

Package: Rmodule (via r-universe)

October 11, 2024

Type Package

Title Automated Markov Chain Monte Carlo for Arbitrarily Structured Correlation Matrices

Version 1.0

Date 2023-08-18

Description Supports automated Markov chain Monte Carlo for arbitrarily structured correlation matrices. The user supplies data, a correlation matrix in symbolic form, the current state of the chain, a function that computes the log likelihood, and a list of prior distributions. The package's flagship function then carries out a parameter-at-a-time update of all correlation parameters, and returns the new state. The method is presented in Hughes (2023), in preparation.

License GPL (>= 2)

Imports Rcpp (>= 1.0.9), utils, Matrix

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.2.3

Encoding UTF-8

NeedsCompilation yes

Author John Hughes [aut, cre]

Maintainer John Hughes <drjphughesjr@gmail.com>

Repository CRAN

Date/Publication 2023-08-18 18:32:36 UTC

Contents

| | |
|--------------|----------|
| update_R | 2 |
| Index | 6 |

 update_R

Update the state vector of the correlation parameters.

Description

Update the state vector of the correlation parameters.

Usage

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

Arguments

| | |
|------------------------------|--|
| <code>r</code> | a p -vector of correlations, the current state of the Markov chain. |
| <code>data</code> | an $n \times d$ matrix such that the rows are iid outcomes for the study in question. |
| <code>R</code> | a $d \times d$ correlation matrix in symbolic form. The off-diagonal elements should be numbered from 2 to $p + 1$. |
| <code>log.f</code> | the log objective function, which must take the dataset, a correlation matrix, and perhaps additional arguments. |
| <code>log.f.args</code> | additional arguments for <code>log.f</code> . |
| <code>log.priors</code> | a list of log prior densities for the correlation parameters, each of which should accept a correlation and perhaps additional arguments. |
| <code>log.priors.args</code> | a list of additional arguments for the functions in <code>log.priors</code> . |
| <code>sigma</code> | a vector, the standard deviations of the Gaussian proposals for the p correlation parameters. This argument must have length 1 or length p . In the former case, all of the random-walk proposals have the same variance. In the latter case, the proposals have distinct variances. |
| <code>n</code> | a positive integer, the number of grid points to employ in root finding. The default value is 100, but in some cases a larger value may be required to avoid missing roots of the determinant function. |

Details

This function takes the current state of the chain and returns the next state. The correlation parameters are updated one at a time by way of a Metropolis-Hastings Gaussian random walk for each parameter. When the set of valid values for the proposal comprises a disconnected subset, i.e., two or more disjoint subintervals, of $(-1, 1)$, the Apes of Wrath algorithm is used to update the parameter in question.

Value

a p -vector, the new state of the chain.

Examples

```
# The following function computes HPD intervals.

hpd = function(x, alpha = 0.05)
{
  n = length(x)
  m = round(n * alpha)
  x = sort(x)
  y = x[(n - m + 1):n] - x[1:m]
  z = min(y)
  k = which(y == z)[1]
  c(x[k], x[n - m + k])
}

# The following function computes the log likelihood.

logL = function(data, R, args)
{
  n = nrow(data)
  Rinv = solve(R)
  detR = -0.5 * n * determinant(R, log = TRUE)$modulus
  qforms = -0.5 * sum(diag(data %*% Rinv %*% t(data)))
  f = detR + qforms
  if (f > 0)
    return(-1e6)
  f
}

# Use a Uniform(-1, 1) prior for each correlation.

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

# Build the list of priors and their arguments.

log.priors = list(logP, logP, logP, logP, logP)
log.priors.args = list(0, 0, 0, 0, 0)

# Simulate a dataset to work with. The dataset will have 32 observations,
# each of length 4. The outcomes will be generated from a Gaussian copula
# model having t-distributed marginal distributions. Then we Gaussianize
```

```

# the ranks for analysis.

n = 16
R = diag(1, 4, 4)
R[1, 2] = R[2, 1] = 2
R[3, 4] = R[4, 3] = 3
R[1, 3] = R[3, 1] = R[2, 4] = R[4, 2] = 4
R[1, 4] = R[4, 1] = 5
R[2, 3] = R[3, 2] = 6
r = c(-0.2, -0.2, -0.4, -0.7, 0.9)
block = R
for (j in 1:5)
  block[block == j + 1] = r[j]
blist = vector("list", n)
for (j in 1:n)
  blist[[j]] = block
C = t(chol(as.matrix(Matrix::bdiag(blist))))
set.seed(42)
z = as.vector(C %%% rnorm(n * 4))
u = pnorm(z)
y = qt(u, df = 3)
data = matrix(y, n, 4, byrow = TRUE)
data = matrix(qnorm(rank(data) / (n * 4 + 1)), n, 4)

# Simulate a sample path of length 1,000.

m = 1000
r.chain = matrix(0, m, 5)
r.chain[1, ] = 0
sigma = c(1, 1, 0.25, 2, 5) # proposal standard deviations
start = proc.time()
for (i in 2:m)
  r.chain[i, ] = update_R(r.chain[i - 1, ], data, R,
                        log.f = logL,
                        log.priors = log.priors,
                        log.priors.args = log.priors.args,
                        sigma = sigma,
                        n = 400)

stop = proc.time() - start
stop
stop[3] / m # 0.001 seconds per iteration on a 3.6 GHz 10-Core Intel Core i9

# Now show trace plots along with the truth and the 95% HPD interval.

dev.new()
plot(r.chain[, 1], type = "l")
abline(h = r[1], col = "orange", lwd = 3)
abline(h = hpd(r.chain[, 1]), col = "blue", lwd = 3)

dev.new()
plot(r.chain[, 2], type = "l")
abline(h = r[2], col = "orange", lwd = 3)
abline(h = hpd(r.chain[, 2]), col = "blue", lwd = 3)

```

```
dev.new()
plot(r.chain[, 3], type = "l")
abline(h = r[3], col = "orange", lwd = 3)
abline(h = hpd(r.chain[, 3]), col = "blue", lwd = 3)
```

```
dev.new()
plot(r.chain[, 4], type = "l")
abline(h = r[4], col = "orange", lwd = 3)
abline(h = hpd(r.chain[, 4]), col = "blue", lwd = 3)
```

```
dev.new()
plot(r.chain[, 5], type = "l")
abline(h = r[5], col = "orange", lwd = 3)
abline(h = hpd(r.chain[, 5]), col = "blue", lwd = 3)
```

Index

update_R, [2](#)