Package 'BayesSummaryStatLM'

March 3, 2015

Type Package

Title MCMC Sampling of Bayesian Linear Models via Summary Statistics

Version 1.0-1

Date 2015-01-01

Author Evgeny Savel'ev, Alexey Miroshnikov, Erin Conlon

Maintainer Evgeny Savel'ev <savelev@vt.edu>

Description Methods for generating Markov Chain Monte Carlo (MCMC) posterior samples of Bayesian linear regression model parameters that require only summary statistics of data as input. Summary statistics are useful for systems with very limited amounts of physical memory. The package provides two functions: one function that computes summary statistics of data and one function that carries out the MCMC posterior sampling for Bayesian linear regression models where summary statistics are used as input. The function read.regress.data.ff utilizes the R package 'ff' to handle data sets that are too large to fit into a user's physical memory, by reading in data in chunks.

Depends R (>= 3.1.1), mvnfast, ff

License GPL (>= 2)

NeedsCompilation no

Repository CRAN

Date/Publication 2015-03-03 01:13:40

R topics documented:

`bayes.regress` *MCMC posterior sampling of Bayesian linear regression model parameters using only summary statistics*

Description

This function generates MCMC posterior samples of the Bayesian linear regression model parameters, using only summary statistics $X'X$, $X'Y$ and $Y'Y$ (e.g. calculated by the function `read.regress.data.ff()` in this package). The samples are generated according to the user specified choices of prior distributions, hyperprior distributions and fixed parameter values where required; the user also specifies starting values for unknown model parameters.

Usage

```
bayes.regress(data.values=NULL,
              beta.prior=list("flat"),
              sigmasq.prior=list("inverse.gamma", 1.0, 1.0, 1.0),
              Tsamp.out=1000, zero.intercept=FALSE)
```

Arguments

`data.values` a list with four (optionally five) components, which are created by the function `read.regress.data.ff()` (in this package):

- `xtx`: a square matrix that stores the product $X'X$, where X is the data from predictor columns with a leading column of 1's for the y-intercept term.
- `xty`: a column vector that stores the product $X'Y$, where X is the same as above and Y is a column of response data values.
- `yty`: a scalar value that stores the product $Y'Y$, where Y is the same as above.
- `numsamp.data`: an integer equal to the number of data values of the predictor variables X.
- `xtx.inv` (optional): the inverse of the matrix `xtx` that is used for the "Uniform" prior distribution for β to speed up computations if the function is used repeatedly with the same `xtx`. If omitted, this inverse will be computed automatically. This component is ignored for other prior distributions.

`beta.prior` a list that specifies the characteristics of the prior distribution for β, the vector of coefficients of the Bayesian linear regression model. There are three possible types:

- `flat`: Uniform distribution.
- `mvnorm.known`: Multivariate Normal with known mean vector μ and known covariance matrix C.
- `mvnorm.unknown`: Multivariate Normal with unknown mean vector μ and unknown covariance matrix C. This prior also includes the hyperpriors for μ and C, where μ~ Multivariate Normal(η, D), and $C^{-1} \sim$ Wishart(d.f. = λ, scale matrix = V); η, D, λ, V assumed known.

In each of these three prior types, the list has a different structure, as follows:

- `beta.prior=list(type = "flat")`: a Uniform prior distribution for β; no other specification is necessary. This prior distribution is used by default.
- `beta.prior=list(type = "mvnorm.known", mean.mu = ..., cov.C = ..., prec.Cinv = ... )`
  - `mean.mu`: the fixed known prior mean vector μ for the Multivariate Normal prior of β. The default is a vector of 0's with length equal to the length of β.
  - `cov.C`: the fixed known prior covariance matrix C for the Multivariate Normal prior of β. The default is an identity matrix with dimension equal to the length of β.
  - `prec.Cinv`: the inverse of the covariance matrix C above. If `cov.C` is not specified, `prec.Cinv` is assigned the identity matrix by default, with dimension equal to the length of β.

  It is advised to supply `prec.Cinv` matrix and omit `cov.C` for speeding up the algorithm. In case both are supplied, the algorithm gives preference to `prec.Cinv`.
- `beta.prior=list(type = "mvnorm.unknown", mu.hyper.mean.eta = ..., mu.hyper.prec.Dinv = ..., Cinv.hyper.df.lambda = ..., Cinv.hyper.invscale.Vinv = ..., mu.init = ..., Cinv.init = ...)`
  - `mu.hyper.mean.eta`: the fixed known hyperparameter mean vector η for the Multivariate Normal hyperprior mean μ. The default is a vector of 0's with length equal to the length of β.
  - `mu.hyper.prec.Dinv`: the fixed known hyperparameter precision matrix D^{-1} for the Multivariate Normal hyperprior mean μ. The default is an identity matrix with dimension equal to the length of β.
  - `Cinv.hyper.df.lambda`: the fixed known degrees of freedom λ for the Wishart hyperprior for C^{-1}. The default value is the length of β .
  - `Cinv.hyper.invscale.Vinv`: the fixed known hyperparameter inverse scale matrix V^{-1} for the Wishart hyperprior for C^{-1}. The default is an identity matrix with dimension equal to the length of β.
  - `mu.init`: initial value for μ for the MCMC chain. The default is a vector of 1's with length equal to the length of β.
  - `Cinv.init`: initial value for C^{-1} for the MCMC chain. The default is an identity matrix with dimension equal to the length of β.

For all three of the above `beta.prior` distributions, only the `type` is mandatory; the remaining parameters are assigned default values if omitted.

`sigmasq.prior` a list that specifies the characteristics of the prior distribution for σ^2 (the variance of ϵ_i, i.e. the variance of the error terms in the Bayesian linear regression model). There are two types:

- `inverse.gamma`: Inverse Gamma distribution with known shape and scale parameters a and b, respectively.
- `sigmasq.inverse`: inverse sigma-squared distribution.

Similar to `beta.prior` above, the structure of the list depends on the type of prior distribution chosen. The list must be supplied in either of the following structures:

- `sigmasq.prior=list(type = "inverse.gamma", inverse.gamma.a = ..., inverse.gamma.b = ..., sigmasq.init = ...)`
  - `inverse.gamma.a`: the shape parameter a for the Inverse Gamma prior distribution, assumed known; default = 1.
  - `inverse.gamma.b`: the scale parameter b for the Inverse Gamma prior distribution, assumed known; default = 1.
  - `sigmasq.init`: the initial value for the unknown σ^2 parameter for the MCMC chain; default = 1.
- `sigmasq.prior=list(type="sigmasq.inverse", sigmasq.init = ...)`.
  - `sigmasq.init`: the initial value for the unknown σ^2 parameter for the MCMC chain; default = 1.

`Tsamp.out`	an optional scalar that specifies the number of MCMC samples to generate; default = 1,000.
`zero.intercept`	an optional logical parameter with default = `FALSE`. If `zero.intercept = TRUE` is specified, the linear regression model sets the y-intercept term β_0 to zero; the corresponding y-intercept terms of the matrices `data.values$xtx` and `data.values$xty` are ignored, and the β vector is revised throughout the models and output automatically by the function.

Details

This function uses the following Bayesian linear regression model:

$$y_i = x_i'\beta + \epsilon_i,$$

where $i = 1, ..., \mathbf{numsamp.data}$; $\epsilon_i \sim N(0, \sigma^2)$; k is the number of predictor variables. The function uses user-supplied prior distributions for β and σ^2.

The Gibbs sampler is used to sample from all full conditional posterior distributions, which only depend on the summary statistics $X'X$, $X'Y$ and $Y'Y$ (and $Y'X = (X'Y)'$); these summary statistics are calculated by the function `read.regress.data.ff()` (in this package), or can be provided by the user. Starting values are not needed for the vector β, since this vector is updated first, conditioned on all other unknown model parameters and the data.

- The full conditional posterior distributions are the following for each prior specification of β; these depend on the data only through summary statistics $X'X$ and $X'Y$:
  - `beta.prior=list(type = "flat")`:

$$\beta|\sigma^2, X, Y \sim Normal_{k+1}(mean = ((X'X)^{-1}(X'Y), covariance = (\sigma^2(X'X)^{-1})))$$

  - `beta.prior=list(type = "mvnorm.known")`:

$$\begin{aligned}\beta|\sigma^2, X, Y \sim Normal_{k+1}(&mean = (C^{-1} + \sigma^{-2}(X'X))^{-1}(C^{-1}\mu + \sigma^{-2}X'Y),\\ &covariance = (C^{-1} + \sigma^{-2}(X'X)^{-1}))\end{aligned}$$

  - `beta.prior=list(type = "mvnorm.unknown")`:

$$\beta|\sigma^2, \mu, C^{-1}, X, Y \sim Normal_{k+1}(mean = (C^{-1} + \sigma^{-2}(X'X))^{-1}(C^{-1}\mu + \sigma^{-2}X'Y),$$

$$covariance = (C^{-1} + \sigma^{-2}(X'X)^{-1}))$$

$$\mu|\beta, \sigma^2, C^{-1}, X, Y \sim Normal_{k+1}(mean = (D^{-1} + C^{-1})^{-1}(C^{-1}\beta + D^{-1}\eta),$$

$$covariance = (D^{-1} + C^{-1})^{-1})$$

$$C^{-1}|\beta, \sigma^2, \mu, X, Y \sim Wishart_{k+1}(d.f. = (1 + \lambda), scalematrix = (V^{-1} + (\beta - \mu)(\beta - \mu)')^{-1})$$

- The full conditional posterior distributions are the following for each prior specification of σ^2; these depend on the data only through summary statistics $X'X$, $X'Y$ and $Y'Y$:
  - `sigmasq.prior=list(type = "inverse.gamma")`:

$$\sigma^2|\beta, X, Y \sim \text{Inv-Gamma}\left(\frac{\text{numsamp.data}}{2} + a, \left(\frac{1}{2}(Y'Y - \beta'X'Y - Y'X\beta + \beta'X'X\beta) + 1/b\right)^{-1}\right)$$

  - `sigmasq.prior=list(type = "sigmasq.inverse")`:

$$\sigma^2|\beta, X, Y \sim \text{Inv-Gamma}\left(\frac{\text{numsamp.data}}{2}, \left(\frac{1}{2}(Y'Y - \beta'X'Y - Y'X\beta + \beta'X'X\beta)\right)^{-1}\right)$$

Value

The returned value is a list containing the MCMC samples of the unknown Bayesian linear regression model parameters; the number of MCMC samples is equal to the argument `Tsamp.out`. Further analysis, including plotting and creating summary statistics, can be carried out using the `'coda'` R package (see References).

References

Carlin, B.P. and Louis, T.A. (2009) *Bayesian Methods for Data Analysis, 3rd ed.*, Boca Raton, FL: Chapman and Hall/CRC Press.

Gelman, A., Carlin, J.B., Stern, H.S., Dunson, D.B., Vehtari, A. and Rubin, D.B. (2013) *Bayesian Data Analysis, 3rd ed.*, Boca Raton, FL: Chapman and Hall/CRC Press.

Plummer, M., Best, N., Cowles, K. and Vines, K. (2006) CODA: Convergence diagnosis and output analysis for MCMC. *R News*, **6**(1), 7-11.

Adler, D., Glaser, C., Nenadic, O., Oehlschlagel, J. and Zucchini, W. (2013) ff: memory-efficient storage of large data on disk and fast access functions. R package: `http://CRAN.R-project.org/package=ff`.

Fasiolo, M. (2014) An introduction to mvnfast. R package: `http://CRAN.R-project.org/package=mvnfast`.

Examples

```
##################################################
## Simulate data
##################################################

set.seed(284698)

num.samp  <- 100 # number of data values to simulate

# The first value of the beta vector is the y-intercept:
beta <- c(-0.33, 0.78, -0.29, 0.47, -1.25)

# Calculate the number of predictor variables:
num.pred <- length(beta)-1

rho       <- 0.0  # correlation between predictors
mean.vec  <- rep(0,num.pred)
sigma.mat <- matrix(rho,num.pred,num.pred) + diag(1-rho,num.pred)
sigmasq.sim <- 0.05

# Simulate predictor variables:
x.pre       <- rmvn(num.samp, mu=mean.vec, sigma=sigma.mat)

# Add leading column of 1's to x.pre for y-intercept:
x <- cbind(rep(1,num.samp),x.pre)

epsilon <- rnorm(num.samp, mean=0, sd=sqrt(sigmasq.sim))

y  <- as.numeric( x %*% as.matrix(beta) +  epsilon)

## Compute summary statistics (alternatively, the
# "read.regress.data.ff() function (in this package) can be
# used to calculate summary statistics; see example below).

xtx <- t(x)%*%x
xty <- t(x)%*%y
yty <- t(y)%*%y

data.values<-list(xtx=xtx, xty=xty, yty=yty,
                  numsamp.data = num.samp,
                  xtx.inv = chol2inv(chol(xtx)))

##########################################################
## Bayesian linear regression analysis
##########################################################

Tsamp.out <- 100 # number of MCMC samples to produce

## Choose priors for beta and sigma-squared. Here,
# beta: Uniform prior; sigma-squared: Inverse Gamma prior.

beta.prior    <- list( type = "flat")
```

```
sigmasq.prior <- list(type = "inverse.gamma", inverse.gamma.a = 1.0,
                      inverse.gamma.b = 1.0, sigmasq.init = 1.0 )

set.seed(284698)

# Run the "bayes.regress" function using the data simulated above.

MCMC.out <- bayes.regress(data.values,
                          beta.prior,
                          sigmasq.prior = sigmasq.prior,
                          Tsamp.out = Tsamp.out)

# Next, print the posterior means of the unknown model parameters.
# Alternatively, the "coda" package can be used for analysis.

print(c(colMeans(MCMC.out$beta), mean(MCMC.out$sigmasq)))

# Check that output is close to simulated values (although num.samp and
# Tsamp.out are small here); note that the output includes both beta and
# sigmasq:
# c(-0.33,  0.78, -0.29,  0.47, -1.25,  0.05)

## Run all 6 combinations of priors for 3 "beta.prior" choices and
#  2 "sigmasq.prior" choices:

beta.priors <- list(
  list( type = "flat"),

  list( type = "mvnorm.known",
        mean.mu = rep(0.0,    (num.pred+1)),
        prec.Cinv = diag(1.0, (num.pred+1))),

  list( type = "mvnorm.unknown",
        mu.hyper.mean.eta         = rep(0.0,(num.pred+1)),
        mu.hyper.prec.Dinv        = diag(1.0, (num.pred+1)),
        Cinv.hyper.df.lambda      = (num.pred+1),
        Cinv.hyper.invscale.Vinv  = diag(1.0, (num.pred+1)),
        mu.init                   = rep(1.0, (num.pred+1)),
        Cinv.init                 = diag(1.0,(num.pred+1)))
)

sigmasq.priors <- list(
  list(type = "inverse.gamma",
       inverse.gamma.a = 1.0,
       inverse.gamma.b = 1.0,
       sigmasq.init = 0.1 ),
  list( type="sigmasq.inverse", sigmasq.init = 0.1)
)

for (beta.prior in beta.priors)
{
  for(sigmasq.prior in sigmasq.priors)
  {
    set.seed(284698)
    MCMC.out <- bayes.regress(data.values,
                              beta.prior,
                              sigmasq.prior = sigmasq.prior,
                              Tsamp.out = Tsamp.out)
    print(c(colMeans(MCMC.out$beta), mean(MCMC.out$sigmasq)))
  }
}

# Check that output is close to simulated values (although num.samp and
# Tsamp.out are small here); note that the output includes both beta and
# sigmasq:
# c(-0.33,  0.78, -0.29,  0.47, -1.25,  0.05):


#######################################################################
## Read the data from a file, calculate the summary statistics and run
## the Bayesian linear regression analysis
#######################################################################

Tsamp.out <- 100

## Assume non-zero y-intercept data.

# Read the files and compute summary statistics using the "read.regress.data.ff()"
# function (in this package).


filename <- system.file('data/regressiondata.nz.all.csv.gz', package='BayesSummaryStatLM')
data.values <- read.regress.data.ff(filename)

# Calculate the number of predictors.

num.pred <- length(data.values$xty)-1

## Run all 6 combinations of priors for 3 "beta.prior" choices and
#  2 "sigmasq.prior" choices:

beta.priors <- list(
  list( type = "flat"),

  list( type = "mvnorm.known",
        mean.mu = rep(0.0,    (num.pred+1)),
        prec.Cinv = diag(1.0, (num.pred+1))),

  list( type="mvnorm.unknown",
        mu.hyper.mean.eta         = rep(0.0,  (num.pred+1)),
        mu.hyper.prec.Dinv       = diag(1.0, (num.pred+1)),
        Cinv.hyper.df.lambda      = (num.pred+1),
        Cinv.hyper.invscale.Vinv  = diag(1.0, (num.pred+1)),
        mu.init                   = rep(1.0, (num.pred+1)),
        Cinv.init                 = diag(1.0,(num.pred+1)))
)
```

```
sigmasq.priors <- list(
  list(type = "inverse.gamma", inverse.gamma.a = 1.0,
               inverse.gamma.b = 1.0, sigmasq.init = 0.5 ),
  list( type = "sigmasq.inverse", sigmasq.init = 0.5)
)

for (beta.prior in beta.priors)
{
  for(sigmasq.prior in sigmasq.priors)
  {

    set.seed(284698)
    MCMC.out <- bayes.regress(data.values,
                             beta.prior,
                             sigmasq.prior = sigmasq.prior,
                             Tsamp.out = Tsamp.out)

    print(c(colMeans(MCMC.out$beta), mean(MCMC.out$sigmasq)))
  }
}

# Check that output is close to simulated values (although num.samp and
# Tsamp.out are small here); note that the output includes both beta and
# sigmasq:
# c( 0.76, -0.92, 0.64, 0.57, -1.65, 0.25)
```

`read.regress.data.ff` *Read in Tabulated Data and Compute Summary Statistics*

Description

This function reads in tabulated data sets and produces summary statistics needed for Bayesian linear regression models for use in the function `bayes.regress()` (in this package). Big data sets that are too large to fit into R memory are handled using functions from package `ff`. The function takes as input data files with the predictor variables X and response values Y, and returns the summary statistics $X'X$, $X'Y$ and $Y'Y$ that are used as an input to the function `bayes.regress()` (in this package) for Bayesian linear regression models. The function supports reading data sets that are split across multiple files.

Usage

```
read.regress.data.ff(filename=NULL,predictor.cols=NA,response.col=NA,update.summaries=NULL
                 , fileEncoding = "", nrows = -1, first.rows = 1e5, next.rows = 1e5
                 , levels = NULL, appendLevels = TRUE,FUN = "read.table",transFUN = NULL
                   , asffdf_args = list(), BATCHBYTES = getOption("ffbatchbytes")
                 , VERBOSE = FALSE, header = FALSE, sep = ",", quote = "\"'", dec = "."
```

```
    , numerals = c("allow.loss", "warn.loss", "no.loss")
    , na.strings = "NA", colClasses = "numeric", skip = 0
    , check.names = TRUE, fill = TRUE, strip.white = FALSE
, blank.lines.skip = TRUE, comment.char = "#", allowEscapes = FALSE
    , flush = FALSE, skipNul = FALSE)
```

Arguments

`filename`	the name of a file or a list of file names, from which the data will be read. By default it is assumed that the file(s) contain data in comma separated format; this can be changed using the `sep` argument. File names must be in the format acceptable by standard functions such as `read.table`. Default = `NULL`.
`predictor.cols`	a vector of integers that specifies the columns to treat as predictor variables, to create the design matrix X. By default, all columns after the first are treated as columns with predictor variables. Default = `NA`.
`response.col`	an integer that specifies the column that contains the response variable values, to create the response vector Y. By default, the first column that is not used as a predictor variable column is selected as the response values column. Default = `NA`.
`update.summaries`	The name of the R object containing previously-calculated summary statistics (if applicable), to be updated with new data. This must be a list similar in structure to the returned value, containing entries `xtx`, `xty`, `yty` and `numsamp.data` of appropriate dimensions and class that match the data contained in `filename`. Default = `NULL`. The remaining arguments are passed directly to the function `read.table.ffdf()`. Below is a short description of the arguments we recommend to set manually in accordance with memory limitations and data structure:
`first.rows`	the number of rows to read in the first chunk of data. Default = 100,000.
`next.rows`	the number of rows to read in the remaining chunks of data. Default = 100,000.
`sep`	the character that separates the columns of data. Default = ",". For the arguments below the default settings should perform well. However, in some situations adjusting these arguments may improve memory use and running time.
`fileEncoding`	a string that describes the file's character encoding
`nrows`	an integer specifying how many rows should be read from the file
`levels`	an optional list of items with `col.names` or factor columns. See `read.table.ffdf()`.
`appendLevels`	a logical vector of permission to expand `levels`
`FUN`	specifies which standard R function is used to read the data.
`transFUN`	an optional filtering function to be applied to each chunk of data. See `read.table.ffdf()`.
`asffdf_args`	an optional list of parameters to be passed to `as.ffdf()` function.
`BATCHBYTES`	an integer limiting the size of the `data.frame` used to store each chunk of the data
`VERBOSE`	See `read.table.ffdf()`.
`header`	a logical value indicating if the first row is the header row

`quote`	a character string specifying which character will be treated as quoting characters
`dec`	a character used for decimal dot
`numerals`	see `read.table()`.
`na.strings`	strings treated as `NA` values
`colClasses`	a vector that describes the data types in each column. Numeric by default.
`skip`	how many first lines in the file should be skipped
`check.names`	see `read.table()`.
`fill`	logical value that turns on automatic padding of the rows in case they have different lengths
`strip.white`	affects the processing of the columns with declared `character` type. See `scan()`.
`blank.lines.skip`	logical value, indicating whether empty lines should be ignored
`comment.char`	character specifying the comment marker
`allowEscapes`	logical. If `TRUE`, the escaped strings will be parsed. See `scan()`.
`flush`	see `read.table()`.
`skipNul`	logical: should nuls be skipped?

Details

The function reads in data and computes summary statistics to be used in Bayesian linear regression by the function `bayes.regress()` (in this package). The function assumes the linear regression model will have a non-zero y-intercept; this option can be changed in the `bayes.regress()` function (see `bayes.regress()` help for details).

Value

The returned value for the `read.regress.data.ff()` function is a list containing the summary statistics named `xtx` (for $X'X$), `xty` (for $X'Y$), `yty` (for $Y'Y$) and the total number of data values `numsamp.data`. The summary statistic `xtx` contains a square matrix obtained by computing a dot product of the predictor variables data X with itself; a leading column of 1's is added to X for the y-intercept term. `xty` contains the vector obtained by computing the dot product of the transposed predictor variables data X with response variable data Y; a leading column of 1's is added to X for the y-intercept term. `yty` contains the dot product of the response variable data Y with itself. `numsamp.data` is the number of data values read from the data file(s); this number may be smaller than the number of rows in the data file, since some of the rows with missing data may be skipped according to specified function arguments. The summary statistics $X'X$, $X'Y$ and $Y'Y$ are summed over data chunks by the following, for $m = 1, ..., M$ chunks:

$$X'X = sum_{m=1}^{M}(X'_m)(X_m)$$

$$X'Y = sum_{m=1}^{M}(X'_m)(Y_m)$$

$$Y'Y = sum_{m=1}^{M}(Y'_m)(Y_m)$$

The returned values are used as input to the function `bayes.regress()` (in this package). Note that the matrix `X` is given a leading column of 1's by default, for the y-intercept term of the Bayesian linear regression model. This can be removed by specifying a model with zero intercept in the function `bayes.regress()` (see `bayes.regress()` help for details).

References

Carlin, B.P. and Louis, T.A. (2009) *Bayesian Methods for Data Analysis, 3rd ed.*, Boca Raton, FL: Chapman and Hall/CRC Press.

Gelman, A., Carlin, J.B., Stern, H.S., Dunson, D.B., Vehtari, A. and Rubin, D.B. (2013) *Bayesian Data Analysis, 3rd ed.*, Boca Raton, FL: Chapman and Hall/CRC Press.

Adler, D., Glaser, C., Nenadic, O., Oehlschlagel, J. and Zucchini, W. (2013) ff: memory-efficient storage of large data on disk and fast access functions. R package: http://CRAN.R-project.org/package=ff.

Examples

```
# The package includes several example data files, illustrated here.

###########
# Example 1
###########
# The following command finds the location of the data file
# that includes 4 predictor variables and 20,000 simulated data values.

filename <- system.file('data/regressiondata.nz.all.csv.gz', package='BayesSummaryStatLM')

# The file is formatted so that the simulated response variable is in the
# first column, and columns 2 to 5 contain simulated predictor variables.
# The simulated coefficients are: beta <- c(0.76, -0.92, 0.64, 0.57, -1.65),
# where the first value is the y-intercept term in the Bayesian linear
# regression model.  The sigma-squared term, i.e. the variance of the normally
# distributed error terms, is simulated as: sigmasq <- 0.25

## Next, read the data and compute the summary statistics using the
# "read.regress.data.ff()" function.  By default, the first column is assumed
# to be the response variable, and the remaining columns are assumed to contain
# predictor variable values.  The function will check if the file exists and
# can be read.

data.values <- read.regress.data.ff(filename)
data.values

###########
# Example 2
###########
## Several files can be given in a list to be read sequentially, as follows.

filenames <- list(
  system.file('data/regressiondata.nz.pt1.csv.gz', package='BayesSummaryStatLM'),
  system.file('data/regressiondata.nz.pt2.csv.gz', package='BayesSummaryStatLM')
)
data.values <- read.regress.data.ff(filenames)
data.values

# The above results can be compared to the "data.values" obtained previously.  They
# are the same, since the current files are just copies of the same data split
# between two files.

###########
# Example 3
###########
## The two files can be read progressively through time, and the summary statistics
# are then updated with data in each file, as follows.

filenames <- list(
  system.file('data/regressiondata.nz.pt1.csv.gz', package='BayesSummaryStatLM'),
  system.file('data/regressiondata.nz.pt2.csv.gz', package='BayesSummaryStatLM')
)
data.values <- read.regress.data.ff(filenames[[1]])
data.values
data.values2 <- read.regress.data.ff(filenames[[2]], update.summaries = data.values)
data.values2

###########
# Example 4
###########
## If not all columns are to be used in regression analysis, one can specify
# which columns to use in the "predictor.cols" and "response.col" options;
# the order of "predictor.cols" can also be changed.  The following command
# reads in predictors from a subset of 3 columns, and changes their order.

filename <- system.file('data/regressiondata.nz.all.csv.gz', package='BayesSummaryStatLM')
data.values <- read.regress.data.ff(filename, predictor.cols=c(4,2,3), response.col=5)
data.values

###########
# Example 5
###########
## If the R session must be terminated, the summary statistics can be saved and then
# loaded using standard methods in R, as follows:

filenames <- list(
  system.file('data/regressiondata.nz.pt1.csv.gz', package='BayesSummaryStatLM'),
  system.file('data/regressiondata.nz.pt2.csv.gz', package='BayesSummaryStatLM')
)
data.values <- read.regress.data.ff(filenames[[1]])

tmpfname <- tempfile()
save(data.values, file = tmpfname)
rm(data.values)

# Now the R session can be terminated.  Note that the filename "tmpfname"
# must be recorded so that it can be used for updating in a later R session.
# Upon starting a new R session, the state of the previously-calculated
# summary statistics in the file named "tmpfname" can be restored and
# then updated, as follows:

load(tmpfname)
unlink(tmpfname)
```

```
# If a new portion of a data set arrives, the summary statistics are updated
# as follows:

data.values2 <- read.regress.data.ff(filenames[[2]], update.summaries = data.values)
data.values2
```

`regressiondata.nz.all` *Simulated data for Bayesian linear regression models, for use in package examples.*

Description

20,000 samples of simulated data. The response values y_i are simulated according to the following:

$$y_i = 0.76 - 0.92 * x_{i1} + 0.64 * x_{i2} + 0.57 * x_{i3} - 1.65 * x_{i4} + \epsilon_i.$$

Here, each predictor variable is simulated from a Normal distribution with mean = 0 and variance = 1.0, and each ϵ_i is simulated from a Normal distribution with mean = 0 and variance = 0.25; the predictor variables are assumed to be independent.

Usage

```
data("regressiondata.nz.all")
```

Format

A data frame with 20,000 observations for five variables: y, x_1, x_2, x_3, x_4

`regressiondata.nz.pt1` *Simulated data for Bayesian linear regression models, for use in package examples.*

Description

10,000 samples of simulated data. The response values y_i are simulated according to the following:

$$y_i = 0.76 - 0.92 * x_{i1} + 0.64 * x_{i2} + 0.57 * x_{i3} - 1.65 * x_{i4} + \epsilon_i.$$

Here, each predictor variable is simulated from a Normal distribution with mean = 0 and variance = 1.0, and each ϵ_i is simulated from a Normal distribution with mean = 0 and variance = 0.25; the predictor variables are assumed to be independent.

This file is a copy of the first 10,000 entries from the file `regressiondata.nz.all`, for illustration in the package examples of using multiple files.

Usage

```
data("regressiondata.nz.pt1")
```

Format

A data frame with 10,000 observations for five variables: y, x_1, x_2, x_3, x_4

`regressiondata.nz.pt2` *Simulated data for Bayesian linear regression models, for use in package examples.*

Description

10,000 samples of simulated data. The response values y_i are simulated according to the following:

$$y_i = 0.76 - 0.92 * x_{i1} + 0.64 * x_{i2} + 0.57 * x_{i3} - 1.65 * x_{i4} + \epsilon_i.$$

Here, each predictor variable is simulated from a Normal distribution with mean = 0 and variance = 1.0, and each ϵ_i is simulated from a Normal distribution with mean = 0 and variance = 0.25; the predictor variables are assumed to be independent.

This file is a copy of the last 10,000 entries from the file `regressiondata.nz.all`, for illustration in the package examples of using multiple files.

Usage

```
data("regressiondata.nz.pt1")
```

Format

A data frame with 10,000 observations for five variables: y, x_1, x_2, x_3, x_4

Index