Package ‘adaptMCMC’

February 19, 2015

Type Package

Title Implementation of a generic adaptive Monte Carlo Markov Chain sampler

Version 1.1

Date 2012-06-22

Author Andreas Scheidegger, <andreas.scheidegger@eawag.ch>, <scheidegger.a@gmail.com>

Maintainer Andreas Scheidegger <andreas.scheidegger@eawag.ch>

Description This package provides an implementation of the generic adaptive Monte Carlo Markov chain sampler proposed by Vihola (2011).

License GPL (>= 2)

LazyLoad yes

Depends R (>= 2.14.1), parallel, coda, Matrix

Repository CRAN

Date/Publication 2012-06-22 09:34:11

NeedsCompilation no

R topics documented:

adaptMCMC-package .................................................. 2
convert.to.coda ......................................................... 2
MCMC ......................................................................... 4
MCMC.add.samples ..................................................... 6
MCMC.parallel .......................................................... 8

Index 10
adaptMCMC-package  

*Generic adaptive Monte Carlo Markov Chain sampler*

**Description**

This package provides an implementation of the generic adaptive Monte Carlo Markov chain sampler proposed by Vihola (2011).

**Details**

<table>
<thead>
<tr>
<th>Package: adaptMCMC</th>
<th>Type: Package</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version: 1.1</td>
<td>Date: 2012-06-22</td>
</tr>
<tr>
<td>License: GPL (&gt;= 2)</td>
<td>LazyLoad: yes</td>
</tr>
</tbody>
</table>

The workhorse function is `mcmc`. Chains can be updated with `mcmc.add.samples`. `MCMC.parallel` is a wrapper to generate independent chains on several CPU’s in parallel using `parallel`. `coda`-functions can be used after conversion with `convert.to.coda`.

**Author(s)**

Andreas Scheidegger, <andreas.scheidegger@eawag.ch> or <scheidegger.a@gmail.com>

**References**


**See Also**

`MCMC, MCMC.add.samples, MCMC.parallel, convert.to.coda`

The package HI provides an adaptive rejection Metropolis sampler with the function `arms`. See also `Metro_Hastings` of the MAdaptive package.

**Description**

`convert.to.coda`  

*Converts chain(s) into coda objects.*

Converts chain(s) produced by `MCMC` or `MCMC.parallel` into `coda` objects.
Usage

```
convert.to.coda(sample)
```

Arguments

- `sample`: output of MCMC or MCMC.parallel.

Details

Converts chain(s) produced by MCMC or MCMC.parallel so that they can be used with functions of the **coda** package.

Value

An object of the class `mcmc` or `mcmc.list`.

Author(s)

Andreas Scheidegger, <andreas.scheidegger@eawag.ch> or <scheidegger.a@gmail.com>

See Also

- `mcmc`
- `mcmc`
- `mcmc.list`

Examples

```r
## ------------------------------
## Banana shaped distribution

## log-pdf to sample from
p.log <- function(x) {
  B <- 0.03
  # controls 'bananacity'
}

## ------------------------------
## generate 200 samples
samp <- MCMC(p.log, n=200, init=c(0, 1), scale=c(1, 0.1),
              adapt=TRUE, acc.rate=0.234)

## ------------------------------
## convert in object of class 'mcmc'
samp.coda <- convert.to.coda(samp)

class(samp.coda)

## ------------------------------
## use functions of package 'coda'
```
require(coda)

plot(samp.coda)
cumuplot(samp.coda)

\begin{verbatim}
MCMC(Adaptive) Metropolis Sampler

Description


Usage

MCMC(p, n, init, scale = rep(1, length(init)),
adapt = !is.null(acc.rate), acc.rate = NULL, gamma = 0.5,
list = TRUE, n.start = 0, ...)

Arguments

p function that returns the log probability density to sample from. Must have two or more dimensions.

n number of samples.

init vector with initial values.

scale vector with the variances or covariance matrix of the jump distribution.

adapt if TRUE, adaptive sampling is used, if FALSE classic metropolis sampling, if a positive integer the adaption stops after adapt iterations.

acc.rate desired acceptance rate (ignored if adapt=FALSE)

gamma controls the speed of adaption. Should be between 0.5 and 1. A lower gamma leads to faster adaption.

list logical. If TRUE a list is returned otherwise only a matrix with the samples.

n.start iteration where the adaption starts. Only internally used.

... further arguments passed to p.

Details

The algorithm tunes the covariance matrix of the (normal) jump distribution to achieve the desired acceptance rate. Classic (non-adaptive) Metropolis sampling can be obtained by setting adapt=FALSE.

Note, due to the calculation for the adaption steps the sampler is rather slow. However, with a suitable jump distribution good mixing can be observed with less samples. This is crucial if the computation of p is slow.

The current implementation does not work for one-dimensional distributions.
\end{verbatim}
Value

If list=FALSE a matrix is with the samples.
If list=TRUE a list is returned with the following components:

- samples: matrix with samples
- log.p: vector with the (unnormalized) log density for each sample
- n.sample: number of generated samples
- acceptance.rate: acceptance rate
- adaption: either logical if adaption was used or not, or the number of adaption steps.
- sampling.parameters: a list with further sampling parameters. Mainly used by MCMC.add.samples().

Note

Due to numerical errors it can happen that the computed covariance matrix is not positive definite. In such a case the nearest positive definite matrix is calculated with nearPD() of the package Matrix.

Author(s)

Andreas Scheidegger, <andreas.scheidegger@eawag.ch> or <scheidegger.a@gmail.com>.
Thanks to David Pleydell for spotting an error in the previous version.

References


See Also

MCMC.parallel, MCMC.add.samples

The package HI provides an adaptive rejection Metropolis sampler with the function arms. See also Metro_Hastings of the MHadaptive package.

Examples

```r
## ------------------------
## Banana shaped distribution

## log-pdf to sample from
p.log <- function(x) {
  B <- 0.03            # controls 'bananacity'
}

## ------------------------
```
## MCMC.add.samples

Add samples to an existing chain.

### Description

Add samples to an existing chain produced by MCMC or MCMC.parallel.

### Usage

MCMC.add.samples(MCMC.object, n.update, ...)

---

```r
## generate samples

## 1) non-adaptive sampling
samp.1 <- MCMC(p.log, n=200, init=c(0, 1), scale=c(1, 0.1), 
               adapt=FALSE)

## 2) adaptive sampling
samp.2 <- MCMC(p.log, n=200, init=c(0, 1), scale=c(1, 0.1), 
               adapt=TRUE, acc.rate=0.234)

## summarize results
str(samp.2)
summary(samp.2$samples)

## covariance of last jump distribution
samp.2$cov.jump

## plot density and samples
x1 <- seq(-15, 15, length=80)
x2 <- seq(-15, 15, length=80)
d.banana <- matrix(apply(expand.grid(x1, x2), 1, p.log), nrow=80)
par(mfrow=c(1,2))
image(x1, x2, exp(d.banana), col=cm.colors(60), asp=1, main="no adaption")
contour(x1, x2, exp(d.banana), add=TRUE, col=gray(0.6))
lines(samp.1$samples, type='b', pch=3)

image(x1, x2, exp(d.banana), col=cm.colors(60), asp=1, main="with adaption")
contour(x1, x2, exp(d.banana), add=TRUE, col=gray(0.6))
lines(samp.2$samples, type='b', pch=3)
```
**MCMC.add.samples**

### Arguments

- **MCMC.object**: a list produced by MCMC or MCMC.parallel with option list = TRUE.
- **n.update**: number of additional samples.
- **...**: further arguments passed to `p`.

### Details

Only objects generated with the option list = TRUE can be updated.

A list of chains produced by MCMC.parallel can be updated. However, the calculations are not performed in parallel (i.e. only a single CPU is used).

### Value

A updated version of `MCMC.object`.

### Author(s)

Andreas Scheidegger, <andreas.scheidegger@eawag.ch> or <scheidegger.a@gmail.com>

### See Also

MCMC, MCMC.parallel

### Examples

```r
## -----------------------------
## Banana shaped distribution

## log-pdf to sample from
p.log <- function(x) {
  B <- 0.03
  # controls 'bananacity'
}

## -----------------------------
## generate 200 samples
samp <- MCMC(p.log, n=200, init=c(0, 1), scale=c(1, 0.1),
               adapt=TRUE, acc.rate=0.234, list=TRUE)

## -----------------------------
## add 200 to the existing chain
samp <- MCMC.add.samples(samp, n.update=200)
str(samp)
```
**Description**

A wrapper function to generate several independent Markov chains by setting up cluster on a multi-core machine. The function is based on the `parallel` package.

**Usage**

```r
MCMC.parallel(p, n, init, n.chain = 4, n.cpu, packages = NULL, dyn.libs=NULL, 
    scale = rep(1, length(init)), adapt = !is.null(acc.rate), 
    acc.rate = NULL, gamma = 0.55, list = TRUE, ...)
```

**Arguments**

- **p**: function that returns the log probability density to sample from. Must have two or more dimensions.
- **n**: number of samples.
- **init**: vector with initial values.
- **n.chain**: number of independent chains.
- **n.cpu**: number of CPUs that should be used in parallel.
- **packages**: vector with name of packages to load into each instance. (Typically, all packages on which `p` depends.)
- **dyn.libs**: vector with name of dynamic link libraries (shared objects) to load into each instance. The libraries must be located in the working directory.
- **scale**: vector with the variances or covariance matrix of the jump distribution.
- **adapt**: if TRUE, adaptive sampling is used, if FALSE classic Metropolis sampling, if a positive integer the adaption stops after `adapt` iterations.
- **acc.rate**: desired acceptance rate (ignored if `adapt=FALSE`)
- **gamma**: controls the speed of adaption. Should be between 0.5 and 1. A lower gamma leads to faster adaption.
- **list**: logical. If TRUE a list of lists is returned otherwise a list of matrices with the samples.
- **...**: further arguments passed to `p`

**Details**

This function is just a wrapper to use `MCMC` in parallel. It is based on `parallel`. Obviously, the application of this function makes only sense on a multi-core machine.

**Value**

A list with a list or matrix for each chain. See `MCMC` for details.
Author(s)

Andreas Scheidegger, <andreas.scheidegger@eawag.ch> or <scheidegger.a@gmail.com>

See Also

MCMC

Examples

```r
## -----------------------
## Banana shaped distribution

## log-pdf to sample from
p.log <- function(x) {
  B <- 0.03 # controls 'bananacity'
  -x[1]^2/200 - 1/2*(x[2]+B*x[1]-100*B)^2
}

## -----------------------
## generate samples
## compute 4 independent chains on 2 CPU's (if available) in parallel

samp <- MCMC.parallel(p.log, n=200, init=c(x1=0, x2=1),
  n.chain=4, n.cpu=2, scale=c(1, 0.1),
  adapt=TRUE, acc.rate=0.234)

str(samp)
```
Index

adaptMCMC (adaptMCMC-package), 2
adaptMCMC-package, 2
arms, 2, 5
convert.to.coda, 2, 2

MCMC, 2, 3, 4, 7–9
mcmc, 3
MCMC.add.samples, 2, 5, 6
mcmc.list, 3
MCMC.parallel, 2, 5, 7, 8
Metro_Hastings, 2, 5