Comment on the AM method

Marko Laine, Jouni Susiluoto, Johanna Tamminen, Heikki Haario
Finnish Meteorological Institute, contact: marko.laine@fmi.fi

The authors are comparing two variants of the Metropolis algorithm to perform Markov chain Monte Carlo (MCMC) analysis for Bayesian inference. The algorithms are DREAM, which uses multiple chains and borrows ideas from differential evolution, and AM with a single chain, which adapts the Gaussian proposal distribution using the chain history. Both of the algorithms should produce ergodic chains that have the target distribution as the invariant distribution of the Markov chain. As such they should provide similar results if the assumptions about the target distribution are met. However, different methods may have varying efficiency in small samples, and there are implementation and tuning issues.

In this comment we show that the authors may not have understood the AM algorithm properly or are using very non-optimal initial values and tuning parameters. In our tests, all the examples work without problems and out-of-the-box by the AM algorithm, and with only 10% of the simulations performed in the article. We do not claim any superiority of AM compared to other methods, but just want to point out that there might be some problems in the authors code. The DREAM algorithm may be very good algorithm for the purpose stated, but the this demonstration does not necessarily have to depend on implementing some other method quite poorly.

The authors claim, that "AM can be unreliable for estimating the PDFs of the parameters that exhibit strong correlation". We have not seen such behavior, and would be happy to see test cases to support the claim. The authors may be mixing SCAM algorithm with AM, of which the first one as a single component method does indeed have a well-documented problem with strong correlations.

It seems, that the authors were not able to perform the relatively simple test cases with the AM algorithm properly. Because of this, there is no reason to believe that the results from the simulations with the DALEC model would appropriately portray the capabilities of AM. While the DREAM algorithm may be superior to AM/DRAM in some settings, we believe that the work presented here does not support such a claim.

A reference implementation of AM is available in the MCMC toolbox for matlab at http://helios.fmi.fi/~lainema/mcmc/. The examples below are computed by the current version downloaded from this site. Alternative AM implementation can be found, for example, in the FME package for R statistical language https://cran.r-project.org/package=FME.

The presented examples are chosen to imitate the test cases in the manuscript. However, as the authors for some reason do not describe them fully, the functions are only our best guesses. In the manuscript, 10^6 MCMC iterations are used. Below, we use only 10^5, as we get convergence with it in all the test cases. We list the complete Matlab commands to generate the chains. A full code, including the plotting commands to generate the figures is included in the MCMC matlab toolbox cited above.

```
nsimu = 100000; % how many simulations
```

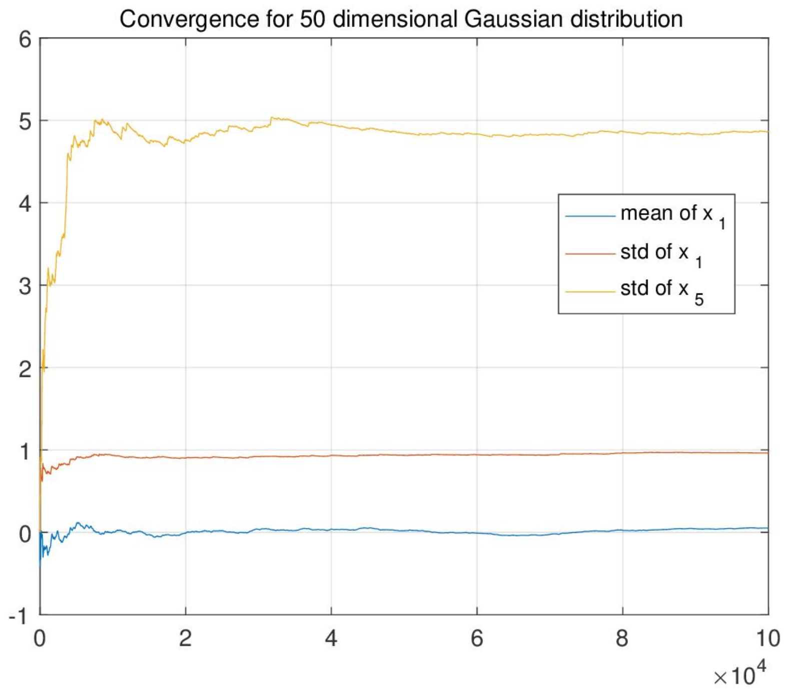
Test I: 50 dimensional Gaussian

The authors do not explicitly state the shape of target. We use 50 dimensional Gaussian, with correlated covariance matrix having marginal variances as 1^2, 2^2, …, 50^2. For MCMC we set a default iid initial scaling of the proposal and set the adaptation interval to every 1000th iteration.

```
npar = 50;
% generate correlated covariance matrix with increasing variances
s = (1:npar)';
ci = inv(cov2cor(covcond(10,ones(npar,1))).*(s*s'));

model.ssfun       = @(x,d) x(:)'*ci*x(:);
options.nsimu     = nsimu;
options.method    = 'am';
options.qcov      = eye(npar)/npar*2.4^2.;
options.adaptint = 1000;
for i=1:npar, params{i} = {sprintf('x_{%d}',i), 0}; end

[res,chain] = mcmcrun(model,[],params,options);
```

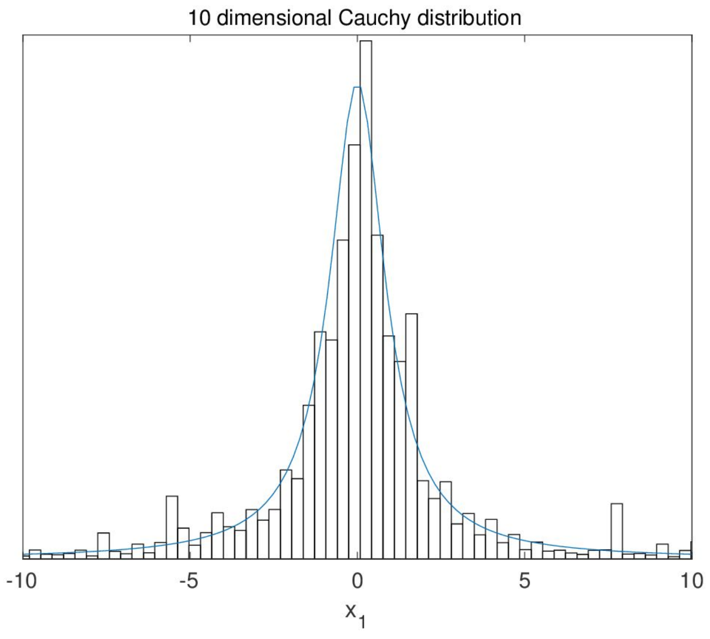


Test II: 10 dimensional Cauchy distribution

Cauchy distribution has heavy tails and no second moments. As the target does not have finite second moments, an adaptive method that depends on covariance matrix of the chain will not converge in theory. In practice, our default AM implementation produces still reasonable chains, similar to those attributed to DREAM. Using an adaptive technique, that does not use the chain variance (such as RAM, http://dx.doi.org/10.1007/s11222-011-9269-5), does not have this problem. However, any random walk algorithm will have slow convergence for targets with tail decaying slower than exponential, see http://www.jstor.org/stable/2337435.

```
npar = 10;
model.ssfun    = @(x,d) 2*sum(log(1+x.^2));
options.nsimu  = nsimu;
options.method = 'am';
for i=1:npar, params{i} = {sprintf('x_{%d}',i), 0}; end

[res,chain] = mcmcrun(model,[],params,options);
```



Alternatively, the same could be run with the RAM-adaptation by changing:

```
options.method = 'ram';
```

Test III, 3 modes 4 dimensional mixed Gaussian

The default AM is typically tuned for one-mode target distribution, but by changing the scaling it will work for multi-modal situations, too. No method will find all the modes if they are too distant and there is no prior information about the locations. The delayed rejection (DR) modification to AM, so called DRAM, allows easy use of different proposals, similarly to

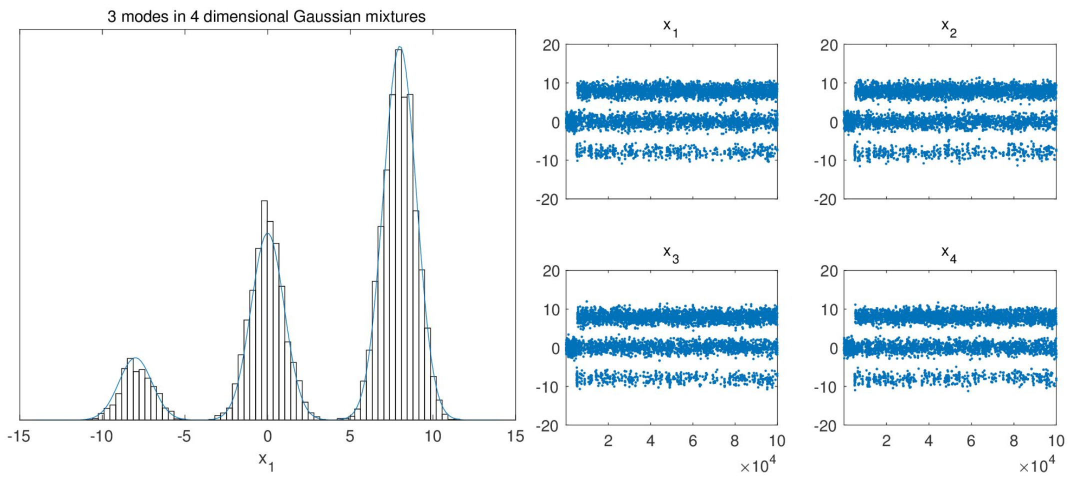
DREAM algorithm which uses larger proposal on every 10th iteration (as far as we know). DRAM (and even AM) can be made to sample multi-modal distributions with suitable initial values. Below, we use DRAM, but the same code works (with a somewhat larger rejection rate) with `options.method = 'am'`, too. Again, a single chain of length 100 000 samples the target quite perfectly.

```
npar = 4;
mu1  = [-8,-8,-8,-8];
mu2  = [ 0, 0, 0, 0];
mu3  = [ 8, 8, 8, 8];
sigs = [ 1, 1, 1, 1];
w    = [0.1, 0.3, 0.6]; % 3 mixture weights

model.ssfun= @(x,d) -2*log(w(1)*mvnorpf(x,mu1,sigs) + ...
  w(2)*mvnorpf(x,mu2,sigs) + ...
  w(3)*mvnorpf(x,mu3,sigs));

options.nsimu     = nsimu;
options.method    = 'dram';
options.qcov      = eye(npar)*5^2;
options.drscale   = 5;
options.adascale = 2.4 / sqrt(npar) * 5;
for i=1:npar, params{i} = {sprintf('x_{%d}',i), 0}; end

[res,chain] = mcmcrun(model,[],params,options);
```



We also noticed that in the manuscript there are some mix-ups with the references related to various versions of adaptive MCMC algorithms. Below are the correct references and other potentially interesting references:

Adaptive Metropolis algorithm (AM): H. Haario, E. Saksman and J. Tamminen, (2001). An adaptive Metropolis algorithm, *Bernoulli*, 7, pp. 223-242. http://dx.doi.org/10.2307/3318737

Single Component Adaptive Metropolis (SCAM): H. Haario, E. Saksman, and J. Tamminen (2005), Componentwise adaptation for high dimensional MCMC. Comput. Stat. 20, 265–274. http://dx.doi.org/10.1007/BF02789703

Delayed Rejection Adaptive Metropolis Algorithm (DRAM): H. Haario, M. Laine, A. Mira and E. Saksman, (2006). DRAM: Efficient adaptive MCMC, *Statistics and Computing*, 16, 339-354. http://dx.doi.org/10.1007/s11222-006-9438-0

Parallel version of Adaptive Metropolis: Solonen A, Ollinaho P, Laine M, Haario H, Tamminen J, Järvinen H (2012). Efficient MCMC for Climate Model Parameter Estimation: Parallel Adaptive Chains and Early Rejection, *Bayesian Analysis*, 7(3), 715-736. http://dx.doi.org/10.1214/12-BA724

Robust version of Adaptive Metropolis for target distributions which do not have finite second moments: M. Vihola (2012). Robust adaptive Metropolis algorithm with coerced acceptance rate, *Stat Comput*. 22: 997. http://dx.doi.org/10.1007/s11222-011-9269-5

Respectfully yours,
Marko Laine
Jouni Susiluoto
Johanna Tamminen
Heikki Haario