Package: demc2 (via r-universe)

November 1, 2024

Type Package

Title demc2 implements adaptive MCMC on real parameter spaces via Differential Evolution Markov Chain

Version 1.0

Date 2015-02-09

Author Cajo J.F. ter Braak

Maintainer Cajo J.F. ter Braak <cajo.terbraak@wur.nl>

Description The package implements both the origal Markovian demc version of ter Braak (2006) and the egodic demc-zc version of ter Braak & Vrugt (2008) using learning from the past and the snooker update. Convergence can be checked with coda. The demc package allows restarts.

License GPL (>= 2)

RoxygenNote 6.1.1

Repository https://cajoterbraak.r-universe.dev

RemoteUrl https://github.com/CajoterBraak/demc2

RemoteRef HEAD

RemoteSha 6094ce74fb8238d19d693523413460b56604a1ca

Contents

demc *generates MCMC samples using Differential Evolution Markov Chain (ter Braak 2006)*

Description

demc implements multi-chain adaptive MCMC on real parameter spaces via Differential Evolution Markov Chain. It allows restart from a previous run. It is the only (?) adaptive MCMC method that is really Markovian and not just ergodic. Required input: starting position for each chain (X) and a unnormalized logposterior function (FUN).

Usage

```
demc(X, FUN, blocks, f = 2.38, n.generation = 1000, n.thin = 1,
  n.burnin = 0, eps.add = 0, p.f.is.1 = 0.1, verbose = FALSE,
  logfitness_X, ...)
```

Arguments

X	matrix of initial values or demc object resulting from a previous run. If matrix, initial parameter values for N chains (rows) for each of the parameters (columns). See Details for choice of N.
FUN	function specifying the (unnormalized) logposterior or target. FUN(theta ,...) with theta a d vector of parameter values and ... other arguments to FUN (e.g. NULL, data, ..). The value of FUN should a single numeric value.
blocks	list of sets of parameters, where each set contains the indices (or names if X has rownames) of variables in theta that are updated jointly, e.g. blocks= list(); blocks[[1]] = c(1,3); blocks[[2]] = c(2,4). The default uses a single block (joint update of all parameters).
f	value specifying the scale of the updates. The scale used is f/sqrt(2k), where k the number of parameters in a block (k=d if there only one block); f can be a vector of length(blocks) to set differential scaling factors for blocks.
n.generation	integer, number of generations, each one generating ncol(X) samples.
n.thin	integer, thinning number, e.g 3 stores every 3rd generation (default 1, no thinning).
n.burnin	integer, number of initial generations that is not stored (default 0).
eps.add	real value . Standard deviation of the (additive) independent random normal step added to the DE update. that guarantees that all parameter values can be reached. Must be postive to garantee that all positions in the space can be reached. For targets (posteriors) without gaps, it can set small or even to 0 (default 0).
p.f.is.1	probability (default 0.1) of using scale 0.98 in the parallel update instead of f/sqrt(2d). Allows exploration of multi-modal posteriors.
verbose	logical (default FALSE). No used currently
logfitness_X	Can be missing. If set, logposterior values for the columns of X; can save some computation if known.

Details

demc This function implements the method of ter Braak (2006) using the differential evolution update (also known as the paralled direction sampling update). See also demc_zs for an egodic variant with learning from the past. The number of initial positions (N, columns of X) should be larger than the number of paramaters (d) in each block. The advice is N=2d. So fewer starting positions are required by specifying blocks with few (or even single) parameters in each block.

Value

an S3 object of class demc which is a list with

$Draws d x (Nchain*n) matrix with n the number of retained simulations [post process with summary or as.coda]. matrix(Draws[p,],nrow= Nchain) is an Nchain x n array (for each parameter p in 1:d)

$accept.prob.mat accept probability matrix of blocks by chains.

$X.final matrix of parameter values of the last generation (same shape as initial X).

$logfitness.X.final vector of logposterior values for columns of X.final (useful for restarting).

$Nchain number of chains.

$demc_zs logical set of FALSE.

References

ter Braak, C. J. F. (2006). A Markov Chain Monte Carlo version of the genetic algorithm Differential Evolution: easy Bayesian computing for real parameter spaces. Statistics and Computing, 16, 239-249.http://edepot.wur.nl/26336

---

demc2 — *Adaptive MCMC on real parameter spaces via Differential Evolution Markov Chain*

Description

demc2 implements both the origal Markovian demc version of ter Braak (2006) and the egodic demc_zc version of ter Braak & Vrugt (2008) using learning from the past and the snooker update. Convergence can be checked with coda. The demc package allows restarts.

Details

The functions that you're likely need from **demc** are demc and demc_zs for generating samples and as.coda to be able to use package coda for convergence checks

References

ter Braak, C. J. F. (2006). A Markov Chain Monte Carlo version of the genetic algorithm Differential Evolution: easy Bayesian computing for real parameter spaces. Statistics and Computing, 16, 239-249.http://edepot.wur.nl/26336 ter Braak C. J. F., and Vrugt J. A. (2008). Differential Evolution Markov Chain with snooker updater and fewer chains. Statistics and Computing, 18 (4), 435-446. http://dx.doi.org/10.1007/s11222-008-9104-9

demc_zs *generates MCMC samples using Differential Evolution Markov Chain (ter Braak & Vrugt 2008)*

Description

demc_zs implements multi-chain adaptive MCMC on real parameter spaces via Differential Evolution Markov Chain with learning from the past and the snooker update. It allows restart from a previous run. Required input: starting position for each chain (X) and a unnormalized logposterior function (FUN).

Usage

```
demc_zs(Nchain = 3, Z, FUN, X, blocks, f = 2.38, pSnooker = 0.1,
  p.f.is.1 = 0.1, n.generation = 1000, n.thin, n.burnin = 0,
  eps.mult = 0.2, eps.add = 0, verbose = FALSE, logfitness_X, ...)
```

Arguments

Nchain	number of (parallel) chains.
Z	matrix of initial values or demc object resulting from a previous run. If matrix, initial parameter values for N chains (columns) for each of the parameters (rows). See Details for choice of N.
FUN	function specifying the (unnormalized) logposterior or target. FUN(theta ,...) with theta a d vector of parameter values and ... other arguments to FUN (e.g. NULL, data, ..). The value of FUN should a single numeric value.
X	optional matrix of initial values (d x Nchain); default: X.final of a previous run or the first Nchain columns of Z if Z is a matrix. If matrix, initial parameter values for N chains (columns) for each of the parameters (rows). See Details for choice of N.
blocks	list of sets of parameters, where each set contains the indices (or names if X has rownames) of variables in theta that are updated jointly, e.g. blocks= list(); blocks[[1]] = c(1,3); blocks[[2]] = c(2,4). The default uses a single block (joint update of all parameters).
f	value specifying the scale of the updates (default 2.38). The scale used is f/sqrt(2k), where k the number of parameters in a block (k=d if there only one block) in the parallel update and k = 1 in the snooker update. f can be a vector of length(blocks) to set differential scaling factors for blocks.
pSnooker	probability of snooker update (default 0.1); 1-pSnooker: prob of the differential evolution (parallel) update
p.f.is.1	probability (default 0.1) of using scale 0.98 in the parallel update instead of f/sqrt(2d). Allows exploration of multi-modal posteriors.
n.generation	integer, number of generations, each one generating ncol(X) samples.

n.thin	integer, thinning number, e.g 3 stores every 3rd generation (default 1, no thinning). In demc_zs and demc_zs_p this value should be set as large as possible, because of autocorrelation that makes the method to adapt to quickly/to the wrong scale and orientation as the many correlated samples overrule any decent initial sample
n.burnin	integer, number of initial generations that is not stored (default 0).
eps.mult	real value (default 0.2). It adds scaled uncorrelated noise to the proposal, by multiplying the scale of the update with a d-dimension uniform variate [1-eps,1+eps].
eps.add	real value . Standard deviation of the (additive) independent random normal step added to the DE update. that guarantees that all parameter values can be reached. Must be postive to garantee that all positions in the space can be reached. For targets (posteriors) without gaps, it can set small or even to 0 (default 0).
verbose	logical (default FALSE). No used currently
logfitness_X	Can be missing. If set, logposterior values for the columns of X; can save some computation if known.

Details

demc_zs This function implements the method of ter Braak & Vrugt (2008) using learning of the past, the differential evolution (parelle direction) update and the snooker update. The number of initial positions (N, columns of Z) should be much larger than the number of paramaters (d) in each block. The advice is N=10d. So fewer initial values are required by specifying blocks with few (or even single) parameters in each block.

Value

an S3 object of class demc which is a list with

$Draws d x (Nchain*n) matrix with n the number of retained simulations [post process with summary or as.coda]. matrix(Draws[p,],nrow= Nchain) is an Nchain x n array (for each parameter p in 1:d)

$accept.prob.mat accept probability matrix of blocks by chains.

$X.final matrix of parameter values of the last generation (same shape as initial X).

$logfitness.X.final vector of logposterior values for columns of X.final (useful for restarting).

$Nchain number of chains.

$demc_zs logical set of FALSE.

References

ter Braak C. J. F., and Vrugt J. A. (2008). Differential Evolution Markov Chain with snooker updater and fewer chains. Statistics and Computing, 18 (4), 435-446. http://dx.doi.org/10.1007/s11222-008-9104-9

demc_zs_p *generates MCMC samples using Differential Evolution Markov Chain (ter Braak & Vrugt 2008)*

Description

demc_zs implements multi-chain adaptive MCMC on real parameter spaces via Differential Evolution Markov Chain with learning from the past and the snooker update. It allows restart from a previous run. Required input: starting position for each chain (X) and a unnormalized logposterior function (FUN).

Usage

```
demc_zs_p(Nchain = 3, Z, FUN, X, blocks, f = 2.38, pSnooker = 0.1,
  p.f.is.1 = 0.1, n.generation = 1000, n.thin, n.burnin = 0,
  eps.mult = 0.2, eps.add = 0, verbose = FALSE, logfitness_X,
  pClust = NULL, ...)
```

Arguments

Nchain	number of (parallel) chains.
Z	matrix of initial values or demc object resulting from a previous run. If matrix, initial parameter values for N chains (columns) for each of the parameters (rows). See Details for choice of N.
FUN	function specifying the (unnormalized) logposterior or target. FUN(theta ,...) with theta a d vector of parameter values and ... other arguments to FUN (e.g. NULL, data, ..). The value of FUN should a single numeric value.
X	optional matrix of initial values (d x Nchain); default: X.final of a previous run or the first Nchain columns of Z if Z is a matrix. If matrix, initial parameter values for N chains (columns) for each of the parameters (rows). See Details for choice of N.
blocks	list of sets of parameters, where each set contains the indices (or names if X has rownames) of variables in theta that are updated jointly, e.g. blocks= list(); blocks[[1]] = c(1,3); blocks[[2]] = c(2,4). The default uses a single block (joint update of all parameters).
f	value specifying the scale of the updates (default 2.38). The scale used is f/sqrt(2k), where k the number of parameters in a block (k=d if there only one block) in the parallel update and k = 1 in the snooker update. f can be a vector of length(blocks) to set differential scaling factors for blocks.
pSnooker	probability of snooker update (default 0.1); 1-pSnooker: prob of the differential evolution (parallel) update
p.f.is.1	probability (default 0.1) of using scale 0.98 in the parallel update instead of f/sqrt(2d). Allows exploration of multi-modal posteriors.
n.generation	integer, number of generations, each one generating ncol(X) samples.

n.thin	integer, thinning number, e.g 3 stores every 3rd generation (default 1, no thinning).
n.burnin	integer, number of initial generations that is not stored (default 0).
eps.mult	real value (default 0.2). It adds scaled uncorrelated noise to the proposal, by multiplying the scale of the update with a d-dimension uniform variate [1-eps,1+eps].
eps.add	real value . Standard deviation of the (additive) independent random normal step added to the DE update. that guarantees that all parameter values can be reached. Must be postive to garantee that all positions in the space can be reached. For targets (posteriors) without gaps, it can set small or even to 0 (default 0).
verbose	logical (default FALSE). No used currently
logfitness_X	Can be missing. If set, logposterior values for the columns of X; can save some computation if known.

Details

demc_zs This function implements the method of ter Braak & Vrugt (2008) using learning of the past, the differential evolution (parelle direction) update and the snooker update. The number of initial positions (N, columns of Z) should be much larger than the number of paramaters (d) in each block. The advice is N=10d. So fewer initial values are required by specifying blocks with few (or even single) parameters in each block.

Value

an S3 object of class demc which is a list with

$Draws d x (Nchain*n) matrix with n the number of retained simulations [post process with summary or as.coda]. matrix(Draws[p,],nrow= Nchain) is an Nchain x n array (for each parameter p in 1:d)

$accept.prob.mat accept probability matrix of blocks by chains.

$X.final matrix of parameter values of the last generation (same shape as initial X).

$logfitness.X.final vector of logposterior values for columns of X.final (useful for restarting).

$Nchain number of chains.

$demc_zs logical set of FALSE.

References

ter Braak C. J. F., and Vrugt J. A. (2008). Differential Evolution Markov Chain with snooker updater and fewer chains. Statistics and Computing, 18 (4), 435-446. http://dx.doi.org/10.1007/s11222-008-9104-9

draws.demc *Extracts the draws from a demc object*

Description

Extracts the draws from a demc object

Usage

```
draws.demc(object, keep.all = 0.9, thin = 1)
```

Arguments

object	demc object
keep.all	logical or fraction of draws to keep. If fraction, the one-complement is discarded as burnin. If logical the first half of the draws are discarded. Default 0.9.
thin	integer (default 1), keep every thin-th draw

Value

a dataframe with parameters as variables and as rows the kept samples

summary.demc *Summarizes a demc object*

Description

Summarizes a demc object

Usage

```
## S3 method for class 'demc'
summary(a, trans = NULL, keep.all = 0.9,
  Rupper.keep = FALSE)
```

Arguments

trans	a vector of length k (default NULL): "" if no transformation, or "log" or "logit"
keep.all	logical or fraction of draws to keep. If fraction, the one-complement is discarded as burnin. If logical the first half of the draws are discarded. Default 0.9.
Rupper.keep	logical (default FALSE); if TRUE: keep Rupper, (Otherwise don't display it.)
object	demc object

Details

The summary is adapted from the monitor function written by Andrew Gelman.

Value

an S3 object of class `demc` which is a list with

$summary $accept.prob.mat

Index