Introduction to SamplerCompare

Madeleine B. Thompson
University of Toronto

Abstract

SamplerCompare is an R package for comparing the performance of Markov chain Monte Carlo samplers. It samples from a collection of distributions with a collection of MCMC methods over a range of tuning parameters. Then, using log density evaluations per uncorrelated observation as a figure of merit, it generates a grid of plots showing the results of the simulation. It comes with a collection of predefined distributions and samplers and provides R and C interfaces for defining additional ones. It also provides the means to import simulation data generated by external systems. This document provides background on the package and demonstrates the basics of running simulations, visualizing results, and defining distributions and samplers in R.

Keywords: MCMC, visualization.

A version of this document was published as Thompson (2011a).

1. Purpose of package

When a researcher develops a new Markov chain Monte Carlo (MCMC) method, they will wish to determine how it compares to existing methods on a representative set of distributions. Similarly, when a statistician specifies a new distribution, they may want to know which common MCMC methods are most efficient at sampling from it. **SamplerCompare** (Thompson 2011b) is an R (R Development Core Team 2010) package that automates these tasks. It draws samples from a collection of probability distributions with a collection of MCMC samplers, with a range of tuning parameters, and presents the results of such simulations graphically. These comparisons allow researchers to better understand which MCMC methods perform best in which circumstances.

The main goal of **SamplerCompare** is to generate a grid of plots in which an individual grid cell corresponds to a single MCMC sampler and a single distribution. In each grid cell, the efficiency of a simulation is summarized by a plot of the product of the number of log density evaluations per iteration multiplied by the autocorrelation time of the slowest-mixing component of the state space for a range of scale tuning parameter values. Autocorrelation time, like effective sample size, which can be computed by `effectiveSize` in the **coda** (Plummer, Best, Cowles, and Vines 2010) package, accounts for the often-substantial correlation between observations in MCMC-generated samples. Log density evaluations accounts for elapsed processor time in a machine-independent way. By viewing the cost measure in a grid of plots, a user can see patterns in the performance more easily than they could if they viewed the same results as numbers in a table.

2. Installation and documentation

Binary packages for MacOS and Windows and a platform-independent source package can be obtained from CRAN at:

http://cran.r-project.org/web/packages/SamplerCompare/index.html

To install it, one must first install the **mvtnorm** package (Genz, Bretz, Miwa, Mi, Leisch, Scheipl, and Hothorn 2011). To use **SamplerCompare**'s graphics, one must install the **ggplot2** package (Wickham 2009). To use multithreading, one must install the **synchronicity** (Kane 2010) package. It is not available for Windows, so Windows users are limited to single-threaded simulations.

Alternatively, one can use the `install.packages` R command:

```
R> install.packages("SamplerCompare", dependencies=c("Depends","Suggests"))
```

More information on **SamplerCompare** is available in the R online help for the package and Thompson (2010b). After the package is installed, a list of online help topics and vignettes can be found by typing:

```
R> library(help='SamplerCompare')
```

Vignettes can be read with the `vignette` command. For example:

```
R> vignette('glue')
```

PDF copies can be found in the `doc` directory of the installed package. Further information on the mathematical background of the comparisons and analysis of the plots is available in Thompson (2010a).

3. Simulations with included samplers and distributions

The three central types of objects in **SamplerCompare** are distributions (which have the class `scdist`), sampler functions, and simulation results. The function `compare.samplers` runs a list of samplers on a list of distributions with a set of tuning parameters and returns a data frame containing simulation results. Sampler functions are assumed to have a single scalar tuning parameter. If they have more, wrapper functions can represent a single sampler with a varying tuning parameter as multiple samplers. **SamplerCompare** comes with a collection of predefined samplers (listed in Table 1) and distributions (listed in Table 2).

Suppose we would like to compare Adaptive Metropolis (`adaptive.metropolis.sample`) and Adaptive Rejection Metropolis (`arms.sample`) with the tuning parameters 0.1, 1, 10, and 100 on two-dimensional Gaussian (`make.gaussian`) and Gamma (`make.mv.gamma.dist`) distributions. We can do this with `compare.samplers` using the R code:

```
library("SamplerCompare")
gauss.cor7 <- make.gaussian(mean = c(1,2), rho = 0.7)
gamma.shape23 <- make.mv.gamma.dist(shape = c(2,3))
```

R function	Sampler
`multivariate.metropolis.sample`	Metropolis–Hastings with spherically symmetric Gaussian proposals
`univar.metropolis.sample`	Metropolis–Hastings with single-coordinate updates
`adaptive.metropolis.sample`	Adaptive Metropolis–Hastings (Roberts and Rosenthal 2009)
`arms.sample`	Adaptive Rejection Metropolis (Gilks, Best, and Tan 1995)
`stepout.slice.sample`	slice sampler with stepping out (Neal 2003, §4)
`interval.slice.sample`	slice sampler without stepping out (Neal 2003, §4)
`univar.eigen.sample`	adaptive slice sampler with univariate steps along eigenvectors of covariance matrix (Thompson 2011c, ch. 3)
`hyperrectangle.sample`	slice sampler with hypercube for initial slice approximation, shrinkage using gradient (Neal 2003, §5.1)
`nograd.hyperrectangle.sample`	slice sampler with hypercube for initial slice approximation, shrinkage in all dimensions (Neal 2003, §5.1)
`oblique.hyperrect.sample`	adaptive slice sampler with hyperrectangle for initial slice approximation (Thompson 2011c, ch. 3)
`nonadaptive.crumb.sample`	slice sampler with Gaussian crumbs (Neal 2003, §5.2)
`cov.match.sample`	covariance-matching slice sampler (Thompson and Neal 2010, §4)
`shrinking.rank.sample`	shrinking rank slice sampler (Thompson and Neal 2010, §5)

Table 1: Predefined samplers; see the R help for the sampler's R function for more information on an individual method.

R symbol	Distribution
`N2weakcor.dist`	weakly correlated two-dimensional Gaussian
`N4poscor.dist`	strongly positively correlated four-dimensional Gaussian
`N4negcor.dist`	strongly negatively correlated four-dimensional Gaussian
`schools.dist`	ten-dimensional multilevel model (Gelman, Carlin, Stern, and Rubin 2004, pp. 138–145)
`funnel.dist`	ten-dimensional distribution with funnel-shaped marginals (Neal 2003, p. 732)

R function	Distributions generated
`make.gaussian`	multivariate Gaussians
`make.cone.dist`	distributions with cone-shaped log density (Roberts and Rosenthal 2002)
`make.multimodal.dist`	mixtures of standard Gaussians
`make.mv.gamma.dist`	distributions with uncorrelated gamma marginals

Table 2: Predefined distributions and functions that generate distributions; see the R help for a symbol for more information on an individual distribution or generator.

```
sampler.comparison <-
  compare.samplers(sample.size = 200,
                   dists = list(gauss.cor7, gamma.shape23),
                   samplers = list(adaptive.metropolis.sample, arms.sample),
                   tuning = 10^seq(-1, 2, by = 1))
```

The call to `compare.samplers` generates the following trace, with one line for each simulation:

Each line in the trace has the distribution name, the sampler name, the number of evaluations per uncorrelated observation with 95% confidence interval in parentheses, the tuning parameter, and the autocorrelation time of the log density.

The return value of `compare.samplers` (`sampler.comparison` in this example) is a data frame with one row per simulation. To see, for example, how many evaluations and how many processor seconds Adaptive Metropolis needed to generate an uncorrelated observation on the two-dimensional Gaussian, one would multiply the `act` column by the `evals` and `cpu` columns:

```
s <- subset(
    sampler.comparison,
    sampler == "Adaptive Metropolis" &
    dist == "N2,rho=0.7" &
    tuning == 1)
print(s$act * s$evals)
print(s$act * s$cpu)
```

This generates the output:

```
[1] 13.27591
[1] 0.0008938892
```

Notice that the first product is the same as the one reported in the corresponding line from the `compare.samplers` trace output (the seventh, `N2,rho=0.7 Adaptive Metropolis . . . tuning=1`). See the R help page for `sampler.comparison` for a full list of the columns of its return value.

4. Visualizing results

To visually compare the efficiency of a collection of simulations, one can use the `comparison.plot` function. It has a single required argument, a data frame containing results from `compare.samplers` or `simulation.result`. The previous section has an example of the use of `compare.samplers`. The online help for `simulation.result` contains an example of its use to load a simulation generated by **JAGS** (Plummer 2010).

`comparison.plot` returns a **ggplot2** plot object. One can call `print` on this object to view the plot; it can also be edited with the **grid** package (Murrell 2005, ch. 5–6). To plot the results from the previous example, one would type:

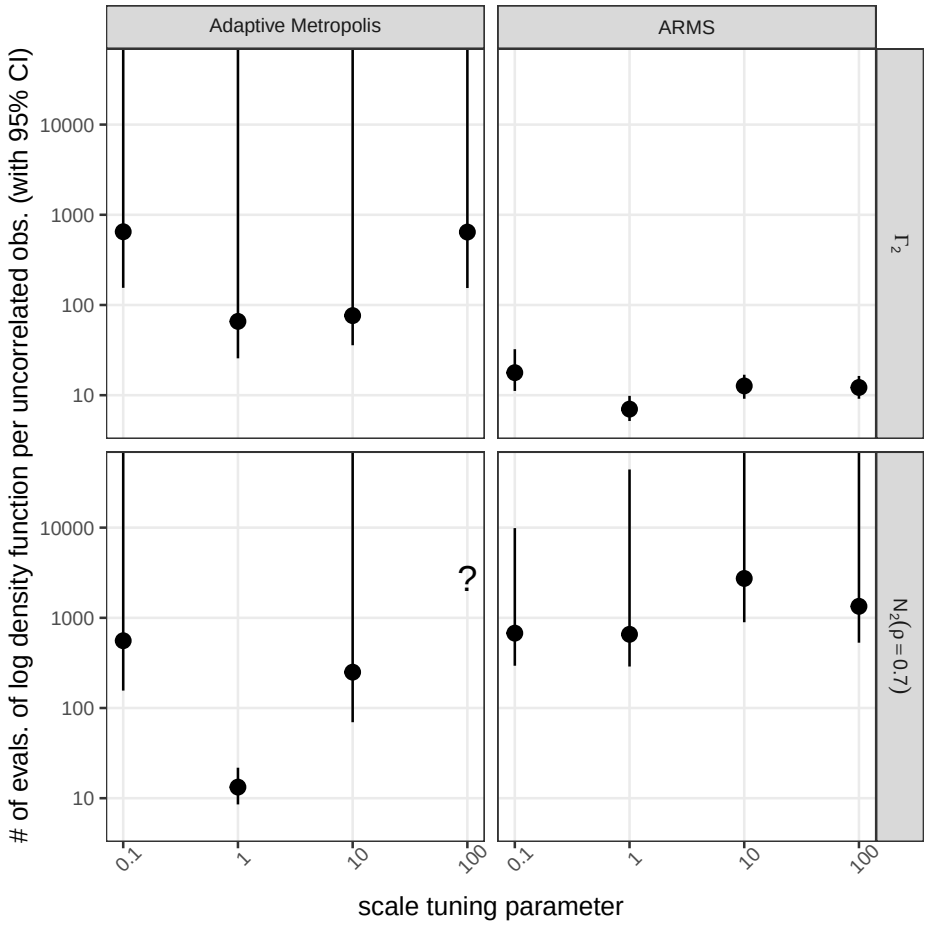


Figure 1: A comparison between Adaptive Metropolis and ARMS on two-dimensional Gaussians and gammas. See section 4 for discussion.

```
R> print(comparison.plot(sampler.comparison))
```

The results are shown in Figure 1.

In this figure, the columns of plots represent the samplers, and the rows of plots represent the distributions. The scale tuning parameter is plotted on the horizontal axis; the number of log density evaluations per uncorrelated observation is plotted as a dot on the vertical axis with a bar for the 95% confidence interval. Log density evaluations per uncorrelated observation is computed by multiplying the average number of log density evaluations per simulation iteration by the autocorrelation time of the slowest-mixing component of the simulation. The autocorrelation time is the ratio of the sample size to the effective sample size. It accounts for linear dependence between successive states. See the help for `ar.act` for more information on how it is computed.

In Figure 1, one can see that on the two distributions compared, when sampling with ARMS, the cost measure does not vary much with the tuning parameter. Adaptive Metropolis seems more sensitive to the tuning parameter. However, when the components of the target dis-

tribution are correlated, it performs better than ARMS when the tuning parameter is well chosen (in this case, equal to one). For more discussion of the interpretation of these plots, see Thompson (2010a, §5).

5. Defining a sampler

MCMC samplers are specified by functions that have the signature:

```
sampler(target.dist, x0, sample.size, tuning)
```

They must also have a `name` attribute, a length-one character vector. The `target.dist` parameter specifies the target distribution; see the R help for `make.dist` for details on its structure. `x0` specifies the start state for the simulation, `sample.size` specifies the sample size, and `tuning` specifies a scalar tuning parameter.

A sampler function should return a list with two elements: `X`, a matrix with one row per observation, and `evals`, a count of the number of times it evaluated the log density (with `target.dist$log.density`). If the sampler evaluates the gradient of the log density (with `target.dist$grad.log.density`), the list should contain a `grads` element, indicating the number of times it did this.

The following code specifies a Metropolis sampler with multivariate proposals:

```
metropolis.sample <- function(target.dist, x0, sample.size, tuning) {
  X <- matrix(nrow = sample.size, ncol = target.dist$ndim)
  state <- x0
  evals <- 1
  state.log.dens <- target.dist$log.density(state)
  for (obs in 1:sample.size) {
    proposal <- rnorm(target.dist$ndim, state, tuning)
    evals <- evals + 1
    proposal.log.dens <- target.dist$log.density(proposal)
    if (runif(1) < exp(proposal.log.dens - state.log.dens)) {
      state <- proposal
      state.log.dens <- proposal.log.dens
    }
    X[obs,] <- state
  }
  return(list(X = X, evals = evals))
}
attr(metropolis.sample, "name") <- "Metropolis"
```

See the R help for `compare.samplers` for more information on writing samplers in R. See the R help for `wrap.c.sampler` and Thompson (2010b) for more information on writing samplers in C.

6. Defining a distribution

`make.dist` can be used to specify a distribution whose log density is expressed in R. (See the R

help for `make.c.dist` and Thompson (2010b) for more information on specifying distributions in C.) Its most important arguments are `ndim`, `name`, and `log.density`. `ndim` specifies the dimension of the distribution and `name` names the distribution. `log.density` is a function of one vector argument of length `ndim` that returns the log density at that point; it should return `-Inf` if the point is outside the support of the distribution. The log density does not need to be normalized.

The following R code defines a Beta(2,3) distribution:

```
beta23.log.dens <-
    function(x) ifelse(x < 0 | x > 1, -Inf, log(x) + 2 * log(1 - x))
beta23.dist <- make.dist(ndim = 1, name = "Beta(2,3)",
                         log.density = beta23.log.dens, mean = 2 / (2 + 3))
```

The optional `mean` argument to `make.dist` makes the autocorrelation time computation in `compare.samplers` more accurate, so it is advisable to specify it when the mean is known.

7. Comparing user-defined distributions and samplers

User-defined samplers and distributions can be used just like the included samplers and distributions. To use the Metropolis sampler defined in section 5 to sample from the beta distribution defined in section 6, one can run the code from those two sections and then run:

```
sim <- metropolis.sample(beta23.dist, x0 = 0.5, sample.size = 100, tuning = 1)
```

User-defined samplers can also be used with `compare.samplers`:

```
sampler.comparison <- compare.samplers(sample.size = 1000,
                                       dists = list(beta23.dist),
                                       samplers = list(metropolis.sample),
                                       tuning = c(0.1, 1, 10),
                                       trace = FALSE)
print(subset(
    sampler.comparison,
    select = c("dist", "sampler", "tuning", "act", "evals", "cpu", "err")))
```

The call to `print(subset(...))` shows some of the columns of the result object:

```
       dist    sampler tuning        act evals   cpu         err
1 Beta(2,3) Metropolis    0.1  18.308804 1.001 5e-06 0.009270565
2 Beta(2,3) Metropolis    1.0   6.166051 1.001 5e-06 0.027165487
3 Beta(2,3) Metropolis   10.0 230.008586 1.001 6e-06 0.036684207
```

One can see that since the evaluations per iteration (`evals`) and processor-seconds per iteration (`cpu`) are similar for each simulation, and the autocorrelation time (`act`) is lowest for a tuning parameter of 1.0, that choice would seem to be better than the other two. However, the plots produced by `comparison.plot` are easier to interpret when more than a few chains are run.

8. Limitations

SamplerCompare was created to support my own research; I am releasing it with the hope that others find it useful. Some current limitations include:

- Distributions are assumed to be continuous and to be of a constant dimension.
- Samplers are assumed to have exactly one scalar tuning parameter.
- All samplers in a given invocation of `compare.samplers` are run with the same simulation length and set of tuning parameters.
- Distributions are defined entirely in terms of their log density; there is no way to specify that a distribution is unimodal or that a particular parameter is always positive.
- Multithreading is not supported on Windows.

References

Gelman A, Carlin JB, Stern HS, Rubin DB (2004). *Bayesian Data Analysis, Second Edition*. Chapman and Hall/CRC. URL http://www.stat.columbia.edu/~gelman/book/.

Genz A, Bretz F, Miwa T, Mi X, Leisch F, Scheipl F, Hothorn T (2011). ***mvtnorm**: Multivariate Normal and t Distributions*. R package version 0.9-96, URL http://CRAN.R-project.org/package=mvtnorm.

Gilks WR, Best NG, Tan KKC (1995). "Adaptive Rejection Metropolis Sampling within Gibbs Sampling." *Applied Statistics*, **44**(4), 455–472. URL http://www.jstor.org/stable/2986138.

Kane MJ (2010). ***synchronicity**: Boost mutex functionality for R*. R package version 1.0.9, URL http://CRAN.R-project.org/package=synchronicity.

Murrell P (2005). *R Graphics*. Chapman and Hall/CRC. URL http://www.stat.auckland.ac.nz/~paul/RGraphics/rgraphics.html.

Neal RM (2003). "Slice Sampling." *The Annals of Statistics*, **31**, 705–767. URL http://projecteuclid.org/getRecord?id=euclid.aos/1056562461.

Plummer M (2010). *JAGS Version 2.2.0 User Manual*. URL http://surfnet.dl.sourceforge.net/project/mcmc-jags/Manuals/2.x/jags_user_manual.pdf.

Plummer M, Best N, Cowles K, Vines K (2010). ***CODA** Reference Manual*.

R Development Core Team (2010). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

Roberts GO, Rosenthal JS (2002). "The Polar Slice Sampler." *Stochastic Models*, **18**(2), 257–280. URL http://www.informaworld.com/openurl?genre=article&issn=1532%2d6349&volume=18&issue=2&spage=257.

Roberts GO, Rosenthal JS (2009). "Examples of Adaptive MCMC." *Journal of Computational and Graphical Statistics*, **18**(2), 349–367. URL http://pubs.amstat.org/doi/abs/10.1198/jcgs.2009.06134.

Thompson MB (2010a). "Graphical Comparison of MCMC Performance." *Technical Report 1010*, Dept. of Statistics, University of Toronto. ArXiv:1011.4457v1 [stat.CO], URL https://arxiv.org/abs/1011.4457.

Thompson MB (2010b). "R/C Glue in SamplerCompare." URL http://cran.r-project.org/web/packages/SamplerCompare/vignettes/glue.pdf.

Thompson MB (2011a). "Introduction to **SamplerCompare**." *Journal of Statistical Software*, **43**(12), 1–10. URL https://www.jstatsoft.org/article/view/v043i12.

Thompson MB (2011b). ***SamplerCompare**: A Framework for Comparing the Performance of MCMC Samplers*. URL http://cran.r-project.org/web/packages/SamplerCompare/index.html.

Thompson MB (2011c). *Slice Sampling with Multivariate Steps*. Ph.D. thesis, University of Toronto. URL http://hdl.handle.net/1807/31955.

Thompson MB, Neal RM (2010). "Covariance-Adaptive Slice Sampling." *Technical Report 1002*, Dept. of Statistics, University of Toronto. ArXiv:1003.3201v1 [stat.CO], URL http://www.cs.toronto.edu/~radford/cass.abstract.html.

Wickham H (2009). ***ggplot2**: Elegant Graphics for Data Analysis*. Springer–Verlag. URL http://had.co.nz/ggplot2/book.

Affiliation:

Madeleine B. Thompson
E-mail: madeleineth@gmail.com
URL: https://mdln.net