`Rmodule`: Automated Markov chain Monte Carlo for arbitrarily structured correlation matrices

John Hughes

Lehigh University

Abstract

Statistical inference for correlation matrices and their functionals can be carried out straightforwardly when the matrix in question has a simple structure or is unstructured. The software described in this article, R package `Rmodule`, allows users to do automated inference for correlation matrices having more complicated structures. This is a challenging problem since updating any given correlation parameter, conditional on the present values of the remaining correlation parameters, may entail proposing a new value from a distribution that has disconnected support, i.e., a support comprising two or more disjoint subintervals of $(-1, 1)$. `Rmodule` solves this problem by applying a new algorithm, the Apes of Wrath algorithm, to handle such proposals. A package user can employ `Rmodule` as a component of a broader Monte Carlo algorithm by supplying a data matrix, a correlation matrix in symbolic form, the current state of the chain, a function that computes the log likelihood, and, optionally, a collection of prior distributions. The package's flagship function then carries out a variable-at-a-time update of all correlation parameters, and returns the new state. The sampler is quite fast since important auxiliary functions have been implemented in C++. The sampler is also easy to tune.

Keywords: correlation matrix, C++, Metropolis–Hastings random walk, R, statistical inference

Email address: `drjphughesjr@gmail.com` (John Hughes)

Current code version	1.0
Permanent link to code/repository used for this code version	Comprehensive R Archive Network `https://tinyurl.com/3azx93ed`
Legal code license	GPL (≥ 2)
Code versioning system used	major.minor-patch
Software code languages, tools, and services used	R and C++
Compilation requirements, operating environments, and dependencies	R packages `Rcpp` and `RcppArmadillo`
Support email for questions	`drjphughesjr@gmail.com`

Table 1: Code metadata

1. Motivation and significance

If one wishes to do Bayesian (or maximum *a posteriori*) inference for a correlation matrix and/or functionals (e.g., Gaussian mutual information) of said matrix, two common approaches are the parameter-at-a-time strategy proposed by Barnard et al. (2000) or a matrix-valued prior distribution such as the Lewandowski–Kurowicka–Joe prior (Lewandowski et al., 2009). These approaches are advantageous when the true correlation matrix is unstructured or has simple and/or soft constraints. These approaches are not suitable for correlation matrices having more complicated structures and hard constraints since in this case it can be challenging to enforce the constraints while also ensuring that the matrix is non-negative definite.

Consider, for example, the 4×4 correlation matrix

$$R = \begin{pmatrix} 1 & r_1 & r_3 & r_4 \\ r_1 & 1 & r_5 & r_3 \\ r_3 & r_5 & 1 & r_2 \\ r_4 & r_3 & r_2 & 1 \end{pmatrix}.$$

Suppose the current state of the Markov chain is

$$(r_1, r_2, r_4, r_5) = (-0.01, -0.18, -0.75, 0.83),$$

i.e.,

$$R(r_3) = \begin{pmatrix} 1 & -0.01 & r_3 & -0.75 \\ -0.01 & 1 & 0.83 & r_3 \\ r_3 & 0.83 & 1 & -0.18 \\ -0.75 & r_3 & -0.18 & 1 \end{pmatrix},$$

and we wish to update r_3 conditional on this state. Then the function

$$f(r_3) = \det\{R(r_3)\}$$

is the quartic function shown in Figure 1 since

$$\begin{aligned} \det(R) &= \boldsymbol{r_3^4} \\ &- 2(r_1 r_2 + r_4 r_5 + 1)\boldsymbol{r_3^2} \\ &+ 2(r_1 r_4 + r_1 r_5 + r_2 r_4 + r_2 r_5)\boldsymbol{r_3} \\ &+ r_1^2 r_2^2 - r_1^2 - r_2^2 + r_4^2 r_5^2 - r_4^2 - r_5^2 - 2r_1 r_2 r_4 r_5 + 1. \end{aligned}$$

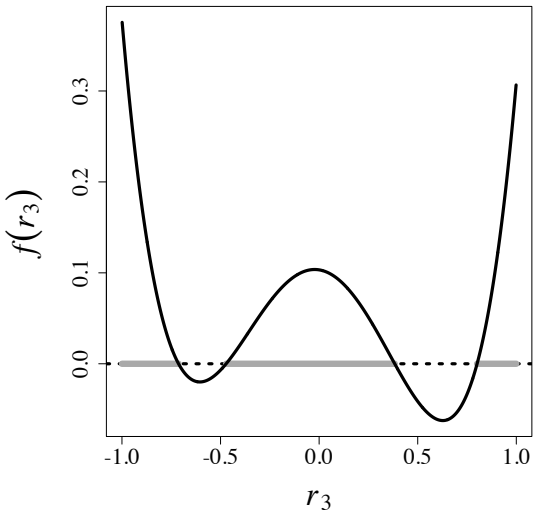

Figure 1: The complicated determinant function for the scenario described in the text

We see that $f(r_3)$ is non-negative only on the subintervals $(-1, -0.71)$, $(-0.47, 0.38)$, and $(0.8, 1)$ (gray line segments), and so a valid proposal for the next state of r_3 must fall within one of these intervals. Scenarios of this sort, which can be arbitrarily complicated depending on the size and structure of

the correlation matrix, cannot be handled by the traditional methods.

Package `Rmodule` handles scenarios like the one described above by (1) using a root-finding algorithm to identify the subintervals on which $f(r_3)$ is non-negative, (2) placing the subintervals end-to-end to produce a single subinterval, (3) mapping the new subinterval onto the real line, (4) using a Gaussian random walk to propose a new value from the real line, (5) mapping the proposed value back to correlation scale, i.e., $(-1, 1)$, and (6) mapping the single subinterval of $(-1, 1)$ back to the original disconnected set. I call this algorithm the Apes of Wrath algorithm because the well-known 1959 Bugs Bunny cartoon *Apes of Wrath* inspired the idea of gathering the collection of subintervals into a single subinterval. In the cartoon, Bugs crosses a rope bridge and tries to keep Elvis, an adult male gorilla, at bay by threatening to cut the rope if Elvis attempts to cross it. Elvis instead pulls the opposing cliffside to him with one impressive yank of the rope (Figure 2).

Figure 2: The inspiration for the Apes of Wrath algorithm (images © Warner Bros. Pictures)

Since it employs numerical root finding, the Apes of Wrath algorithm can easily handle arbitrarily complicated functions $f(r) = \det\{R(r)\}$. The root finding is computationally efficient because I implemented a vectorized version of the determinant function in C++. Steps 2–6 of the algorithm are also implemented in C++.

Incidentally, the maps $(a, b) \leftrightarrow \mathbb{R}$ are given by

$$T(x) = \tan\left\{\frac{\pi}{b-a}\left(x - \frac{a+b}{2}\right)\right\}$$

and

$$T^{-1}(y) = \frac{b-a}{\pi} \arctan(y) + \frac{a+b}{2},$$

where $(a, b) \subset (-1, 1)$ is the interval to be mapped. And, on a side note, the name '`Rmodule`' is a fun triple pun:

1. `Rmodule` is an R module,
2. `Rmodule` infers correlation matrices, which are often denoted as R, and
3. linear/matrix algebra can be expressed more generally in terms of R-modules, where the underlying field F is replaced by ring R.

2. Software description

Package `Rmodule` exports exactly one function to the R namespace. The function's signature is as follows.

```
update_R(
  r,
  data,
  R,
  log.f,
  log.f.args,
  log.priors,
  log.priors.args,
  sigma,
  n = 100
)
```

The first argument, `r`, is a p-vector of correlations and the current state of the Markov chain. It is assumed that these values correspond to a valid correlation matrix since function `update_R` updates the state of the Markov chain conditional on this value of the state vector.

The second argument, `data`, is the data matrix. This argument must be an R matrix with each row an observation. It is assumed that the rows are independent and have the same correlation structure. Whether the rows are assumed to be identically distributed is up to the user and can be handled via argument `log.f`.

The third argument, `R`, is the posited correlation matrix in symbolic form. Since we assume this matrix has 1's on its diagonal, the remaining entries

should be numbered using consecutive integers starting with 2. For example, correlation matrix

$$R = \begin{pmatrix} 1 & r_1 & r_3 & r_4 \\ r_1 & 1 & r_5 & r_3 \\ r_3 & r_5 & 1 & r_2 \\ r_4 & r_3 & r_2 & 1 \end{pmatrix}.$$

might be represented symbolically as

$$R = \begin{pmatrix} 1 & 2 & 4 & 5 \\ 2 & 1 & 6 & 4 \\ 4 & 6 & 1 & 3 \\ 5 & 4 & 3 & 1 \end{pmatrix}.$$

Function `update_R` will then update the correlations one at a time: first 2, then 3, and so on until all p correlations have been updated.

The fourth argument, `log.f`, is the log objective function. The objective function might be a likelihood or a composite likelihood or an approximate likelihood, for example. This function must take exactly three arguments: the data matrix, the correlation matrix (not the symbolic form of same), and a list, `log.f.args`, containing any additional arguments. This function must return exactly one scalar value, the value of the log objective function evaluated at the given value of the correlation matrix, i.e., $\log f(R \mid \text{data}, \cdot)$, where $\cdot$ denotes any additional arguments required to compute $\log f$. This function is used by `update_R` to compute $\log f(R^* \mid \text{data}, \cdot) - \log f(R \mid \text{data}, \cdot)$ in the Metropolis–Hastings acceptance probability, where R^* is the proposal and R is the current state.

Argument `log.priors` must be a list of p log prior distributions. Common choices of prior distribution are the Uniform$(-1, 1)$ prior, the Jeffreys prior $(1 - r^2)^{-1.5}$, and, more generally, $(1 - r^2)^c$, where $c \in \mathbb{R}$. Additional arguments for the priors can be passed as the list `log.priors.args`.

Argument `sigma` is a vector of standard deviations for the Gaussian random walk proposals. These are the only necessary tuning parameters for the sampler. If `sigma` has length 1, that value will be used for all p correlation parameters. Otherwise `sigma` should have length p.

The final argument is `n`. This is the number of grid points to use in root finding. The default value is 100. For some problems, a larger value may be required to avoid overlooking roots of $\det(R)$. Note that the choice of `n` does

not affect execution time much because auxiliary function `detR` is a vectorized C++ function that employs the Armadillo linear algebra library (Sanderson and Curtin, 2016) by way of R package `RcppArmadillo` (Eddelbuettel and Sanderson, 2014):

```
Rcpp::NumericVector detR(Rcpp::NumericVector r, int param,
                         const arma::mat& R, arma::mat R_)
{
    arma::uword n = R.n_rows;
    arma::uword m = r.length();
    Rcpp::NumericVector dets(m);
    arma::uword i, j;
    for (arma::uword k = 0; k < m; k++)
    {
        for (i = 0; i < n; i++)
            for (j = i + 1; j < n; j++)
                if (R(i, j) == param)
                {
                    R_(i, j) = r[k];
                    R_(j, i) = r[k];
                }
        dets[k] = det(R_);
    }
    return dets;
}
```

3. Illustrative examples

In this section I illustrate the use of `Rmodule` by employing the package in an algorithm that measures intra-coder and inter-coder agreement for a biomaging study of congenital diaphragmatic hernia, in which a hole in the diaphragm permits abdominal organs to enter the chest (Longoni et al., 2020). The data for this example are liver-herniation scores (in $\{1, \ldots, 5\}$) assigned by two coders (radiologists) to magnetic resonance images (MRI) of the liver. The five grades are described in Table 2.

Each coder scored each of the 47 images twice, and so we can assess both

Table 2: Liver herniation grades for the CDH study

Grade	Description
1	No herniation of liver into the fetal chest
2	Less than half of the ipsilateral thorax is occupied by the fetal liver
3	Greater than half of the thorax is occupied by the fetal liver
4	The liver dome reaches the thoracic apex
5	The liver dome not only reaches the thoracic apex but also extends across the thoracic midline

intra-coder and inter-coder agreement. I used the 4×4 correlation structure

$$R = \begin{pmatrix} 1 & r_1 & r_3 & r_3 \\ r_1 & 1 & r_3 & r_3 \\ r_3 & r_3 & 1 & r_2 \\ r_3 & r_3 & r_2 & 1 \end{pmatrix}.$$

In this scheme r_1 captures intra-coder agreement for the first radiologist (first two columns of the data matrix), r_2 captures intra-coder agreement for the second radiologist (third and fourth columns of the data matrix), and r_3 measures inter-coder agreement.

I fit a Gaussian copula model to the data, assuming the rows are iid, follow a categorical distribution, and have the underlying correlation structure just described. Rather than inferring the probabilities of the categorical response distribution, I compute the empirical probabilities, i.e., sample proportions, and focus on inferring the copula correlation matrix. This means my approach is not fully Bayesian. Extension to a fully Bayesian procedure would be straightforward. As for prior distributions, I assigned independent $\textsc{Uniform}(-1, 1)$ priors to r_1, r_2, and r_3.

The R code is shown below in Figure 3. I have omitted the details of the copula likelihood computation. The form of the copula likelihood for discrete outcomes can be found in Xue-Kun Song (2000), for example. The code refers to Miwa's algorithm (Miwa et al., 2003), which I used to approximate

multinormal probabilities. Since the likelihood is $\Theta(2^n)$ and approximation of multinormal probabilities is expensive, computing the likelihood is the computational bottleneck in this scenario.

Usage of the package's flagship function, `update_R`, appears at the end of the code snippet. Note that I chose `n = 800` as the grid size for the root finder. For some datasets a smaller grid is sufficient to avoid numerical problems. The default grid size is 100.

Results of the analysis are shown in Table 3. For each parameter I have provided the estimated posterior mean, the 95% highest posterior density interval, and the Monte Carlo standard error. We see that all three agreement measures are quite high (very close to 1) and the posterior is highly concentrated. I also used the samples to estimate, as an overall measure of agreement for the dataset, the Gaussian mutual information $-\frac{1}{2}\log\det(R)$. The estimate was 10.4, which indicates quite strong overall agreement.

Table 3: Results from analysis of the liver herniation data

Parameter	Estimated Posterior Mean	95% HPD Interval	MCSE
r_1	0.995278	$(0.991900, 0.997721)$	0.000085
r_2	0.999988	$(0.999966, 0.999998)$	0.000002
r_3	0.996699	$(0.995248, 0.998098)$	0.000052

4. Impact

By extending the parameter-at-a-time strategy proposed by Barnard et al. (2000), package `Rmodule` permits principled MCMC-based inference for arbitrarily structured correlation matrices and functionals thereof. This is immensely useful since modeling correlation structures is among the most common and important and difficult tasks in statistical analysis. Package `Rmodule` should greatly ease this burden by providing a flexible, modular, automated, and computationally efficient means of sampling correlation matrices for a very large class of statistical models. Examples include copula models (e.g., Gaussian, Student t, χ^2, Fisher), general location-scale models, shrinkage estimation of regression coefficients, and multivariate analysis of variance models.

I chose to produce package `Rmodule` because I wanted to use it to explore Bayesian Gaussian copula agreement models, such as the model I applied to the liver data in Section 3. Having `Rmodule` in my toolkit has allowed me to focus on the frontend of these explorations while outsourcing the backend work to the package. This advantageous modularization was illustrated by the liver data example, wherein sampling the copula correlation matrix could be done straightforwardly through repeated calls to function `Rmodule::update_R`, the setup for which was simple. I hope `Rmodule` will facilitate many similar explorations by both methodologists and practitioners.

5. Conclusions

R package `Rmodule` is a software tool that permits MCMC-based inference for arbitrarily structured correlation matrices. The package's functionality can easily be integrated into larger algorithms for which correlation modeling is an important subtask. I illustrated such an integration by using `Rmodule` to apply a Gaussian copula agreement model to ordinal radiologist-generated scores from an imaging study of congenital diaphragmatic hernia.

Planned enhancements/extensions of the package include permitting Hamiltonian Monte Carlo (Hoffman et al., 2014; Duane et al., 1987; Neal et al., 2011) updates in place of Gaussian random walk updates. Hamiltonian Monte Carlo uses ideas from the Hamiltonian formulation of classical mechanics (Hamilton, 1833) to realize a fast-mixing Markov chain. Thus an HMC-enabled version of `Rmodule::update_R` would require a much shorter sample path to provide the same quality of inference as the random walk.

Declaration of competing interest

The author declares that he has no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The data that support the illustration in this article are available from the author upon reasonable request.

References

Barnard, J., McCulloch, R., Meng, X.L., 2000. Modeling covariance matrices in terms of standard deviations and correlations, with application to shrinkage. Statistica Sinica , 1281–1311.

Duane, S., Kennedy, A., Pendleton, B.J., Roweth, D., 1987. Hybrid Monte Carlo. Physics Letters B 195, 216–222.

Eddelbuettel, D., Sanderson, C., 2014. RcppArmadillo: Accelerating R with high-performance C++ linear algebra. Computational Statistics and Data Analysis 71, 1054–1063.

Hamilton, W.R., 1833. On a General Method of Expressing the Paths of Light, & of the Planets, by the Coefficients of a Characteristic Function. PD Hardy.

Hoffman, M.D., Gelman, A., et al., 2014. The No-U-Turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo. J. Mach. Learn. Res. 15, 1593–1623.

Lewandowski, D., Kurowicka, D., Joe, H., 2009. Generating random correlation matrices based on vines and extended onion method. Journal of Multivariate Analysis 100, 1989–2001.

Longoni, M., Pober, B.R., High, F.A., 2020. Congenital diaphragmatic hernia overview, in: GeneReviews. University of Washington, Seattle.

Miwa, T., Hayter, A., Kuriki, S., 2003. The evaluation of general non-centred orthant probabilities. Journal of the Royal Statistical Society Series B: Statistical Methodology 65, 223–234.

Neal, R.M., et al., 2011. MCMC using Hamiltonian dynamics. Handbook of Markov chain Monte Carlo 2, 2.

Sanderson, C., Curtin, R., 2016. Armadillo: A template-based C++ library for linear algebra. Journal of Open Source Software 1, 26.

Xue-Kun Song, P., 2000. Multivariate dispersion models generated from Gaussian copula. Scandinavian Journal of Statistics 27, 305–320.

```
library(Rmodule)               # Load the package.
data = read.csv("liver.csv")   # Read in the liver data.
data = as.matrix(data)
n.c = ncol(data)

# Build the symbolic correlation matrix. The parameters are numbered 2, 3, and 4.

R = diag(1, n.c)
R[1, 2] = R[2, 1] = 2
R[3, 4] = R[4, 3] = 3
R[1, 3] = R[3, 1] = R[1, 4] = R[4, 1] = R[2, 3] = R[3, 2] = R[2, 4] = R[4, 2] = 4

# The following function computes the copula log likelihood. The first argument
# is the data matrix. The second argument is the correlation matrix (not the
# symbolic matrix but the matrix carrying the current state of the chain). The
# third argument is a list of additional arguments.

loglike.miwa = function(data, R, args)
{
    # details omitted in the interest of brevity
}
log.f = compiler::cmpfun(loglike.miwa)   # Compile to gain speed.

# Now compute the empirical probabilities.

prob = table(data)
prob = prob / sum(prob)

# The additional arguments for log.f are the number of steps for Miwa's method
# and the empirical probabilities.

log.f.args = list(steps = 16, prob = prob)

# The prior for the correlations is Uniform(-1, 1), and so the second argument,
# 'args', is not needed in this example.

logP = function(r, args) dunif(r, -1, 1, log = TRUE)

# Build the list of log priors and their arguments.

log.priors = list(logP, logP, logP)
log.priors.args = list(0, 0, 0)       # dummy arguments in this example

m = 100000                            # Simulate a sample path of length 100,000.
r.chain = matrix(0, m, 3)
r.chain[1, ] = c(0.98, 0.99, 0.96)    # good starting values
sigma = c(1, 4, 4)                    # Standard deviations for the three random
                                      # walk proposals.
for (i in 2:m)
    r.chain[i, ] = update_R(r.chain[i - 1, ], data, R,   # R in symbolic form
                            log.f = log.f,
                            log.f.args = log.f.args,
                            log.priors = log.priors,
                            log.priors.args = log.priors.args,
                            sigma = sigma,
                            n = 800)          # grid size for the root finder
```

Figure 3: Code for using `Rmodule` in analyzing the liver data