R package for Calderhead (2014) A general construction for parallelizing MetropolisHastings algorithms

Alejandra Avalos-Pacheco Tom Jin Valerio Perrone

February 26, 2015

Abstract

`GPMH` is a package for implementing the `rcGMH` algorithm with a toy example provided.

1 Introduccion

The `GPMH` package provides an implementation for the general construction for parallelizing Metropolis-Hastings algorithms provided by Calderhead (2014). This packages provides different functions for some technologies to evaluate de algorithm in parallel:`xMH` `rGPMH`, `rcGPMH`, ADD TOM's technologies (functions explained bellow).

2 Package installation

The package `GPMH` is located in
`https://github.com/tom-jin/GPMH.git`.
Once downloaded, load the package by:

```
library(GPMH)
```

3 Algorithm overview

In a typical Bayesian framework, one of the main aims is computing posterior expectations with respect to a target probability distribution π. This problem corresponds to evaluating integrals of the type of $I(f) := \int_\chi f(x)\pi(dx)$, where $f \in L_1(X, \pi) = \{f : \pi(|f|) < \infty\}$. The Markov Chain Monte Carlo (MCMC) approach consists in simulating an ergodic Markov Chain $(X_n)_{n\geq 0}$ with limiting distribution π in order to estimate I by $\hat{I}(f) := \frac{1}{N}\sum_{i=t}^{t+N} f(X_i)$, for some $t >$

0. A solution is provided by the Metropolis-Hasting (MH) algorithm, which constructs a Markov Chain converging to the distribution π. Given the current value X_n, Y_{n+1} is sampled from a given proposal distribution $K(X_n, Y_{n+1})$ and then accepted with probability

$$\alpha(X_n, Y_{n+1}) = \min\left\{1, \frac{\pi(Y_{n+1})K(Y_{n+1}, X_n)}{\pi(X_n)K(X_n, Y_{n+1})}\right\}. \quad (1)$$

If the proposed value is accepted, we set $X_{n+1} = Y_{n+1}$. Otherwise, $X_{n+1} = X_n$.

The computational efficiency of the Metropolis-Hastings can be improved. An intuition is to make use of the increasingly low-cost parallelism and propose multiple points in parallel at each iteration. Many existing attempts to parallelise Metropolis-Hasting rely on retaining just one of these points. These procedures, however, suffer from the computational inefficiency of not using the remaining points. The Generalised Metropolis-Hastings algorithm proposed by Calderhead (2014) builds on these ideas by retaining the information contained in all proposed points. This is done by weighting them with proper likelihoods and sampling from them as shown in Algorithm (1).

Algorithm 1 `GMH` Calderhead (2014)

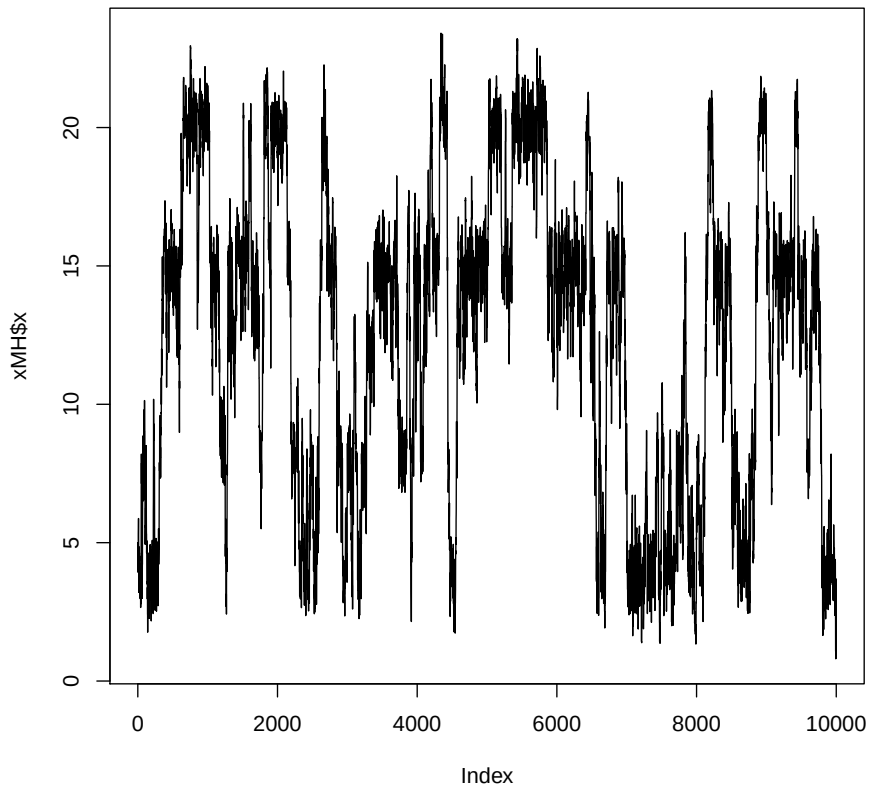
Input $X_1, I = 1, i = 0$.
Output $(X_1, ..., X_{n\times(N+1)})$.
for $i = 1$ `to` n **do**
  `Draw` N `points` $x_2, \ldots, x_{N+1}$ `from` $K(x_I, \cdot)$.
  `Compute` $A(j) = \pi(x_j)K(x_j, x_{\setminus j}) \quad \forall j \in [1, \ldots, N+1]$.
  `Sample with replacement the` $N + 1$ `points with probabilities` $A(j)$ `to obtain the sample` X_k, $k = (i-1) * N + i, \ldots, i * (N+1)$
  `Draw` $I \sim U[1, \ldots, N+1]$, `set` $i = i + N$
end for

4 Functions overview

4.1 `MH`

Runs the standard Metropolis-Hastings algorithm for sampling from a given target distribution.

```
xMH <- MH(target=function(x) {0.2*dnorm(x,12,1)+0.3*dnorm(x,8,1)+0.4*dnorm(x,4,1)+0.6*dnorm(
          kernel=function(x) {rnorm(n=1, x)},
          dkernel=function(x,y) {dnorm(y, x)},
          init.state=5,
          n=10000)

plot(xMH$x, type="l")
```

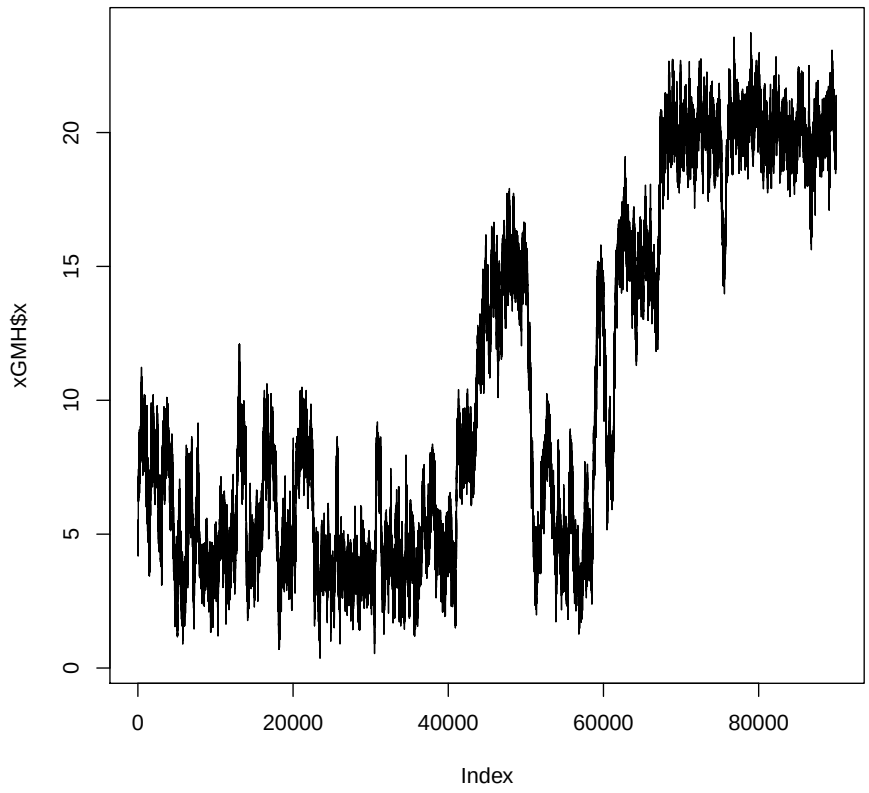



```
summary(xMH$x)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.803   7.580  13.900  12.400  16.000  23.400
```

4.2 rGPMH

Runs the Generalised Metropolis-Hastings algorithm for sampling from a given target distribution. At each iteration, the algorithm draws N new points from the proposal distribution given the current state, then it computes the likelihoods of all N+1 points (the N new points plus the current state). Then, these N+1 points are sampled with replacement according to their likelihoods and the new sample of N+1 points is obtained. Finally, a state from these N+1 points is randomly chosen to generate the N new points in the successive iteration.

```
xGMH<-rGPMH(target=function(x) {0.2*dnorm(x,12,1)+0.3*dnorm(x,8,1)+0.4*dnorm(x,4,1)+0.6*dnor
          kernel=function(x) {rnorm(n=1, x)},
          dkernel=function(x,y) {dnorm(y, x)},
          init.state=5,
          n=10000,
          N=8)

plot(xGMH$x, type="l")
```

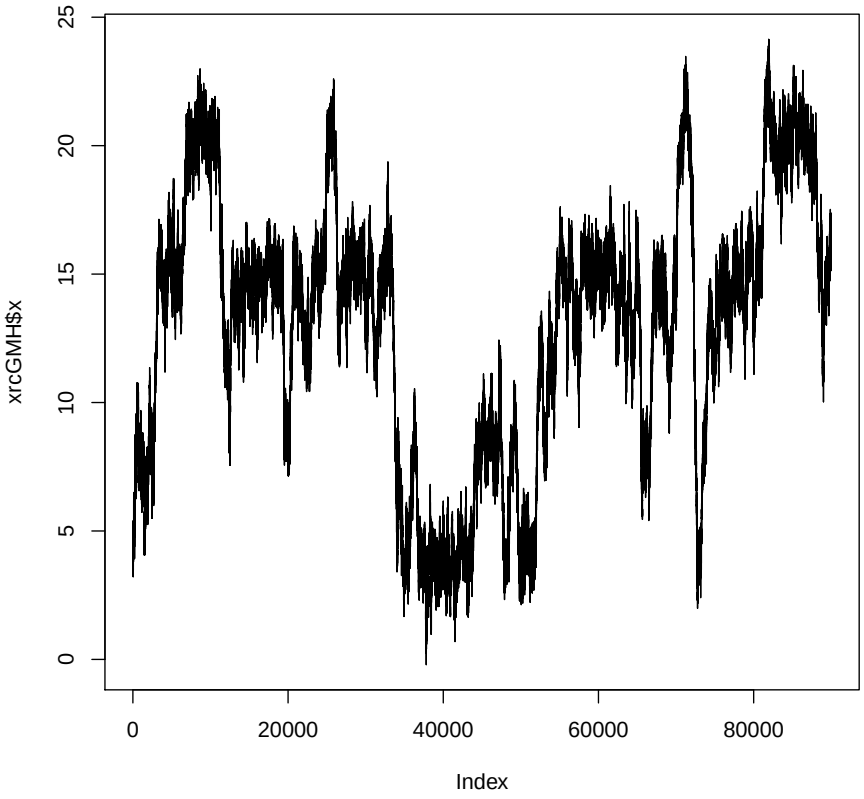



```
summary(xGMH$x)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.364   4.370   8.000  10.500  17.000  23.700
```

4.3 rcGPMH

Runs the Generalised Metropolis-Hastings algorithm for sampling from a given target distribution. At each iteration, the algorithm draws N new points from the proposal distribution given the current state, then it calls the C function "Cgmh.c" to compute the likelihoods of all N+1 points (the N new points plus the current state). Then, these N+1 points are sampled with replacement according to their likelihoods and the new sample of N+1 points is obtained. Finally, a state from these N+1 points is randomly chosen to generate the N new points in the successive iteration.

```
xrcGMH<-rcGPMH(target=function(x) {0.2*dnorm(x,12,1)+0.3*dnorm(x,8,1)+0.4*dnorm(x,4,1)+0.6*d
               kernel=function(x) {rnorm(n=1, x)},
               dkernel=function(x,y) {dnorm(y, x)},
               init.state=5,
               n=10000,
               N=8)

plot(xrcGMH$x, type="l")
```




```
summary(xrcGMH$x)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   -0.21    8.90   14.20   12.90   15.80   24.10
```

4.4 C

4.5 CUDA

5 Comparison of technologies

In the following examples, we set a mixture of univariate normals as target distribution. We compared the running times of the standard Metropolis-Hastings (MH) algorithm with the Generalised Metropolis-Hastings (GMH) using different technologies and different values of N (the number of points drawn at

each iteration from the proposal distribution). The results, which are plotted in Figure 1, reveal the great sensitivity of running time to the technology used. The blue point at $N = 1$ corresponds to the regular MH algorithm. The green points correspond to the pure R implementation of the GMH, while the blue and red ones to the versions in which the vector of likelihoods A is computed in parallel within a C function. More specifically, for the blue points the algorithm was run in the CDT desktop ???, while for the red ones we used OPEN MP???. The results illustrate that the parallelised versions of GMH clearly outperform both the pure R implementation and the standard MH, with the difference becoming more evident as the number N of parallely drawn points increases. On the contrary, the pure R version gets increasingly slower as N increases, which is perfectly consistent with the low performance of R in computing for loops.

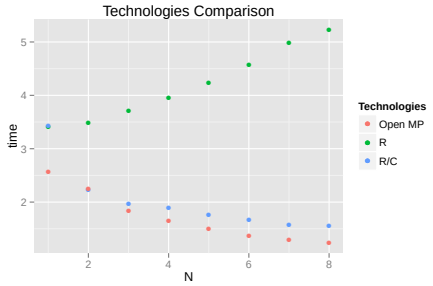


Figure 1: Time comparison between different technologies.

6 Extensions

Write here all the stuff that Tom wanted to do but didn't have time to do it

References

Ben Calderhead. A general construction for parallelizing metropolis?hastings algorithms. *Proceedings of the National Academy of Sciences*, 111(49):17408–17413, 2014. doi: 10.1073/pnas.1408184111. URL `http://www.pnas.org/content/111/49/17408.abstract`.