Package ‘metropolis’

January 23, 2020

Title The Metropolis Algorithm
Version 0.1.5
Date 2020-01-12
Author Alexander Keil [aut, cre]
Maintainer Alexander Keil <akeil@unc.edu>
Description Learning and using the Metropolis algorithm for
Bayesian fitting of a generalized linear model. The package vignette
includes examples of hand-coding a logistic model using several
variants of the Metropolis algorithm. The package also contains R
functions for simulating posterior distributions of Bayesian
generalized linear model parameters using guided, adaptive,
guided-adaptive and random walk Metropolis algorithms. The random walk
Metropolis algorithm was originally described in Metropolis et al

License GPL (>= 2)
Depends coda, R (>= 3.5.0)
Imports stats
Suggests knitr
VignetteBuilder knitr
Encoding UTF-8
Language en-US
LazyData true
RoxygenNote 7.0.2
NeedsCompilation no
Repository CRAN
Date/Publication 2020-01-23 17:30:07 UTC
R topics documented:

as.mcmc.metropolis.samples ........................................... 2
expit ............................................................................. 3
logistic_ll ................................................................. 3
magfields ................................................................. 4
metropolis.control ....................................................... 4
metropolis_glm ........................................................... 5
normal_ll ................................................................. 7
plot.metropolis.samples ............................................... 8
print.metropolis.samples .............................................. 8
summary.metropolis.samples ........................................... 9

Index 11

as.mcmc.metropolis.samples

Convert glm_metropolis output to 'mcmc' object from package coda

Description

Allows use of useful functions from 'coda' package

Usage

## S3 method for class 'metropolis.samples'
as.mcmc(x, ...)

Arguments

x an object from the function "metropolis"
...

not used

Details

TBA

Value

An object of type "mcmc" from the coda package

Examples

library("coda")
dat = data.frame(y = rbinom(100, 1, 0.5), x1=runif(100), x2 = runif(100))
res = metropolis_glm(y ~ x1 + x2, data=dat, family=binomial(), iter=10000, burnin=3000,
adapt=TRUE, guided=TRUE, block=FALSE)
res2 = as.mcmc(res)
summary(res2)
expit  

inverse logit transform

Description
Inverse logit transform

Usage
expit(mu)

Arguments
mu log-odds

Value
returns a scalar or vector the same length as mu with values that are the inverse logit transform of mu

Examples
logodds = rnorm(10)
expit(logodds)
logodds = log(1.0)
expit(logodds)

logistic_ll  

logistic log likelihood

Description
logistic log likelihood

Usage
logistic_ll(y, X, par)

Arguments
y binary outcome
X design matrix
par vector of model coefficients

Value
a scalar quantity proportional to a binomial likelihood with logistic parameterization, given y, X, and par
**magfields**


**Description**


**Usage**

```r
magfields
```

**Format**

A data frame with 234 rows and 2 variables:

- `y` childhood leukemia
- `x` exposure to magnetic field

---

**metropolis.control**

**Description**

metropolis.control

**Usage**

```r
metropolis.control(  
adapt.start = 25,  
adapt.window = 200,  
adapt.update = 25,  
min.sigma = 0.001,  
prop.sigma.start = 1,  
scale = 2.4  
)
```
Arguments

- `adapt.start`: start adapting after this many iterations; set to iter+1 to turn off adaptation
- `adapt.window`: base acceptance rate on maximum of this many iterations
- `adapt.update`: frequency of adaptation
- `min.sigma`: minimum of the proposal distribution standard deviation (if set to zero, posterior may get stuck)
- `prop.sigma.start`: starting value, or fixed value for proposal distribution's standard deviation
- `scale`: scale value for adaptation (how much should the posterior variance estimate be scaled by?). Scale/sqrt(p) is used in `metropolis_glm` function, and Gelman et al. (2014, ISBN: 9781584883883) recommend a scale of 2.4

@return A list of parameters used in fitting with the following named objects: `adapt.start`, `adapt.window`, `adapt.update`, `min.sigma`, `prop.sigma.start`, `scale`

---

**metropolis_glm**

*Use the Metropolis Hastings algorithm to estimate Bayesian glm parameters*

---

**Description**

This function carries out the Metropolis algorithm.

**Usage**

```r
metropolis_glm(
    f,
    data,
    family = binomial(),
    iter = 100,
    burnin = round(iter/2),
    pm = NULL,
    pv = NULL,
    chain = 1,
    prop.sigma.start = 0.1,
    inits = NULL,
    adaptive = TRUE,
    guided = FALSE,
    block = TRUE,
    saveproposal = FALSE,
    control = metropolis.control()
)
```
Arguments

- **f**: an R style formula (e.g. \( y \sim x1 + x2 \))
- **data**: an R data frame containing the variables in f
- **family**: R glm style family that determines model form: normal() or binomial()
- **iter**: number of iterations after burnin to keep
- **burnin**: number of iterations at the beginning to throw out (also used for adaptive phase)
- **pm**: vector of prior means for normal prior on log(scale) (if applicable) and regression coefficients (set to NULL to use uniform priors)
- **pv**: vector of prior variances for normal prior on log(scale) (if applicable) and regression coefficients (set to NULL to use uniform priors)
- **chain**: chain id [plan to deprecate]
- **prop.sigma.start**: proposal distribution standard deviation (starting point if adapt=TRUE)
- **inits**: NULL, a vector with length equal to number of parameters (intercept + x + scale [gaussian() family only model only]), or "glm" to set priors based on an MLE fit
- **adaptive**: logical, should proposal distribution be adaptive? (TRUE usually gives better answers)
- **guided**: logical, should Gustafson’s "guided" algorithm be used (TRUE usually gives better answers)
- **block**: logical or a vector that sums to total number of parameters (e.g. if there are 4 random variables in the model, including intercept, then block=c(1,3) will update the intercept separately from the other three parameters.) If TRUE, then updates each parameter 1 by 1. Using "guide=TRUE" with blocking=\(<\text{vector}>\) is not advised
- **saveproposal**: (logical, default=FALSE) save the rejected proposals (block=TRUE only)?
- **control**: parameters that control fitting algorithm. See metropolis.control()

Details

Implements the Metropolis algorithm, which allows user specified proposal distributions or implements an adaptive algorithm as described by Gelman et al. (2014, ISBN: 9781584883883). This function also allows the "Guided" Metropolis algorithm of Gustafson (1998) \(<\text{doi:10.1023/A:100880707168}>\). Note that by default all parameters are estimated simultaneously via "block" sampling, but this default behavior can be changed with the "block" parameter. When using guided=TRUE, block should be set to FALSE.

Value

An object of type "metropolis.samples" which is a named list containing posterior MCMC samples as well as some fitting information.
Examples

dat = data.frame(y = rbinom(100, 1, 0.5), x1=runif(100), x2 = runif(100))

res = metropolis_glm(y ~ x1 + x2, data=dat, family=binomial(), iter=1000, burnin=3000, adapt=TRUE, guided=TRUE, block=FALSE)
res
summary(res)
apply(res$parms, 2, mean)

glm(y ~ x1 + x2, family=binomial(), data=dat)

dat = data.frame(y = rnorm(100, 1, 0.5), x1=runif(100), x2 = runif(100), x3 = rpois(100, .2))

res = metropolis_glm(y ~ x1 + x2 + factor(x3), data=dat, family=gaussian(), inits="glm", iter=10000, burnin=3000, adapt=TRUE, guide=TRUE, block=FALSE)
apply(res$parms, 2, mean)

glm(y ~ x1 + x2+ factor(x3), family=gaussian(), data=dat)

---

**normal_ll**  
*Gaussian log likelihood*

**Description**

Gaussian log likelihood

**Usage**

`normal_ll(y, X, par)`

**Arguments**

- `y`: binary outcome
- `X`: design matrix
- `par`: vector of gaussian scale parameter followed by model coefficients

**Value**

a scalar quantity proportional to a normal likelihood with linear parameterization, given y, X, and par
plot.metropolis.samples

Plot the output from the metropolis function

Description

This function allows you to summarize output from the metropolis function.

Usage

## S3 method for class 'metropolis.samples'
plot(x, keepburn = FALSE, parms = NULL, ...)

Arguments

x the outputted object from the "metropolis_glm" function
keepburn keep the burnin iterations in calculations (if adapt=TRUE, keepburn=TRUE
parms names of parameters to plot (plots the first by default, if TRUE, plots all)
... other arguments to plot

Details

TBA

Value

None

Examples

dat = data.frame(y = rbinom(100, 1, 0.5), x1=runif(100), x2 = runif(100))
res = metropolis_glm(y ~ x1 + x2, data=dat, family=binomial(), iter=10000, burnin=3000,
adapt=TRUE, guided=TRUE, block=FALSE)
plot(res)

print.metropolis.samples

Print a metropolis.samples object

Description

This function allows you to summarize output from the "metropolis_glm" function.
**Usage**

```r
## S3 method for class 'metropolis.samples'
print(x, ...)
```

**Arguments**

- `x`: a "metropolis.samples" object from the function "metropolis_glm"
- `...`: not used.

**Details**

None

**Value**

An unmodified "metropolis.samples" object (invisibly)

---

**Summary.metropolis.samples**

*Summarize a probability distribution from a Markov Chain*

---

**Description**

This function allows you to summarize output from the metropolis function.

**Usage**

```r
## S3 method for class 'metropolis.samples'
summary(object, keepburn = FALSE, ...)
```

**Arguments**

- `object`: an object from the function "metropolis"
- `keepburn`: keep the burnin iterations in calculations (if adapt=TRUE, keepburn=TRUE will yield potentially invalid summaries)
- `...`: not used

**Details**

TBA

**Value**

returns a list with the following fields:
- `nsamples`: number of simulated samples
- `sd`: standard deviation of parameter distributions
- `se`: standard deviation of parameter distribution means
- `ESS_parms`: effective sample size of parameter distribution means
- `postmean`: posterior means and normal based 95
- `postmedian`: posterior medians and percentile based 95
- `postmode`: posterior modes and highest posterior density based 95
Examples

dat = data.frame(y = rbinom(100, 1, 0.5), x1=runif(100), x2 = runif(100))
res = metropolis_glm(y ~ x1 + x2, data=dat, family=binomial(), iter=10000, burnin=3000,
adapt=TRUE, guided=TRUE, block=FALSE)
summary(res)
Index

*Topic datasets
  magfields, 4
as.mcmc.metropolis.samples, 2
expit, 3
logistic_ll, 3
magfields, 4
metropolis.control, 4
metropolis_glm, 5
normal_ll, 7
plot.metropolis.samples, 8
print.metropolis.samples, 8
summary.metropolis.samples, 9