMC methods. Here, I provide a source code of the basic DREAM algorithm. This source code has few options available to the user and Section 4 therefore introduces all the elements of the MATLAB toolbox of DREAM. This section is especially concerned with the input and output arguments of the DREAM program and the various functionalities, capabilities, and options available to the user. Section 5 of this paper illustrates the practical application of the DREAM toolbox to seven different case studies. These examples involve a wide variety of problem features, and illustrate some of the main capabilities of the MATLAB toolbox. In Section 6, I then discuss a few of the functionalities of the DREAM code not demonstrated explicitly in the present paper. Examples include Bayesian model selection using a new and robust integration method for inference of the marginal likelihood, $p(\tilde{\mathbf{Y}})$ (Volpi et al., 2015), the use of diagnostic Bayes to help detect epistemic errors (Vrugt, submitted for publication), and the joint treatment of parameter, model input (forcing) and output (calibration/evaluation) data uncertainty. In the penultimate section of this paper, I discuss relatives of the DREAM algorithm including DREAM$_{(ZS)}$, DREAM$_{(D)}$ (Vrugt et al., 2011), DREAM$_{(ABC)}$ (Sadegh and Vrugt, 2014), and MT-DREAM$_{(ZS)}$ (Laloy and Vrugt, 2012) and describe briefly how their implementation in MATLAB differs from the present toolbox. Finally, Section 8 concludes this paper with a summary of the work presented herein.

## 2. Posterior exploration

A key task in Bayesian inference is to summarize the posterior distribution. When this task cannot be carried out by analytical means nor by analytical approximation, Monte Carlo simulation methods can be used to generate a sample from the posterior distribution. The desired summary of the posterior distribution is then obtained from the sample. The posterior distribution, also referred to as the target or limiting distribution, is often high dimensional. A large number of iterative methods have been developed to generate samples from the posterior distribution. All these methods rely in some way on Monte Carlo simulation. The next sections discuss several different posterior sampling methods.

### 2.1. Monte Carlo simulation

Monte Carlo methods are a broad class of computational algorithms that use repeated random sampling to approximate some multivariate probability distribution. The simplest Monte Carlo method involves random sampling of the prior distribution. This method is known to be rather inefficient, which I can illustrate with a simple example. Lets consider a circle with unit radius in a square of size $\mathbf{x} \in [-2,2]^2$. The circle (posterior distribution) has an area of $\pi$ and makes up $\pi/16 \approx 0.196$ of the prior distribution. I can now use Monte Carlo simulation to estimate the value of $\pi$. I do so by randomly sampling $N = 10{,}000$ values of $\mathbf{x}$ from the prior distribution. The $M$ samples of $\mathbf{x}$ that fall within the circle are posterior solutions and indicated with the plus symbol in Fig. 3. Samples that fall outside the circle are rejected and printed with a dot. The value of can now be estimated using $\pi \approx 16M/N$ which in this numerical experiment with $N = 10{,}000$ samples equates to 3.0912.

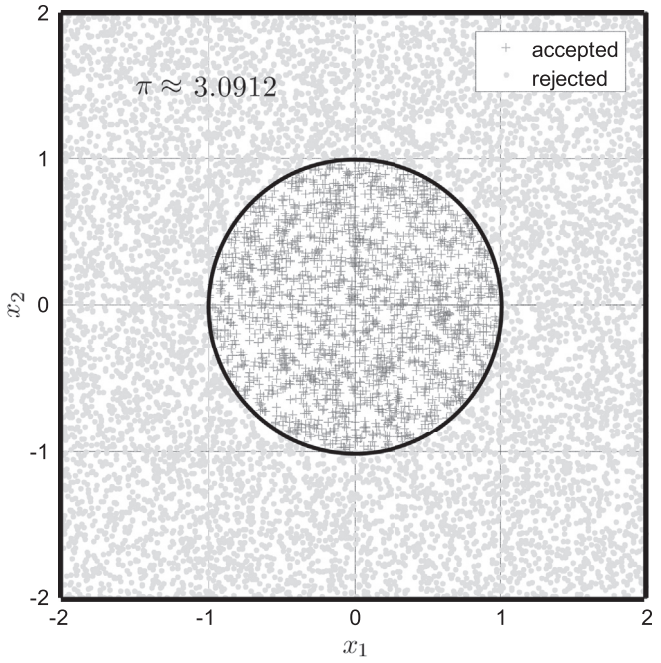


**Fig. 3.** Example target distribution: A square with unit radius (in black) centered at the origin. The Monte Carlo samples are coded in dots (rejected) and plusses (accepted). The number of accepted samples can now be used to estimate the value of $\pi \approx 3.0912$.

The target distribution is relatively simple to sample in the present example. It should be evident however that uniform random sampling will not be particularly efficient if the hypercube of the prior distribution is much larger. Indeed, the chance that a random sample of $\mathbf{x}$ falls within the unit circle decreases rapidly (quadratically) with increasing size of the prior distribution. If a much higher dimensional sample were considered then rejection sampling would quickly need many millions of Monte Carlo samples to delineate reasonably the posterior distribution and obtain an accurate value of $\pi$. What is more, in the present example all solutions within the circle have an equal density. If this were not the case then many accepted samples are required to approximate closely the distribution of the probability mass within the posterior distribution. Indeed, methods such as the generalized likelihood uncertainty estimation (GLUE) that rely on uniform sampling (such as rejection sampling) can produce questionable results if the target distribution is somewhat complex and/or comprises only a relatively small part of the prior distribution (Vrugt, 2015a). In summary, standard Monte Carlo simulation methods are computationally inefficient for anything but very low dimensional problems.

This example is rather simple but conveys what to expect when using simple Monte Carlo simulation methods to approximate complex and high-dimensional posterior distributions. I therefore

resort to Markov chain Monte Carlo simulation to explore the posterior target distribution.

### 2.2. Markov chain Monte Carlo simulation

The basis of MCMC simulation is a Markov chain that generates a random walk through the search space and successively visits solutions with stable frequencies stemming from a stationary distribution, $\overline{\pi}(\cdot)$.[3] To explore the target distribution, $\overline{\pi}(\cdot)$, a MCMC algorithm generates trial moves from the current state of the Markov chain $\mathbf{x}_{t-1}$ to a new state $\mathbf{x}_p$. The earliest MCMC approach is the random walk Metropolis (RWM) algorithm introduced by Metropolis et al. (1953). This scheme is constructed to maintain detailed balance with respect to $\overline{\pi}(\cdot)$ at each step in the chain. If $p(\mathbf{x}_{t-1})$ ($p(\mathbf{x}_p)$) denotes the probability to find the system in state $\mathbf{x}_{t-1}$ ($\mathbf{x}_p$) and $q(\mathbf{x}_{t-1} \rightarrow \mathbf{x}_p)(q(\mathbf{x}_p \rightarrow \mathbf{x}_{t-1}))$ is the conditional probability to perform a trial move from $\mathbf{x}_{t-1}$ to $\mathbf{x}_p$ ($\mathbf{x}_p$ to $\mathbf{x}_{t-1}$), then the probability $p_{acc}(\mathbf{x}_{t-1} \rightarrow \mathbf{x}_p)$ to accept the trial move from $\mathbf{x}_{t-1}$ to $\mathbf{x}_p$ is related to $p_{acc}(\mathbf{x}_p \rightarrow \mathbf{x}_{t-1})$ according to

$$p(\mathbf{x}_{t-1})q(\mathbf{x}_{t-1} \rightarrow \mathbf{x}_p)p_{acc}(\mathbf{x}_{t-1} \rightarrow \mathbf{x}_p) = p(\mathbf{x}_p)q(\mathbf{x}_p \rightarrow \mathbf{x}_{t-1})p_{acc}(\mathbf{x}_p \rightarrow \mathbf{x}_{t-1}) \quad (12)$$

If a symmetric jumping distribution is used, that is $q(\mathbf{x}_{t-1} \rightarrow \mathbf{x}_p) = q(\mathbf{x}_p \rightarrow \mathbf{x}_{t-1})$, then it follows that

$$\frac{p_{acc}(\mathbf{x}_{t-1} \rightarrow \mathbf{x}_p)}{p_{acc}(\mathbf{x}_p \rightarrow \mathbf{x}_{t-1})} = \frac{p(\mathbf{x}_p)}{p(\mathbf{x}_{t-1})} \quad (13)$$

This Equation does not yet fix the acceptance probability. Metropolis et al. (1953) made the following choice

$$p_{acc}(\mathbf{x}_{t-1} \rightarrow \mathbf{x}_p) = \min\left[1, \frac{p(\mathbf{x}_p)}{p(\mathbf{x}_{t-1})}\right], \quad (14)$$

to determine whether to accept a trial move or not. This selection rule has become the basic building block of many existing MCMC algorithms. Hastings (1970) extended Equation (14) to the more general case of non-symmetrical jumping distributions

$$p_{acc}(\mathbf{x}_{t-1} \rightarrow \mathbf{x}_p) = \min\left[1, \frac{p(\mathbf{x}_p)q(\mathbf{x}_p \rightarrow \mathbf{x}_{t-1})}{p(\mathbf{x}_{t-1})q(\mathbf{x}_{t-1} \rightarrow \mathbf{x}_p)}\right], \quad (15)$$

in which the forward ($\mathbf{x}_{t-1}$ to $\mathbf{x}_p$) and backward ($\mathbf{x}_p$ to $\mathbf{x}_{t-1}$) jump do not have equal probability, $q(\mathbf{x}_{t-1} \rightarrow \mathbf{x}_p) \neq q(\mathbf{x}_p \rightarrow \mathbf{x}_{t-1})$. This generalization is known as the Metropolis-Hastings (MH) algorithm and broadens significantly the type of proposal distribution that can be used for posterior inference.

The core of the RWM algorithm can be coded in just a few lines (see Algorithm 1) and requires only a jumping distribution, a function to generate uniform random numbers, and a function to calculate the probability density of each proposal. Note, for the time being I conveniently assume the use of a noninformative prior distribution. This simplifies the Metropolis acceptance probability to the ratio of the densities of the proposal and the current state of the chain. The use of an informative prior distribution will be considered at a later stage.

**ALGORITHM 1.** MATLAB function script of the Random Walk Metropolis (RWM) algorithm. Notation matches variable names used in main text. Based on input arguments prior, pdf, T and d, the RWM algorithm creates a Markov chain, x and corresponding densities, p_x. prior() is an anonymous function that draws N samples from a d-variate prior distribution. This function generates the initial state of the Markov chain. pdf() is another anonymous function that computes the density of the target distribution for a given vector of parameter values, x. Input arguments T and d signify the number of samples of the Markov chain and dimensionality of the parameter space, respectively. Built-in functions of MATLAB are highlighted with a low dash. The function handle q(C,d) is used to draw samples from a d-variate normal distribution, mvnrnd() with zero mean and covariance matrix, C. rand draws a value from a standard uniform distribution on the open interval (0,1), min() returns the smallest element of two different scalars, zeros() creates a zeroth vector (matrix), eye() computes the d × d identity matrix, sqrt() calculates the square root, and nan() fills each entry of a vector (matrix) with not a number.

```
function [x,p_x] = rwm(prior,pdf,T,d)
% Random Walk Metropolis (RWM) algorithm

q = @(C,d) mvnrnd(zeros(1,d),C);                % d-variate normal proposal distribution
C = (2.38/sqrt(d))^2 * eye(d);                  % Covariance matrix proposal distribution
x = nan(T,d); p_x = nan(T,1);                   % Preallocate memory for chain and density
x(1,1:d) = prior(1,d);                          % Initialize chain by sampling from prior
p_x(1) = pdf(x(1,1:d));                         % Compute density initial state chain

for t = 2:T,    % Dynamic part: evolution of chain
    xp = x(t-1,1:d) + q(C,d);                   % Create candidate point
    p_xp = pdf(xp);                             % Calculate density proposal
    p_acc = min(1,p_xp/p_x(t-1));               % Compute p_accept
    if p_acc > rand,                            % p_acc larger than U[0,1]?
        x(t,1:d) = xp; p_x(t) = p_xp;           % True: accept proposal
    else
        x(t,1:d) = x(t-1,1:d); p_x(t) = p_x(t-1); % False: copy old state
    end
end             % End dynamic part
```

In words, assume that the points $\{\mathbf{x}_0,\ldots,\mathbf{x}_{t-1}\}$ have already been sampled, then the RWM algorithm proceeds as follows. First, a candidate point $\mathbf{x}_p$ is sampled from a proposal distribution $q$ that depends on the present location, $\mathbf{x}_{t-1}$ and is symmetric, $q(\mathbf{x}_{t-1},\mathbf{x}_p) = q(\mathbf{x}_p, \mathbf{x}_{t-1})$. Next, the candidate point is either accepted or rejected using the Metropolis acceptance probability (Equation (14)). Finally,

[3] This notation for the target distribution has nothing to do with the value of $\pi = 3.1415...$ subject to inference in Fig. 3.

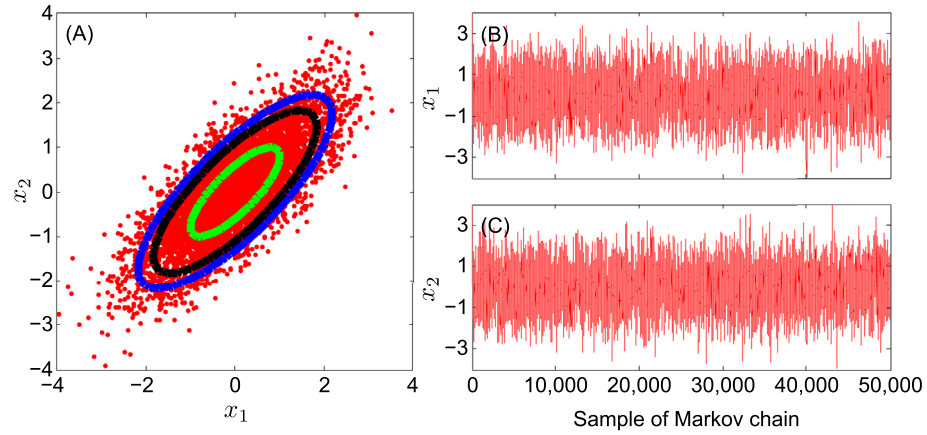


**Fig. 4.** (A) bivariate scatter plots of the RWM derived posterior samples. The green, black and blue contour lines depict the true 68, 90 and 95% uncertainty intervals of the target distribution, respectively. (B,C) traceplot of the sampled values of $x_1$ (top) and $x_2$ (bottom). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

if the proposal is accepted the chain moves to $\mathbf{x}_p$, otherwise the chain remains at its current location $\mathbf{x}_{t-1}$. Repeated application of these three steps results in a Markov chain which, under certain regularity conditions, has a unique stationary distribution with posterior probability density function, $\overline{\pi}(\cdot)$. In practice, this means that if one looks at the values of $\mathbf{x}$ sufficiently far from the arbitrary initial value, that is, after a burn-in period, the successively generated states of the chain will be distributed according to $\overline{\pi}(\cdot)$, the posterior probability distribution of $\mathbf{x}$. Burn-in is required to allow the chain to explore the search space and reach its stationary regime.

Fig. 4 illustrates the outcome of the RWM algorithm for a simple $d = 2$-dimensional multivariate normal target distribution with correlated dimensions. This target distribution is specified as anonymous function (a function not stored as program file) in MATLAB

$$\text{pdf} = @(\text{x})\ \text{mvnpdf}(\text{x}, [0\ 0],\ [1\ 0.8;\ 0.8\ 1]) \tag{16}$$

where the @ operator creates the handle, and the parentheses contain the actual function itself. This anonymous function accepts a single input $\mathbf{x}$, and implicitly returns a single output, a vector (or scalar) of posterior density values with same number of rows as $\mathbf{x}$.

The chain is initialized by sampling from $\mathscr{U}_2[-10, 10]$, where $\mathscr{U}_d(a, b)$ denotes the $d$-variate uniform distribution with lower and upper bounds $a$ and $b$, respectively, and thus

$$\text{prior} = @(\text{N}, \text{d})\ \text{unifrnd}(-10,\ 10,\ \text{N},\ \text{d}) \tag{17}$$

The left graph presents a scatter plot of the bivariate posterior samples using a total of $T = 50{,}000$ function evaluations and burn-in of 50%. The contours depict the 68, 90, and 95% uncertainty intervals of the target distribution. The right graph displays a plot of the generation number against the value of parameter, $x_1$ and $x_2$ at each iteration. This is also called a traceplot.

Perhaps not surprisingly, the bivariate samples of the RWM algorithm nicely approximate the target distribution. The acceptance rate of 23% is somewhat lower than considered optimal in theory but certainly higher than derived from Monte Carlo simulation. The posterior mean and covariance are in excellent agreement with their values of the target distribution (not shown).

This simple example just serves to demonstrate the ability of RWM to approximate the posterior target distribution. The relative ease of implementation of RWM and its theoretical underpinning have led to widespread application and use in Bayesian inference. However, the efficiency of the RWM algorithm is determined by the choice of the proposal distribution, $q(\cdot)$ used to create trial moves (transitions) in the Markov chain. When the proposal distribution is too wide, too many candidate points are rejected, and therefore the chain will not mix efficiently and converge only slowly to the target distribution. On the other hand, when the proposal distribution is too narrow, nearly all candidate points are accepted, but the distance moved is so small that it will take a prohibitively large number of updates before the sampler has converged to the target distribution. The choice of the proposal distribution is therefore crucial and determines the practical applicability of MCMC simulation in many fields of study (Owen and Tribble, 2005).

## 3. Automatic tuning of proposal distribution

In the past decade, a variety of different approaches have been proposed to increase the efficiency of MCMC simulation and enhance the original RWM and MH algorithms. These approaches can be grouped into single and multiple chain methods.

### 3.1. Single-chain methods

The most common adaptive single chain methods are the adaptive proposal (AP) (Haario et al., 1999), adaptive Metropolis (AM) (Haario et al., 2001) and delayed rejection adaptive metropolis (DRAM) algorithm (Haario et al., 2006), respectively. These methods work with a single trajectory, and continuously adapt the covariance, $\Sigma$ of a Gaussian proposal distribution, $q_t(\mathbf{x}_{t-1}, \cdot) = \mathscr{N}_d(\mathbf{x}_{t-1}, s_d\Sigma)$ using the accepted samples of the chain, $\Sigma = \text{cov}(\mathbf{x}_0,\ldots,\mathbf{x}_{t-1}) + \varphi\mathbf{I}_d$. The variable $s_d$ represents a scaling factor (scalar) that depends only on the dimensionality $d$ of the problem, $\mathbf{I}_d$ signifies the $d$-dimensional identity matrix, and $\varphi = 10^{-6}$ is a small scalar that prevents the sample covariance matrix to become singular. As a basic choice, the scaling factor is chosen to be $s_d = 2.38^2/d$ which is optimal for Gaussian target and proposal distributions (Gelman et al., 1996; Roberts et al., 1997) and should give an acceptance rate close to 0.44 for $d = 1$, 0.28 for $d = 5$ and 0.23 for large $d$. A MATLAB code of the AM algorithm is given on the next page (see Algorithm 2).

**ALGORITHM 2**. Basic MATLAB code of adaptive Metropolis (AM) algorithm. This code is similar to that of the RWM algorithm in Algorithm 1 but the d × d covariance matrix, C of the proposal distribution, q() is adapted using the samples stored in the Markov chain. Built-in functions of MATLAB are highlighted with a low dash. mod() signifies the modulo operation, and cov() computes the covariance matrix of the chain samples, x.

```
function [x,p_x] = am(prior,pdf,T,d)
% Adaptive Metropolis (AM) algorithm

q = @(C,d) mvnrnd(zeros(1,d),C);                % d-variate normal proposal distribution
C = (2.38/sqrt(d))^2 * eye(d);                  % Covariance matrix proposal distribution
x = nan(T,d); p_x = nan(T,1);                   % Preallocate memory for chain and density
x(1,1:d) = prior(1,d);                          % Initialize chain by sampling from prior
p_x(1) = pdf(x(1,1:d));                         % Compute density initial state chain

for t = 2:T,    % Dynamic part: evolution of chain
    % --------------- Adaptation covariance matrix of proposal distribution ----------------
    if ( mod(t,10) == 0 )
        C = (2.38/sqrt(d))^2 * (cov(x(1:t-1,1:d)) + 1e-4*eye(d));
        % Note: recursive formulae for C much more CPU-efficient!
    end
    % ---------------------------------- End adaptation -----------------------------------
    xp = x(t-1,1:d) + q(C,d);                   % Create candidate point
    p_xp = pdf(xp);                             % Calculate density proposal
    p_acc = min(1,p_xp/p_x(t-1));               % Compute p_accept
    if p_acc > rand,                            % p_acc larger than U[0,1]?
        x(t,1:d) = xp; p_x(t) = p_xp;           % True: accept proposal
    else
        x(t,1:d) = x(t-1,1:d); p_x(t) = p_x(t-1); % False: copy old state
    end
end             % End dynamic part
```

Single-site updating of **x** (Haario et al., 2005) is possible to increase efficiency of AM for high-dimensional problems (large $d$). In addition, for the special case of hierarchical Bayesian inference of hydrologic models, Kuczera et al. (2010) proposed to tune Σ using a limited-memory multi-block pre-sampling step, prior to a classical single block Metropolis run.

Another viable adaptation strategy is to keep the covariance matrix fixed (identity matrix) and to update during burn-in the scaling factor, $s_d$ until a desired acceptance rate is achieved. This approach differs somewhat from the AM algorithm but is easy to implement (see Algorithm 3).

**ALGORITHM 3**. Metropolis algorithm with adaptation of the scaling factor, $s_d$ rather than covariance matrix instead. The scaling factor is updated after each 25 successive generations to reach a desired acceptance rate between 20 and 30%. The multipliers of 0.8 and 1.2 are by no means generic values and should be determined through trial-and-error. Note, adaptation is restricted to the the first half of the Markov chain to ensure reversibility of the last 50% of the samples.

```
function [x,p_x] = am_sd(prior,pdf,T,d)
% Metropolis algorithm with adaptive scaling factor (jump rate)

q = @(C,d) mvnrnd(zeros(1,d),C);                % d-variate normal proposal distribution
C = (2.38/sqrt(d))^2 * eye(d);                  % Covariance matrix proposal distribution
x = nan(T,d); p_x = nan(T,1);                   % Preallocate memory for chain and density
x(1,1:d) = prior(1,d);                          % Initialize chain by sampling from prior
p_x(1) = pdf(x(1,1:d));                         % Compute density initial state chain
accept = 1;                                     % First sample has been accepted

for t = 2:T,    % Dynamic part: evolution of chain
    % ----------------- Adaptation scaling factor of proposal distribution -----------------
    if (mod(t,25)==0) && (t<T/2)
        AR = 100*(accept/(t-1));                % Calculate acceptance rate
        if AR < 20, s_d = 0.8*s_d; end          % Reduce scaling factor
        if AR > 30, s_d = 1.2*s_d; end          % Increase scaling factor
    end
    C = (s_d+1e-4)*eye(d);                      % Compute covariance prop. dist.
    % ---------------------------------- End adaptation -----------------------------------
    xp = x(t-1,1:d) + q(C,d);                   % Create candidate point
    p_xp = pdf(xp);                             % Calculate density proposal
    p_acc = min(1,p_xp/p_x(t-1));               % Compute p_accept
    if p_acc > rand,                            % p_acc larger than U[0,1]?
        x(t,1:d) = xp; p_x(t) = p_xp;           % True: accept proposal
    else
        x(t,1:d) = x(t-1,1:d); p_x(t) = p_x(t-1); % False: copy old state
    end
    accept = accept + idx;                      % How many samples accepted?
end             % End dynamic part
```

Whether a specific adaptation scheme of the scaling factor (also called jump rate) works well in practice depends on the properties of the target distribution. Some tuning is hence required to achieve adequate results. Practical experience suggests that covariance matrix adaptation (AM) is preferred over scaling factor adaptation. The proposals created with the AM algorithm will more rapidly behave as the target distribution.

The use of a multivariate normal proposal distribution with adaptive covariance matrix or jump rate works well for Gaussian-shaped target distributions, but does not converge properly for multimodal distributions with long tails, possibly infinite first and second moments (as demonstrated in section). Experience further suggests that single chain methods are unable to traverse efficiently complex multi-dimensional parameter spaces with multiple different regions of attraction and numerous local optima. The use of an overly dispersed proposal distribution can help to avoid premature convergence, but with a very low acceptance rate in return. With a single chain it is also particularly difficult to judge when convergence has been achieved. Even the most powerful diagnostics that compare the sample moments of the first and second half of the chain cannot guarantee that the target distribution has been sampled. Indeed, the sample moments of both parts of the chain might be identical but the chain is stuck in a local optimum of the posterior surface or traverses consistently only a portion of the target distribution (Gelman and Shirley, 2009). In fact, single chain methods suffer many similar problems as local optimizers and cannot guarantee that the full parameter space has been explored adequately in pursuit of the target distribution.

### 3.2. Multi-chain methods: DE-MC

Multiple chain methods use different trajectories running in parallel to explore the posterior target distribution. The use of multiple chains has several desirable advantages, particularly when dealing with complex posterior distributions involving long tails, correlated parameters, multi-modality, and numerous local optima (Gilks et al., 1994; Liu et al., 2000; ter Braak, 2006; ter Braak and Vrugt, 2008; Vrugt et al., 2009a; Radu et al., 2009). The use of multiple chains offers a robust protection against premature convergence, and opens up the use of a wide arsenal of statistical measures to test whether convergence to a limiting distribution has been achieved (Gelman and Rubin, 1992). One popular multi-chain method that has found widespread application and use in hydrology is the Shuffled Complex Evolution Metropolis algorithm (SCEM-UA, Vrugt et al., 2003). Although the proposal adaptation of SCEM-UA violates Markovian properties, numerical benchmark experiments on a diverse set of multi-variate target distributions have shown that the method is efficient and close to exact. The difference between the limiting distribution of SCEM-UA and the true target distribution is negligible in most reasonable cases and applications. The SCEM-UA method can be made an exact sampler if the multi-chain adaptation of the covariance matrix is restricted to the burn-in period only. In a fashion similar to the AP (Haario et al., 1999) and AM algorithm, the method then derives an efficient Gaussian proposal distribution for the standard Metropolis algorithm. Nevertheless, I do not consider the SCEM-UA algorithm herein.

ter Braak (2006) proposed a simple adaptive RWM algorithm called Differential Evolution Markov chain (DE-MC). DE-MC uses differential evolution as genetic algorithm for population evolution with a Metropolis selection rule to decide whether candidate points should replace their parents or not. In DE-MC, $N$ different Markov chains are run simultaneously in parallel. If the state of a single chain is given by the $d$-vector $\mathbf{x}$, then at each generation $t-1$ the $N$ chains in DE-MC define a population, $\mathbf{X}$ which corresponds to an $N \times d$ matrix, with each chain as a row. Then multivariate proposals, $\mathbf{X}_p$ are generated on the fly from the collection of chains, $\mathbf{X} = \{\mathbf{x}_{t-1}^1, \ldots, \mathbf{x}_{t-1}^N\}$ using differential evolution (Storn and Price, 1997; Price et al., 2005)

$$\mathbf{X}_p^i = \gamma_d\left(\mathbf{X}^a - \mathbf{X}^b\right) + \zeta_d, \quad a \neq b \neq i, \tag{18}$$

where $\gamma$ denotes the jump rate, $a$ and $b$ are integer values drawn without replacement from $\{1,\ldots,i-1,i+1,\ldots,N\}$, and $\zeta \overset{D}{\sim} \mathcal{N}_d(0, c_*)$ is drawn from a normal distribution with small standard deviation, say $c_* = 10^{-6}$. By accepting each proposal with Metropolis probability

$$p_{\text{acc}}\left(\mathbf{X}^i \rightarrow \mathbf{X}_p^i\right) = \min\left[1, p\left(\mathbf{X}_p^i\right) \Big/ p\left(\mathbf{X}^i\right)\right], \tag{19}$$

a Markov chain is obtained, the stationary or limiting distribution of which is the posterior distribution. The proof of this is given in ter Braak and Vrugt (2008). Thus, if $p_{\text{acc}}(\mathbf{X}^i \rightarrow \mathbf{X}_p^i)$ is larger than some uniform label drawn from $\mathcal{U}(0,1)$ then the candidate point is accepted and the $i$th chain moves to the new position, that is $\mathbf{x}_t^i = \mathbf{X}_p^i$, otherwise $\mathbf{x}_t^i = \mathbf{x}_{t-1}^i$.

Because the joint pdf of the $N$ chains factorizes to $\overline{\pi}(\mathbf{x}^1|\cdot) \times \ldots \times \overline{\pi}(\mathbf{x}^N|\cdot)$, the states $\mathbf{x}^1 \ldots \mathbf{x}^N$ of the individual chains are independent at any generation after DE-MC has become independent of its initial value. After this burn-in period, the convergence of a DE-MC run can thus be monitored with the $\widehat{R}$-statistic of Gelman and Rubin (1992). If the initial population is drawn from the prior distribution, then DE-MC translates this sample into a posterior population. From the guidelines of $s_d$ in RWM the optimal choice of $\gamma = 2.38/2d$. With a 10% probability the value of $\gamma = 1$, or $p_{(\gamma=1)} = 0.1$ to allow for mode-jumping (ter Braak, 2006; ter Braak and Vrugt, 2008; Vrugt et al., 2008a, 2009a) which is a significant strength of DE-MC as will be shown later. If the posterior distribution consists of disconnected modes with in-between regions of low probability, covariance based MCMC methods will be slow to converge as transitions between probability regions will be infrequent.

The DE-MC method can be coded in MATLAB in about 20 lines (Algorithm 4), and solves an important practical problem in RWM, namely that of choosing an appropriate scale and orientation for the jumping distribution. Earlier approaches such as (parallel) adaptive direction sampling (Gilks et al., 1994; Roberts and Gilks, 1994; Gilks and Roberts, 1996) solved the orientation problem but not the scale problem.

Based on input arguments, prior, pdf, $N$, $T$, and $d$, defined by the user de_mc returns a sample from the posterior distribution. prior is an anonymous function that draws $N$ samples from a $d$-variate prior distribution, and similarly pdf is a function handle which computes the posterior density of a proposal (candidate point).

To demonstrate the advantages of DE-MC over single chain methods please consider Fig. 5 that presents histograms of the posterior samples derived from AM (left plot) and DE-MC (right plot) for a simple univariate target distribution consisting of a mixture of two normal distributions

$$p(x) = \frac{1}{6}\psi(-8,1) + \frac{5}{6}\psi(10,1), \tag{20}$$

where $\psi(a,b)$ denotes the probability density function (pdf) of a normal distribution with mean $a$ and standard deviation $b$. The target distribution is displayed with a solid black line, and in MATLAB language equivalent to

**ALGORITHM 4.** MATLAB code of differential evolution-Markov chain (DE-MC) algorithm. Variable use is consistent with symbols used in main text. Based on input arguments prior, pdf, N, T and d, the DE-MC algorithm evolves N different trajectories simultaneously to produce a sample of the posterior target distribution. Jumps in each chain are computed from the remaining N-1 chains. The output arguments x and p_x store the sampled Markov chain trajectories and corresponding density values, respectively. Built-in functions are highlighted with a low dash. randsample draws with replacement 'true' the value of the jump rate, gamma from the vector [gamma_RWM 1] using selection probabilities [0.9 0.1]. randn() returns a row vector with d draws from a standard normal distribution. I refer to introductory textbooks and/or the MATLAB "help" utility for its built-in functions setdiff(), and reshape().

```
function [x,p_x] = de_mc(prior,pdf,N,T,d)
% Differential Evolution Markov Chain (DE-MC) algorithm

gamma_RWM = 2.38/sqrt(2*d);                              % Calculate default jump rate
x = nan(T,d,N); p_x = nan(T,N);                          % Preallocate chains and density
X = prior(N,d);                                          % Create initial population
for i = 1:N, p_X(i,1) = pdf(X(i,1:d)); end               % Compute density initial population
x(1,1:d,1:N) = reshape(X',1,d,N); p_x(1,1:N) = p_X';     % Store initial states and density
for i = 1:N, R(i,1:N-1) = setdiff(1:N,i); end            % R-matrix: index of chains for DE

for t = 2:T,    % Dynamic part: Evolution of N chains
    [~,draw] = sort(rand(N-1,N));                        % Permute [1,...,N-1] N times
    g = randsample([gamma_RWM 1],1,'true',[0.9 0.1]);    % Select gamma: 90/10 mix [default 1]
    for i = 1:N,                                         % Create proposals and accept/reject
        a = R(i,draw(1,i)); b = R(i,draw(2,i));          % Extract a and b not equal i
        Xp(i,1:d) = X(i,1:d) + g*(X(a,1:d)-X(b,1:d))...
            + 1e-6*randn(1,d);                           % Create ith proposal with diff. evol.
        p_Xp(i,1) = pdf(Xp(i,1:d));                      % Calculate density ith proposal
        p_acc = min(1,p_Xp(i,1)/p_X(i,1));               % Compute acceptance probability
        if p_acc > rand,                                 % p_acc larger than U[0,1]?
            X(i,1:d) = Xp(i,1:d); p_X(i,1) = p_Xp(i,1);  % True: Accept proposal
        end
    end
    x(t,1:d,1:N) = reshape(X',1,d,N); p_x(t,1:N) = p_X'; % Append current X and density
end             % End dynamic part
```

$$\texttt{pdf} = @(x)\ 1/6*\texttt{normpdf}(x,-8,1) + 5/6*\texttt{normpdf}(x,10,1). \tag{21}$$

The initial state of the Markov chain(s) is sampled from $\mathscr{U}[-20,20]$ using

$$\texttt{prior} = @(N,d)\ \texttt{unifrnd}(-20,20,N,d). \tag{22}$$

The AM algorithm produces a spurious approximation of the bimodal target distribution. The variance (width) of the proposal distribution is insufficient to enable the chain to adequately explore both modes of the target distribution. A simple remedy to this problem is to increase the (default) initial variance of the univariate normal proposal distribution. This would allow the AM sampler to take much larger steps and jump directly between both modes, but at the expense of a drastic reduction in the acceptance rate and search efficiency. Indeed, besides the occasional successful jumps many other proposals will overshoot the target distribution, receive a nearly zero density, and consequently be rejected.

This rather simple univariate example illustrates the dilemma of RWM how to determine an appropriate scale and orientation of the proposal distribution. Fortunately, the histogram of the posterior samples derived with the DE-MC algorithm matches perfectly the mixture distribution. Periodic use of $\gamma = 1$ enables the $N = 10$ different Markov chains of DE-MC to transition directly between the two disconnected posterior modes (e.g. ter Braak and Vrugt (2008); Vrugt et al. (2008a); Laloy and Vrugt (2012)) and

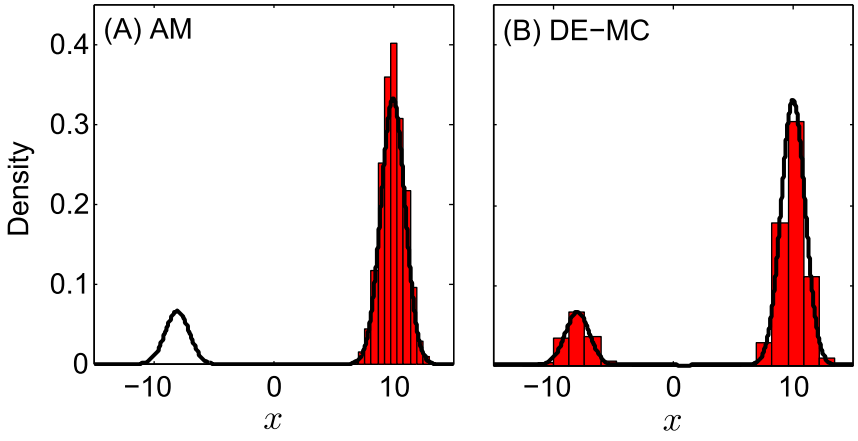


**Fig. 5.** Histogram of the posterior distribution derived from the (A) AM (single chain), and (B) DE-MC (multi-chain) samplers. The solid black line displays the pdf of the true mixture target distribution.

rapidly converge to the exact target distribution. The initial states of the DE-MC chains should be distributed over the parameter space so that both modes can be found. What is more the use of $N$ trajectories allows for a much more robust assessment of convergence.

In previous work (Vrugt et al., 2008a, 2009a) we have shown that the efficiency of DE-MC can be enhanced, sometimes dramatically, by using adaptive randomized subspace sampling, multiple chain pairs for proposal creation, and explicit consideration of aberrant trajectories. This method, entitled **D**iffe**R**ential **E**volution **A**daptive **M**etropolis (DREAM) maintains detailed balance and ergodicity and has shown to exhibit excellent performance on a wide range of problems involving nonlinearity, high-dimensionality, and multimodality. In these and other papers [e.g (Laloy and Vrugt, 2012)] benchmark experiments have shown that DREAM outperforms other adaptive MCMC sampling approaches, and, in high-dimensional search/variable spaces, can even provide better solutions than commonly used optimization algorithms.

### 3.3. Multi-chain methods: DREAM

The DREAM algorithm has it roots within DE-MC but uses subspace sampling and outlier chain correction to speed up convergence to the target distribution. Subspace sampling is implemented in DREAM by only updating randomly selected dimensions of $\mathbf{x}$ each time a proposal is generated. If A is a subset of $d^*$-dimensions of the original parameter space, $\mathbb{R}^{d^*} \subseteq \mathbb{R}^d$, then a jump, $\mathbf{dX}^i$ in the $i$th chain, $i = \{1,\ldots,N\}$ at iteration $t = \{2,\ldots,T\}$ is calculated from the collection of chains, $\mathbf{X} = \{\mathbf{x}_{t-1}^1, \ldots, \mathbf{x}_{t-1}^N\}$ using differential evolution (Storn and Price, 1997; Price et al., 2005)

$$\begin{aligned} \mathbf{dX}_{\mathrm{A}}^i &= \zeta_{d^*} + (1_{d^*} + \lambda_{d^*})\gamma_{(\delta,d^*)} \sum_{j=1}^{\delta} \left( \mathbf{X}_{\mathrm{A}}^{\mathbf{a}_j} - \mathbf{X}_{\mathrm{A}}^{\mathbf{b}_j} \right) \\ \mathbf{dX}_{\neq \mathrm{A}}^i &= 0, \end{aligned} \tag{23}$$

where

$$\gamma = \frac{2.38}{\sqrt{2\delta d^*}}$$

is the jump rate, $\delta$ denotes the number of chain pairs used to generate the jump, and $\mathbf{a}$ and $\mathbf{b}$ are vectors consisting of $\delta$ integers drawn without replacement from $\{1,\ldots,i-1,i+1,\ldots,N\}$. The default value of $\delta = 3$, and results, in practice, in one-third of the proposals being created with $\delta = 1$, another third with $\delta = 2$, and the remaining third using $\delta = 3$. The values of $\lambda$ and $\zeta$ are sampled independently from $\mathscr{U}_{d^*}(-c, c)$ and $\mathscr{N}_{d^*}(0, c_*)$, respectively, the multivariate uniform and normal distribution with, typically, $c = 0.1$ and $c^*$ small compared to the width of the target distribution, $c^* = 10^{-6}$ say. Compared to DE-MC, $p_{(\gamma=1)} = 0.2$ to enhance the probability of jumps between disconnected modes of the target distribution. The candidate point of chain $i$ at iteration $t$ then becomes

$$\mathbf{X}_{\mathrm{p}}^i = \mathbf{X}^i + \mathbf{dX}^i, \tag{24}$$

and the Metropolis ratio of Equation (19) is used to determine whether to accept this proposal or not. If $p_{\mathrm{acc}}(\mathbf{X}^i \rightarrow \mathbf{X}_{\mathrm{p}}^i) \geq \mathscr{U}(0,1)$ the candidate point is accepted and the $i$th chain moves to the new position, that is $\mathbf{x}_t^i = \mathbf{X}_{\mathrm{p}}^i$, otherwise $\mathbf{x}_t^i = \mathbf{x}_{t-1}^i$. The default equation for $\gamma$ should, for Gaussian and Student target distribution, result in optimal acceptance rates close to 0.44 for $d = 1$, 0.28 for $d = 5$, and 0.23 for large $d$ (please refer to Section 7.84 of Roberts and Casella (2004) for a cautionary note on these references acceptance rates).

The $d^*$-members of the subset A are sampled from the entries $\{1,\ldots,d\}$ (without replacement) and define the dimensions of the parameter space to be sampled by the proposal. This subspace spanned by A is construed in DREAM with the help of a crossover operator. This genetic operator is applied before each proposal is created and works as follows. First, a crossover value, cr is sampled from a geometric sequence of $n_{\mathrm{CR}}$ different crossover probabilities, $\mathrm{CR} = \left\{ \frac{1}{n_{\mathrm{CR}}}, \frac{2}{n_{\mathrm{CR}}}, \ldots, 1 \right\}$ using the discrete multinomial distribution, $\mathscr{M}(\mathrm{CR}, \mathbf{p}_{\mathrm{CR}})$ on CR with selection probabilities $\mathbf{p}_{\mathrm{CR}}$. Then, a $d$-vector $\mathbf{z} = \{z_1,\ldots,z_d\}$ is drawn from a standard multivariate normal distribution, $\mathbf{z} \overset{D}{\sim} \mathscr{U}_d(0,1)$. All those values $j$ which satisfy $z_j \leq \mathrm{cr}$ are stored in the subset A and span the subspace of the proposal that will be sampled using Equation (23). If A is empty, one dimension of $\{1,\ldots,d\}$ will be sampled at random to avoid the jump vector to have zero length.

The use of a vector of crossover probabilities enables single-site Metropolis (A has one element), Metropolis-within-Gibbs (A has one or more elements) and regular Metropolis sampling (A has $d$ elements), and constantly introduces new directions in the parameter space that chains can take outside the subspace spanned by their current positions. What is more, the use of subspace sampling allows using $N < d$ in DREAM, an important advantage over DE-MC that requires $N = 2d$ chains to be run in parallel (ter Braak, 2006). Subspace sampling as implemented in DREAM adds one extra algorithmic variable, $n_{\mathrm{CR}}$ to the algorithm. The default setting of $n_{\mathrm{CR}} = 3$ has shown to work well in practice, but larger values of this algorithmic variable might seem appropriate for high-dimensional target distributions, say $d > 50$, to preserve the frequency of low-dimensional jumps. Note, more intelligent subspace selection methods can be devised for target distributions involving many highly correlated parameters. These parameters should be sampled jointly in a group, otherwise too many of the (subspace) proposals will be rejected and the search can stagnate. This topic will be explored in future work.

To enhance search efficiency the selection probability of each crossover value, stored in the $n_{\mathrm{CR}}$-vector $\mathbf{p}_{\mathrm{CR}}$, is tuned adaptively during burn-in by maximizing the distance traveled by each of the $N$ chains. This adaptation is described in detail in Vrugt et al. (2008a, 2009a), and a numerical implementation of this approach appears in the MATLAB code of DREAM on the next page.

The core of the DREAM algorithm can be written in about 30 lines of code (see Algorithm 5). The input arguments are similar to those used by DE-MC and include the function handles prior and pdf and the values of $N$, $T$, and $d$.

The MATLAB code listed above implements the different steps of the DREAM algorithm as detailed in the main text. Structure, format and notation matches that of the DE-MC code, and variable names correspond with their symbols used in Equations (23) and (24). Indents and comments are used to enhance readability and to convey the main intent of each line of code. Note that this code does not monitor convergence of the sampled chain trajectories, an important hiatus addressed in the MATLAB toolbox of DREAM discussed in the next sections. The computational efficiency of this code can be improved considerably, for instance through vectorization of the inner for loop, but this will affect negatively readability.

The source code of DREAM listed below differs in several important ways from the basic code of the DE-MC algorithm presented in Section 3.2. These added features increase the length of the code with about 20 lines, but enhance significantly the convergence speed of the sampled chains to a limiting distribution. For reasons of simplicity, a separate function is used for

**ALGORITHM 5.** MATLAB code of the differential evolution adaptive Metropolis (DREAM) algorithm. The script is similar to that of DE-MC but uses (a) more than one chain pair to create proposals, (b) subspace sampling, and (c) outlier chain detection, to enhance convergence to the posterior target distribution. Built-in functions are highlighted with a low dash. The jump vector, dX(i,1:d) of the ith chain contains the desired information about the scale and orientation of the proposal distribution and is derived from the remaining N-1 chains. deal() assigns default values to the algorithmic variables of DREAM, std() returns the standard deviation of each column of X, and sum() computes the sum of the columns A of the chain pairs a and b. The function check() is a remedy for outlier chains.

```
function [x,p_x] = dream(prior,pdf,N,T,d)
% DiffeRential Evolution Adaptive Metropolis (DREAM) algorithm

[delta,c,c_star,n_CR,p_g] = deal(3,0.1,1e-12,3,0.2);      % Default of algorithmic parameters
x = nan(T,d,N); p_x = nan(T,N);                           % Preallocate chains and density
[J,n_id] = deal(zeros(1,n_CR));                           % Variables selection prob. crossover
for i = 1:N, R(i,1:N-1) = setdiff(1:N,i); end             % R-matrix: index of chains for DE
CR = [1:n_CR]/n_CR; pCR = ones(1,n_CR)/n_CR;              % Crossover values and select. prob.

X = prior(N,d);                                           % Create initial population
for i = 1:N, p_X(i,1) = pdf(X(i,1:d)); end                % Compute density initial population
x(1,1:d,1:N) = reshape(X',1,d,N); p_x(1,1:N) = p_X';      % Store initial states and density

for t = 2:T,    % Dynamic part: Evolution of N chains
    [~,draw] = sort(rand(N-1,N));                         % Permute [1,...,N-1] N times
    dX = zeros(N,d);                                      % Set N jump vectors to zero
    lambda = unifrnd(-c,c,N,1);                           % Draw N lambda values
    std_X = std(X);                                       % Compute std each dimension
    for i = 1:N,                                          % Create proposals and accept/reject
        D = randsample([1:delta],1,'true');               % Select delta (equal select. prob.)
        a = R(i,draw(1:D,i)); b = R(i,draw(D+1:2*D,i));   % Extract vectors a and b not equal i
        id = randsample(1:nCR,1,'true',pCR);              % Select index of crossover value
        z = rand(1,d);                                    % Draw d values from U[0,1]
        A = find(z < CR(id));                             % Derive subset A selected dimensions
        d_star = numel(A);                                % How many dimensions sampled?
        if d_star == 0, [~,A] = min(z); d_star = 1; end   % A must contain at least one value
        gamma_d = 2.38/sqrt(2*D*d_star);                  % Calculate jump rate
        g = randsample([gamma_d 1],1,'true',[1-p_g p_g]); % Select gamma: 80/20 mix [default 1]
        dX(i,A) = c_star*randn(1,d_star) + ...
            (1+lambda(i))*g*sum(X(a,A)-X(b,A),1);         % Compute ith jump diff. evol.
        Xp(i,1:d) = X(i,1:d) + dX(i,1:d);                 % Compute ith proposal
        p_Xp(i,1) = pdf(Xp(i,1:d));                       % Calculate density ith proposal
        p_acc = min(1,p_Xp(i,1)./p_X(i,1));               % Compute acceptance probability
        if p_acc > rand,                                  % p_acc larger than U[0,1]?
            X(i,1:d) = Xp(i,1:d); p_X(i,1) = p_Xp(i,1);   % True: Accept proposal
        else
            dX(i,1:d) = 0;                                % Set jump back to zero for pCR
        end
        J(id) = J(id) + sum((dX(i,1:d)./std_X).^2);       % Update jump distance crossover idx
        n_id(id) = n_id(id) + 1;                          % How many times idx crossover used
    end
    x(t,1:d,1:N) = reshape(X',1,d,N); p_x(t,1:N) = p_X';  % Append current X and density
    if t<T/10, pCR = J./n_id; pCR = pCR/sum(pCR); end     % Update selection prob. crossover
    [X,p_X] = check(X,mean(log(p_x(ceil(t/2):t,1:N))));   % Outlier detection and correction
end             % End dynamic part
```

one of these features, the correction of outlier chains. This function is called check (line 44) and patches a critical vulnerability of multi-chain MCMC methods such as SCEM-UA, DE-MC, and DREAM (Vrugt et al., 2003; ter Braak and Vrugt, 2008; Vrugt et al., 2008a, 2009a). The performance of these methods is impaired if one or more of their sampled chains have become trapped in an unproductive area of the parameter space while in pursuit of the target distribution. The state of these outlier chains will not only contaminate the jumping distribution of Equation (23) and thus slow down the evolution and mixing of the other "good" chains, what is much worse, dissident chains make it impossible to reach convergence to a limiting distribution. For as long as one of the chains samples a disjoint part of the parameter space, the $\widehat{R}$-diagnostic of Gelman and Rubin (1992) cannot reach its stipulated threshold of 1.2 required to officially declare convergence.

The problem of outlier chains is well understood and easily demonstrated with an example involving a posterior response surface with one or more local area of attractions far removed from the target distribution. Chains that populate such local optima can continue to persist indefinitely if the size of their jumps is insufficient to move the chain outside the space spanned by this optima (see Fig. 2 of ter Braak and Vrugt (2008)). Dissident chains will occur most frequent in high-dimensional target distributions, as they require the use of a large $N$, and complex posterior response surfaces with many areas of attraction.

The function check is used as remedy for dissident chains. The mean log density of the samples stored in the second half of each chain is used as proxy for the "fitness" of each trajectory, and these $N$ data points are examined for anomalies using an outlier detection test. Those chains (data points) that have been labeled as outlier will relinquish their dissident state and move to the position of one of the other chains (chosen at random). Details of this procedure can be found in Vrugt et al. (2009a). The MATLAB toolbox of DREAM implements four different outlier detection methods the user can choose from. Details will be presented in the next section.

Those proficient in statistics, computer coding and numerical computation, will be able to personalize this code for their own applications. Yet, for others this code might not suffice as it has very few built-in options and capabilities. To satisfy these potential users, I have therefore developed a MATLAB toolbox for DREAM. This package has many built-in functionalities and is easy to use in practice. The next sections will introduce the various elements of the DREAM package, and use several examples to illustrate how the package can be used to solve a wide variety of Bayesian inference problems involving (among others) simple functions, dynamic simulation models, formal and informal likelihood functions, informative and noninformative prior distributions, limits of acceptability, summary statistics, diagnostic model evaluation, low and high-dimensional parameter spaces, and distributed computing.

Before I proceed to the next section, a few remarks are in order. The code of DREAM listed on the previous page does not adapt the selection probabilities of the individual crossover values nor does it monitor the convergence of the sampled chain trajectories. These functionalities appear in the toolbox of DREAM. In fact, several different metrics are computed to help diagnose convergence of the sampled chains to a limiting distribution.

The MATLAB code of DREAM listed in Algorithm 5 evolves each of the $N$ chains sequentially. This serial implementation satisfies DREAM's reversibility proof (ter Braak and Vrugt, 2008; Vrugt et al., 2009a), but will not be efficient for CPU-intensive models. We can adapt DREAM to a multi-core implementation in which the $N$ proposals are evaluated simultaneously in parallel using the distributed computing toolbox of MATLAB (Algorithm 6).

Numerical experiments with a large and diverse set of test functions have shown that the parallel implementation of DREAM converges to the correct target distribution. I will revisit this topic in Section 7.1 of this paper.

## 4. MATLAB implementation of DREAM

The basic code of DREAM listed in Algorithm 5 was written in 2006 but many new functionalities and options have been added to the source code in recent years due to continued research developments and to support the needs of a growing group of users. The DREAM code can be executed from the MATLAB prompt by the command

**ALGORITHM 6.** Distributed implementation of DREAM in MATLAB. This code differs from the standard code of DREAM in that the proposals are evaluated in parallel on different cores using the built-in parfor function of the parallel computing toolbox.

```
function [x,p_x] = par_dream(prior,pdf,N,T,d)
% DiffeRential Evolution Adaptive Metropolis (DREAM) algorithm

[delta,c,c_star,n_CR,p_g] = deal(3,0.1,1e-12,3,0.2);      % Default of algorithmic parameters
x = nan(T,d,N); p_x = nan(T,N);                           % Preallocate chains and density
CR = [1:n_CR]/n_CR; p_CR = ones(1,n_CR)/n_CR;             % Crossover values and select. prob.
[J,n_id] = deal(zeros(1,n_CR));                           % Variables selection prob. crossover
for i = 1:N, R(i,1:N-1) = setdiff(1:N,i); end             % R-matrix: index of chains for DE

X = prior(N,d);                                           % Create initial population
for i = 1:N, p_X(i,1) = pdf(X(i,1:d)); end                % Compute density initial population
x(1,1:d,1:N) = reshape(X',1,d,N); p_x(1,1:N) = p_X';      % Store initial states and density

for t = 2:T,    % Dynamic part: Evolution of N chains
    [~,draw] = sort(rand(N-1,N));                         % Permute [1,...,N-1] N times
    dX = zeros(N,d);                                      % Set N jump vectors to zero
    lambda = unifrnd(-c,c,N,1);                           % Draw N lambda values
    std_X = std(X);                                       % Compute std each sampling dimension
    for i = 1:N,                                          % Create proposals
        D = randsample([1:delta],1,'true');               % Select delta (equal select. prob.)
        a = R(i,draw(1:D,i)); b = R(i,draw(D+1:2*D,i));   % Extract vectors a and b not equal i
        id(i) = randsample(1:n_CR,1,'true',p_CR);         % Select index of crossover value
        z = rand(1,d);                                    % Draw d values from U[0,1]
        A = find(z < CR(id(i)));                          % Derive subset A selected dimensions
        d_star = numel(A);                                % How many dimensions sampled?
        if (d_star == 0), [~,A] = min(z); d_star = 1; end % A must contain at least one value
        gamma_d = 2.38/sqrt(2*D*d_star);                  % Calculate jump rate
        g = randsample([gamma_d 1],1,'true',[1-p_g p_g]); % Select gamma: 80/20 mix [default 1]
        dX(i,A) = c_star*randn(1,d_star) + ...
            (1+lambda(i))*g*sum(X(a,A)-X(b,A),1);         % Compute ith jump diff. evol.
        Xp(i,1:d) = X(i,1:d) + dX(i,1:d);                 % Compute ith proposal
    end
    parfor i = 1:N,                                       % Accept/reject proposals (parallel)
        p_Xp(i,1) = pdf(Xp(i,1:d));                       % Calculate density ith proposal
        p_acc = min(1,p_Xp(i,1)./p_X(i,1));               % Compute acceptance probability
        if p_acc > rand,                                  % p_acc larger than U[0,1]?
            X(i,1:d) = Xp(i,1:d); p_X(i,1) = p_Xp(i,1);   % True: Accept proposal
        else
            dX(i,1:d) = 0;                                % Set jump back to zero for p_CR
        end
        J(id(i)) = J(id(i)) + sum((dX(i,1:d)./std_X).^2); % Update jump distance crossover idx
        n_id(id(i)) = n_id(id(i)) + 1;                    % How many times idx crossover used
    end
    x(t,1:d,1:N) = reshape(X',1,d,N); p_x(t,1:N) = p_X';  % Append current X and density
    if t<T/10, p_CR = J./n_id; p_CR = p_CR/sum(p_CR); end % Update selection prob. crossover
    [X,p_X] = check(X,mean(log(p_x(ceil(t/2):t,1:N))));   % Check for outlier and correct
end             % End dynamic part
```

$$[\text{chain, output, fx}] = \text{DREAM}(\text{Func\_name, DREAMPar, Par\_info})$$

where Func_name (string), DREAMPar (structure array), and the variable Par_info (structure array), etc. are input arguments defined by the user, and chain (matrix), output (structure array) and fx (matrix) are output variables computed by DREAM and returned to the user. To minimize the number of input and output arguments in the DREAM function call and related primary and secondary functions called by this program, I use MATLAB structure arrays and group related variables in one main element using data containers called fields. Two optional input arguments that the user can pass to DREAM are Meas_info and options and their content and usage will be discussed later.

The DREAM function uses more than twenty other functions to implement its various steps and functionalities and generate samples from the posterior distribution. All these functions are summarized briefly in Appendix A. In the subsequent sections I will discuss the MATLAB implementation of DREAM. This, along with prototype case studies presented herein and template examples listed in RUNDREAM should help users apply Bayesian inference to their data and models.

### 4.1. Input argument 1: Func_Name

The variable Func_Name defines the name (enclosed in quotes) of the MATLAB function (.m file) used to calculate the likelihood (or proxy thereof) of each proposal. The use of a m-file rather than anonymous function (e.g. pdf example), permits DREAM to solve inference problems involving, for example, dynamic simulation models, as they can generally not be written in a single line of code. If Func_name is conveniently assumed to be equivalent to 'model' then the call to this function becomes

$$Y = \text{model}(x) \tag{25}$$

where $\mathbf{x}$ (input argument) is a $1 \times d$ vector of parameter values, and Y is a return argument whose content is either a likelihood, log-likelihood, or vector of simulated values or summary statistics, respectively. The content of the function model needs to be written by the user - but the syntax and function call is universal. Appendix C provides seven different templates of the function model which are used in the case study presented in Section 5.

### 4.2. Input argument 2: DREAMPar

The structure DREAMPar defines the computational settings of DREAM. Table 1 lists the different fields of DREAMPar, their default values, and the corresponding variable names used in the mathematical description of DREAM in Section 3.3.

The field names of DREAMPar match exactly the symbols (letters) used in the (mathematical) description of DREAM in Equations (23) and (24). The values of the fields d, N, T depend on the dimensionality of the target distribution. These variables are problem dependent and should hence be specified by the user. Default settings are assumed in Table 1 for the remaining fields of DREAMPar with the exception of GLUE and lik whose values will be discussed in the next two paragraphs. To create proposals with Equation (23), the value of N should at least be equivalent to $2\delta+1$ or $N = 7$ for the default of $\delta = 3$. This number of chains is somewhat excessive for low dimensional problems involving just a few parameters. One could therefore conveniently set $\delta = 1$ for small $d$. The default settings of DREAMPar are easy to modify by the user by declaring individual fields and their respective value.

The DREAM algorithm can be used to sample efficiently the behavioral solution space of informal and likelihood functions used within GLUE (Beven and Binley, 1992; Beven and Freer, 2001). In fact, as will be shown later, DREAM can also solve efficiently the limits of acceptability framework of Beven (2006). For now it suffices to say that the field GLUE of structure DREAMPar stores the value of the shaping factor used within the (pseudo)likelihood functions of GLUE. I will revisit GLUE and informal Bayesian inference at various places in the remainder of this paper. The content of the field lik of DREAMPar defines the choice of likelihood function used to compare the output of the function model with the available calibration data. Table 2 lists the different options for lik the user can select from. The choice of likelihood function depends in large part on the content of the return argument Y of the function model, which is either a (log)-likelihood, a vector with simulated values, or a vector with summary statistics, respectively.

If the return argument, Y of function model is equivalent to a likelihood or log-likelihood then field lik of DREAMPar should be

**Table 1**
Main algorithmic variables of DREAM: mathematical symbols, corresponding fields of DREAMPar and default settings. These default settings have been determined in previous work and shown to work well for a range of target distributions.

| Symbol | Description | Field DREAMPar | Default |
|---|---|---|---|
| **Problem dependent** | | | |
| $d$ | Number of parameters | d | $\geq 1$ |
| $N$ | Number of Markov chains | N | $\leq 2\delta+1$ |
| $T$ | Number of generations | T | $\geq 1$ |
| $\mathscr{L}(\mathbf{x}\|\tilde{\mathbf{Y}})$ | (Log)-Likelihood function | lik | [1,2], [11−17], [21−23], [31−34] |
| **Default variables**[a] | | | |
| $n_{cr}$ | Number of crossover values | $n_{CR}$ | 3 |
| $\delta$ | Number chain pairs proposal | delta | 3 |
| $\boldsymbol{\lambda}$[b] | Randomization | lambda | 0.1 |
| $\boldsymbol{\zeta}$[c] | Ergodicity | zeta | $10^{-12}$ |
| $p_{(\gamma=1)}$ | Probability unit jump rate | p_unit_gamma | 0.2 |
| | Outlier detection test | outlier | 'iqr' |
| $K$ | Thinning rate | thinning | 1 |
| | Adapt crossover probabilities? | adapt_pCR | 'yes' |
| $G$[d] | Shaping factor | GLUE | >0 |
| $\beta_0$[e] | Scaling factor jump rate | beta0 | 1 |

[a] A change to the default values of DREAM will affect the convergence (acceptance) rate.
[b] $\boldsymbol{\lambda} \overset{D}{\sim} \mathscr{U}_{d^*}(-\text{DREAMPar.lambda}, \text{DREAMPar.lambda})$.
[c] $\boldsymbol{\zeta} \overset{D}{\sim} \mathscr{N}_{d^*}(0, \text{DREAMPar.zeta})$.
[d] For pseudo-likelihood functions of GLUE (Beven and Binley, 1992).
[e] Multiplier of the jump rate, $\gamma = \beta_0 \gamma$, default $\beta_0 = 1$.

set equivalent to 1 or 2, respectively. This choice is appropriate for problems involving some prescribed multivariate probability distribution whose density can be evaluated directly. Examples of such functions are presented in the first two case studies of Section 5. Option 1 and 2 also enable users to evaluate their own preferred likelihood function directly in the model script. In principle, these two options are therefore sufficient to apply the DREAM code to a large suite of problems. Nevertheless, to simplify implementation and use, the DREAM package contains about 15 different built-in likelihood functions.

Likelihood functions 11–17 and 31–34 are appropriate if the output of model consists of a vector of simulated values of some variable(s) of interest. Some of these likelihood functions (e.g., 12–14, 16, 17) contain extraneous variables (nuisance coefficients) whose values need to be inferred jointly with the model parameters, **x**. Practical examples of joint inference are provided in the RUNDREAM script and Appendix B. Likelihood functions 21 and 22 are appropriate if the return argument Y of model consists of one or more summary statistics of the simulated data. These two likelihood functions allow use of approximate Bayesian computation and diagnostic model evaluation (Vrugt and Sadegh, 2013; Sadegh and Vrugt, 2014; Vrugt, submitted for publication). Finally, likelihood function 23 enables use of the limits of acceptability framework (Beven, 2006; Beven and Binley, 2014; Vrugt, 2015a). Section 5 presents the application of different likelihood functions and provides templates for their use. Appendix B provides the mathematical formulation of each of the likelihood functions listed in Table 2. Note, likelihood 22 and 23 use a modified Metropolis selection rule to accept proposals or not. This issue is revisited in Section 7 of this paper.

The generalized likelihood (GL) function of Schoups and Vrugt (2010) (14) is most advanced in that it can account explicitly for bias, correlation, non-stationarity, and nonnormality of the error residuals trough the use of nuisance coefficients. In a recent paper, Scharnagl et al. (2015) has introduced a skewed student likelihood function (17) as modification to the GL formulation (14) to describe adequately heavy-tailed error residual distributions. Whittle's likelihood (Whittle, 1953) (15) is a frequency-based approximation of the Gaussian likelihood and can be interpreted as minimum distance estimate of the distance between the parametric spectral density and the (nonparametric) periodogram. It also minimizes the asymptotic Kullback–Leibler divergence and, for autoregressive processes, provides asymptotically consistent estimates for Gaussian and non-Gaussian data, even in the presence of long-range dependence (Montanari and Toth, 2007). Likelihood function 16, also referred to as Laplace or double exponential distribution, differs from all other likelihood functions in that it assumes a $\ell^1$-norm of the error residuals. This approach weights all error residuals equally and the posterior inference should therefore not be as sensitive to outliers.

Likelihood functions 11–17 and 31–34 represent a different school of thought. Formulations 11–17 are derived from first-order statistical principles about the expected probabilistic properties of the error residuals, $\mathbf{E}(\mathbf{x}) = \tilde{\mathbf{Y}} - \mathbf{Y}(\mathbf{x})$. These functions are also referred to as formal likelihood functions. For example if the error residuals are assumed to be independent (uncorrelated) and normally distributed then the likelihood function is simply equivalent to formulation 11 or 12, depending on whether the measurement data error is integrated out (11) or explicitly considered (12).

The second class of likelihood functions, 31–34, avoids overconditioning of the likelihood surface in the presence of epistemic and other sources, and their mathematical formulation is guided by trial-and-error, expert knowledge, and commonly used goodness-of-fit criteria (Beven and Binley, 1992; Freer et al., 1996; Beven and Freer, 2001). These informal likelihood functions enable users to implement the GLUE methodology of Beven and

**Table 2**
Built-in likelihood functions of the DREAM package. The value of field lik of DREAMPar depends on the content of the return argument Y from the function model; [1] likelihood, [2] log-likelihood, [11–17] vector of simulated values, [21–23] vector of summary statistics, and [31–34] vector of simulated values. The mathematical formulation of each likelihood function is given in Appendix B.

| lik | Description | References |
|---|---|---|
| **User-free likelihood functions** | | |
| 1 | Likelihood, $L(\mathbf{x}\|\tilde{\mathbf{Y}})$ | e.g. Equation (7) |
| 2 | Log-likelihood, $\mathscr{L}(\mathbf{x}\|\tilde{\mathbf{Y}})$ | e.g. Equations (8), (10) and (11) |
| **Formal likelihood functions** | | |
| 11 | Gaussian likelihood: measurement error integrated out | Thiemann et al. (2001) see also footnote 1 |
| 12[a] | Gaussian likelihood: homos/heteroscedastic data error | Equation (7) |
| 13[a,b] | Gaussian likelihood: with AR-1 model of error residuals | Equations (10) and (11) |
| 14[c] | Generalized likelihood function | Schoups and Vrugt (2010a) |
| 15 | Whittle's likelihood (spectral analysis) | Whittle (1953) |
| 16[a] | Laplacian likelihood: homos/heteroscedastic data error | Laplace (1774) |
| 17[c] | Skewed Student likelihood function | Scharnagl et al. (2015) |
| **ABC – diagnostic model evaluation** | | |
| 21[d] | Noisy ABC: Gaussian likelihood | Turner and Sederberg (2012) |
| 22[d,e] | ABC: Boxcar likelihood | Sadegh and Vrugt (2014) |
| **GLUE – limits of acceptability** | | |
| 23[e] | Limits of acceptability | Vrugt (2015a) |
| **GLUE – informal likelihood functions** | | |
| 31[f] | Inverse error variance with shaping factor | Beven and Binley (1992) |
| 32[f] | Nash and Sutcliffe efficiency with shaping factor | Freer et al. (1996) |
| 33[f] | Exponential transform error variance with shaping factor | Freer et al. (1996) |
| 34[f] | Sum of absolute error residuals | Beven and Binley (1992) |

[a] Measurement data error in field Sigma of Par_info or inferred jointly with parameters.
[b] First-order autoregressive coefficient is nuisance variable.
[c] Nuisance variables for model bias, correlation, non-stationarity and nonnormality residuals.
[d] Default of $\varepsilon = 0.025$ in field epsilon of options to delineate behavioral space.
[e] Uses a modified Metropolis selection rule to accept proposals or not.
[f] Shaping factor, $G$ defined in field GLUE of DREAMPar (default $G = 10$).

Binley (1992). The use of DREAM enhances, sometimes dramatically, the computational efficiency of GLUE (Blasone et al., 2008).

The field thinning of DREAMPar allows the user to specify the thinning rate of each Markov chain to reduce memory requirements for high-dimensional target distributions. For instance, for a $d = 100$ dimensional target distribution with $N = 100$ and $T = 10{,}000$, MATLAB would need a staggering 100-million bytes of memory to store all the samples of the joint chains. Thinning applies to all the sampled chains, and stores only every $K$th visited state. This option reduces memory storage with a factor of $T/K$, and also decreases the autocorrelation between successively stored chain samples. A default value of $K = 1$ (no thinning) is assumed in DREAM. Note, large values for $K$ ($K \gg 10$) can be rather wasteful as many visited states are not used in the computation of the posterior moments and/or plotting of the posterior parameter distributions.

Multi-chain methods can suffer convergence problems if one or more of the sampled chains have become stuck in a local area of attraction while in pursuit of the target distribution. This fallacy has been addressed in the basic source code of DREAM listed in Algorithm 5 and the function check was used to detect and resolve aberrant trajectories. Dissident chains are more likely to appear if the target distribution is high-dimensional and the posterior response surface is non-smooth with many local optima and regions of attraction. These non-ideal properties are often the consequence of poor model numerics (Clark and Kavetski, 2010; Schoups et al., 2010) and hinder convergence of MCMC simulation methods to the target distribution. The field outlier of DREAMPar lists (in quotes) the name of the outlier detection test that is used to expose dissident chains. Options available to the user include the 'iqr' (Upton and Cook, 1996), 'grubbs' (Grubbs, 1950), 'peirce' (Peirce, 1852), and 'chauvenet' (Chauvenet, 1960) method. These nonparametric methods diagnose dissident chains by comparing the mean log-density values of each of the $N$ sampled trajectories. The premise of this comparison is that the states visited by an outlier chain should have a much lower average density than their counterparts sampling the target distribution. Those chains diagnosed as outlier will give up their present position in the parameter space in lieu of the state of one of the other $N - 1$ chains, chosen at random. This correction step violates detailed balance (irreversible transition) but is necessary in some cases to reach formally convergence to a limiting distribution. Numerical experiments have shown that the default option DREAMPar.outlier = 'iqr' works well in practice. Note, the problem of outlier chains would be resolved if proposals are created from past states of the chains as used in DREAM$_{(ZS)}$, DREAM$_{(DZS)}$ and MT-DREAM$_{(ZS)}$. Dissident chains can then sample their own position and jump directly to the mode of the target if $\gamma = 1$ (ter Braak and Vrugt, 2008; Laloy and Vrugt, 2012). We will revisit this issue in Section 7 of this paper.

The field adapt_pCR of DREAMPar defines whether the crossover probabilities, $\mathbf{p}_{CR}$ are adaptively tuned during a DREAM run so as to maximize the normalized Euclidean distance between two successive chain states. The default setting of 'yes', can be set to 'no' and thus switched off by the user. The selection probabilities are tuned only during burn-in of the chains to not destroy reversibility of the sampled chains.

The default choice of the jump rate in DREAM is derived from the value of $s_d = 2.38^2/d$ in the RWM algorithm. This setting should lead to optimal acceptance rates for Gaussian and Student target distributions, but might not yield adequate acceptance rates for real-word studies involving complex multivariate posterior parameter distributions. The field beta0 of structure DREAMPar allows the user to increase (decrease) the value of the jump rate, $\gamma = 2.38\beta_0/2\delta d^*$, thereby improving the mixing of the individual chains. This $\beta_0$-correction is applied to all sampled proposals, with the exception of the unit jump rate used for mode jumping. Values of $\beta_0 \in [1/4,1/2]$ have shown to enhance significantly the convergence rate of DREAM for sampling problems involving parameter-rich groundwater and geophysical models (e.g. Laloy et al. (2015)).

### 4.3. Input argument 3: Par_info

The structure Par_info stores all necessary information about the parameters of the target distribution, for instance their prior uncertainty ranges (for bounded search problems), starting values (initial state of each Markov chain), prior distribution (defines Metropolis acceptance probability) and boundary handling (what to do if out of feasible space), respectively. Table 3 lists the different fields of Par_info and summarizes their content, default values and variable types.

The field initial of Par_info specifies with a string enclosed between quotes how to sample the initial state of each of the $N$ chains. Options available to the user include (1) 'uniform' (2) 'latin' (3) 'normal' and (4) 'prior', and they create the initial states of the chains by sampling from (1) a uniform prior distribution, (2) a Latin hypercube (McKay et al., 1979), (3) a multivariate normal distribution, and (4) a user defined prior distribution. The first three options assume the prior distribution to be noninformative (uniform/flat), and consequently the posterior density of each proposal to be directly proportional to its likelihood. On the contrary, if the option 'prior' is used and a non-flat (informative) prior distribution of the parameters is specified by the user, then the density of each proposal becomes equivalent to the product of the (multiplicative) prior density and likelihood derived from the output of model.

Option (1) and (2) require specification of the fields min and max of Par_info. These fields contain in a $1 \times d$-vector the lower and upper bound values of each of the parameters, respectively. If option (3) 'normal' is used then the fields mu ($1 \times d$-vector) and cov ($d \times d$-matrix) of Par_info should be defined by the user. These

**Table 3**
DREAM input argument Par_info: Different fields, their default settings and variable types.

| Field Par_info | Description | Options | Default | Type |
|---|---|---|---|---|
| initial | Initial sample | 'uniform'/'latin'/'normal'/'prior' | | String |
| min | Minimum values | | $-\infty_d$ | $1\times d$-vector |
| max | Maximum values | | $\infty_d$ | $1\times d$-vector |
| boundhandling | Boundary handling | 'eflect'/'bound'/'fold'/'none' | 'none' | String |
| mu | Mean 'normal' | | | $1\times d$-vector |
| cov | Covariance 'normal' | | | $d\times d$-matrix |
| prior | Prior distribution | | | Cell array[a]/function handle[b] |

[a] Multiplicative case: each cell of the $d$-array contains a different marginal prior pdf.
[b] Multivariate case: an Anonymous function with prior pdf is provided by user.

fields store the mean and covariance matrix of the multivariate normal distribution. We will revisit the option 'prior' at the end of this section.

The fields min and max of the structure Par_info serve two purposes. First, they define the feasible parameter space from which the initial state of each of the chains is drawn if 'uniform' random or 'latin' hypercube sampling is used. Second, they can define a bounded search domain for problems involving one or more parameters with known physical/conceptual ranges. This does however require the bound to be actively enforced during chain evolution. Indeed, proposals generated with Equations (23) and (24) can fall outside the hypercube defined by min and max even if the initial state of each chain are well within the feasible search space. The field boundhandling of Par_info provides several options what to do if the parameters are outside their respective ranges. The four different options that are available are (1) 'bound', (2) 'eflect', (3) 'fold', and (4) 'none' (default). These methods are illustrated graphically in Fig. 6 and act on one parameter at a time.

The option 'bound' is most simplistic and simply sets each parameter value that is out of bound of equal to its closest bound. The option 'eflect' is somewhat more refined and treats the boundary of the search space as a mirror through which each individual parameter value is reflected backwards into the search space. The reflection step size is simply equivalent to the "amount" of boundary violation. The 'bound' and 'eflect' options are used widely in the optimization literature in algorithms concerned only with finding the minimum (maximum, if appropriate) of a given cost or objective function. Unfortunately, these two proposal correction methods violate detailed balance in the context of MCMC simulation. It is easy to show for both boundary handling methods that the forward (correction step) and backward jump cannot be construed with equal probability. The third option 'fold' treats the parameter space as a continuum representation by simply connecting the upper bound of each dimension to its respective lower bound. This folding approach does not destroy the Markovian properties of the $N$ sampled chains, and is therefore preferred statistically. However, this approach can provide "bad" proposals (reduced acceptance rate) if the posterior distribution is located at the edges of the search domain. Then, the parameters can jump from one side of the search domain to the opposite end.

The option 'bound' is least recommended in practice as it collapses the parameter values to a single point. This not only relinquishes unnecessarily sample diversity but also inflates artificially the solution density (probability mass) at the bound. The loss of chain diversity also causes a-periodicity (proposal and current state are similar for selected dimensions) and distorts convergence to the target distribution. A simple numerical experiment with a truncated normal target distribution will demonstrate the superiority of the folding approach. This results in an exact inference of the target distribution whereas a reflection step overestimates the probability mass at the bound. For most practical applications, a reflection step will provide accurate results unless too many dimensions of the target distribution find their highest density in close vicinity of the bound.

What is left is a discussion of the use of 'prior' as initial sampling distribution of the chains. This option is specifically implemented to enable the use of an informative (non-flat) prior distribution. The user can select among two choices for 'prior', that is the use of a multiplicative prior or multivariate prior distribution. In the multiplicative case each parameter has its own prior distribution, and the field prior of Par_info should be declared a cell array. Each cell then specifies between quotes the density of the corresponding parameter in the vector **x**, for example

$$\text{Par\_info.prior} = \{\text{'normpdf}(-2, 0.1)\text{'}, \text{ 'tpdf}(10)\text{'}, \text{ 'unifpdf}(-2, 4)\text{'}\} \tag{26}$$

uses a normal distribution with mean of $-2$ and standard deviation of $-0.1$ for the first parameter, a Student distribution with $\nu = 10$ degrees of freedom for the second dimension, and a uniform distribution between $-2$ and 4 for the third and last parameter of the target distribution, respectively. The prior density of some parameter vector is then simply equivalent to the product of the individual densities specified in field prior of Par_info. The user can select from the following list of built-in density functions in MATLAB: beta, chi-square, extreme value, exponential, F, gamma, geometric, generalized extreme value, generalized Pareto, hypergeometric, lognormal, noncentral F, noncentral t, noncentral chi-square, normal (Gaussian), Poisson, Rayleigh, T, uniform, and the Weibull density. The function name of each density and corresponding input variables is easily found by typing "help stats" in the MATLAB prompt.

The multiplicative prior assumes the parameters of the prior distribution to be uncorrelated, an assumption that might not be justified for some inference problems. The second option for 'prior' includes the use of a multivariate prior distribution, and declares the field prior of structure Par_info to be an anonymous function, for example

$$\text{Par\_info.prior} = \text{@(x, a, b) mvnpdf(x, a, b)} \tag{27}$$

where mvnpdf(x,a,b) is the $d$-variate normal distribution, $\mathcal{N}_d(a, b)$, evaluated at **x** and with mean $a$ and covariance matrix, $b$, respectively. The input variables, $a$ and $b$ should be specified as separate fields of structure Par_info, for example Par_info.a = zeros(1,d) and

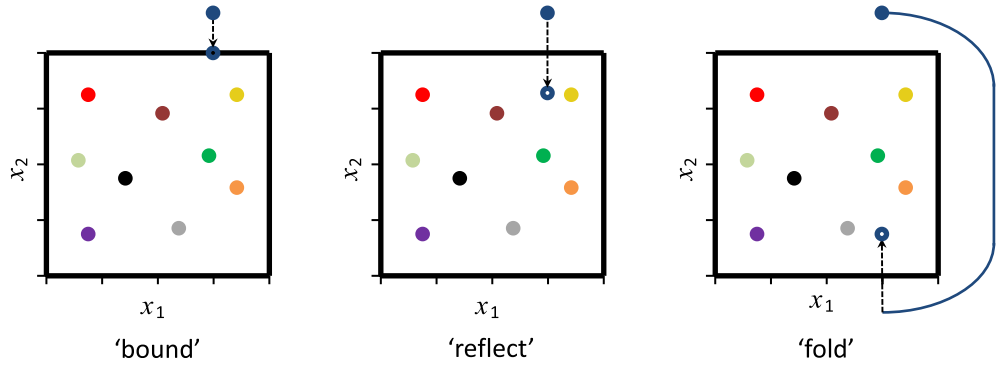


**Fig. 6.** Different options for parameter treatment in bounded search spaces in the DREAM package. a) Set to bound, b) reflection, and c) folding. The option folding is the only boundary handling approach that maintains detailed balance.

**Table 4**
Content of (optional) input structure Meas_info. This fourth input argument of DREAM is required if the return argument of model constitutes a vector of simulated values (or summary statistics) of one or more variables.

| Field Meas_info | Description | Type |
|---|---|---|
| Y | Measurement data | $n\times1$-vector |
| Sigma | Measurement error | scalar or $n\times1$-vector |
| S | Summary statistics (ABC) | $m\times1$-vector |

Par_info.b = eye(d). The use of a multivariate prior allows the user to take into explicit consideration parameter interdependencies. Options available to the user include the multivariate normal and multivariate t-distribution, respectively.

If the standard built-in densities of MATLAB (univariate and multivariate) are insufficient for a given application then the user is free to contribute their own function for the prior distribution. This subroutine should follow exactly the same format as the standard MATLAB densities, and the name of the function should end with "pdf", for example ownpdf.m. What is more, the user has to supply a second function ending with "rnd", (e.g. ownrnd.m) which returns random samples from the user-defined prior pdf. This function should match exactly the format of standard built-in univariate and multivariate random number generators such as lognrnd and mvnrnd, respectively, and will be used by DREAM to sample the initial states of the $N$ different chains. If this second code, ownrnd.m is too difficult to write then the user can always choose to draw the initial states of the chains in DREAM from an noninformative prior, using for instance Latin hypercube sampling. That is, Par_info.initial = 'latin' with min and max of structure Par_info defining the sampling ranges of the parameters. This alternative approach might be favored in practice anyway as it will allow DREAM to explore more thoroughly, at least in the first generations, the parameter space outside the prior pdf. Unless of course, the parameter space defined by min and max is limited to the area of ownpdf.m with high prior density.

### 4.4. (Optional) input argument 4: Meas_info

The fourth input argument Meas_info of the DREAM function is mandatory if the output of model constitutes a vector of simulated values or summary metrics of one or more entities of interest. Table 4 describes the different fields of Meas_info, their content and type.

The field Y of Meas_info stores the $n \geq 1$ observations of the calibration data, $\tilde{\mathbf{Y}}$ against which the output, $\mathbf{Y}$ of model is compared. The $n$-vector of error residuals, $\mathbf{E}(\mathbf{x}) = \tilde{\mathbf{Y}} - \mathbf{Y}(\mathbf{x})$ is then translated into a log-likelihood value using one of the formal (11–17) or informal (31–34) likelihood functions listed in Table 2 and defined by the user in field lik of structure DREAMPar. The field S of Par_info stores $m \geq 1$ summary statistics of the data, and is mandatory input for likelihood functions 21, 22, 23 used for ABC, diagnostic model evaluation, and limits of acceptability. Examples of these approaches are given in the case studies section of this paper. The number of elements of Y and S should match exactly the output of the script model written by the user.

The field Sigma of structure Meas_info stores the measurement error of each entry of the field Y. This data error is necessary input for likelihood functions 12, 13 and 16. A single value for Sigma suffices if homoscedasticity of the data error is expected, otherwise $n$-values need to be declared and specify the heteroscedastic error of the observations of Y.

In case the measurement error of the data Y is unknown, three different approaches can be implemented. The first option is to select likelihood function 11. This function is derived from Equation (7) by integrating over (out) the data measurement error. Field Sigma of Meas_info can then be left blank (empty). The second option uses likelihood function 12, 13, or 16 and estimates the measurement data error along with the model parameters using nuisance variables. The field Sigma of Meas_info should then be used as inline function, for example, Meas_info. Sigma = inline('a + bY'), which defines mathematically the relationship between the observed data, Y and corresponding measurement data error, Sigma. The scalars a and b are nuisance variables and their values append the vector of model parameters, which increases the dimensionality of the target distribution to $d + 2$. If the initial states of the chains are sampled from a uniform distribution (Par_info.initial = 'uniform') then the ranges of a and b augment the $d$-vectors of fields min and max. Note, care should be exercised that Sigma $> 0$ $\forall a,b$. The user is free to define the measurement error function, as long as the nuisance variables used in the inline function are in lower caps, and follow the order of the alphabet. The third and last option uses likelihood function 14 (Schoups and Vrugt, 2010) or 17 (Scharnagl et al., 2015). These functions do not use field Sigma (can be left empty) but rather use their own built-in measurement error model. The coefficients of the error models are part of a larger set of nuisance parameters that allow these likelihood functions to adapt to nontraditional error residual distributions. Appendix B details how to use and adapt likelihood function 11 and 17.

### 4.5. (Optional) input argument 5: options

The structure options is optional and passed as fifth input argument to DREAM. The fields of this structure can activate (among others) file writing, distributed multi-core calculation, storage of the model output simulations, ABC, diagnostic model evaluation, diagnostic Bayes, and the limits of acceptability framework. Table 5 summarizes the different fields of options and their default settings.

**Table 5**
Content of (optional) input structure options. This fifth input argument of the main DREAM code is required to activate several of its built-in capabilities such as distributed multi-processor calculation, workspace saving, ABC, diagnostic model evaluation, diagnostic Bayes and limits of acceptability.

| Field options | Description | Options | Type |
|---|---|---|---|
| parallel | Distributed multi-core calculation? | no/yes | String |
| IO | If parallel, IO writing of model? | no/yes | String |
| modout | Store output of model? | no/yes | String |
| save | Save DREAM workspace? | no/yes | String |
| restart | Restart run? ('save' required) | no/yes | String |
| DB | Diagnostic Bayes? | no/yes | String |
| epsilon | ABC cutoff threshold | | scalar or $m\times1$-vector[a] |
| rho | ABC distance function | | inline function[b] |
| linux | Execute in Linux/unix? | no/yes | String |
| diagnostics | Within chain convergence diagnostics? | no/yes | String |

[a] Default setting of options.epsilon = 0.025.
[b] Default is inline('abs(Meas_info.S − Y)') or $\rho(S(\tilde{\mathbf{Y}}), S(\mathbf{Y}(\mathbf{x}))) = \left|S(\tilde{\mathbf{Y}}) - S(\mathbf{Y}(\mathbf{x}))\right|$.

Multi-core calculation takes advantage of the MATLAB Parallel Computing Toolbox and evaluates the $N$ different proposals created with Equations (23) and (24) on a different processor. Parallel computing is built-in the DREAM code and can be activated automatically if the user sets the field parallel of field options equal to 'yes' (default 'no'). Such distributed calculation can significantly reduce the run time of DREAM for CPU-demanding forward models. For simple models that require only a few seconds to run the time savings of a parallel run is usually negligible due to latency (transport delay) of the hardware and operating system. In fact, for the mixture distribution of Equation (20) multi-core evaluation of the $N$ proposals increases the wall-time of DREAM as compared to sequential calculation.

The field IO (input/output) of options allows the user to communicate to DREAM the desired setup of their distributed computing environment. If file writing is used in model to communicate the parameter values of the DREAM proposal to some external program coded in Fortran or C then the field IO of options should be set equal to 'yes'. Then, DREAM will create automatically, during initialization, $N$ different copies of the model directory (and underlying folders) to satisfy each individual processor. This method avoids the corruption of model input and output files that were to happen if the external program were executed at the same time by different processors working in the same directory. At the end of each DREAM trial, the duplicate directories are removed automatically. This approach to parallelization was used in the HYDRUS-1D case study in Section 5.3. If, on the contrary, the model function involves MATLAB code only then a common directory suffices for all the different workers as all input and output arguments can be passed directly through shared memory. The field IO of options can then be set to 'no'. The same holds if the model function involves use of shared libraries linked through the built-in MEX-compiler of MATLAB (see Case study 4).

For CPU-intensive forward models it would be desirable to not only store the parameter samples but also keep in memory their corresponding model simulations returned by model and used to calculate the likelihood of each proposal. This avoids having to rerun the model script many times after DREAM has terminated to assess model predictive (simulation) uncertainty. The field modout of options allows the user to store the output of the model script. If simulation output storage is desired then modout should be set equal to 'yes', and the $N$ simulations of $\mathbf{X}$ are stored, after each generation, in a binary file "Z.bin". These simulations are then returned to the user as third output argument, fx of DREAM. If chain thinning is activated (please check Table 1) then this applies to the simulations stored in fx as well so that the rows of fx match their samples stored in the chains.

To help evaluate the progress of DREAM, it can be useful to periodically store the MATLAB workspace of the main function "DREAM.m" to a file. This binary MATLAB file, "DREAM.mat", is written to the main directory of DREAM if the field save of structure options is set equal to 'yes'. This binary file can then be loaded into the workspace of another MATLAB worker and used to evaluate the DREAM results during its execution. What is more, the "DREAM.mat" file is required if the user wishes to reboot a prematurely aborted DREAM trial or continue with sampling if convergence (e.g. section 4.7) has not been achieved with the assigned computational budget in field T of DREAMPar. A reboot is initiated by setting the field restart of structure options equal to 'yes'. In case of a prematurely terminated DREAM run, rebooting will finalize the computational budget assigned to this trial. If lack of convergence was the culprit, then a restart run will double the number of samples in each chain, or, add to the existing chains whatever new number of samples specified by the user in field T of DREAMPar.

The MATLAB code of DREAM was developed in Windows. For a unix/linux operating system the user should set the field linux of options to 'yes'. The field diagnostics controls the computation of within-chain convergence diagnostics (see section 4.7). The default setting of this field is 'no'. The single chain diagnostics augment the multi-chain $\widehat{R}$-statistic of Gelman and Rubin (1992) and enable a more robust assessment of convergence.

For ABC or diagnostic model evaluation the fields rho and epsilon of options need to be specified unless their default settings are appropriate. The field rho is an inline function object which specifies the mathematical formulation of the distance function between the simulated and observed summary statistics. In practice, a simple difference operator rho = inline('abs(Meas_info.S − Y)') (default) suffices, where Y (output of model) and field S of Meas_info denote the observed and measured summary statistics, respectively. The field epsilon of options stores a small positive value (default of 0.025) which is used to truncate the behavioral (posterior) parameter space.

If ABC is used then the user can select two different implementations to solve for the target distribution. The first approach, adopted from Turner and Sederberg (2012), uses likelihood function 21 to transform the distance function between the observed and simulated summary metrics in a probability density that DREAM uses to derive the target distribution. This approach can produce nicely bell-shaped marginal distributions, but does not guarantee that the posterior summary metrics fall within epsilon of their observed values. A more viable and powerful approach was introduced recently by Sadegh and Vrugt (2014) and uses likelihood function 22 with the following modified Metropolis acceptance probability to decide whether to accept proposals or not

$$p_{\mathrm{acc}}\left(\mathbf{X}^i \rightarrow \mathbf{X}_{\mathrm{p}}^i\right) = \begin{cases} I\left(f\left(\mathbf{X}_{\mathrm{p}}^i\right) \geq f\left(\mathbf{X}^i\right)\right) & \text{if } f\left(\mathbf{X}_{\mathrm{p}}^i\right) < 0 \\ 1 & \text{if } f\left(\mathbf{X}_{\mathrm{p}}^i\right) \geq 0 \end{cases}, \tag{28}$$

where $I(a)$ is an indicator function that returns one if $a$ is true, and zero otherwise. The mathematical expression of the fitness (likelihood) function 22 is given in Table B1 (in Appendix B). Equation (28) is implemented in an extension of DREAM called DREAM$_{(\mathrm{ABC})}$ and rapidly guides the posterior summary metrics to lie within epsilon of their observed counterparts. Section 5.4 of this paper demonstrates the application of ABC to diagnostic inference using an illustrative case study involving a catchment hydrologic model.

### 4.6. Output arguments

I now briefly discuss the three output (return) arguments of DREAM including chain, output and fx. These three variables summarize the results of the DREAM algorithm and are used for convergence assessment, posterior analysis and plotting.

The variable chain is a matrix of size $T \times d + 2 \times N$. The first $d$ columns of chain store the sampled parameter values (state), whereas the subsequent two columns list the associated log-prior and log-likelihood values respectively. If thinning is applied to each of the Markov chains then the number of rows of chain is equivalent to $T/K + 1$, where $K \geq 2$ denotes the thinning rate. If a non-informative (uniform) prior is used then the values in column $d + 1$ of chain are all zero and consequently, $p(\mathbf{x}|\tilde{\mathbf{Y}}) \propto L(\mathbf{x}|\tilde{\mathbf{Y}})$. With an informative prior, the values in column $d + 1$ are non-zero and the posterior density, $p(\mathbf{x}|\tilde{\mathbf{Y}}) \propto p(\mathbf{x})L(\mathbf{x}|\tilde{\mathbf{Y}})$.

The following MATLAB command

$$\text{plot}(\text{chain}(1:\text{end},1,2),\ \text{'r+'}) \tag{29}$$

creates a traceplot (using red dots) of the first parameter of the second chain. By plotting in the same Figure the remaining $N-1$ chains (using different colors/symbols), the mixing of the Markov chains can be assessed visually.

The structure output contains important (diagnostic) information about the progress of the DREAM algorithm. The field RunTime (scalar) stores the wall-time (seconds), R_stat (matrix), AR (matrix) and CR (matrix) list for a given number of generations the $\widehat{R}$ convergence diagnostic for each individual parameter of the target distribution, the average acceptance rate, and the selection probability of each of the $n_{CR}$ crossover values, respectively, and outlier (vector) contains the index of all outlier chains (often empty). The MATLAB command

$$\text{output.RunTime} \tag{30}$$

displays the wall time of DREAM, and the command

$$\text{plot}(\text{output.AR}(:,1),\ \text{output.AR}(:,2)) \tag{31}$$

plots the acceptance rate of proposals (in %) as function of generation number. This plot reveals important information about the performance of the DREAM algorithm but cannot be used to judge when convergence has been achieved (see next section).

Finally, the matrix fx stores the output Y of model. If this return argument constitutes a vector of simulated values (summary metrics) then fx is of size $NT \times n$ ($NT \times m$), otherwise fx is a vector of $NT \times 1$ with likelihood or log-likelihood values. If thinning is used then this applies to fx as well and the number of rows of fx becomes equivalent to $NT/K+1$; $K \geq 2$.

The directory "../postprocessing" (under main directory) contains a number of different functions that can be used to visualize the different output arguments of DREAM. The script DREAM_postproc can be executed from the MATLAB prompt after the main DREAM function has terminated. Appendix A summarizes briefly the graphical output of the post-processing scripts.

### 4.7. Convergence diagnostics & burn-in

From MCMC theory, the chains are expected to eventually converge to a stationary distribution, which should be the desired target distribution. But, how do we actually assess that convergence has been achieved in practice, without knowledge of the actual target distribution?

One way to check for convergence is to see how well the chains are mixing, or moving around the parameter space. For a properly converged MCMC sampler, the chains should sample, for a sufficiently long period, the approximate same part of the parameter space, and mingle readily and in harmony with one another around some fixed mean value. This can be inspected visually for each dimension of $\mathbf{x}$ separately, and used to diagnose convergence informally.

Another proxy for convergence monitoring is the acceptance rate. A value between 15 and 30% is usually indicative of good performance of a MCMC simulation method. Much lower values usually convey that the posterior surface is difficult to traverse in pursuit of the target distribution. A low acceptance rate can have different reasons, for instance poor model numerics, or the presence of multimodality and local optima. The user can enhance the acceptance rate by declaring a value for $\beta_0 < 1$ in field beta0 of structure DREAMPar (see Table 1). This multiplier will reduce the jumping distance, $\mathbf{dX}$ in Equation (23) and thus proposals will remain in closer vicinity of the current state of each chain. This should enhance the acceptance rate and mixing of individual chains. Note, the acceptance rate can only diagnose whether a MCMC method such as DREAM is achieving an acceptable performance, it cannot be used to determine when convergence has been achieved.

The MATLAB code of DREAM includes various non-parametric and parametric statistical tests to determine when convergence of the sampled chains to a limiting distribution has been achieved. The most powerful of these convergence tests is the multi-chain $\widehat{R}$-statistic of Gelman and Rubin (1992). This diagnostic compares for each parameter $j = \{1,\ldots,d\}$ the within-chain

$$W_j = \frac{2}{N(T-2)} \sum_{r=1}^{N} \sum_{i=\lfloor T/2 \rfloor}^{T} \left( \mathbf{x}_{i,j}^r - \overline{\mathbf{x}}_j^r \right)^2 \qquad \overline{\mathbf{x}}_j^r = \frac{2}{T-2} \sum_{i=\lfloor T/2 \rfloor}^{T} \mathbf{x}_{i,j}^r \tag{32}$$

and between-chain variance

$$B_j/T = \frac{1}{2(N-1)} \sum_{r=1}^{N} \left( \overline{\mathbf{x}}_j^r - \overline{\overline{\mathbf{x}}}_j \right)^2 \qquad \overline{\overline{\mathbf{x}}}_j = \frac{1}{N} \sum_{r=1}^{N} \overline{\mathbf{x}}_j^r \tag{33}$$

using

$$\widehat{R}_j = \sqrt{\frac{N+1}{N} \frac{\widehat{\sigma}_+^{2(j)}}{W_j} - \frac{T-2}{NT}}, \tag{34}$$

where $T$ signifies the number of samples in each chain, $\lfloor \cdot \rfloor$ is the integer rounding operator, and $\widehat{\sigma}_+^{2(j)}$ is an estimate of the variance of the $j$th parameter of the target distribution

$$\widehat{\sigma}_+^{2(j)} = \frac{T-2}{T} W_j + \frac{2}{T} B_j. \tag{35}$$

To official declare convergence, the value $\widehat{R}_j \leq 1.2$ for each parameter, $j \in \{1,\ldots,d\}$, otherwise the value of $T$ should be increased and the chains run longer. As the $N$ different chains are launched from different starting points, the $\widehat{R}$-diagnostic is a relatively robust estimator.

The DREAM code computes automatically during execution the $\widehat{R}$-statistic for each parameter. This statistic is returned to the user in the field R_stat of options. After termination, the following MATLAB command

$$\text{plot}(\text{output.R\_stat}(1{:}\ \text{end},\ 2{:}\ \text{DREAMPar.d}+1)) \tag{36}$$

creates a traceplot of the $\widehat{R}$-convergence diagnostic for each of the $d$ parameters of the target distribution. This plot can be used to determine when convergence has been achieved and thus which samples of chain to use for posterior estimation and analysis. The other samples can simply be discarded from the chains as burn-in. An example of how to use the $\widehat{R}$-statistic for convergence analysis will be provided in case study 2 in section 5.2, a 100-dimensional Student distribution.

The DREAM package also includes several within-chain diagnostics but their calculation is optional and depends on the setting of the field diagnostics of structure options. If activated by the user then DREAM computes, at the end of its run, the autocorrelation function, the Geweke (1992), and Raftery and Lewis (1992)-diagnostics.

The autocorrelation function for each parameter $j = \{1,\ldots,d\}$ is defined as

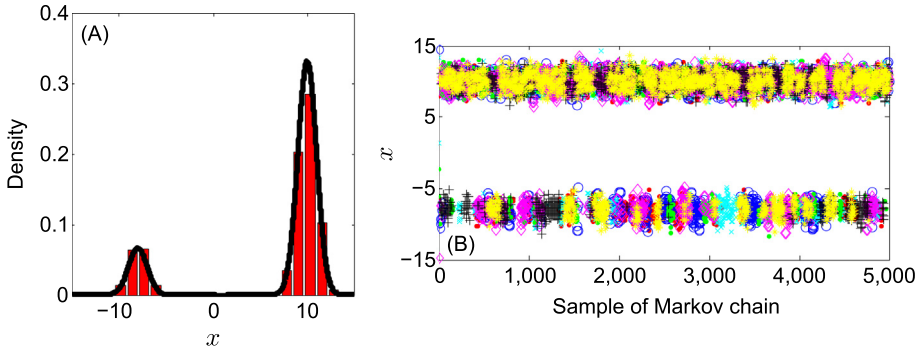


**Fig. 7.** (A) Histogram of posterior distribution derived from DREAM using $N$=10 chains, and $T$=5,000 generations. The solid black line depict the target distribution. (B) Trace plot. Individual chains are coded with a different color (symbol). (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$$\rho_{j,k}^{r} = \frac{\sum_{i=1}^{T-k}\left(\mathbf{x}_{i,j}^{r} - \overline{x}_{j}^{r}\right)\left(\mathbf{x}_{i+k,j}^{r} - \overline{x}_{j}^{r}\right)}{\sum_{i=1}^{T}\left(\mathbf{x}_{i,j}^{r} - \overline{x}_{j}^{r}\right)^{2}}, \tag{37}$$

and returns the correlation between two samples $k$ iterations apart in the $r$th chain, $r = \{1,\ldots,N\}$. Compared to rejection sampling which, per construction, produces uncorrelated samples, MCMC chain trajectories exhibit autocorrelation as the current state of the chain is derived from its previous state. This correlation is expected to decrease with increasing lag $k$. The autocorrelation function is a useful proxy to assess sample variability and mixing, but does not convey when convergence has been achieved. A high autocorrelation, say $|\rho| > 0.8$, at lags, say $k \geq 5$, simply demonstrates a rather poor mixing of the individual chains.

The Geweke (1992)-diagnostic compares the means of two nonoverlapping parts of the Markov chain using a standard Z-score adjusted for autocorrelation. The Raftery and Lewis (1992)-statistic calculates the number of iterations, $T$ and length of burn-in necessary to satisfy the condition that some posterior quantile of interest, say $q$ has a probability, $p$ of lying within interval $[q-r,q+r]$. Default values are $q = 0.025$, $p = 0.95$, and $r = 0.01$, respectively. Details of how to compute and interpret these two statistics is found in the cited references.

The three within-chain diagnostics are calculated for each of the $N$ chains and $d$ parameters separately (if options.diagnostics = 'yes') and results stored in a file called "DREAM_diagnostics.txt". This file is subsequently printed to the screen in the MATLAB editor after DREAM has terminated its run unless the user is running in a unix/linux environment (options.linux = 'yes').

Altogether, joint interpretation of the different diagnostics should help assess convergence of the sampled chain trajectories. Of all these metrics, the $\widehat{R}$-statistic provides the best guidance on exactly when convergence has been achieved. This happens as soon as this statistic drops below the critical threshold of 1.2 for all $d$ parameters of the target distribution. Suppose this happens at $T^*$ iterations (generations) then the first $(T^*-1)$ samples of each chain are simply discarded as burn-in and the remaining $N(T-T^*)$ samples from the joint chains are used for posterior analysis. Note, I always recommend to verify convergence of DREAM by visually inspecting the mixing of the different chain trajectories.

In practice, one has to make sure that a sufficient number of chain samples is available for the inference, otherwise the posterior estimates can be biased. For convenience, I list here the total number of posterior samples, $N(T-T^*)$ (in brackets) one would need for a reliable inference with DREAM for a given dimensionality of the target distribution: $d = 1$ (500); $d = 2$ (1000); $d = 5$ (5000); $d = 10$ (10,000); $d = 25$ (50,000); $d = 50$ (200,000); $d = 100$ (1,000,000); $d = 250$ (5,000,000). These listed numbers are only a rough guideline, and based on several assumptions such as a reasonable acceptance rate (> 10%) and not too complicated shape of the posterior distribution. In general, the number of posterior samples required increases with rejection rate and complexity of the target distribution.

### 4.8. Miscellaneous

The main reason to write this toolbox of DREAM in MATLAB is its relative ease of implementation, use, and graphical display. What is more, the computational complexity of DREAM is rather limited compared to the forward models in script model the code is designed to work with. Indeed, the CPU-time of DREAM is determined in large part by how long it takes to evaluate the density of the target distribution. Relatively little time savings are therefore expected if DREAM were written and executed in a lower level language such as Fortran or C.

The toolbox described herein has been developed for MATLAB 7.10.0.499 (R2010a). The current source code works as well for the most recent MATLAB releases. Those that do not have access to MATLAB, can use GNU Octave instead. This is a high-level interpreted language as well, and intended primarily for numerical computations. The Octave language is quite similar to MATLAB so that most programs are easily portable. GNU Octave is open-source and can be downloaded for free from the following link: http://www.gnu.org/software/octave/.

Finally, likelihood option 1 and 2 allow the user to return the density of their own likelihood function (and prior distribution) immediately to the main DREAM program to satisfy the needs of their own specific inference problems and case studies. The same holds for the use of summary statistics. The built-in likelihood functions 21, 22 and 23 allow the use of any type of summary statistic (or combination thereof) the user deems appropriate for their study.

## 5. Numerical examples

I now demonstrate the application of the MATLAB DREAM package to seven different inference problems. These case studies cover a diverse set of problem features and involve (among others) bimodal and high-dimensional target distributions, summary statistics, dynamic simulation models, formal/informal likelihood functions, diagnostic model evaluation, Bayesian model averaging, limits of acceptability, and informative/noninformative prior parameter distributions.

### 5.1. Case study I: one-dimensional mixture distribution

I revisit the bimodal target distribution of Equation (20). The modes at −8 and 10 are so far separated that it is notoriously

difficult for regular covariance based proposal distributions (AM and RWM) to sample correctly the target distribution. The initial state of the chains is sampled from $\mathscr{U}[-20, 20]$. The following MATLAB script defines the problem setup.

DREAM SETUP CASE STUDY 1 : Univariate mixture distribution.

```
%% Problem settings defined by user
DREAMPar.d = 1;              % Dimension of the problem
DREAMPar.N = 10;             % Number of Markov chains
DREAMPar.T = 5000;           % Number of generations
DREAMPar.lik = 1;            % Model output is likelihood

%% Initial sampling and parameter ranges
Par_info.initial = 'latin'; % Latin hypercube sampling
Par_info.min = [ -20 ];      % Lower bound parameters
Par_info.max = [  20 ];      % Upper bound parameters

%% Define name of function (.m file) for posterior exploration
Func_name = 'mixture';

%% Run the DREAM algorithm
[chain,output,fx] = DREAM(Func_name,DREAMPar,Par_info);
```

The initial sample is drawn using Latin hypercube sampling, and the target distribution is defined in the script mixture of Appendix C. Fig. 7 plots a histogram of the posterior samples, and (right) a traceplot of the sampled value of $x$ in each of the Markov chains. The average acceptance rate is about 36.3%.

The sampled distribution is in excellent agreement with the target distribution, in large part due to the ability of DREAM to jump directly from one mode to the other when $\gamma = 1$. The traceplot shows periodic moves of all chains between both modes of the target distribution and an excelling mixing of the sampled trajectories. The time each chain spends in each of the two modes of the mixture is consistent with their weight in Equation (20).

### 5.2. Case study II: 100-dimensional t-distribution

Our second case study involves a 100-dimensional Student distribution with 60° of freedom. The target distribution, defined in the script t_distribution of Appendix C, is centered at the zeroth vector, with all pairwise correlations equivalent to 0.5. The problem setup is given below.

The initial sample is drawn using Latin hypercube sampling, and thinning is applied to each Markov chain to reduce memory storage. Fig. 8 compares histograms of the sampled marginal distribution of dimensions {25, 50, …, 100} with the actual target distribution (black line). The sampled distributions are in excellent agreement with their observed counterparts. The $\widehat{R}$ diagnostic illustrates that about 500,000 function evaluations are required to reach convergence to a stationary distribution. The acceptance rate of 15.9% is close to optimal.

The marginal distributions derived from DREAM closely approximate their true histograms of the 100-dimensional target. In particular, the tails of the sampled distribution are very well represented with mean correlation of the $d = 100$ dimensions of 0.50 and standard deviation of 0.015.

### 5.3. Case study III: dynamic simulation model

The third case study considers HYDRUS-1D, a variably saturated porous flow model written in Fortran by Šimůnek et al. (1998). This case study is taken from Scharnagl et al. (2011), and involves inference of the soil hydraulic parameters $\theta_r$, $\theta_s$, $\alpha$, $n$, $K_s$ and $\lambda$ (van Genuchten, 1980) and the lower boundary condition (constant head) using time-series of observed soil water contents in the

DREAM SETUP CASE STUDY 2 : 100-dimensional Student distribution.

```
%% Problem settings defined by user
DREAMPar.d = 100;                       % Dimension of the problem
DREAMPar.N = 50;                        % Number of Markov chains
DREAMPar.T = 10000;                     % Number of generations
DREAMPar.lik = 2;                       % Model output is log-likelihood
DREAMPar.thinning = 5;                  % Store each 5th chain sample

%% Initial sampling and parameter ranges
Par_info.initial = 'latin';             % Latin hypercube sampling
Par_info.min = -5 * ones(1,DREAMPar.d); % Lower bound parameters
Par_info.max = 15 * ones(1,DREAMPar.d); % Upper bound parameters

%% Define name of function (.m file) for posterior exploration
Func_name = 't_distribution';

%% Run the DREAM algorithm
[chain,output,fx] = DREAM(Func_name,DREAMPar,Par_info);
```

unsaturated zone. The following MATLAB script defines the problem setup.

DREAM SETUP CASE STUDY 3 : Variably saturated water flow.

```
%% Problem settings defined by user
DREAMPar.d = 7;                     % Dimension of the problem
DREAMPar.N = 10;                    % Number of Markov chains
DREAMPar.T = 2500;                  % Number of generations
DREAMPar.lik = 11;                  % Model output is simulation

%% Initial sampling and parameter ranges
Par_info.initial = 'prior';         % Sample initial state of chains from prior distribution
Par_info.prior = {                  % Marginal prior distribution
    'normpdf(0.0670,0.0060)','normpdf(0.4450,0.0090)','normpdf(-2.310,0.0600)',...
    'normpdf(0.2230,0.0110)','normpdf(-1.160,0.2700)','normpdf(0.3900,1.4700)',...
    'unifpdf(-250,-50)' };
Par_info.boundhandling = 'fold';    % Explicit boundary handling (folding)
% Par. names      thetar thetas log10(alpha) log10(n) log10(Ks) lambda hLB
Par_info.min = [ 0.043  0.409   -2.553       0.179    -2.237   -5.49 -250 ];  % Lower bound
Par_info.max = [ 0.091  0.481   -2.071       0.267    -0.080    6.27 -50  ];  % Upper bound

%% Load calibration data vector (against which model simulation is compared for likelihood)
data = Load_data; Meas_info.Y = data.water;

%% Define name of function (.m file) for posterior exploration
Func_name = 'hydrus';

%% Optional settings
options.parallel = 'yes';           % Run chains in parallel
options.IO = 'yes';                 % MATLAB uses file-writing to communicate with HYDRUS

%% Run the DREAM algorithm
[chain,output,fx] = DREAM(Func_name,DREAMPar,Par_info,Meas_info,options);
```

An explicit prior distribution is used for the soil hydraulic parameters to make sure that their posterior estimates remain in close vicinity of their respective values derived from surrogate soil data using the Rosetta toolbox of hierarchical pedo-transfer functions (Schaap et al., 1998, 2001). The initial state of each chain is sampled from the prior distribution, and boundary handling is applied to enforce the parameters to stay within the hypercube specified by min and max. To speed-up posterior exploration, the $N = 10$ different chains are ran in parallel using the MATLAB parallel computing toolbox.

The hydrus script is given in Appendix C. The Fortran executable of HYDRUS-1D is called from within MATLAB using the dos command. File writing and reading is used to communicate the parameter values of DREAM to HYDRUS-1D and to load the output of this executable back into MATLAB. The output, Y of hydrus constitutes a vector of simulated soil moisture values which are compared against their observed values in Meas_info.Y using likelihood function 11.

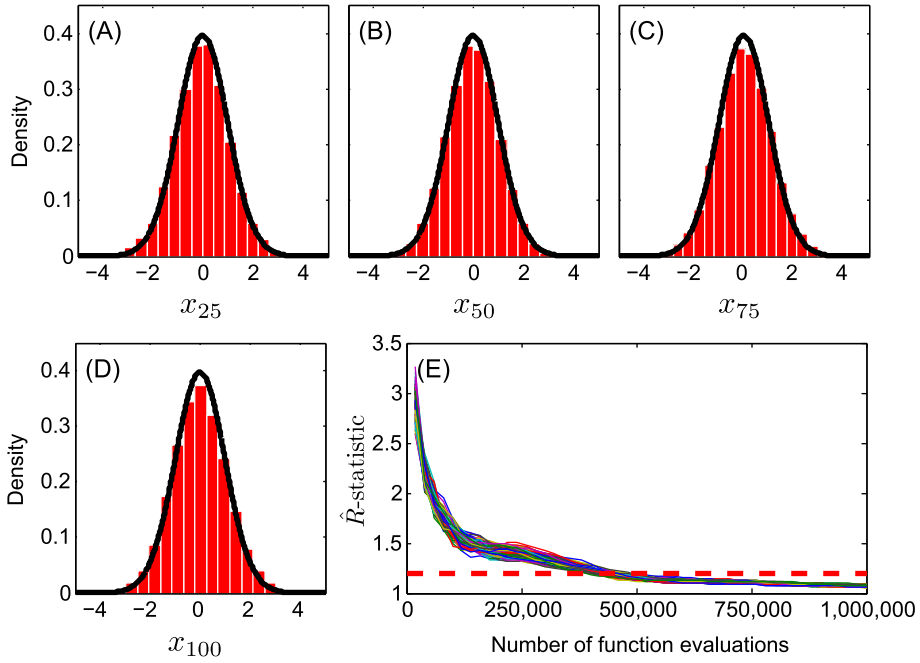


**Fig. 8.** DREAM derived posterior marginal distribution of dimensions (A) 25, (B) 50, (C) 75, and (D) 100 of the $d = 100$ multivariate Student distribution. The solid black line depicts the target distribution. (E) Evolution of the $\hat{R}$ convergence diagnostic of Gelman and Rubin (1992). The horizontal line depicts the threshold of 1.2, necessary to officially declare convergence to a limiting distribution.

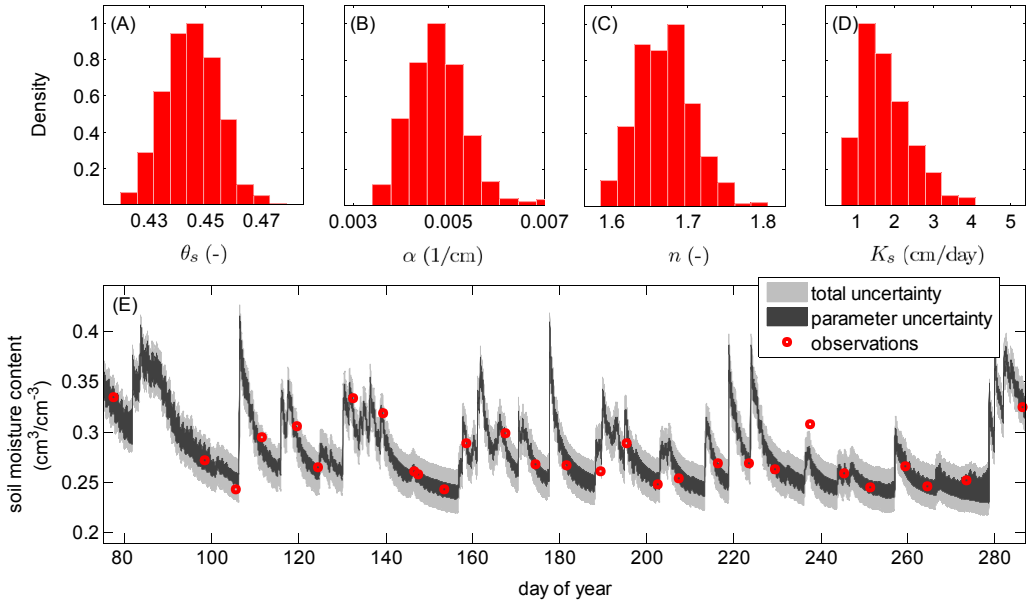


**Fig. 9.** Histograms of the marginal posterior distribution of the soil hydraulic parameters, (A) $\theta_s$, (B) $\alpha$, (C) $n$, and (D) $K_s$, and (E) HYDRUS-1D 95% simulation uncertainty intervals due to parameter (dark region) and total uncertainty (light gray). The observed soil moisture value are indicated with a red circle. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Fig. 9 presents histograms of the marginal posterior distribution of four of the seven parameters considered in this study. The bottom panel presents a time series plot of simulated soil moisture contents. The dark gray region constitutes the 95% HYDRUS-1D simulation uncertainty due to parameter uncertainty, whereas the light gray region denotes the total simulation uncertainty (parameter + randomly sampled additive error). The observed soil moisture values are indicated with a red circle.

The HYDRUS-1D model closely tracks the observed soil moisture contents with Root Mean Square Error (RMSE) of the posterior mean simulation of about 0.01 cm$^3$/cm$^3$. About 95% of the observations lies within the gray region, an indication that the simulation uncertainty ranges are statistically adequate. The acceptance rate of DREAM averages about 12.6% – about half of its theoretical optimal value of 22 – 25% (for Gaussian and Student target distributions). This deficiency is explained in part by the high nonlinearity of retention and hydraulic conductivity functions, and numerical errors of the implicit, time-variable, solver of the Richards' equation. This introduces irregularities (e.g. local optima) in the posterior response surface and makes the journey to and sampling from the target distribution more difficult.

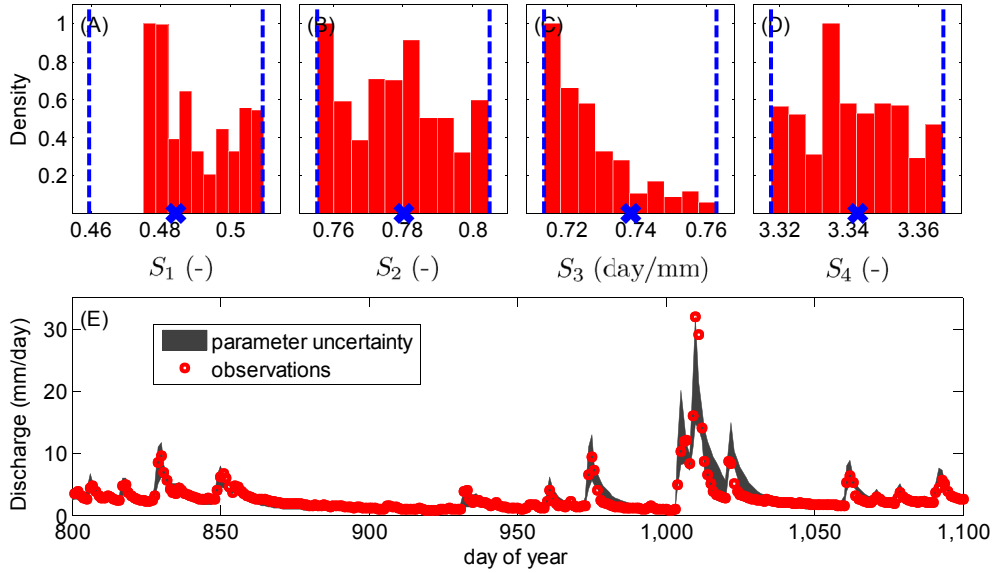


**Fig. 10.** Application of DREAM$_{(ABC)}$ to the hmodel using historical data from the Guadalupe River at Spring Branch, Texas. Posterior marginal distribution of the summary metrics (A) $S_1$ (runoff coefficient), (B) $S_2$ (baseflow index), (C) $S_3$ and (D) $S_4$ (two coefficients of the flow duration curve). The blue vertical lines are epsilon removed from the observed summary metrics (blue cross) and delineate the behavioral (posterior) model space. The bottom panel (E) presents the 95% simulation uncertainty ranges of the hmodel for a selected 300-day portion of the calibration data set. The observed discharge data are indicated with red circles. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

### 5.4. Case study IV: diagnostic model evaluation

The fourth case study illustrates the ability of DREAM to be used for diagnostic model evaluation. A rather parsimonious 7-parameter lumped watershed model (also known as hmodel) is used with historical data from the Guadalupe River at Spring Branch, Texas. This is the driest of the 12 MOPEX basins described in the study of Duan et al. (2006). The model structure and hydrologic process representations are found in Schoups and Vrugt (2010). The model transforms rainfall into runoff at the watershed outlet using explicit process descriptions of interception, throughfall, evaporation, runoff generation, percolation, and surface and subsurface routing.

Daily discharge, mean areal precipitation, and mean areal potential evapotranspiration were derived from Duan et al. (2006) and used for diagnostic model evaluation with DREAM$_{(ABC)}$ (Sadegh and Vrugt, 2014). Details about the basin and experimental data, and likelihood function can be found there, and will not be discussed herein. The same model and data was used in a previous study Schoups and Vrugt (2010), and used to introduce the generalized likelihood function of Table 1.

Four different summary metrics of the discharge data are used for ABC inference (activated with likelihood function 22), including $S_1$ (-) the annual runoff coefficient, $S_2$ (-) the annual baseflow coefficient, and $S_3$ (day/mm) and $S_4$ (-) two coefficients of the flow duration curve (Vrugt and Sadegh, 2013; Sadegh et al., 2015a). The following setup is used in the MATLAB package of DREAM.

The simulated summary metrics cluster closely (within epsilon) around their observed counterparts. About 15,000 function evaluations were required with DREAM$_{(ABC)}$ to converge to a limiting distribution (not shown). This is orders of magnitude more efficient than commonly used rejection samplers (Sadegh and Vrugt, 2014). Note that the hmodel nicely mimics the observed discharge dynamics with simulation uncertainty ranges that envelop a large portion of the discharge observations. Thus, the four summary metrics used herein contain sufficient information to provide a reasonably adequate calibration. The interested reader is referred to Vrugt and Sadegh (2013) and Vrugt (submitted for publication) for a much more detailed ABC analysis with particular focus on diagnosis and detection of epistemic errors.

### 5.5. Case study V: Bayesian model averaging

Ensemble Bayesian Model Averaging (BMA) proposed by Raftery et al. (2005) is a widely used method for statistical post-processing of forecasts from an ensemble of different models. The BMA predictive distribution of any future quantity of interest is a weighted average of probability density functions centered on the bias-corrected forecasts from a set of individual models. The weights are the estimated posterior model probabilities, representing each model's relative forecast skill in the training (calibration) period.

Successful application of BMA requires estimates of the weights and variances of the individual competing models in

DREAM SETUP CASE STUDY 4 : Diagnostic model evaluation.

```
%% Problem settings defined by user
DREAMPar.d = 7;                                          % Dimension of the problem
DREAMPar.N = 10;                                         % Number of Markov chains
DREAMPar.T = 5000;                                       % Number of generations
DREAMPar.lik = 22;                                       % Model output summary statistics (DREAM_ABC)

%% Initial sampling and parameter ranges
Par_info.initial = 'latin';                              % Latin hypercube sampling
Par_info.boundhandling = 'reflect';                      % Explicit boundary handling: reflection
% Par. names:   Imax Smax Qsmax alE alF K_F K_S
Par_info.min = [ 0.5  10    0    0  -10  0   0  ];      % Lower bound parameters
Par_info.max = [ 10   1000  100  100  10 10  150 ];      % Upper bound parameters

%% Define name of function (.m file) for posterior exploration
Func_name = 'hmodel';

%% Load calibration data vector (summary metrics with which simulations are compared)
daily_data = load('03451500.dly');                       % Load French Broad dataset
Meas_info.S = Calc_metrics( daily_data(731:end,6) )'; % Calculate summary statistics

%% Optional settings
options.parallel = 'yes';                                % Run chains in parallel
options.IO = 'no';                                       % No file-writing to communicate with hmodel

%% Run the DREAM algorithm (diagnostic model evaluation)
[chain,output,fx] = DREAM(Func_name,DREAMPar,Par_info,Meas_info,options);
```

The function Calc_metrics returns the values of the four summary statistics using as input a record of daily discharge values. The actual model crr_model is written in the C-language and linked to MATLAB into a shared library called a MEX-file. The use of such MEX function significantly reduces the wall-time of DREAM.

Fig. 10 (top panel) presents histograms of the marginal distributions of the summary statistics. The posterior summary metrics lie within epsilon of their observed values, a necessary requirement for successful ABC inference. The bottom panel presents a time series plot of the observed (red dots) and hmodel simulated streamflow values. The dark gray region constitutes the 95% simulation uncertainty of the hmodel due to parameter uncertainty.

the ensemble. In their seminal paper, Raftery et al. (2005) recommends using the Expectation Maximization (EM) algorithm (Dempster et al., 1997). This method is relatively easy to implement, computationally efficient, but does not provide uncertainty estimates of the weights and variances. Here I demonstrate the application of DREAM to BMA model training using a 36-year record of daily streamflow observations from the Leaf River basin in the USA. An ensemble of eight different calibrated watershed models is taken from Vrugt and Robinson (2007a) and used in the present analysis. The names of these models and the RMSE ($m^3/s$) of their forecast error are listed in Table 6.

Theory, concepts and applications of DREAM$_{(BMA)}$ have been presented by Vrugt et al. (2008c) and interested readers are

referred to this publication for further details. Here, I restrict attention to the setup of BMA in the MATLAB package of DREAM.

The values of the weights depend somewhat on the assumed conditional distribution of the deterministic model

DREAM SETUP CASE STUDY 5 : Bayesian model averaging.

```
%% Problem settings defined by user
DREAMPar.T = 5000;                  % Number of generations
DREAMPar.lik = 2;                   % Model returns log-likelihood

%% Initial sampling and parameter ranges
Par_info.initial = 'latin';         % Latin hypercube sampling
Par_info.boundhandling = 'reflect'; % Explicit boundary handling

%% Define BMA as a global variable
global BMA

%% Define name of function (.m file) for posterior exploration
Func_name = 'BMA_calc';

%% Load data from Vrugt and Robinson, WRR, 43, W01411, doi:10.1029/2005WR004838, 2007
load data.txt;                      % Daily streamflow simulations eight watershed models
load Y.txt;                         % Daily streamflow observations
StartT = 1; EndT = 3000;            % Start/End day training period
BMA.PDF = 'gamma';                  % pdf predictor: normal/heteroscedastic/gamma
BMA.VAR = 'multiple';               % variance pdf: single/multiple (multiple for 'normal')

%% Setup the BMA model (apply linear bias correction)
[DREAMPar,BMA,Par_info] = Setup_BMA(DREAMPar,Par_info,BMA,data,Y,StartT,EndT);

%% Run the DREAM algorithm
[chain,output,fx] = DREAM(Func_name,DREAMPar,Par_info);
```

The predictive distribution of each constituent member of the ensemble is assumed to follow a gamma distribution with unknown heteroscedastic variance. The BMA_calc script is listed in Appendix C.

Table 6 summarizes the results of DREAM$_{(BMA)}$ and presents (in column "Gamma") the maximum a-posteriori (MAP) values of the BMA weights for the different models of the ensemble. Values listed in parentheses denote the posterior standard deviation derived from the DREAM sample. I also summarize the MAP values of the weights for a Gaussian (conditional) distribution (columns "Normal") with homoscedastic (left) or heteroscedastic (right) error variance, and report the average RMSE ($m^3/s$), coverage (%) and spread ($m^3/s$) of the resulting BMA model during the 26-year evaluation period.

**Table 6**
Results of DREAM$_{(BMA)}$ by application to eight different watershed models using daily discharge data from the Leaf River in Mississippi, USA. I list the individual forecast errors of the models for the training data period, the corresponding MAP values of the weights for a Gamma (default) and Gaussian forecast distribution (numbers between parenthesis list the posterior standard deviation), and present the results of the BMA model (bottom panel) during the evaluation period. The spread ($m^3/s$) and coverage (%) are derived from a 95% prediction interval.

| Model | RMSE | Gamma | Normal[a] | Normal[b] |
|---|---|---|---|---|
| ABC | 31.67 | 0.02 *(0.006)* | 0.03 *(0.010)* | 0.00 *(0.002)* |
| GR4J | 19.21 | 0.21 *(0.016)* | 0.14 *(0.013)* | 0.10 *(0.013)* |
| HYMOD | 19.03 | 0.03 *(0.008)* | 0.13 *(0.046)* | 0.00 *(0.005)* |
| TOPMO | 17.68 | 0.03 *(0.006)* | 0.08 *(0.047)* | 0.03 *(0.010)* |
| AWBM | 26.31 | 0.05 *(0.009)* | 0.01 *(0.010)* | 0.00 *(0.002)* |
| NAM | 20.22 | 0.05 *(0.011)* | 0.14 *(0.048)* | 0.11 *(0.014)* |
| HBV | 19.44 | 0.24 *(0.017)* | 0.13 *(0.034)* | 0.31 *(0.016)* |
| SACSMA | 16.45 | 0.37 *(0.017)* | 0.34 *(0.022)* | 0.43 *(0.017)* |
| BMA: log-likelihood | | −9775.1 | −9950.5 | −9189.4 |
| BMA: RMSE | | 22.54 | 23.22 | 23.16 |
| BMA: spread | | 39.74 | 46.98 | 46.54 |
| BMA: coverage | | 93.65% | 92.59% | 95.71% |

[a] Homoscedastic (fixed) variance.
[b] Heteroscedastic variance.

forecasts of the ensemble. The GR4J, HBV and SACSMA models consistently receive the highest weights and are thus most important in BMA model construction for this data set. Note also that TOPMO receives a very low BMA weight, despite having the second lowest RMSE value of the training data period. Correlation between the individual forecasts of the watershed models affects strongly the posterior distribution of the BMA weights. The gamma distribution is preferred for probabilistic streamflow forecasting with 95% simulation uncertainty ranges that, on average, are noticeably smaller than their counterparts derived from a normal distribution. The interested reader is referred to Vrugt and Robinson (2007a) and Rings et al. (2012) for a more detailed analysis of the BMA results, and a comparison with filtering methods.

Fig. 11 presents histograms of the marginal posterior distribution of the BMA weights for each of the models of the ensemble. The MAP values of the weights are separately indicated with a blue cross.

The distributions appear rather well-defined and exhibit an approximate Gaussian shape. The posterior weights convey which models of the ensemble are of importance in the BMA model and which models can be discarded without harming the results. The use of fewer models is computational appealing as it will reduce the CPU-time to generate the ensemble.

### 5.6. Case study VI: generalized likelihood uncertainty estimation

Our sixth case study reports on GLUE and involves application of an informal likelihood function to the study of animal population dynamics. One of the first models to explain the interactions between predators and prey was proposed in 1925 by the American biophysicist Alfred Lotka and the Italian mathematician Vito Volterra. This model, one of the earliest in theoretical ecology, has been widely used to study population dynamics, and is given by the following system of two coupled differential equations

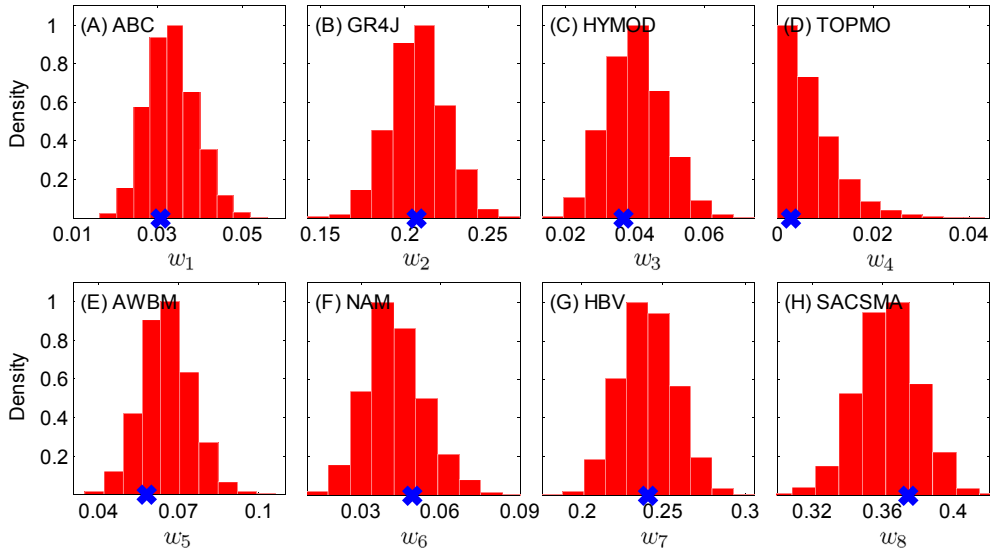


**Fig. 11.** Histograms of the marginal posterior distribution of the weights and variances of each individual model of the ensemble. The MAP values of the weights are denoted with a blue cross. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$$\frac{dP_t^1}{dt} = \alpha P_t^1 - \beta P_t^1 P_t^2$$
$$\frac{dP_t^2}{dt} = -\gamma P_t^2 + \delta P_t^1 P_t^2, \tag{38}$$

where $P_t^1$ and $P_t^2$ denote the size of the prey and predator population at time $t$ respectively, $\alpha$ (-) is the prey growth rate (assumed exponential in the absence of any predators), $\beta$ (-) signifies the attack rate (prey mortality rate for per-capita predation), $\gamma$ (-) represents the exponential death rate for predators in the absence of any prey, and $\delta$ (-) is the efficiency of conversion from prey to predator.

A synthetic monthly data set of a prey and predator population is created by solving Equation (38) numerically for a 20-year period using an implicit, time-variable, integration method (built-in ode solver of MATLAB). The initial states, $P_0^1 = 30$ and $P_0^2 = 4$ and parameter values $\alpha = 0.5471$, $\beta = 0.0281$, $\gamma = 0.8439$ and $\delta = 0.0266$ correspond to data collected by the Hudson Bay Company between 1900 and 1920. These synthetic monthly observations are subsequently perturbed with a homoscedastic error, and this corrupted data set is saved as text file "abundances.txt" and used for inference. The following setup of DREAM is used in MATLAB.

DREAM SETUP CASE STUDY 6 : Population dynamics modeling.

```
%% Problem settings defined by user
DREAMPar.d = 4;                              % Dimension of the problem
DREAMPar.N = 10;                             % Number of Markov chains
DREAMPar.T = 2500;                           % Number of generations
DREAMPar.lik = 32;                           % Model output simulation: Informal likelihood
DREAMPar.GLUE = 10;                          % Value of likelihood shape parameter (for GLUE)

%% Initial sampling and parameter ranges
Par_info.initial = 'latin';                  % Latin hypercube sampling
Par_info.boundhandling = 'reflect';          % Explicit boundary handling: reflection
% Par. names:    alpha beta gamma delta
Par_info.min = [   0     0    0     0    ]; % Lower bound parameters
Par_info.max = [   1     10   1     10   ]; % Upper bound parameters

%% Load calibration data vector
Meas_info.Y = load('abundances.txt');        % Load food web dataset

%% Define name of function (.m file) for posterior exploration
Func_name = 'lotka_volterra';

%% Optional settings
options.parallel = 'yes';                    % Run chains in parallel
options.modout = 'yes';                      % Store model simulations

%% Run the DREAM algorithm (diagnostic model evaluation)
[chain,output,fx] = DREAM(Func_name,DREAMPar,Par_info,Meas_info,options);
```

An informal likelihood function (32) is used to transform the difference between the observed and simulated predator-prey populations in a likelihood. The forward model script lotka_volterra can be found in Appendix C.

Fig. 12 presents the marginal posterior distributions of the parameters $\alpha$, $\beta$, $\gamma$ and $\delta$ (top panel) and displays (bottom panel) the 95% uncertainty (dark grey) of the simulated prey and predator populations. The observed species abundances are separately indicated with the red circles.

The parameter $\delta$ appears best defined by calibration against the observed species abundances with posterior ranges that are rather tight. The histograms of $\alpha$ and $\gamma$ are rather dispersed with posterior uncertainty ranges that encompass a large part of the prior distribution. This relatively large parameter uncertainty translates into an unrealistically large prediction uncertainty (bottom panel). Of course, the results of DREAM depend strongly on the value of the shaping factor, GLUE of DREAMPar in likelihood function 32. If this value is taken to be much larger (e.g. 100), then the marginal distributions would be much peakier and center on the "true" Lokta–Volterra parameter values used to generate the synthetic record of predator and prey populations. Moreover, the spread of the 95% prediction uncertainty ranges would be much smaller. Blasone et al. (2008) presents a more in-depth analysis of the application of MCMC simulation to GLUE inference.

### 5.7. Case study VII: limits of acceptability

In the manifesto for the equifinality thesis, Beven (2006) suggested that a more rigorous approach to model evaluation would involve the use of limits of acceptability for each individual observation against which model simulated values are compared. Within this framework, behavioral models are defined as those that satisfy the limits of acceptability for each observation. Our seventh and last case study briefly describes the application of DREAM to sampling the behavioral parameter space that satisfies the limits of acceptability of each observation.

I use a simple illustrative example involving modeling of the soil temperature $T$ in degrees Celsius using the following analytic equation

$$T(t,z) = T_0 + A_0 \exp\left(-\frac{z}{d}\right)\sin\left(\omega(t-\phi) - \frac{z}{d}\right), \tag{39}$$

where $t$ (hr) denotes time, $T_0$ ($^o$C) is the annual average temperature at the soil surface, $A_0$ ($^\circ$C) is the amplitude of the temperature fluctuation, $\omega = 2\pi/24$ (hr$^{-1}$) signifies the angular frequency, $\phi$ (hr) is a phase constant, $z$ (cm) is the depth in the soil profile (positive downward) and $d$ (cm) denotes the characteristic damping depth.

A synthetic record of hourly soil temperature observations at $z = 5$, $z = 10$, and $z = 15$ cm depth is used to illustrate the DREAM setup and results. This data set was created by solving Equation (39) in the model script heatflow (see Appendix C) for a 2-day period using $T_0 = 20^0$ C, $A_0 = 5^0$ C, $\phi = 8$ (hr) and $d = 20$ (cm). The hourly data was subsequently perturbed with a normally distributed error of $0.5^0$ C and used in the analysis. The limits of acceptability were set to be equal to 2 $^\circ$C for each of the $m = 144$ temperature observations. The four parameters $T_0$, $A_0$, $\phi$ and $d$ are determined from the observed temperature data using the following setup of DREAM in MATLAB.

DREAM SETUP CASE STUDY 7 : Temperature modeling.

```
%% Problem settings defined by user
DREAMPar.d = 4;                         % Dimension of the problem
DREAMPar.N = 10;                        % Number of Markov chains
DREAMPar.T = 5000;                      % Number of generations
DREAMPar.lik = 23;                      % Model output simulation (limits of acceptability)

%% Initial sampling and parameter ranges
Par_info.initial = 'latin';             % Latin hypercube sampling
Par_info.boundhandling = 'reflect';     % Explicit boundary handling: reflection
% Par. names:     T0   A0   phi   d
Par_info.min = [ 10   0    0     0   ];     % Lower bound parameters
Par_info.max = [ 30   10   24    50 ];     % Upper bound parameters

%% Define name of function (.m file) for posterior exploration
Func_name = 'heat_flow';

%% Load calibration data vector (summary metrics with which simulations are compared)
Meas_info.S = load('temp_data.txt');    % Temperature data are summary statistics

%% Optional settings
options.epsilon = 2;
options.modout = 'yes';

%% Run the DREAM algorithm (diagnostic model evaluation)
[chain,output,fx] = DREAM(Func_name,DREAMPar,Par_info,Meas_info,options);
```

The value of the effective observation error is assumed to be a constant, and consequently a scalar declaration suffices for this field epsilon of structure Meas_info. If the limits of acceptability are observation dependent then a vector, with in this case $m = 144$ values, should be defined.

Fig. 13 presents the results of the analysis. The top panel presents marginal distributions of the parameters (A) $T_0$, (B) $A_0$, (C) $\phi$, and (D) $d$, whereas the bottom panel presents time series plots of (E) the original temperature data before corruption with a measurement error, and the behavioral simulation space of Equation (39) in model at (F) 5, (G) 10 and (H) 15 cm depth in the soil profile. The gray region satisfies the limits of acceptability of each temperature observation and measurement depth.

The histograms center around their true values (denoted with a blue cross). The parameters $T_0$, $A_0$, and $\phi$ appear well defined, whereas the damping depth $d$ exhibits a large uncertainty. This uncertainty translates in a rather large uncertainty of the apparent soil thermal diffusivity, $K_T = \frac{1}{2}\omega d^2$ (cm$^2$ hr$^{-1}$). This concludes the numerical experiments. The interested reader is referred to Vrugt

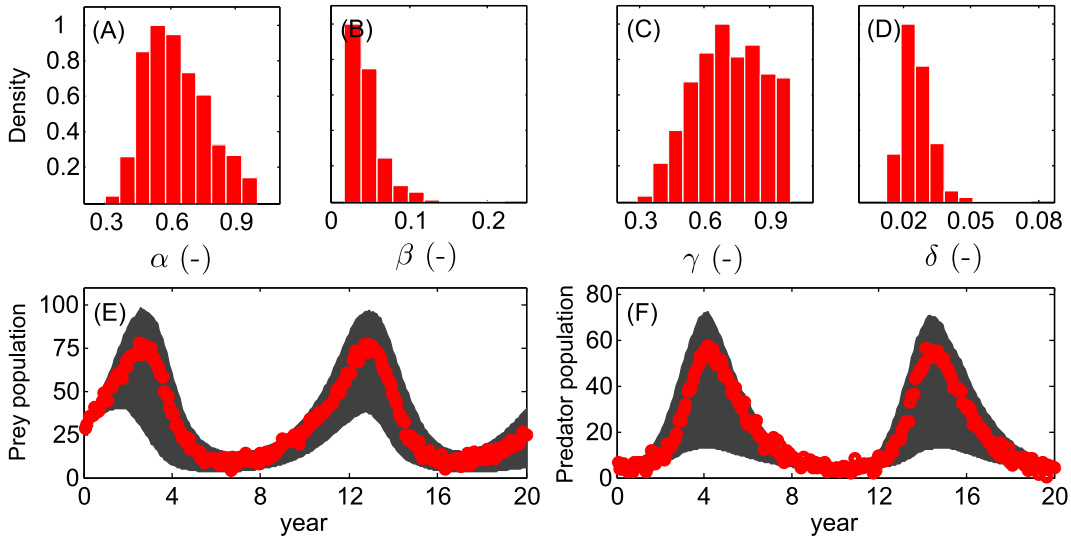


**Fig. 12.** Histograms (top panel) of the marginal posterior distribution of the Lotka–Volterra model parameters (A) $\alpha$, (B) $\beta$, (C) $\gamma$, and (D) $\delta$. Time series plot (bottom panel) of 95% simulation uncertainty ranges of the (E) prey and (F) predator populations. The observed data are indicated with the red circles. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

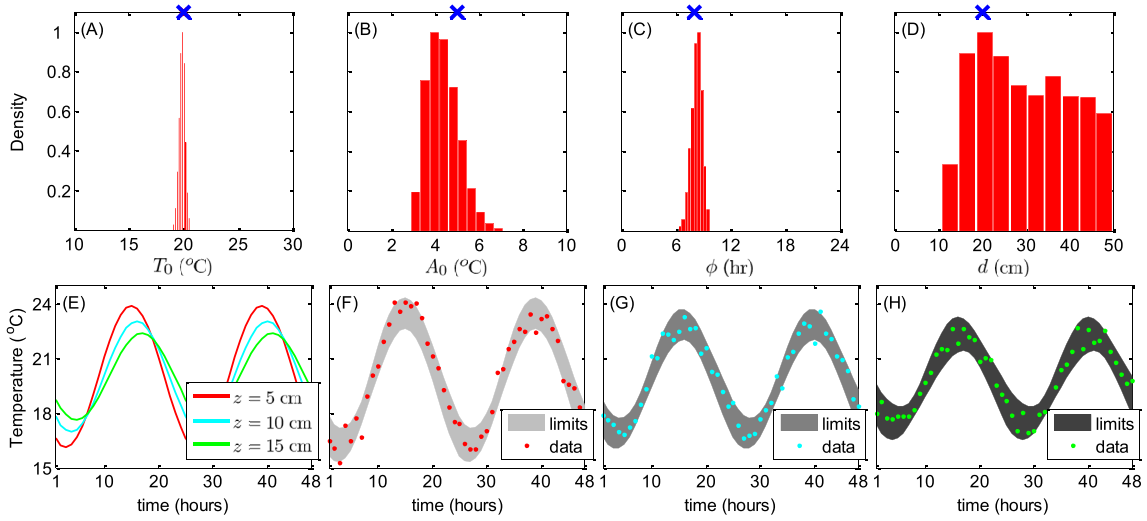


**Fig. 13.** Histograms (top panel) of the marginal posterior distribution of the heat flow parameters (A) $T_0$, (B) $A_0$, (C) $\phi$, and (D) $d$. The true values of the parameters are separately indicated with a blue cross. Time series plot (bottom panel) of (E) original data (before corruption) at the 5, 10 and 15 cm depth in the soil profile, and (F)–(H) behavioral simulation space (gray region) that satisfies the effective observation error (2 °C) of each temperature measurement. The corrupted data are separately indicated with dots. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

(2015a) for a more detailed exposition of membership-set likelihood functions such as those used in the GLUE limits of acceptability framework.

## 6. Additional options

The seven case studies presented herein illustrate only some of the capabilities of the DREAM software package. The script RUNDREAM presents a more exhaustive overview of the different functionalities of the DREAM toolbox, and includes 23 prototype example studies involving among others much more complex and higher dimensional target distributions as well, for example estimation of the two- and/or three-dimensional soil moisture distribution from travel time data of ground penetrating radar (Laloy et al., 2012; Linde and Vrugt, 2013) and treatment of rainfall uncertainty in hydrologic modeling (Vrugt et al., 2008a). Users can draw inspiration from these different test problems and use them as templates for their own modeling and inference problems. I now list a few important topics that have not been explicitly addressed herein.

### 6.1. Diagnostic Bayes

A recurrent issue with the application of ABC is self-sufficiency of the summary metrics, $S(\tilde{\mathbf{Y}})$. In theory, $S(\cdot)$ should contain as much information as the original data itself, yet complex systems rarely admit sufficient statistics. Vrugt (submitted for publication) therefore proposed in another recent article a hybrid approach, coined diagnostic Bayes, that uses the summary metrics as prior distribution and the original data in the likelihood function, or $p(\mathbf{x}|\tilde{\mathbf{Y}}) \propto p(\mathbf{x}|\mathbf{S}(\tilde{\mathbf{Y}}))L(\mathbf{x}|\tilde{\mathbf{Y}})$. This approach guarantees that no information is lost during the inference. The use of summary metrics as prior distribution is rather unorthodox and arguments of in favor of this approach are provided by Vrugt (submitted for publication).

Diagnostic Bayes is easily setup and executed within DREAM. The user has to set the field DB of structure options equal to 'yes'. Then, the observations of the calibration data and related summary statistics are stored in fields Y and S of structure Meas_info, respectively. The output of the model script consists of the simulated data, $\mathbf{Y}$ augmented at the end of the return vector Y with the values of the simulated summary statistics, $\mathbf{S}(\mathbf{Y}(\mathbf{x}))$.

### 6.2. Joint parameter and state estimation

The return argument Y of the function script model usually involves the output of some model, $\mathbf{Y} \leftarrow \mathscr{F}(\mathbf{x}, \cdot)$. The computation in this script can involve state estimation as well. The return argument of model then involves a time-series of forecasts derived from the Kalman filter. This approach, assumes time-invariant parameter values and is at the heart of SODA and particle-DREAM (Vrugt et al., 2005, 2013b).

### 6.3. Bayesian model selection

Inferences about the model parameters are typically made from the unnormalized posterior density, $p(\mathbf{x}|\tilde{\mathbf{Y}})$ in Equation (5). This Equation ignores the normalization constant, $p(\tilde{\mathbf{Y}})$. This constant, also referred to as marginal likelihood or evidence can be derived from multi-dimensional integration of the posterior distribution, $p(\tilde{\mathbf{Y}}) = \int_{\boldsymbol{\chi}} p(\mathbf{x}) L(\mathbf{x}|\tilde{\mathbf{Y}}) \mathrm{d}\boldsymbol{\chi}$, where $\mathbf{x} \in \boldsymbol{\chi} \in \mathbb{R}^d$. In the case of multiple competing model hypotheses

$$p\left(\tilde{\mathbf{Y}}\middle|H\right) = \int_{-\infty}^{\infty} p(H) L\left(H\middle|\tilde{\mathbf{Y}}\right) \mathrm{d}H \tag{40}$$

the model with the largest value of $p(\tilde{\mathbf{Y}}|H)$ is preferred statistically.

The statistical literature has introduced several methods to determine $p(\tilde{\mathbf{Y}}|H)$ (Chib, 1995; Kass and Raftery, 1995; Meng and Wong, 1996; Lewis and Raftery, 1997; Gelman and Meng, 1998). Numerical experiments with a suite of different benchmark functions have shown that these approaches are not particularly accurate for high-dimensional and complex target distributions that deviate markedly from multi-normality. Volpi et al. (2015) have therefore presented a new estimator of the marginal likelihood which works well for a large range of posterior target distributions. This algorithm uses the posterior samples derived from DREAM and is integrated in the MATLAB package.

### 6.4. Improved treatment of uncertainty

Most applications of Bayesian inference in Earth and environmental modeling assume the model to be a perfect representation of reality, the input (forcing) data to be observed without error, and consequently the parameters to be the only source of uncertainty. These assumptions are convenient in applying statistical theory but often not borne out of the properties of the error residuals whose probabilistic properties deviate often considerably from normality with (among others) non-constant variance, heavy tails, and varying degrees of skewness and temporal and/or spatial correlation. Bayes law allows for treatment of all sources of modeling error through the use of nuisance variables, $\boldsymbol{\beta}$ for instance

$$p\left(\mathbf{x}, \boldsymbol{\beta}, \tilde{\mathbf{U}}, \tilde{\psi}_0 \middle| \tilde{\mathbf{Y}}\right) \propto p(\mathbf{x}) p(\boldsymbol{\beta}) p\left(\tilde{\mathbf{U}}\right) p(\tilde{\psi}_0) L\left(\mathbf{x}, \boldsymbol{\beta}, \tilde{\mathbf{U}}, \tilde{\psi}_0 \middle| \tilde{\mathbf{Y}}\right). \tag{41}$$

The nuisance variables are coefficients in error models of the initial states and forcing data, respectively and their values subject to inference with the parameters using the observed data, $\tilde{\mathbf{Y}}$. The BATEA framework is an example of this more advanced approach (Kavetski et al., 2006a,b; Kuczera et al., 2006; Renard et al., 2010, 2011), and can be implemented with DREAM as well (Vrugt et al., 2008a, 2009a,b). The formulation of Equation (41) is easily adapted to include errors in the calibration data as well (see Appendix B) though it remains difficult to treat epistemic errors. What is more, this approach with many nuisance variables will only work satisfactorily if a sufficiently strong prior is used for each individual error source. Otherwise the inference can rapidly degenerate and become meaningless.

One can also persist in treating model parameter uncertainty only, and use instead an advanced likelihood function whose nuisance variables render it flexible enough to mimic closely complex nontraditional error residual distributions (Schoups and Vrugt, 2010; Evin et al., 2013; Scharnagl et al., 2015). The results of such approach might be statistically meaningful in that the assumptions of the likelihood function are matched by the actual residual properties, yet this methodology provides little guidance on structural model errors.

The answer to this complicated problem of how to detect, diagnose and resolve model structural errors might lie in the use of summary statistics of the data rather than the data itself. A plea for this approach has been made by Gupta et al. (2008) and Vrugt and Sadegh (2013) have provided the mathematical foundation for diagnostic model evaluation using ABC. Subsequent work by Sadegh et al. (2015b) has shown the merits of this methodology by addressing the stationarity paradigm. Other recent work demonstrates that the use of summary metrics provides much better guidance on model malfunctioning (Vrugt, submitted for publication).

### 6.5. $\ell^{\Omega}$-norm of error residuals

Likelihood functions play a key role in statistical inference of the model parameters. Their mathematical formulation depends on the assumptions that are made about the probabilistic properties of the error residuals. The validity of these assumptions can be verified a-posteriori by inspecting the actual error residual time series of the posterior mean simulation. Likelihood functions based on a $\ell^2$-norm (squared residuals) are most often used in practical applications, despite their relative sensitivity to peaks and outlier data points. Their use is motivated by analytic tractability - that is - with relatively little ease confidence intervals of the parameters can be construed from a classical first-order approximation around the optimum. This attractive feature of a $\ell^2$-type likelihood function was of imminent importance in past eras without adequate computational resources but is a far less desirable quality nowadays with availability of powerful computing capabilities and efficient algorithms. Indeed, methods such as DREAM can solve for likelihood functions with any desired norm, $\Omega \in \mathbb{N}^+$. For instance, the Laplacian likelihood (see Table B1) uses a $\ell^1$ norm of the error residuals and therefore should be less sensitive to peaks and outliers. Unless there are very good reasons to adopt a $\ell^2$-type likelihood function, their use might otherwise be a historical relic (Beven and Binley, 2014).

### 6.6. Convergence monitoring

The most recent version of DREAM also includes calculation of the multivariate $\widehat{R}$-statistic of Gelman and Rubin (1992). This statistic, hereafter referred to as $\widehat{R}^d$-diagnostic, is defined in Brooks and Gelman (1998) and assesses convergence of the $d$ parameters simultaneously by comparing their within and between-sequence covariance matrix. Convergence is achieved when a rotationally

invariant distance measure between the two matrices indicates that they are "sufficiently" close. Then, the multivariate $\widehat{R}^d$-statistic achieves a value close to unity, otherwise its value is much larger. In fact, the $\widehat{R}$ and $\widehat{R}^d$-statistic take on a very similar range of values, hence simplifying analysis of when convergence has been achieved.

The $\widehat{R}^d$-statistic is particularly useful for high-dimensional target distributions involving complicated multi-dimensional parameter interactions. We do not present this statistic in the present paper, but the DREAM package returns its value at different iteration numbers in field MR_stat of structure options.

### 6.7. Prior distribution

The prior distribution, $p(\mathbf{x})$ describes all knowledge about the model parameters before any data is collected. Options include the use of noninformative (flat, uniform) and informative prior distributions. These built-in capabilities will not suffice for applications involving complex prior parameter distributions defined (or not) by a series of simulation steps rather than some analytic distribution. Such priors are used abundantly in the fields of geostatistics and geophysics, and have led Mosegaard and Tarantola (1995) to develop the extended Metropolis algorithm (EMA). This algorithm builds on the standard RWM algorithm, but samples proposals from the prior distribution instead, thereby honoring existing data and the spatial structure of the variable of interest (Hansen et al., 2012; Laloy et al., 2015). The acceptance probability in Equation (14) then becomes

$$p_{\text{acc}}(\mathbf{x}_{t-1} \rightarrow \mathbf{x}_{\text{p}}) = \min\left[1, \frac{L(\mathbf{x}_{\text{p}})}{L(\mathbf{x}_{t-1})}\right], \tag{42}$$

and the resulting chain simulated by EMA satisfies detailed balance. This approach, also known as sequential simulation or sequential geostatistical resampling, can handle complex geostatistical priors, yet its efficiency is critically dependent on the proposal mechanism used to draw samples from the prior distribution (Laloy et al., 2015; Ruggeri et al., 2015).

The basic idea of EMA is readily incorporated in DREAM by replacing the parallel direction jump of Equation (23) with a swap type proposal distribution used in the DREAM$_{(\text{D})}$ algorithm (see section). For instance, the most dissimilar entries of two other chains can be used to guide which coordinates to draw from the prior distribution. This adaptive approach shares information about the topology of the search space between the different chains, a requirement to speed up the convergence to the target distribution. I will leave this development for future research.

Dimensionality reduction methods provide an alternative to EMA and represent the spatial structure of the variable of interest with much fewer parameters than required for pixel based inversion while maintaining a large degree of fine-scale information. This allows for the use of standard closed-form prior distributions for the reduced set of parameters. Examples of such approaches include the discrete cosine transform (Jafarpour et al., 2009, 2010; Linde and Vrugt, 2013; Lochbühler et al., 2015), wavelet transform (Davis and Li, 2011; Jafarpour, 2011), and singular value decomposition (Laloy et al., 2012; Oware et al., 2013).

## 7. The DREAM family of algorithms

In the past years, several other MCMC algorithms have appeared in the literature with a high DREAM pedigree. These algorithms use DREAM as their basic building block but include special extensions to simplify inference (among others) of discrete and combinatorial search spaces, and high-dimensional and CPU-intensive system models. These algorithms have their own individual MATLAB toolboxes identical to what is presented herein for DREAM, but with unique algorithmic parameters. I briefly describe each of these algorithms below, and discuss their algorithmic parameters in the MATLAB code.

### 7.1. DREAM$_{(ZS)}$

This algorithm creates the jump vector in Equation (23) from the past states of the joint chains. This idea is implemented as follows. If $\mathbf{Z} = \{\mathbf{z}^1,\ldots,\mathbf{z}^m\}$ is a matrix of size $m \times d$ which thinned history of each of the $N$ chains, then the jump is calculated using

$$\begin{aligned} \mathbf{dX}^i_{\text{A}} &= \zeta_{d^*} + (1_{d^*} + \lambda_{d^*})\gamma_{(\delta,d^*)} \sum_{j=1}^{\delta} \left(\mathbf{Z}^{\mathbf{a}_j}_{\text{A}} - \mathbf{Z}^{\mathbf{b}_j}_{\text{A}}\right) \\ \mathbf{dX}^i_{\neq \text{A}} &= 0, \end{aligned} \tag{43}$$

where $\mathbf{a}$ and $\mathbf{b}$ are $2\delta N$ integer values drawn without replacement from $\{1,\ldots,m\}$.

The DREAM$_{(ZS)}$ algorithm contains two additional algorithmic variables compared to DREAM, including $m_0$, the initial size (number of rows) of the matrix $\mathbf{Z}$ and $k$ the rate at which samples are appended to this external archive. Their recommended default values are $m_0 = 10d$ and $k = 10$ iterations respectively. The initial archive $\mathbf{Z}$ is drawn from the prior distribution of which the last $N$ draws are copied to the matrix $\mathbf{X}$ which stores the current state of each chain. After each $k$ draws (generations) in each Markov chain, the matrix $\mathbf{X}$ is appended to $\mathbf{Z}$.

The use of past samples in the jump distribution of Equation (43) has three main advantages. First, a much smaller number of chains suffices to explore the target distribution. This not only minimizes the number of samples required for burn-in, but also simplifies application of DREAM$_{(ZS)}$ to high-dimensional search spaces. Indeed, whereas DREAM requires at least $N \geq d/2$, benchmark experiments with DREAM$_{(ZS)}$ have shown that $N = 3$ chains (set as default) suffices for a large range of target dimensionalities. Second, because the proposal distribution in DREAM$_{(ZS)}$ uses past states of the chains only, each trajectory can evolve on a different processor. Such distributed implementation is used within DREAM as well, but violates, at least theoretically, the convergence proof (see Section 3.3). Third, outlier chains do not need forceful treatment. Such chains can always sample their own past and with a periodic value of $\gamma = 1$ jump directly to the mode of the target.

The sampling from an external archive of past states violates the Markovian principles of the sampled chains, and turns the method into an adaptive Metropolis sampler (Roberts and Rosenthal, 2007; ter Braak and Vrugt, 2008). To ensure convergence to the exact target distribution the adaptation should decrease in time, a requirement satisfied by DREAM$_{(ZS)}$ as $\mathbf{Z}$ grows by an order of $N/m = k/t$ which hence slows down with generation $t$ (ter Braak and Vrugt, 2008).

To enhance the diversity of the proposals created by DREAM$_{(ZS)}$, the algorithm includes a mix of parallel direction and snooker jumps (ter Braak and Vrugt, 2008). This snooker jump is depicted schematically in Fig. 14 and uses an adaptive step size. The indexes $a$, $b$ and $c$ are drawn randomly from the integers $\{1,\ldots,m\}$ (without replacement).

The orientation of the snooker jump is determined by the line $\mathbf{X}^i\mathbf{Z}^a$ going through the current state of the $i$th chain and sample $a$ of the external archive. The snooker axis is now defined by the line $\mathbf{Z}^b\mathbf{Z}^c$ and is projected orthogonally on to the line $\mathbf{X}^i\mathbf{Z}^a$. The difference between the two projection points $\mathbf{Z}^b_\perp$ and $\mathbf{Z}^c_\perp$ now defines the length of the snooker jump as follows

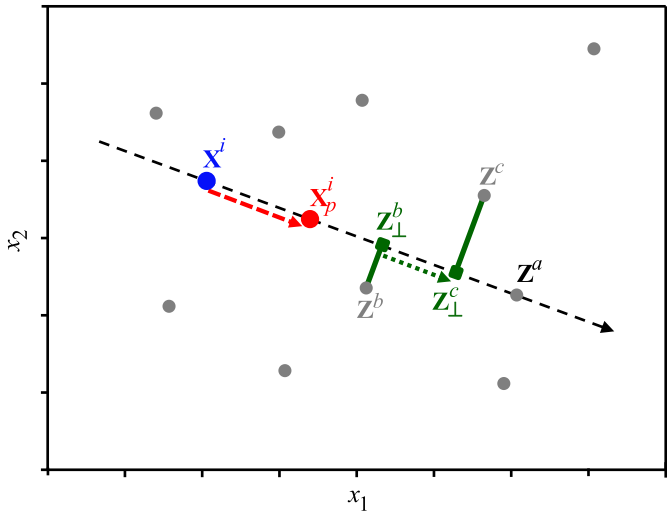


**Fig. 14.** DREAM$_{(ZS)}$ algorithm: Explanation of the snooker update for a hypothetical two-dimensional problem using some external archive of $m$=10 points (grey dots). Three points of this archive $\mathbf{Z}^a$, $\mathbf{Z}^b$ and $\mathbf{Z}^c$ are sampled at random and define the jump of the $i$th chain, $\mathbf{X}^i$ (blue) as follows. The points $\mathbf{Z}^b$ and $\mathbf{Z}^c$ are projected orthogonally on to the dotted $\mathbf{X}^i\mathbf{Z}^a$ line. The jump is now defined as a multiple of the difference between the projections points, $\mathbf{Z}^b_\perp$ and $\mathbf{Z}^c_\perp$ (green squares) and creates the proposal, , $\mathbf{X}^i_p$. The DREAM$_{(ZS)}$ algorithm uses a 90/10% mix of parallel direction and snooker updates, respectively. The probability of a snooker update is stored in field psnooker of structure DREAMPar. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

$$\mathbf{dX}^i = \gamma_s\left(\mathbf{Z}^b_\perp - \mathbf{Z}^c_\perp\right) + \zeta_d, \tag{44}$$

where $\gamma_s \overset{D}{\sim} \mathscr{U}[1.2, 2.2]$ signifies the snooker jump rate (ter Braak and Vrugt, 2008). The proposal point is then calculated using Equation (24).

The MATLAB code of DREAM$_{(ZS)}$ uses the exact same coding terminology and variables as DREAM but includes three additional fields in structure DREAMPar, that is m0, k and psnooker with default values of 10$d$, 10 and 0.1, respectively. These are the algorithmic parameters that determine the initial size of the external archive, the rate at which proposals are appended to this archive, and the probability of a snooker jump. Furthermore, a default value of $N = 3$ is used for the number of chains.

### 7.2. DREAM$_{(D)}$

The DREAM$_{(D)}$ code is especially developed to sample efficiently non-continuous, discrete, and combinatorial target distributions. This method helps solve experimental design problems involving real-time selection of measurements that discriminate best among competing hypothesis (models) of the system under consideration (Vrugt and ter Braak, 2011; Kikuchi et al., 2015). The DREAM$_{(D)}$ algorithm uses DREAM as its main building block and implements two different proposal distributions to recognize explicitly differences in topology between discrete and Euclidean search spaces. The first proposal distribution is a regular parallel direction jump

$$\begin{aligned} \mathbf{dX}^i_A &= \left\lfloor \zeta_{d^*} + (1_{d^*} + \lambda_{d^*})\gamma_{(\delta,d^*)} \sum_{j=1}^{\delta}\left(\mathbf{X}^{\mathbf{a}_j}_A - \mathbf{X}^{\mathbf{b}_j}_A\right) \right\rfloor_{d^*} \\ \mathbf{dX}^i_{\neq A} &= 0, \end{aligned} \tag{45}$$

but with each of the sampled dimensions of the jump vector rounded to the nearest integer using the operator, $\lfloor(\cdot)\rfloor$. The integer-valued proposals, $\mathbf{X}^i_p \in \mathbb{N}^d$; $i = \{1, \ldots, N\}$ can be transformed to non-integer values using a simple linear transformation

$$\mathbf{X}^i_p = \Delta\mathbf{x} \odot \mathbf{X}^i_p \tag{46}$$

where $\Delta\mathbf{x} = \{\Delta x_1, \ldots, \Delta x_d\}$ is a $1 \times d$-vector with discretization interval of each dimension of $\mathbf{x}$ and $\odot$ denotes element-by-element multiplication. For instance, consider a two-dimensional problem with prior $\mathscr{U}_2[-2, 6]$ (and thus Par_info.min = [−2 −2], Par_info.max = [6 6]) and $\Delta\mathbf{x} = \{1/4, 1/2\}$, then DREAM$_{(D)}$ samples the integer space, $x_1 \in [0{-}33]$ and $x_2 \in [0{-}17]$, respectively. A proposal, $\mathbf{X}^i_p = \{16, 9\}$ is then equivalent to $\{-2,-2\} + \{16\times 1/4, 9\times 1/2\} = \{2, 5/2\}$. The field steps of structure Par_info stores in a $1 \times d$-vector the values of $\Delta\mathbf{x}$. For an integer space, the value of $\Delta\mathbf{x} = 1_d$.

The parallel direction jump of Equation (45) works well for discrete problems but is not necessary optimal for combinatorial problems in which the optimal values are known a-priori but not their location in the parameter vector. The topology of such search problems differs substantially from Euclidean search problems. Vrugt et al. (2011) therefore introduce two alternative proposal distributions for combinational problems. The first of these two proposal distributions swaps randomly two coordinates in each individual chain. If the current state of the $i$th chain is given by $\mathbf{X}^i_{t-1} = [X^i_{t-1,1}, \ldots, X^i_{t-1,j}, \ldots, X^i_{t-1,k}, \ldots, X^i_{t-1,d}]$ then the candidate point becomes, $\mathbf{X}^i_p = [X^i_{p,1}, \ldots, X^i_{p,k}, \ldots, X^i_{p,j}, \ldots, X^i_{p,d}]$ where $j$ and $k$ are sampled without replacement from the integers $\{1,\ldots,d\}$. It is straightforward to see that this proposal distribution satisfies detailed balance as the forward and backward jump have equal probability.

This coordinate swapping does not exploit any information about the topology of the solution encapsulated in the position of the other $N-1$ chains. Each chain essentially evolves independently to the target distribution. This appears rather inefficient, particularly for complicated search problems. The second proposal distribution takes explicit information from the dissimilarities in coordinates of the $N$ chains evolving in parallel. This idea works as follows Vrugt et al. (2011). Let $\mathbf{X}^a$ and $\mathbf{X}^b$ be two chains that are chosen at random from the population $\mathbf{X}_{t-1}$. From the dissimilar coordinates of $a$ and $b$ two different dimensions, say $j$ and $k$, are picked at random, and their values swapped within each chain. The resulting two proposals, $\mathbf{X}^a_p$ and $\mathbf{X}^b_p$ are subsequently evaluated by model and the product of their respective Metropolis ratios calculated, $p_{acc}(\mathbf{X}^a \rightarrow \mathbf{X}^a_p) p_{acc}(\mathbf{X}^b \rightarrow \mathbf{X}^b_p)$ using Equation (14). If this product is larger than the random label drawn from $\mathscr{U}(0, 1)$ then both chains move to their respective candidate points, that is, $\mathbf{x}^a_t = \mathbf{X}^a_p$ and $\mathbf{x}^b_t = \mathbf{X}^b_p$, otherwise they remain at their current state, $\mathbf{x}^a_t = \mathbf{x}^a_{t-1}$ and $\mathbf{x}^b_t = \mathbf{x}^b_{t-1}$.

The dimensions $j$ and $k$ are determined by the dissimilarities of the $d$ coordinate values of two different chains. Unlike the random swap this second proposal distribution (also referred to as directed swap) shares information between two chains about their state. Those coordinates of the chains that are dissimilar are swapped, a strategy that expedites convergence to the target distribution. The swap move is fully Markovian, that is, it uses only information from the current states for proposal generation, and maintains detailed balance (Vrugt et al., 2011). If the swap is not feasible (less than two dissimilar coordinates), the current chain is simply sampled again. This is necessary to avoid complications with unequal probabilities of move types (Denison et al., 2002), the same trick is applied in reversible jump MCMC (Green, 1995). Restricting the swap to dissimilar coordinates does not destroy detailed balance, it just selects a subspace to sample on the basis of the current state. For combinational search problems, the DREAM$_{(D)}$ algorithm uses a default 90/10% mix of directed and random swaps, respectively.

The field prswap of structure DREAMPar in DREAM$_{(D)}$ defines the probability of a random swap (default DREAMPar.prswap = 0.1).

### 7.3. DREAM$_{(DZS)}$

Alternative (not published) discrete variant of DREAM$_{(ZS)}$. This code uses discrete (snooker) sampling from the past to explore target distributions involving non-continuous and/or combinatorial parameter spaces.

### 7.4. DREAM$_{(ABC)}$

This code has been developed by Sadegh and Vrugt (2014) for diagnostic model evaluation and can be activated from within DREAM using likelihood function 22. Implementation details have been discussed in the main text of this paper and in Appendix B and C.

### 7.5. DREAM$_{(BMA)}$

Specific implementation of DREAM for Bayesian model averaging. Theory and application of this method have been discussed in Vrugt et al. (2008c) and an example has been presented in the case studies section of this paper.

### 7.6. MT-DREAM$_{(ZS)}$

The MT-DREAM$_{(ZS)}$ algorithm uses multiple-try sampling (Liu et al., 2000), snooker updating, and sampling from an archive of past states to enhance the convergence speed of CPU-intensive and parameter rich models. Benchmark experiments in geophysics, hydrology and hydrogeology have shown that this sampler is able to sample correctly high-dimensional target distributions (Laloy and Vrugt, 2012; Laloy et al., 2012, 2013; Linde and Vrugt, 2013; Carbajal et al., 2014; Lochbühler et al., 2014, 2015).

The MT-DREAM$_{(ZS)}$ algorithm uses as basic building block the DREAM$_{(ZS)}$ algorithm and implements multi-try sampling in each of the chains. This multi-try scheme is explained in detail by Laloy and Vrugt (2012) and creates $\mu$ different proposals in each of the $N = 3$ (default) chains. If we use symbol $J_d(\cdot)$ to denote the jumping distributions in Equation (43) or (44) then this scheme works as follows. For convenience whenever the symbol $j$ is used I mean 'for all $j \in \{1,\ldots,\mu\}$'.

(1) Create $\mu$ proposals, $\mathbf{X}_p^j = \mathbf{X}^i + J_d(\cdot)$.
(2) Calculate $w_p^j$, the product of prior and likelihood of $\mathbf{X}_p^j$ and store values in $\mu$-vector, $\mathbf{w}_p = \{w_p^1, \ldots, w_p^\mu\}$.
(3) Select $\mathbf{X}_p^i$ from $\mathbf{X}_p$ using selection probabilities $\mathbf{w}_p$.
(4) Set $\mathbf{X}_r^1 = \mathbf{X}_p^i$ and create remaining $\mu-1$ points of reference set, $\mathbf{X}_r^j = \mathbf{X}_p^i + J_d(\cdot)$.
(5) Calculate $w_r^j$, the product of prior and likelihood of $\mathbf{X}_r^j$ and store values in $\mu$-vector, $\mathbf{w}_r = \{w_r^1, \ldots, w_r^\mu\}$.
(6) Accept $\mathbf{X}_p^i$ with probability

$$p_{acc}\left(\mathbf{X}^i \rightarrow \mathbf{X}_p^i\right) = \min\left[1, \frac{\left(w_r^1 + \ldots + w_r^\mu\right)}{\left(w_p^1 + \ldots + w_p^\mu\right)}\right]. \quad (47)$$

It can be shown that this method satisfies the detailed balance condition and therefore produces a reversible Markov chain with the target distribution as the stationary distribution (Liu et al., 2000).

The advantage of this multi-try scheme is that the $\mu$ proposals can be evaluated in parallel. With the use of $N = 3$ chains this would require only $N \times mt$ processors, which is much more practical for large $d$ than running DREAM in parallel with large $N$ (Laloy and Vrugt, 2012). Compared to DREAM$_{(ZS)}$ the MT-DREAM$_{(ZS)}$ algorithm has one more algorithmic parameter, $\mu$, the number of multi-try proposals in each of the $N$ chains. This variable is stored in field mt of DREAMPar and assumes a default value of $\mu = 5$.

## 8. Summary

In this paper I have reviewed the basic theory of Markov chain Monte Carlo (MCMC) simulation and have introduced a MATLAB package of the DREAM algorithm. This toolbox provides scientists and engineers with an arsenal of options and utilities to solve posterior sampling problems involving (amongst others) bimodality, high-dimensionality, summary statistics, bounded parameter spaces, dynamic simulation models, formal/informal likelihood functions, diagnostic model evaluation, data assimilation, Bayesian model averaging, distributed computation, and informative/noninformative prior distributions. The DREAM toolbox supports parallel computing and includes tools for convergence analysis of the sampled chain trajectories and post-processing of the results. Seven different case studies were used to illustrate the main capabilities and functionalities of the MATLAB toolbox. These example studies are easy to run and adapt and serve as templates for other inference problems.

A graphical user interface (GUI) of DREAM is currently under development and will become available in due course.

## Acknowledgments

The comments of Niklas Linde and two anonymous referees are greatly appreciated. Guilherme Gomez is thanked for careful proof reading of the revised manuscript, and Gerrit Schoups and Benedikt Scharnagl are acknowledged for contributing code for likelihood functions 14 and 17, respectively. I highly appreciate the feedback of the many users. Their comments and suggestions have been a source of inspiration for code improvement, and implementation of new concepts and functionalities. The MATLAB toolbox of DREAM is available upon request from the author, jasper@uci.edu.

## Appendices

### Appendix A

Table A1 summarizes, in alphabetic order, the different function/program files of the DREAM package in MATLAB. The main program RUNDREAM contains 23 different prototype studies which cover a large range of problem features. These example studies have been published in the geophysics, hydrologic, pedometrics, statistics and vadose zone literature, and provide a template for users to setup their own case study. The last line of each example study involves a function call to DREAM, which uses all the other functions listed on the next page to generate samples of the posterior distribution. Each example problem of RUNDREAM has its own directory which stores the model script written by the user and all other files (data file(s), MATLAB scripts, external executable(s), etc.) necessary to run this script and compute the return argument Y.

If activated by the user (field diagnostics of structure options is set to 'yes'), then at the end of each DREAM trial, the autocorrelation function, Geweke (1992) and Raftery and Lewis (1992) convergence diagnostic are computed separately for each of the $N$ chains using the CODA toolbox written by James P. LeSage (http://www.spatial-econometrics.com/). These functions are stored in the folder "../diagnostics" under the main DREAM directory and produce an output file called "DREAM_diagnostics.txt" which is printed to the screen in the MATLAB editor at the end of each DREAM trial. These within-chain convergence diagnostics were designed specifically for single-chain Metropolis samplers, and augment the multi-chain univariate and multivariate $\hat{R}$ and $\hat{R}^d$-statistic of Gelman and Rubin (1992) and Brooks and Gelman (1998) stored in fields R_stat and MR_stat of structure output, respectively. Joint interpretations of all these different convergence

diagnostics allows for a better assessment of when convergence to the target distribution has been achieved. The single-chain diagnostics, require each chain to have at least 200 samples otherwise the file "DREAM_diagnostics.txt" is returned empty.

The directory "../postprocessing" contains a number of different functions designed to visualize the results (output arguments) of DREAM. The program DREAM_POSTPROC creates a large number of MATLAB figures, including (among others) traceplots of the sampled chain trajectories, bivariate scatter plots and histograms of the posterior samples, traceplots of the $\widehat{R}$- and $\widehat{R}^d$-diagnostics, autocorrelation functions of the sampled parameter values, quantile–quantile plots of the error residuals, time series plots of the 95% simulation (prediction) uncertainty intervals. If ABC or diagnostic Bayes is used then marginal distributions of the sampled summary statistics are plotted as well. The number of figures that is plotted depends on the dimensionality of the target distribution, the number of chains used, and the type of output argument (e.g. likelihood/simulation/summary metrics or combination thereof) that is returned by the function model written by the user.

**Table A1**
Description of the MATLAB functions and scripts (.m files) used by DREAM, version 3.0.

| Name of function | Description |
|---|---|
| ADAPT_PCR | Calculates the selection probabilities of each crossover value |
| BOUNDARY_HANDLING | Corrects those parameter values of each proposal that are outside the search domain (if so desired) |
| CALC_DELTA | Calculates the normalized Euclidean distance between successive samples of each chain |
| CALC_DENSITY | Calculates the log-likelihood of each proposal |
| CALC_PROPOSAL | Computes proposals (candidate points) using differential evolution (see Equations (23) and (24)) |
| CHECK_SIGMA | Verifies whether the measurement error is estimated along with the parameters of the target distribution |
| DRAW_CR | Draws crossover values from discrete multinomial distribution |
| DREAM | Main DREAM function that calls different functions and returns sampled chains, diagnostics, and/or simulations |
| DREAM_CALC_SETUP | Setup of computational core of DREAM and (if activated) the distributed computing environment |
| DREAM_CHECK | Verifies the DREAM setup for potential errors and/or inconsistencies in the settings |
| DREAM_END | Terminates computing environment, calculates single-chain convergence diagnostics, and checks return arguments |
| DREAM_INITIALIZE | Samples the initial state of each chain |
| DREAM_SETUP | Setup of the main variables used by DREAM (pre-allocates memory) |
| DREAM_STORE_RESULTS | Appends model simulations to binary file "Z.bin" |
| EVALUATE_MODEL | Evaluates the proposals (executes function/model script Func_name) |
| GELMAN | Calculates the $\widehat{R}$ convergence diagnostic of Gelman and Rubin (1992) |
| GL | Evaluates generalized likelihood function of Schoups and Vrugt (2010a) |
| LATIN | Latin hypercube sampling |
| METROPOLIS_RULE | Computes Metropolis selection rule to accept/reject proposals |
| MOMENT_TPDF | Calculate absolute moments of the skewed standardized t-distribution |
| REMOVE_OUTLIER | Verifies presence of outlier chains and resets their states |
| RUNDREAM | Setup of 17 different example problems and calls the main DREAM script |
| WHITTLE | Evaluates Whittle's likelihood function (Whittle, 1953) |

## Appendix B

The mathematical formulations of the built-in likelihood functions of DREAM in Table 2 are given in Table B1 below. For convenience, $\mathbf{E}(\mathbf{x}) = \{e_1(\mathbf{x}),\ldots,e_n(\mathbf{x})\}$ signifies the $n$-vector of residuals, $\tilde{\mathbf{S}} = \{S_1(\tilde{\mathbf{Y}}),\ldots,S_m(\tilde{\mathbf{Y}})\}$ and $\mathbf{S} = \{S_1(\mathbf{Y}(\mathbf{x})),\ldots,S_m(\mathbf{Y}(\mathbf{x}))\}$ are $m$-vectors with observed and simulated summary statistics, respectively, and $\mathbf{A} = \{a_1,\ldots,a_n\}$ is a $n$-vector of filtered residuals in likelihood function 14 using an autoregressive model with coefficients, $\boldsymbol{\phi} = \{\phi_1,\ldots,\phi_4\}$.

**Table B1**
Mathematical formulation of built-in likelihood functions of DREAM. Option (1) and (2) return directly a likelihood and log-likelihood value, respectively, and their formulation is defined in the model script by the user.

| lik | Mathematical formulation | Nuisance variables | Note |
|---|---|---|---|
| | **Formal likelihood functions** | | |
| 11 | $\mathscr{L}(\mathbf{x}\|\tilde{\mathbf{Y}}) = -\frac{n}{2}\log\{\sum_{t=1}^{n} e_t(\mathbf{x})^2\}$ | none | |
| 12 | $\mathscr{L}(\mathbf{x}\|\tilde{\mathbf{Y}}) = -\frac{n}{2}\log(2\pi) - \sum_{t=1}^{n}\{\log(\sigma_t)\} - \frac{1}{2}\sum_{t=1}^{n}\left(\frac{e_t(\mathbf{x})}{\sigma_t}\right)^2$ | $\sigma_t$;$t\in\{1,\ldots,n\}$ | † |
| 13 | $\mathscr{L}(\mathbf{x}\|\tilde{\mathbf{Y}}) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log\left(\frac{\sigma_1^2}{(1-\phi^2)}\right) - \frac{1}{2}(1-\phi^2)\left(\frac{e_1(\mathbf{x})}{\sigma_1}\right)^2 - \sum_{t=2}^{n}\{\log(\sigma_t)\} - \frac{1}{2}\sum_{t=2}^{n}\left(\frac{(e_t(\mathbf{x}) - \phi e_{t-1}(\mathbf{x}))}{\sigma_t}\right)^2$ | $\sigma_t,\phi$; $t\in\{1,\ldots,n\}$ | † |
| 14 | $\mathscr{L}(\mathbf{x}\|\tilde{\mathbf{Y}}) \simeq n\log\left(\omega_\beta\frac{2\sigma_\xi}{(\xi+\xi^{-1})}\right) - \sum_{t=1}^{n}\{\log(\sigma_t)\} - c_\beta\sum_{t=1}^{n}\left\|a_{\xi,t}\right\|^{2/(1+\beta)} + (\lambda_{\mathrm{BC}}-1)\sum_{t=1}^{n}(\tilde{y}_t + K_{\mathrm{BC}})$ | $\sigma_0,\sigma_1,\beta,\xi,\mu_1,\phi,K_{\mathrm{BC}},\lambda_{\mathrm{BC}}$ | ‡§ |
| 15 | $\mathscr{L}\left(\mathbf{x}\middle\|\tilde{\mathbf{Y}}\right) = \sum_{j=1}^{\lfloor n/2\rfloor}\left\{\log(f_{\mathscr{F}}(\lambda_j,\mathbf{x}) + f_{\mathbf{E}}(\lambda_j,\Phi)) + \frac{g(\lambda_j)}{f_{\mathscr{F}}(\lambda_j,\mathbf{x})+f_{\mathbf{E}}(\lambda_j,\Phi)}\right\}$ | none | ¶ |
| 16 | $\mathscr{L}\left(\mathbf{x}\middle\|\tilde{\mathbf{Y}}\right) = -\sum_{t=1}^{n}\{\log(2\sigma_t)\} - \sum_{t=1}^{n}\left(\frac{\|e_t(\mathbf{x})\|}{\sigma_t}\right)$ | $\sigma_t$;$t\in\{1,\ldots,n\}$ | † |
| 17 | $\mathscr{L}(\mathbf{x}\|\tilde{\mathbf{Y}}) = \sum_{t=2}^{n}\Big\{\log(2c_2\Gamma((\nu+1)/2)\sqrt{\nu/(\nu-2)}) - \log((\kappa+\kappa^{-1})\Gamma(\nu/2)\sqrt{\pi\nu}\sqrt{(1-\phi^2)}\sigma_t) - ((\nu+1)/2)\log\left(1+(1/(\nu-2))\left(\frac{c_1+c_2\underline{\eta}_t}{\kappa^{\mathrm{sign}(c_1+c_2\underline{\eta}_t)}}\right)^2\right)\Big\}$ | $\sigma_a,\sigma_b,\sigma_c,\sigma_d,\phi,\nu,\kappa$ | §¥ |
| | **ABC – diagnostic model evaluation** | | |
| 21 | $\mathscr{L}(\mathbf{x}\|\tilde{\mathbf{Y}}) = -\frac{m}{2}\log(2\pi) - m\log(\varepsilon) - \frac{1}{2}\varepsilon^{-2}\sum_{j=1}^{m}\rho(S_j(\tilde{\mathbf{Y}}),S_j(\mathbf{Y}(\mathbf{x})))^2$ | none | £♯ |
| 22 | $\mathscr{L}(\mathbf{x}\|\tilde{\mathbf{Y}}) = \min_{j=1:m}(\varepsilon_j - \rho(S_j(\tilde{\mathbf{Y}}),S_j(\mathbf{Y}(\mathbf{x}))))$ | none | £♯ |
| | **GLUE – limits of acceptability** | | |
| 23 | $\mathscr{L}(\mathbf{x}\|\tilde{\mathbf{Y}}) = \sum_{j=1}^{m}\{I(\left\|S_j(\tilde{\mathbf{Y}}) - S_j(\mathbf{Y}(\mathbf{x}))\right\| \le \varepsilon_j)\}$ | none | ♣♯ |
| | **GLUE - informal likelihood functions** | | |
| 31 | $\mathscr{L}(\mathbf{x}\|\tilde{\mathbf{Y}}) = -G\log\{\mathrm{Var}[\mathbf{E}(\mathbf{x})]\}$ | none | ⋄ |
| 32 | $\mathscr{L}(\mathbf{x}\|\tilde{\mathbf{Y}}) = G\log\left(1 - \frac{\mathrm{Var}[\mathbf{E}(\mathbf{x})]}{\mathrm{Var}[\tilde{\mathbf{Y}}]}\right)$ | none | ⋄ |
| 33 | $\mathscr{L}(\mathbf{x}\|\tilde{\mathbf{Y}}) = -G\mathrm{Var}[\mathbf{E}(\mathbf{x})]$ | none | ⋄ |
| 34 | $\mathscr{L}(\mathbf{x}\|\tilde{\mathbf{Y}}) = -\log\{\sum_{t=1}^{n}\left\|e_t(\mathbf{x})\right\|\}$ | none | ⋄ |

† Measurement error, $\sigma_t$ defined in field Sigma of Meas_info or inferred jointly with $\mathbf{x}$ (see Appendix B).
‡ Measurement error defined as $\sigma_t = \sigma_0 + \sigma_1 y_t(\mathbf{x})$; Scalars $\omega_\beta$, $\sigma_\xi$ and $c_\beta$ derived from values of $\xi$ and $\beta$; $\boldsymbol{\phi} = \{\phi_1,\ldots,\phi_4\}$ stores coefficients autoregressive model of error residuals.
§ User is free to select exact formulation (depends on selection nuisance variables).
¶ Fourier frequencies, $\lambda_j$, spectral density function, $f_{\mathbf{E}}(\cdot)$ and periodogram, $g(\cdot)$ defined in Whittle (1953).
¥ Scalars $c_1$ and $c_2$ computed from $\nu>2$ and $\kappa>0$; $\underline{\eta}$ signifies $(n-1)$-vector of restandardized first-order decorrelated residuals; $\Gamma(\cdot)$ and sign denote the gamma and signum function, respectively.
£ ABC distance function, $\rho(S(\tilde{\mathbf{Y}}),S(\mathbf{Y}(\mathbf{x})))$ specified as inline function in field rho of structure options.
♯ $\varepsilon$ (scalar or $m$-vector) stored in field epsilon of options.
♣ Variable $I(a)$ returns one if $a$ is true, zero otherwise.
⋄ Shaping factor, $G$ defined in field GLUE of structure DREAMPar. Default setting of $G = 10$.

The generalized likelihood function of Schoups and Vrugt (2010) allows for bias correction, which is applied to the first or higher order filtered residuals prior to calculation of the likelihood. I refer to Schoups and Vrugt (2010) and Scharnagl et al. (2015) for an exact derivation and detailed analysis of likelihood functions 14 and 17, respectively, and Whittle (1953) for an introduction to likelihood 15. The ABC likelihood functions 21 and 22 are described and discussed in detail by Turner and Sederberg (2012) and Sadegh and Vrugt (2014), whereas the limits of acceptability function 23 is introduced and tested in Vrugt (2015a). The pseudo-likelihoods in 31, 32, 33 and 34 are explicated in the GLUE papers of Beven and coworkers (Beven and Binley, 1992; Freer et al., 1996; Beven and Freer, 2001; Beven, 2006). The derivation and explanation of the remaining likelihood functions, 11, 12, 13, and 16 can be found in introductory textbooks on time-series analysis and Bayesian inference.

Likelihood functions 14 and 17 extend the applicability of the other likelihood functions to situations where residual errors are correlated, heteroscedastic, and non-Gaussian with varying degrees of kurtosis and skewness. For instance, consider Fig. 15 which plots the density of the generalized likelihood function for different values of the skewness, $\beta$ and kurtosis, $\xi$. The density is symmetric for $\xi = 1$, positively skewed for $\xi > 1$ and negatively skewed for $\xi < 1$. If $\xi = 1$, then for $\beta = -1(0)[1]$ this density reduces to a uniform (Gaussian) [double-exponential] distribution.

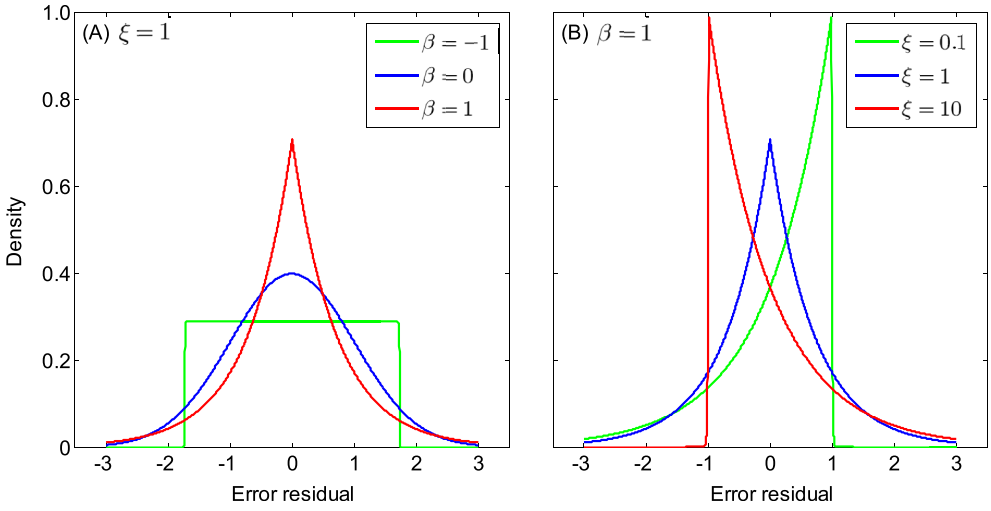


**Fig. 15.** Densities of the generalized likelihood function of Schoups and Vrugt (2010) for different values of the kurtosis ($\beta$) and skewness ($\xi$).

The Student likelihood function, 17, of Scharnagl et al. (2015) is designed in part to better mimic residual distributions with heavy tails (see Fig. 16).

By fixing some of the values of the nuisance variables the likelihood function can be simplified to a specific family of probability distributions.

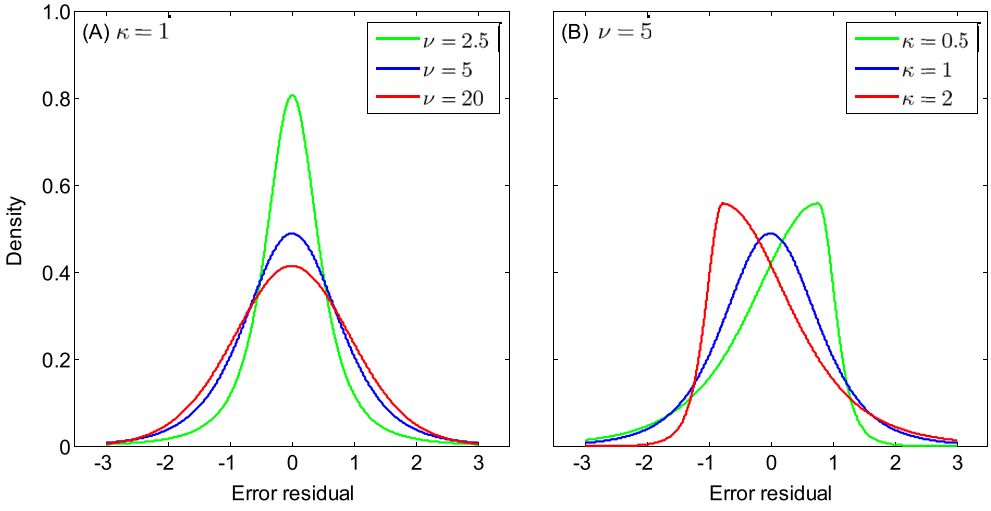


**Fig. 16.** Densities of the skewed Student likelihood function of Scharnagl et al. (2015) for different values of the skewness ($\kappa$) and kurtosis ($\xi$).

Table B2 summarizes several commonly used formal likelihood functions in hydrologic modeling applications and lists how likelihood function 14 can be reduced to these by making specific assumptions about the error residuals (see also Schoups and Vrugt (2010)).

I am now left to describe how to setup the joint inference of the model and nuisance parameters using the data stored in field Y of structure Meas_info. The MATLAB script on the next page provides an example for likelihood function 14 involving a model with $d = 3$ parameters, their names referred to in the excerpt as A, B and C.

**Table B2**
Relationship of likelihood functions used/proposed in the hydrologic literature and the likelihood function 14 of the DREAM package.

| Reference | Implementation using 14 |
|---|---|
| Standard least squares | $\phi_1 = 0$; $\phi_2 = 0$; $\phi_3 = 0$; $\phi_4 = 0$; $\sigma_1 = 0$; $\xi = 1$; $\beta = 0$ |
| Sorooshian and Dracup (1980): Equation (20) | $\phi_2 = 0$; $\phi_3 = 0$; $\phi_4 = 0$; $\sigma_1 = 0$; $\xi = 1$; $\beta = 0$ |
| Sorooshian and Dracup (1980): Equation (26) | $\phi_1 = 0$; $\phi_2 = 0$; $\phi_3 = 0$; $\phi_4 = 0$; $\xi = 1$; $\beta = 0$ |
| Kuczera (1983) | $\beta = 0$ |
| Bates and Campbell (2001) | $\beta = 0$ |
| Thiemann et al. (2001) | $\phi_1 = 0$; $\phi_2 = 0$; $\phi_3 = 0$; $\phi_4 = 0$ |

EXCERPT OF DREAM INPUT FILE FOR LIKELIHOOD FUNCTION 14

```
DREAMPar.d = 3;
DREAMPar.lik = 14; global LV
%% index:     1     2     3 |  d+1   d+2   d+3  d+4  d+5  d+6   d+7   d+8   d+9  d+10 d+11
%% name:      A     B     C | sig0  sig1  beta  xi   mu1  phi1  phi2  phi3  phi4  K  lambda
LV.fpar = [ nan   nan   nan   0.1     0     0    1    0    0     0     0     0    1  ]; % Default
parmin =  [ .5    10    -2      0     0    -1   0.1  0    0     0     0     0    0.1 ]; % Lower
parmax =  [ 10   1000   1      1     1     1   10   100  1     1     1     1    1  ]; % Upper
%% ranges from: Schoups and Vrugt, Water Resour. Res., 46 (10), W10531, pp. 1-17, 2010
LV.idx_vpar = [ 1:d d+1:d+3 d+6 d+10:d+11 ]; % Index parameters and latent variables for inference
Par_info.min = parmin(LV.idx_vpar);          % Lower bound parameters and latent variables
Par_info.max = parmax(LV.idx_vpar);          % Upper bound parameters and latent variables
```

Each nuisance variable in the DREAM package is assigned a unique label, hereafter also referred to as index or identifier. For example, the coefficients, $\sigma_0$, $\phi_1$, and $K_{BC}$ in likelihood function 14 have index, $d$+1, $d$+6 and $d$+9, respectively which equates to an index of 4, 9, and 12 for a model involving $d = 3$ parameters. Those indexes of the nuisance variables which are stored in the field idx_vpar of global variable **LV** will be subject to inference. These nuisance variables of the likelihood function augment the parameters. Nuisance variables not selected for inference are held constant at their default value declared by the user in field fpar of **LV**. Thus, in the MATLAB except above the nuisance variables {$\sigma_0$, $\sigma_1$, $\beta$, $\phi_1$, $K_{BC}$, $\lambda_{BC}$} are subject to inference, whereas the remaining coefficients, {$\xi$, $\mu_1$, $\phi_2$, $\phi_3$, $\phi_4$} of likelihood 14 will assume their respective default values of fpar.

A similar setup is used for likelihood function 17 (see below), except that the user has to separately define the values of the fields a, b, c, and d of structure **LV**. These values define the anchor points to be used with piecewise cubic hermite interpolation, details of which are given by Scharnagl et al. (2015).

All nuisance variables are selected for inference in this setup of likelihood function 17, except the skewness parameter $\kappa$ which is assumed to be unity (no skew).

For completeness, I also consider an example for likelihood function 13 involving joint inference of the model parameters, the measurement error of the data, $\sigma_t$; $t \in \{1,\ldots,n\}$, and the first order autoregressive parameter, $\phi$.

The user is free to determine the measurement error model of the data as long as this is specified as an inline function object. In the present example a heteroscedastic error model was assumed. If homoscedasticity of the measurement error is expected then the user can resort to another formulation of the inline function, for instance without the parameter a. Whatever mathematical formulation of the measurement data error model is used the ranges of its parameters should augment those of the parameters stored in field min and max of structure Par_info. These ranges are then followed by those of the first-order autoregressive coefficient, $\phi$. This order is consistent with that specified for likelihood function

EXCERPT OF DREAM INPUT FILE FOR LIKELIHOOD FUNCTION 17

```
DREAMPar.d = 3;
DREAMPar.lik = 17; global LV
%% Define anchor points used to estimate sigma of error residuals (from Scharnagl)
LV.a = 0.25; LV.b = 0.30; LV.c = 0.35; LV.d = 0.41;
%% index:     1     2     3 |  d+1   d+2   d+3   d+4  d+5  d+6  d+7
%% name:      A     B     C | siga  sigb  sigc  sigd  phi  nu  kappa
LV.fpar = [ nan   nan   nan  0.02  0.02  0.02  0.02   0    0    1  ]; % Default
parmin =  [ .5    10    -2     0     0     0     0    0    2   0.5 ]; % Lower
parmax =  [ 10   1000    1   0.05  0.05  0.05  0.05   1   1e3  2.0 ]; % Upper
%% ranges from: Scharnagl et al., Hydrol. Earth Syst. Sci. Discuss., 12, pp. 2155-2199, 2015
LV.idx_vpar = [ 1:d d+1:d+7 ];       % Index parameters and latent variables for inference
Par_info.min = parmin(LV.idx_vpar); % Lower bound parameters and latent variables
Par_info.max = parmax(LV.idx_vpar); % Upper bound parameters and latent variables
```

EXCERPT OF DREAM INPUT FILE FOR LIKELIHOOD FUNCTION 13

```
DREAMPar.d = 3;
DREAMPar.lik = 13;                                % Model output is simulation
%% index:          1    2    3   | d+1   d+2   d+3
%% name:           A    B    C   |  a     b    phi
Par_info.min = [ .5   10   -2      0     0    -1 ];  % Lower bound parameters, "a", "b" and phi
Par_info.max = [ 10 1000   1      1     1     1 ];  % Upper bound parameters, "a", "b" and phi
Meas_info.Sigma = inline('a * y + b');           % Define measurement error model
```

EXCERPT OF DREAM INPUT FILE FOR LIKELIHOOD FUNCTIONS 12 AND 16

```
DREAMPar.d = 3;
DREAMPar.lik = 12; % (DREAMPar.lik = 16)    % Model output is simulation
%% index:          1    2    3   | d+1   d+2
%% name:           A    B    C   |  a     b
Par_info.min = [ .5   10   -2      0     0  ];  % Lower bound parameters, "a", "b"
Par_info.max = [ 10 1000   1      1     1  ];  % Upper bound parameters, "a", "b"
Meas_info.Sigma = inline('a * y + b');       % Define measurement error model
```

13 in the third column of Table B1.

The last example considered herein involves the use of likelihood function 12 and 16 both of which share $\sigma_t$; $t \in \{1,\ldots,n\}$, the measurement error of the data. This variable can be specified by the user in field Sigma of structure Meas_info (see section 4.4), or alternatively be estimated along with the parameters of the target distribution. I follow this second approach in the script listed on the previous page.

This setup is similar to that of likelihood function 13, except without the use of $\phi$. Likelihood function 12 and 16 do not assume such first-order autoregressive correction of the error residuals.

## Appendix C

This Appendix presents the different model functions used in the six case studies presented in Section 5 of this paper. These functions (.m file) serve as a template for users to help define their own forward model in DREAM. All model functions have as input argument, **x** a row-vector with $d$ parameter values, and a single output argument which contains the likelihood, log-likelihood, a vector with simulated values or vector with summary statistics, respectively. A low-dash is used in the print out of each model script to denote the use of a standard built-in function of MATLAB.

*Case study I: one-dimensional mixture distribution*

The function mixture listed below uses the built-in normal probability density function of MATLAB, normpdf() to calculate the density (likelihood) of the mixture distribution for a given candidate point, **x**.

*Case study II: 100-dimensional t-distribution*

The function t_distribution listed below takes advantage of the built-in functions, log() and mvtpdf() of MATLAB to calculate the log-density (log-likelihood) of the multivariate t-probability density function with covariance matrix C and degrees of freedom, df.

The persistent declaration helps retain variables C and df in local memory after the first function call has been completed. This is computationally appealing, as it avoids having to recompute these variables in subsequent function calls.

*Case study III: dynamic simulation model*

The MATLAB function hydrus listed on the next page executes the HYDRUS-1D porous flow model and returns a vector with simulated soil moisture values.

The HYDRUS-1D model is an executable file encoded with instructions in Fortran, and consequently it is not possible to pass the $d$ parameter values, **x** in MATLAB directly to this stand-alone program. I therefore have to resort to an alternative, and somewhat less efficient approach. First, in the MATLAB script hydrus a file writing command is used to replace the current values of the parameters in the input files of HYDRUS-1D with those of the proposal, **x**. Then, HYDRUS-1D is executed from within MATLAB using the dos command and functionality. After this call has terminated, a load-command is used to read in MATLAB workspace the output files created by the HYDRUS-1D program. The simulated soil moisture values are then isolated from the data stored in MATLAB memory and returned to the main DREAM program. To maximize computational efficiency, the option persistent is used to retain the structure data in local memory after the first function call has been completed.

*Case study IV: likelihood-free inference*

The MATLAB function hmodel listed on the next page simulates the rainfall-runoff transformation for parameter values, **x** and returns four summary statistics (signatures) of watershed behavior.

SCRIPT OF FUNCTION MODEL CASE STUDY 1 : Univariate mixture distribution.

```
function [ L ] = mixture ( x );
% 1-dimensional mixture distribution

L = @(x) 1/6*normpdf(x,-8,1) + 5/6*normpdf(x,10,1)
```

SCRIPT OF FUNCTION MODEL CASE STUDY 2 : 100-dimensional Student distribution.

```
function [ log_L ] = t_distribution ( x );
% Multivariate Student distribution, t_{df}(x,C)

persistent df C                          % Retain variables df and C in local memory

if isempty(df),
    % Dimensionality of target distribution
    d = size(x,2);
    % Degrees of freedom
    df = 60;
    % Define correlation matrix
    A = 1/2*eye(d) + 1/2*ones(d);
    % Calculate covariance matrix
    for i = 1 : d,
        for j = 1 : d,
            C(i,j) = A(i,j) * sqrt(i * j);
        end
    end
end
log_L = log(mvtpdf(x,C,df));             % Compute log-density multivariate t-distribution
                                         % Note: More efficient if using log-formulation
```

SCRIPT OF FUNCTION MODEL CASE STUDY 3 : HYDRUS-1D model.

```
function [ soil_moisture ] = hydrus ( x )
% Executes HYDRUS-1D and returns vector with simulated soil moisture contents

persistent data                             % Retain structure data in memory after first call

if isempty(data),
    % Write level_01.dir (directory with files for HYDRUS-1D)
    fid = fopen('level_01.dir','w+'); fprintf(fid,'%s',pwd); fclose(fid);
    % Structure with observations and data to modify initial and boundary conditions
    data = Load_data; data.N = numel(data.hoy);
end

x(3:5) = 10.^x(3:5);                        % Back-transform parameters from log-space
data.initial(:,3) = x(7);                   % Change column of initial conditions
data.boundcon(:,7) = x(7);                  % Change column of boundary conditions

try     % Try running HYDRUS-1D model

    ModifySelectorIn(x);                    % Write current parameter values to file "SELECTOR.IN"
    ModifyProfileDat(data);                 % Write new initial condition to file "PROFILE.DAT"
    ModifyAtmosphIn(data);                  % Write new boundary conditions to file "ATMOSPH.IN"
    dos('HYDRUS-1D.EXE');                   % Run HYDRUS-1D in Fortran using dos command
    Y = ReadObsNodeOut;                     % Load simulation data from output files
    for i = 1:data.N,
        ind(i) = find(Y.hoy==data.hoy(i));  % Find "right" measurement times
    end
    soil_moisture = Y.water(ind);           % Get simulated soil moisture values

catch   % If HYDRUS-1D has crashed/failed to run completely

    soil_moisture = inf(data.N,1);          % Return "bad" moisture values

end
```

SCRIPT OF FUNCTION MODEL CASE STUDY 4 : Diagnostic model evaluation.

```
function [ S_sim ] = hmodel ( x );
% 7-parameter hmodel - Calculates summary metrics from simulated streamflow record

persistent func y0 options tout                         % Retain variables in local memory

if isempty(tout),
    % Load data and define forcing data
    daily_data = load('03451500.dly');                  % Load the French Broad data
    func.idx = [731:size(daily_data,1)]';               % First two years are warm-up
    func.P = daily_data(:,4); func.Ep = daily_data(:,5); % Define the PET and precipitation.
    % Initial conditions (state variables hmodel)
    y0 = 1e-5 * ones(5,1);                              % Initial conditions
    % Numerical solver
    options.InitialStep = 1;                            % initial time-step (d)
    options.MaxStep     = 1;                            % maximum time-step (d)
    options.MinStep     = 1e-5;                         % minimum time-step (d)
    options.RelTol      = 1e-3;                         % relative tolerance
    options.AbsTol      = 1e-3*ones(5,1);               % absolute tolerances (mm)
    options.Order       = 2;                            % 2nd order accurate method (Heun)
    % Run time
    tout = [ 0 : size(func.P,1) ];
end

% Assign parameters
func.Imax  = x(1);                                      % interception storage capacity     (mm)
func.Sumax = x(2);                                      % unsaturated zone storage capacity (mm)
func.Qsmax = x(3);                                      % maximum percolation rate          (mm/d)
func.aE    = x(4);                                      % evaporation coefficient           (-)
func.aF    = x(5);                                      % runoff coefficient                (-)
func.aS    = 1e-6;                                      % percolation coefficient           (-)
func.Kf    = x(6);                                      % fast-flow response time           (d)
func.Ks    = x(7);                                      % slow-flow response time           (d)

% Run the model and calculate summary metrics
y = crr_model(tout,y0,func,options);                    % Run model in C language
Y = diff(y(5,:))'; Y = Y(func.idx);                      % Now compute discharge
S_sim = Calc_metrics(Y,func.P(func.idx))';              % Now calculate simulated summary metrics
```

The source code of the hmodel is written in C and linked into a shared library using the MEX-compiler of MATLAB. This avoids file writing, and enables a direct passing of the parameter values, forcing data, and numerical solver settings to crr_model. A second-order time-variable integration method is used to solve the differential equations of the hmodel. The function Calc_-metrics computes the four summary metrics using as input arguments the simulated discharge record and observed precipitation data.

*Case study V: Bayesian model averaging*

The MATLAB function BMA_calc returns the log-likelihood of the BMA model for a given proposal, **x** consisting of weights and variances.

SCRIPT OF FUNCTION MODEL CASE STUDY 5 : Bayesian model averaging.

```
function [ log_L ] = BMA_calc ( x );
% This function calculates the log likelihood corresponding to the weights and sigma's

global BMA                                          % Request the BMA structure
L = 0;                                              % Set likelihood equivalent to zero
w = x(1:BMA.k);                                     % Unpack weights

switch lower(BMA.PDF)       % Now check which BMA model is used

    case {'normal'}             % Normal distribution with homoscedastic error variance

        if strcmp(lower(BMA.VAR),'single');         % One or multiple variances?
            sigma = x(BMA.k+1) * ones(1,BMA.k);
        elseif strcmp(lower(BMA.VAR),'multiple');
            sigma = x(BMA.k + 1 : numel(x));
        else
            error('do not know this option for variance treatment')
        end

        for i = 1:BMA.k,                            % Mixture model
            L = L + w(i)*exp(-1/2*((BMA.Ycal-BMA.Xcal(:,i))./sigma(i)).^2)./ ...
                (sqrt(2*pi).*sigma(i));             % Now calculate likelihood
        end

    case {'heteroscedastic'}    % Normal distribution with heteroscedastic error variance

        b = x(BMA.k+1);                             % Unpack variance parameter
        for i = 1:BMA.k,                            % Mixture model
            sigma = abs(b*BMA.Xcal(:,i));           % Calculate measurement error of data
            L = L + w(i)*exp(-1/2*((BMA.Ycal-BMA.Xcal(:,i))./sigma).^2)./ ...
                (sqrt(2*pi).*sigma);                % Calculate likelihood
        end

    case {'gamma'}              % Gamma distribution

        b = x(BMA.k+1:2*BMA.k); c = x(2*BMA.k+1);   % Unpack variables gamma distribution
        for i = 1:BMA.k,                            % Mixture model
            mu = abs(BMA.Xcal(:,i));                % Derive mean of gamma distribution
            var = abs(c + b(i) * BMA.Xcal(:,i));    % Derive variance of gamma distribution
            A = mu.^2./var; B = var./mu;            % Derive A and B of gamma distribution
            z = BMA.Ycal./B;                        % Compute help variable
            u = (A - 1).*log(z) - z - gammaln(A);   % Compute help variable
            L = L + w(i) * (exp(u)./B);             % Calculate likelihood
        end

end                         % End check which BMA model is used

L(L==0) = 1e-300;                                   % Replace zero likelihood with 1e-300
log_L = sum(log(L));                                % Compute log-likelihood of the BMA model
```

The log-likelihood of the BMA model is computed as the log of the sum of the likelihoods of each of ensemble member. In the example considered herein, the conditional distribution of each ensemble member is assumed to be Gaussian and with unknown variance.

*Case study VI: generalized likelihood uncertainty estimation*

The MATLAB function lotka_volterra solves the predator-prey system for the proposal, **x** and returns in a single vector, Y their simulated abundances as function of time.

A time-variable integration method, ode45 is used for numerical solution of the two coupled differential equations. The parameters are defined as additional state variables of the Lotka–Volterra model so that their values can be passed directly to the inline function within the built-on ODE solver.

*Case study VII: limits of acceptability*

The MATLAB function heat_flow returns the simulated time series of soil temperatures at 5, 10 and 15 cm depth in the soil profile.

SCRIPT OF FUNCTION MODEL CASE STUDY 6 : Predator-prey model.

```
function [ Y ] = lotka_volterra ( x );
% Lotka-Volterra model of population dynamics

persistent dydt y0 tout;                % Retain variables in memory after first call

if isempty(dydt),
    % Initial conditions (initial population of prey and predator population)
    y0 = [30 4];
    % Print time in years (start at t = 0 )
    tout = [ 0 : 1/12 : 20 ];
    % Define dydt --> the Lotka-Volterra partial differential equations
    % dydt = inline('[ alpha*y(1) - beta*y(1)*y(2) ; -gamma*y(2) + delta*y(1)*y(2) ]', ...
    %   't','y','alpha','beta','gamma','delta'); --> does not work with built-in ODE solver
    % Trick to use built-in ODE solver with inline function and additional parameters
    dydt = inline('[ y(3)*y(1)-y(4)*y(1)*y(2) ; -y(5)*y(2)+y(6)*y(1)*y(2) ; zeros(4,1) ]','t','y');
end

y = [ y0 x ]';                          % Append initial states to parameters and all are states
[ t , Y ] = ode45 ( dydt , tout , y );  % Use ode45 integration method to solve system equations
Y = Y(2:size(Y,1),1:2); Y = Y(:);       % Remove first row Y (initial state); return as one vector
```

SCRIPT OF FUNCTION MODEL CASE STUDY 7 : Heat flow equation.

```
function [ soil_temp ] = heat_flow ( x )
% Calculates the soil temperature at depth z and time t

persistent z omega t                          % Retain structure data in local memory

if isempty(z),
    % Define depth of simulation, omega and time of calculation
    z = [5 10 15]; omega = 2*pi/24; t = [1:1:48];
end

T_surf = x(1); A0 = x(2); phi = x(3); d = x(4); % Unpack parameter values

for i = 1:3
    soil_temp(:,i) = T_surf + A0*exp(-z(i)/d).*sin(omega*(t-phi) - z(i)/d);
end

soil_temp = soil_temp(:);                     % Return soil_temp as a single vector
```

## References


Ahrens, B., Reichstein, M., 2014. Reconciling 14C and minirhizotron-based estimates of fine-root turnover with survival functions. J. Plant Nutr. Soil Sci. 177, 287–296. http://dx.doi.org/10.1002/jpln.201300110.

Barthel, M., Hammerle, A., Sturm, P., Baur, T., Gentsch, L., Knohl, A., 2011. The diel imprint of leaf metabolism on the $\delta^{13}$ 13C signal of soil respiration under control and drought conditions. New Phytol. 192, 925–938. http://dx.doi.org/10.1111/j.1469-8137.2011.03848.x.

Bates, B.C., Campbell, E.P., 2001. A Markov chain Monte Carlo scheme for parameter estimation and inference in conceptual rainfall-runoff modeling. Water Resour. Res. 37 (4), 937–947.

Bauwens, L., de Backer, B., Dufays, A., 2011. Estimating and Forecasting Structural Breaks in Financial Time Series. Economics, Finance, Operations Research, Econometrics, and Statistics, Discussion paper, pp. 1–23.

Beven, K., 2006. A manifesto for the equifinality thesis. J. Hydrol. 320 (1), 18–36.

Beven, K.J., Binley, A.M., 1992. "The future of distributed models: model calibration and uncertainty prediction. Hydrol. Process. 6, 279–298.

Beven, K.J., Binley, A.M., 2014. GLUE: 20 years on. Hydrol. Process. 28, 5879–5918. http://dx.doi.org/10.1002/hyp.10082.

Beven, K., Freer, J., 2001. "Equifinality, data assimilation, and uncertainty estimation in mechanistic modelling of complex environmental systems using the GLUE methodology. J. Hydrol. 249 (1–4), 11–29. http://dx.doi.org/10.1016/S0022-1694(01)00421-8.

Bikowski, J., Huisman, J.A., Vrugt, J.A., Vereecken, H., van der Kruk, J., 2012. Inversion and sensitivity analysis of ground penetrating radar data with waveguide dispersion using deterministic and Markov chain Monte Carlo methods. Near Surf. Geophys. 10 (6), 641–652. http://dx.doi.org/10.3997/1873-0604.2012041.

Blasone, R.S., Vrugt, J.A., Madsen, H., Rosbjerg, D., Zyvoloski, G.A., Robinson, B.A., 2008. Generalized likelihood uncertainty estimation (GLUE) using adaptive Markov chain Monte Carlo sampling. Adv. Water Resour. 31, 630–648. http://dx.doi.org/10.1016/j.advwatres.2007.12.003.

Braakhekke, M.C., Wutzler, T., Beer, C., Kattge, J., Schrumpf, M., Ahrens, B., Schöning, I., Hoosbeek, M.R., Kruijt, B., Kabat, P., Reichstein, M., 2013. Modeling the vertical soil organic matter profile using Bayesian parameter estimation. Biogeosciences 10, 399–420. http://dx.doi.org/10.5194/bg-10-399-2013.

Brooks, S.P., Gelman, A., 1998. General methods for monitoring convergence of iterative simulations. J. Comput. Graph. Stat. 7, 434–455.

Chauvenet, W., 1960. Reprint of 1891, fifth ed.. A Manual of Spherical and Practical Astronomy V. II. 1863, pp. 474–566 Dover, N.Y.

Chib, S., 1995. Marginal likelihood from the Gibbs output. J. Am. Stat. Assoc. 90, 1313–1321.

Clark, M.P., Kavetski, D., 2010. Ancient numerical daemons of conceptual hydrological modeling: 1. Fidelity and efficiency of time stepping schemes. Water Resour. Res. 46, W10510. http://dx.doi.org/10.1029/2009WR008894.

Coelho, F.C., Codeço, C.T., Gomes, M.G.M., 2011. A Bayesian framework for parameter estimation in dynamical models. PLoS ONE 6 (5), e19616. http://dx.doi.org/10.1371/journal.pone.0019616.

Davis, K., Li, Y., 2011. Fast solution of geophysical inversion using adaptive mesh, space-filling curves and wavelet compression. Geophys. J. Int. 185 (1), 157–166.

DeCaluwe, S.C., Kienzle, P.A., Bhargava, P., Baker, A.M., Dura, J.A., 2014. Phase segregation of sulfonate groups in Nafion interface lamellae, quantified via neutron reflectometry fitting techniques for multi-layered structures. Soft Matter 10, 5763–6577. http://dx.doi.org/10.1039/C4SM00850B.

Dekker, S.C., Vrugt, J.A., Elkington, R.J., 2010. Significant variation in vegetation characteristics and dynamics from ecohydrologic optimality of net carbon profit. Ecohydrology 5, 1–18. http://dx.doi.org/10.1002/eco.177.

Dempster, A.P., Laird, N.M., Rubin, D.B., 1977. "Maximum likelihood from incomplete data via the EM algorithm. J. R. Stat. Soc. 39 (B), 1–39.

Denison, D.G.T., Holmes, C.C., Mallick, B.K., Smith, A.F.M., 2002. Bayesian Methods for Nonlinear Classification and Regression. John Wiley & Sons, Chicester.

Duan, Q., Schaake, J., Andréassian, V., Franks, S., Goteti, G., Gupta, H.V., Gusev, Y.M., Habets, F., Hall, A., Hay, L., Hogue, T., Huang, M., Leavesley, G., Liang, X.,

Nasonova, O.N., Noilhan, J., Oudin, L., Sorooshian, S., Wagener, T., Wood, E.F., 2006. "Model parameter estimation Experiment (MOPEX): an overview of science strategy and major results from the second and third workshops. J. Hydrol. 320, 3–17. http://dx.doi.org/10.1016/j.jhydrol.2005.07.031.

Dumont, B., Leemans, V., Mansouri, M., Bodson, B., Destain, J.-P., Destain, M.-F., 2014. "Parameter identification of the STICS crop model, using an accelerated formal MCMC approach. Environ. Model. Softw. 52, 121–135.

Dura, J.A., Kelly, S.T., Kienzle, P.A., Her, J.-H., Udovic, T.J., Majkrzak, C.F., Chung, C.-J., Clemens, B.M., 2011. "Porous Mg formation upon dehydrogenation of $MgH_2$ thin films. J. Appl. Phys. 109, 093501.

Evin, G., Kavetski, D., Thyer, M., Kuczera, G., 2013. "Pitfalls and improvements in the joint inference of heteroscedasticity and autocorrelation in hydrological model calibration. Water Resour. Res. 49, 4518–4524. http://dx.doi.org/10.1002/wrcr.20284.

Freer, J., Beven, K.J., Ambroise, B., 1996. "Bayesian estimation of uncertainty in runoff prediction and the value of data: an application of the GLUE approach. Water Resour. Res. 32, 2161–2173.

Gelman, A., Meng, X.L., 1998. Simulating normalizing constants: from importance sampling to bridge sampling to path sampling. Stat. Sci. 13 (2), 163–185.

Gelman, A.G., Rubin, D.B., 1992. "Inference from iterative simulation using multiple sequences. Stat. Sci. 7, 457–472.

Gelman, A.G., Shirley, K., 2009. Inference from simulations and monitoring convergence. In: Meng, X.L., Gelman, A., Jones, G. (Eds.), The Handbook of Markov Chain Monte Carlo. Chapman & Hall/CRC Press, pp. 163–174.

Gelman, A.G., Roberts, G.O., Gilks, W.R., 1996. Bayesian Statistics. Oxford University Press, pp. 599–608.

Gentsch, L., Hammerle, A., Sturm, P., Ogée, J., Wingate, L., Siegwolf, R., Plüss, P., Baur, T., Buchmann, N., Knohl, A., 2014. Carbon isotope discrimination during branch photosynthesis of Fagus sylvatica: a Bayesian modeling approach. Plant Cell Environ. 37, 1516–1535. http://dx.doi.org/10.1111/pce.12262.

Geweke, J., 1992. Evaluating the accuracy of sampling-based approaches to the calculation of posterior moments. In: Bernardo, J.M., Berger, J.O., Dawid, A.P., Smith, A.F.M. (Eds.), Bayesian Statistics 4Oxford University Press, pp. 169–193.

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

Gilks, W.R., Roberts, G.O., George, E.I., 1994. "Adaptive direction sampling. Statistician 43, 179–189.

Green, P.J., 1995. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. Biometrika 82, 711–732.

Grubbs, F.E., 1950. "Sample criteria for testing outlying observations. Ann. Math. Stat. 21 (1), 27–58. http://dx.doi.org/10.1214/aoms/1177729885.

Gupta, H.V., Wagener, T., Liu, Y., 2008. Reconciling theory with observations: elements of a diagnostic approach to model evaluation. Hydrol. Process. 22 (18), 3802–3813.

Haario, H., Saksman, E., Tamminen, J., 1999. "Adaptive proposal distribution for random walk Metropolis algorithm. Comput. Stat. 14, 375–395.

Haario, H., Saksman, E., Tamminen, J., 2001. "An adaptive Metropolis algorithm. Bernoulli 7, 223–242.

Haario, H., Saksman, E., Tamminen, J., 2005. Componentwise adaptation for high dimensional MCMC. Stat. Comput. 20, 265–274.

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

Hansen, T.M., Cordua, K.S., Mosegaard, K., 2012. Inverse problems with non-trivial priors: efficient solution through sequential Gibbs sampling. Comput. Geosci. 16, 593–611.

Hastings, H., 1970. "Monte Carlo sampling methods using Markov chains and their applications. Biometrika 57, 97–109.

Hinnell, A.W., Ferré, T.P.A., Vrugt, J.A., Moysey, S., Huisman, J.A., Kowalsky, M.B., 2010. "Improved extraction of hydrologic information from geophysical data through coupled hydrogeophysical inversion. Water Resour. Res. 46, W00D40. http://dx.doi.org/10.1029/2008WR007060.

Horowitz, V.R., Aleman, B.J., Christle, D.J., Cleland, A.N., Awschalom, D.D., 2012. "Electron spin resonance of nitrogen-vacancy centers in optically trapped nanodiamonds. Proc. Natl. Acad. U. S. A. 109 (34), 13493–13497. http://dx.doi.org/10.1073/pnas.1211311109.

Iizumi, T., Tanaka, Y., Sakurai, G., Ishigooka, Y., Yokozawa, M., 2014. "Dependency of parameter values of a crop model on the spatial scale of simulation. J. Adv. Model. Earth Syst. 06 http://dx.doi.org/10.1002/2014MS000311.

Jafarpour, B., 2011. "Wavelet reconstruction of geologic facies from nonlinear dynamic flow measurements. IEEE Trans. Geosci. Remote Sens. 49 (5), 1520–1535.

Jafarpour, B., Goyal, V.K., McLaughlin, D.B., Freeman, W.T., 2009. "Transform-domain sparsity regularization for inverse problems in geosciences. Geophysics 74 (5), R69–R83.

Jafarpour, B., Goyal, V.K., McLaughlin, D.B., Freeman, W.T., 2010. "Compressed history matching: exploiting transform-domain sparsity for regularization of nonlinear dynamic data integration problems. Math. Geosci. 42 (1), 1–27.

Joseph, J.F., Guillaume, J.H.A., 2013. Using a parallelized MCMC algorithm in R to identify appropriate likelihood functions for SWAT. Environ. Model. Softw. 46, 292–298.

Kass, R.E., Raftery, A.E., 1995. Bayes factors. J. Am. Stat. Assoc. 90, 773–795.

Kavetski, D., Kuczera, G., Franks, S.W., 2006a. Bayesian analysis of input uncertainty in hydrological modeling: 1. Theory. Water Resour. Res. 42 (3), W03407. http://dx.doi.org/10.1029/2005WR004368.

Kavetski, D., Kuczera, G., Franks, S.W., 2006b. Bayesian analysis of input uncertainty in hydrological modeling: 2. Application. Water Resour. Res. 42 (3), W03408. http://dx.doi.org/10.1029/2005WR004376.

Keating, E.H., Doherty, J., Vrugt, J.A., Kang, Q., 2010. Optimization and uncertainty assessment of strongly nonlinear groundwater models with high parameter dimensionality. Water Resour. Res. 46, W10517. http://dx.doi.org/10.1029/2009WR008584.

Kikuchi, C.P., Ferré, T.P.A., Vrugt, J.A., 2015. "Discrimination-inference for measurement selection. Water Resour. Res. XX http://dx.doi.org/10.1002/wrcr.XXXX. XX–XX.

Kirby, B.J., Rahman, M.T., Dumas, R.K., Davies, J.E., Lai, C.H., Liu, K., 2013. "Depth-resolved magnetization reversal in nanoporous perpendicular anisotropy multilayers. J. Appl. Phys. 113, 033909. http://dx.doi.org/10.1063/1.4775819.

Kow, W.Y., Khong, W.L., Chin, Y.K., Saad, I., Teo, K.T.K., 2012. "Enhancement of Markov chain monte Carlo convergence speed in vehicle tracking using genetic operator. In: 2012 Fourth International Conference on Computational Intelligence, Modeling and Simulation (CIMSiM), pp. 270–275. http://dx.doi.org/10.1109/CIMSim.2012.61.

Krayer, L., Lau, J.W., Kirby, B.J., 2014. "Structural and magnetic etch damage in CoFeB. J. Appl. Phys. 115, 17B751.

Kuczera, G., 1983. "Improved parameter inference in catchment models, 1. Evaluating parameter uncertainty. Water Resour. Res. 19 (5), 1151–1162. http://dx.doi.org/10.1029/WR019i005p01151.

Kuczera, G., Kavetski, D., Franks, S., Thyer, M., 2006. "Towards a Bayesian total error analysis of conceptual rainfall-runoff models: characterising model error using storm-dependent parameters. J. Hydrol. 331 (1), 161–177.

Kuczera, G., Kavetski, D., Renard, B., Thyer, M., 2010. "A limited memory acceleration strategy for MCMC sampling in hierarchical Bayesian calibration of hydrological models. Water Resour. Res. 46, W07602. http://dx.doi.org/10.1029/2009WR008985.

Laloy, E., Vrugt, J.A., 2012. "High-dimensional posterior exploration of hydrologic models using multiple-try $DREAM_{(ZS)}$ and high-performance computing. Water Resour. Res. 48, W01526. http://dx.doi.org/10.1029/2011WR010608.

Laloy, E., Linde, N., Vrugt, J.A., 2012. "Mass conservative three-dimensional water tracer distribution from Markov chain Monte Carlo inversion of time-lapse ground-penetrating radar data. Water Resour. Res. 48, W07510. http://dx.doi.org/10.1029/2011WR011238.

Laloy, E., Rogiers, B., Vrugt, J.A., Jacques, D., Mallants, D., 2013. "Efficient posterior exploration of a high-dimensional groundwater model from two-stage Markov chain Monte Carlo simulation and polynomial chaos expansion. Water Resour. Res. 49 (5), 2664–2682. http://dx.doi.org/10.1002/wrcr.20226.

Laloy, E., Linde, N., Jacques, D., Vrugt, J.A., 2015. "Probabilistic inference of multi-Gaussian fields from indirect hydrological data using circulant embedding and dimensionality reduction. Water Resour. Res. 51, 4224–4243. http://dx.doi.org/10.1002/2014WR016395.

Laplace, P.S., 1774. "Mémoire sur la probabilité des causes par les évènements. Mém. l'Acad. R. Sci. Present. Divers Savan 6, 621–656.

Leventhal, G.E., Günthard, H.F., Bonhoeffer, S., Stadler, T., 2013. "Using an epidemiological model for phylogenetic inference reveals density dependence in HIV transmission. Mol. Biol. Evol. 31 (1), 6–17. http://dx.doi.org/10.1093/molbev/mst172.

Lewis, S.M., Raftery, A.E., 1997. "Estimating Bayes factors via posterior simulation with the Laplace-Metropolis estimator. J. Am. Stat. Assoc. 92 (438), 648–655.

Linde, N., Vrugt, J.A., 2013. "Distributed soil moisture from crosshole ground-penetrating radar travel times using stochastic inversion. Vadose Zone J. 12 (1) http://dx.doi.org/10.2136/vzj2012.0101.

Lise, J., 2013. "On the job search and precautionary savings. Rev. Econ. Stud. 80 (3), 1086–1113. http://dx.doi.org/10.1093/restud/rds042.

Lise, J., Meghir, C., Robin, J.-M., 2012. Mismatch, Sorting and Wage Dynamics. Working paper, 18719. National Bureau of Economic Research, pp. 1–43. http://www.nber.org/papers/w18719.

Liu, J.S., Liang, F., Wong, W.H., 2000. "The multiple-try method and local optimization in metropolis sampling. J. Am. Stat. Assoc. 95 (449), 121–134. http://dx.doi.org/10.2307/2669532.

Lochbühler, T., Vrugt, J.A., Sadegh, M., Linde, N., 2015. Summary statistics from training images as prior information in probabilistic inversion. Geophys. J. Int. 201, 157–171. http://dx.doi.org/10.1093/gji/ggv008.

Lochbühler, T., Breen, S.J., Detwiler, R.L., Vrugt, J.A., Linde, N., 2014. "Probabilistic electrical resistivity tomography for a $CO_2$ sequestration analog. J. Appl. Geophys. 107, 80–92. http://dx.doi.org/10.1016/j.jappgeo.2014.05.013.

Lu, D., Ye, M., Hill, M.C., Poeter, E.P., Curtis, G.P., 2014. "A computer program for uncertainty analysis integrating regression and Bayesian methods. Environ. Model. Softw. 60, 45–56.

Malama, B., Kuhlman, K.L., James, S.C., 2013. "Core-scale solute transport model selection using Monte Carlo analysis. Water Resour. Res. 49, 3133–3147. http://dx.doi.org/10.1002/wrcr.20273.

Mari, L., Bertuzzo, E., Righetto, L., Casagrandi, R., Gatto, M., Rodriguez-Iturbe, I., Rinaldo, A., 2011. "Modeling cholera epidemics: the role of waterways, human mobility and sanitation. J. R. Soc. Interface 9 (67), 376–388.

McKay, M.D., Beckman, R.J., Conover, W.J., 1979. "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 21 (2), 239–245. http://dx.doi.org/10.2307/1268522.

Meng, X.L., Wong, W.H., 1996. "Simulating ratios of normalizing constants via a simple identity: a theoretical exploration. Stat. Sin. 6, 831–860.

Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E., 1953. "Equation of state calculations by fast computing machines. J. Chem. Phys. 21,

1087–1092.
Minasny, B., Vrugt, J.A., McBratney, A.B., 2011. "Confronting uncertainty in model-based geostatistics using Markov chain Monte Carlo simulation. Geoderma 163, 150–622. http://dx.doi.org/10.1016/j.geoderma.2011.03.011.
Montanari, A., Toth, E., 2007. "Calibration of hydrological models in the spectral domain: an opportunity for scarcely gauged basins? Water Resour. Res. 43, W05434. http://dx.doi.org/10.1029/2006WR005184.
Mosegaard, K., Tarantola, A., 1995. "Monte Carlo sampling of solutions to inverse problems. J. Geophys. Res. 100 (B7), 12431–12447.
Nott, D.J., Marshall, L., Brown, J., 2012. "Generalized likelihood uncertainty estimation (GLUE) and approximate Bayesian computation: what's the connection? Water Resour. Res. 48 (12) http://dx.doi.org/10.1029/2011WR011128.
Oware, E., Moysey, S., Khan, T., 2013. "Physically based regularization of hydrogeophysical inverse problems for improved imaging of process-driven systems. Water Resour. Res. 49 (10), 6238–6247.
Owejan, J.E., Owejan, J.P., DeCaluwe, S.C., Dura, J.A., 2012. "Solid electrolyte interphase in Li-ion batteries: evolving structures measured in situ by neutron reflectometry. Chem. Mater. 24, 2133–2140.
Owen, A.B., Tribble, S.D., 2005. "A quasi-Monte Carlo metropolis algorithm. Proc. Natl. Acad. Sci. U. S. A. 102, 8844–8849.
Partridge, D.G., Vrugt, J.A., Tunved, P., Ekman, A.M.L., Gorea, D., Sorooshian, A., 2011. "Inverse modeling of cloud-aerosol interactions – part I: detailed response surface analysis. Atmos. Chem. Phys. 11, 4749–4806. http://dx.doi.org/10.5194/acpd-11-4749-2011.
Partridge, D.G., Vrugt, J.A., Tunved, P., Ekman, A.M.L., Struthers, H., Sorooshian, A., 2012. "Inverse modeling of cloud-aerosol interactions - Part II: sensitivity tests on liquid phase clouds using Markov chain Monte Carlo simulation approach. Atmos. Chem. Phys. 12, 2823–2847. http://dx.doi.org/10.5194/acp-12-2823-2012.
Peirce, B., 1852. "Criterion for the rejection of doubtful observations. Astron. J. II 45.
Price, K.V., Storn, R.M., Lampinen, J.A., 2005. Differential Evolution, a Practical Approach to Global Optimization. Springer, Berlin.
Radu, V.C., Rosenthal, J., Yang, C., 2009. "Learn from the thy neighbor: parallel-chain and regional adaptive MCMC. J. Am. Stat. Assoc. 104 (488), 1454–1466.
Raftery, A.E., Lewis, S.M., 1992. "One long run with diagnostics: Implementation strategies for Markov chain Monte Carlo. Stat. Sci. 7, 493–497.
Raftery, A.E., Gneiting, T., Balabdaoui, F., Polakowski, M., 2005. "Using Bayesian model averaging to calibrate forecast ensembles. Mon. Weather Rev. 133, 1155–1174.
Renard, B., Kavetski, D., Kuczera, G., Thyer, M., Franks, S.W., 2010. "Understanding predictive uncertainty in hydrologic modeling: the challenge of identifying input and structural errors. Water Resour. Res. 46, W05521. http://dx.doi.org/10.1029/2009WR008328.
Renard, B., Kavetski, D., Leblois, E., Thyer, M., Kuczera, G., Franks, S.W., 2011. "Toward a reliable decomposition of predictive uncertainty in hydrological modeling: characterizing rainfall errors using conditional simulation. Water Resour. Res. 47 (11), W11516. http://dx.doi.org/10.1029/2011WR010643.
Rinaldo, A., Bertuzzo, E., Mari, L., Righetto, L., Blokesch, M., Gatto, M., Casagrandi, R., Murray, M., Vesenbeckh, S.M., Rodriguez-Iturbe, I., 2012. "Reassessment of the 2010–2011 Haiti cholera outbreak and rainfall-driven multiseason projections. Proc. Natl. Acad. U. S. A. 109 (17), 6602–6607.
Rings, J., Vrugt, J.A., Schoups, G., Huisman, J.A., Vereecken, H., 2012. "Bayesian model averaging using particle filtering and Gaussian mixture modeling: theory, concepts, and simulation experiments. Water Resour. Res. 48, W05520. http://dx.doi.org/10.1029/2011WR011607.
Roberts, C.P., Casella, G., 2004. Monte Carlo Statistical Methods, second ed. Springer, New York.
Roberts, G.O., Gilks, W.R., 1994. "Convergence of adaptive direction sampling. J. Multivar. Anal. 49, 287–298.
Roberts, G.O., Rosenthal, J.S., 2007. "Coupling and ergodicity of adaptive Markov chain Monte Carlo algorithms. J. Appl. Probab. 44, 458–475.
Roberts, G.O., Gelman, A., Gilks, W.R., 1997. "Weak convergence and optimal scaling of random walk Metropolis algorithms. Ann. Appl. Probab. 7, 110–120.
Rosas-Carbajal, M., Linde, N., Kalscheuer, T., Vrugt, J.A., 2014. "Two-dimensional probabilistic inversion of plane-wave electromagnetic data: methodology, model constraints and joint inversion with electrical resistivity data. Geophys. J. Int. 196 (3), 1508–1524. http://dx.doi.org/10.1093/gji/ggt482.
Ruggeri, P., Irving, J., Holliger, K., 2015. "Systematic evaluation of sequential geostatistical resampling within MCMC for posterior sampling of near-surface geophysical inverse problems. Geophys. J. Int. 202, 961–975. http://dx.doi.org/10.1093/gji/ggv196.
Sadegh, M., Vrugt, J.A., 2013. "Approximate Bayesian Computation in hydrologic modeling: equifinality of formal and informal approaches. Hydrol. Earth Syst. Sci. - Discuss. 10, 4739–4797. http://dx.doi.org/10.5194/hessd-10-4739-2013, 2013.
Sadegh, M., Vrugt, J.A., 2014. "Approximate Bayesian computation using Markov chain monte Carlo simulation: DREAM$_{(ABC)}$. Water Resour. Res. 50 http://dx.doi.org/10.1002/2014WR015386.
Sadegh, M., Vrugt, J.A., Gupta, H.V., 2015a. "The soil water characteristic as new class of closed-form parametric expressions for the flow duration curve. Water Resour. Res. XX http://dx.doi.org/10.1002/2014WRXXXX.
Sadegh, M., Vrugt, J.A., Xu, C., Volpi, E., 2015b. "The stationarity paradigm revisited: hypothesis testing using diagnostics, summary metrics, and DREAM$_{(ABC)}$. Water Resour. Res. XX. http://onlinelibrary.wiley.com/doi/10.1002/2014WR016805/ful, (in press), Published online.
Schaap, M.G., Leij, F.J., van Genuchten, M.Th, 1998. "Neural network analysis for hierarchical prediction of soil water retention and saturated hydraulic conductivity. Soil Sci. Soc. Am. J. 62, 847–855.
Schaap, M.G., Leij, F.J., van Genuchten, M.Th., 2001. "Rosetta: a computer program for estimating soil hydraulic parameters with hierarchical pedotransfer functions. J. Hydrol. 251, 163–176.
Scharnagl, B., Vrugt, J.A., Vereecken, H., Herbst, M., 2010. "Information content of incubation experiments for inverse estimation of pools in the Rothamsted carbon model: a Bayesian perspective. Biogeosciences 7, 763–776.
Scharnagl, B., Vrugt, J.A., Vereecken, H., Herbst, M., 2011. "Bayesian inverse modeling of soil water dynamics at the field scale: using prior information about the soil hydraulic properties. Hydrol. Earth Syst. Sci. 15, 3043–3059. http://dx.doi.org/10.5194/hess-15-3043-2011.
Scharnagl, B., Iden, S.C., Durner, W., Vereecken, H., Herbst, M., 2015. "Inverse modelling of in situ soil water dynamics: accounting for heteroscedastic, autocorrelated, and non-Gaussian distributed residuals. Hydrol. Earth Syst. Sci. Discuss. 12, 2155–2199.
Schoups, G., Vrugt, J.A., 2010. "A formal likelihood function for parameter and predictive inference of hydrologic models with correlated, heteroscedastic and non-gaussian errors. Water Resour. Res. 46, W10531. http://dx.doi.org/10.1029/2009WR008933.
Schoups, G., Vrugt, J.A., Fenicia, F., van de Giesen, N.C., 2010. "Corruption of accuracy and efficiency of Markov chain Monte Carlo simulation by inaccurate numerical implementation of conceptual hydrologic models. Water Resour. Res. 46, W10530. http://dx.doi.org/10.1029/2009WR008648.
Shafii, M., Tolson, B., Matott, L.S., 2014. Uncertainty-based multi-criteria calibration of rainfall-runoff models: a comparative study. Stoch. Environ. Res. Risk Assess. 28 (6), 1493–1510.
Šimůnek, J., Šejna, M., van Genuchten, M. Th., 1998. The HYDRUS-1D software package for simulating the one-dimensional movement of water, heat, and multiple solutes in variably-saturated Media. In: V1.0, IGWMC-TPS-70, International Ground Water Modeling Center. Colorado School of Mines, Golden, CO, p. 186.
Smith, T., Sharma, A., Marshall, L., Mehrotra, R., Sisson, S., 2010. "Development of a formal likelihood function for improved Bayesian inference of ephemeral catchments. Water Resour. Res. 46, W12551. http://dx.doi.org/10.1029/2010WR009514.
Sorooshian, S., Dracup, J.A., 1980. Stochastic parameter estimation procedures for hydrologic rainfall-runoff models: correlated and heteroscedastic error cases. Water Resour. Res. 16 (2), 430–442.
Starrfelt, J., Kaste, Ø, 2014. Bayesian uncertainty assessment of a semi-distributed integrated catchment model of phosphorus transport. Environ. Sci. Process. Impacts 16, 1578–1587. http://dx.doi.org/10.1039/C3EM00619K.
Storn, R., Price, K., 1997. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. J. Glob. Optim. 11, 341–359.
Sun, X.-L., Wu, S.-C., Wang, H.-L., Zhao, Yu-G., Zhang, G.-L., Man, Y.B., Wong, M.H., 2013. Dealing with spatial outliers and mapping uncertainty for evaluating the effects of urbanization on soil: a case study of soil pH and particle fractions in Hong Kong. Geoderma 195–196, 220–233.
Tarasevich, B.J., Perez-Salas, U., Masic, D.L., Philo, J., Kienzle, P., Krueger, S., Majkrzak, C.F., Gray, J.L., Shaw, W.J., 2013. Neutron reflectometry studies of the adsorbed structure of the amelogenin, LRAP. J. Phys. Chem. B 117 (11), 3098–3109. http://dx.doi.org/10.1021/jp311936j.
ter Braak, C.J.F., 2006. A Markov chain Monte Carlo version of the genetic algorithm differential evolution: easy Bayesian computing for real parameter spaces. Stat. Comput. 16, 239–249.
ter Braak, C.J.F., Vrugt, J.A., 2008. Differential evolution Markov chain with snooker updater and fewer chains. Stat. Comput. 18 (4), 435–446. http://dx.doi.org/10.1007/s11222-008-9104-9.
Thiemann, M., Trosset, M., Gupta, H., Sorooshian, S., 2001. Bayesian recursive parameter estimation for hydrologic models. Water Resour. Res. 37 (10), 2521–2535. http://dx.doi.org/10.1029/2000WR900405.
Toyli, D.M., Christle, D.J., Alkauskas, A., Buckley, B.B., van de Walle, C.G., Awschalom, D.D., 2012. Measurement and control of single nitrogen-vacancy center spins above 600 K. Phys. Rev. X 2, 031001. http://dx.doi.org/10.1103/PhysRevX.2.031001.
Turner, B.M., Sederberg, P.B., 2012. Approximate Bayesian computation with differential evolution. J. Math. Psychol. 56 (5), 375–385. http://dx.doi.org/10.1016/j.jmp.2012.06.004.
Upton, G., Cook, I., 1996. Understanding Statistics. Oxford University Press, p. 55.
van Genuchten, M.Th., 1980. A closed-form equation for predicting the hydraulic conductivity of unsaturated soils. Soil Sci. Soc. Am. J. 44 (5), 892–898. http://dx.doi.org/10.2136/sssaj1980.03615995004400050002x.
Volpi, E., Vrugt, J.A., Schoups, G., 2015. Bayesian model selection with DREAM: multi-dimensional integration of the evidence. Water Resour. Res. XX http://dx.doi.org/10.1002/2014WRXXXX.
Vrugt, J.A., 2015. To be coherently incoherent: GLUE done with DREAM but much more accurate and efficient. J. Hydrol. XX doi:XX/XX.XX.
Vrugt, J.A., 2015. The scientific method, Bayes theorem, diagnostic model evaluation, and summary metrics as prior information. Hydrol. Process. (submitted for publication).
Vrugt, J.A., Robinson, B.A., 2007a. Treatment of uncertainty using ensemble methods: comparison of sequential data assimilation and Bayesian model averaging. Water Resour. Res. 43, W01411. http://dx.doi.org/10.1029/2005WR004838.

Vrugt, J.A., Sadegh, M., 2013. Toward diagnostic model calibration and evaluation: approximate Bayesian computation. Water Resour. Res. 49 http://dx.doi.org/10.1002/wrcr.20354.

Vrugt, J.A., Gupta, H.V., Bouten, W., Sorooshian, S., 2003. A shuffled complex evolution metropolis algorithm for optimization and uncertainty assessment of hydrologic model parameters. Water Resour. Res. 39 (8), 1201. http://dx.doi.org/10.1029/2002WR001642.

Vrugt, J.A., Diks, C.G.H., Bouten, W., Gupta, H.V., Verstraten, J.M., 2005. Improved treatment of uncertainty in hydrologic modeling: combining the strengths of global optimization and data assimilation. Water Resour. Res. 41 (1), W01017. http://dx.doi.org/10.1029/2004WR003059.

Vrugt, J.A., ter Braak, C.J.F., Clark, M.P., Hyman, J.M., Robinson, B.A., 2008a. Treatment of input uncertainty in hydrologic modeling: doing hydrology backward with Markov chain Monte Carlo simulation. Water Resour. Res. 44, W00B09. http://dx.doi.org/10.1029/2007WR006720.

Vrugt, J.A., Diks, C.G.H., Clark, M.P., 2008c. Ensemble Bayesian model averaging using Markov chain Monte Carlo sampling. Environ. Fluid Mech. 8 (5–6), 579–595. http://dx.doi.org/10.1007/s10652-008-9106-3.

Vrugt, J.A., ter Braak, C.J.F., Diks, C.G.H., Higdon, D., Robinson, B.A., Hyman, J.M., 2009a. Accelerating Markov chain Monte Carlo simulation by differential evolution with self-adaptive randomized subspace sampling. Int. J. Nonlinear Sci. Numer. Simul. 10 (3), 273–290.

Vrugt, J.A., ter Braak, C.J.F., Gupta, H.V., Robinson, B.A., 2009b. Equifinality of formal (DREAM) and informal (GLUE) Bayesian approaches in hydrologic modeling. Stoch. Environ. Res. Risk Assess. 23 (7), 1011–1026. http://dx.doi.org/10.1007/s00477-008-0274-y.

Vrugt, J.A., ter Braak, C.J.F., Diks, C.G.H., Schoups, G., 2013. Advancing hydrologic data assimilation using particle Markov chain Monte Carlo simulation: theory, concepts and applications. Adv. Water Resour. 51, 457–478. http://dx.doi.org/10.1016/j.advwatres.2012.04.002. Anniversary Issue – 35 Years.

Whittle, P., 1953. Estimation and information in stationary time series. Ark. Mat. 2, 423–434.

Wikle, C.K., Hooten, M.B., 2010. A general science-based framework for dynamic spatio-temporal models. Test 19, 417–451. http://dx.doi.org/10.1007/s11749-010-0209-z.

Wöhling, T., Vrugt, J.A., 2011. "Multi-response multi-layer vadose zone model calibration using Markov chain Monte Carlo simulation and field water retention data. Water Resour. Res. 47, W04510. http://dx.doi.org/10.1029/2010WR009265.

Yale, C.G., Buckley, B.B., Christle, D.J., Burkard, G., Heremans, F.J., Bassett, L.C., Awschalom, D.D., 2013. "All-optical control of a solid-state spin using coherent dark states. Proc. Natl. Acad. Sci. U. S. A. 110 (19), 7595–7600. http://dx.doi.org/10.1073/pnas.1305920110.

Zaoli, S., Giometto, A., Formentin, M., Azaele, S., Rinaldo, A., Maritan, A., 2014. "Phenomenological modeling of the motility of self-propelled microorganisms. arXiv, 1407.1762. .

Zilliox, C., Gosselin, Frédéric, 2014. "Tree species diversity and abundance as indicators of understory diversity in French mountain forests: Variations of the relationship in geographical and ecological space. For. Ecol. Manag. 321 (1), 105–116.