

Package ‘MCMCpack’

January 28, 2006

Version 0.7-1

Date 2006-1-28

Title Markov chain Monte Carlo (MCMC) Package

Author Andrew D. Martin <admartin@wustl.edu>, and Kevin M. Quinn <kevin_quinn@harvard.edu>

Maintainer Andrew D. Martin <admartin@wustl.edu>

Depends R (>= 2.2.0), coda (>= 0.10-3), MASS

Description This package contains functions to perform Bayesian inference using posterior simulation for a number of statistical models. All simulation is done in compiled C++ written in the Scythe Statistical Library Version 1.0. All models return coda mcmc objects that can then be summarized using the coda package. MCMCpack also contains some useful utility functions, including some additional density functions and pseudo-random number generators for statistical distributions, a general purpose Metropolis sampling algorithm, and tools for visualization.

License GPL version 2

URL <http://mcmcpack.wustl.edu>

R topics documented:

BayesFactor	2
MCMCSVDreg	3
MCMCdynamicEI	6
MCMCfactanal	9
MCMChierEI	12
MCMCirt1d	14
MCMCirtKd	18
MCMCirtKdRob	21
MCMClogit	26
MCMCmetrop1R	29
MCMCmixfactanal	32
MCMCmnl	36
MCMCoprobit	40
MCMCordfactanal	42
MCMCpanel	46
MCMCpoisson	48
MCMCprobit	50

MCMCregress	52
MCMCtobit	54
MCbinomialbeta	57
MCmultinomdirichlet	58
MCnormalnormal	59
MCpoissongamma	60
Nethvote	61
PErisk	62
PostProbMod	63
Senate	64
SupremeCourt	65
choicevar	65
Dirichlet	66
dtomogplot	67
InvGamma	69
InvWishart	69
NoncenHypergeom	70
procrustes	71
read.Scythe	72
tomogplot	73
vech	74
Wishart	75
write.Scythe	75
xpnd	76
Index	78

BayesFactor	<i>Create an object of class BayesFactor from MCMCpack output</i>
-------------	---

Description

This function creates an object of class `BayesFactor` from `MCMCpack` output.

Usage

```
BayesFactor(...)  
is.BayesFactor(BF)
```

Arguments

- `...` `MCMCpack` output objects. These have to be of class `mcmc` and have a `logmarglike` attribute. In what follows, we let M denote the total number of models to be compared.
- `BF` An object to be checked for membership in class `BayesFactor`.

Value

An object of class `BayesFactor`. A `BayesFactor` object has four attributes. They are: `BF.mat` an $M \times M$ matrix in which element i, j contains the Bayes factor for model i relative to model j ; `BF.log.mat` an $M \times M$ matrix in which element i, j contains the natural log of the Bayes factor for model i relative to model j ; `BF.logmarglike` an M vector containing the log marginal likelihoods for models 1 through M ; and `BF.call` an M element list containing the calls used to fit models 1 through M .

See Also

[MCMCregress](#)

Examples

```
## Not run:
data(birthwt)

model1 <- MCMCregress(bwt~age+lwt+as.factor(race) + smoke + ht,
  data=birthwt, b0=c(2700, 0, 0, -500, -500,
                    -500, -500),
  B0=c(1e-6, .01, .01, 1.6e-5, 1.6e-5, 1.6e-5,
        1.6e-5), c0=10, d0=4500000,
  marginal.likelihood="Chib95", mcmc=10000)

model2 <- MCMCregress(bwt~age+lwt+as.factor(race) + smoke,
  data=birthwt, b0=c(2700, 0, 0, -500, -500,
                    -500),
  B0=c(1e-6, .01, .01, 1.6e-5, 1.6e-5, 1.6e-5),
  c0=10, d0=4500000,
  marginal.likelihood="Chib95", mcmc=10000)

model3 <- MCMCregress(bwt~as.factor(race) + smoke + ht,
  data=birthwt, b0=c(2700, -500, -500,
                    -500, -500),
  B0=c(1e-6, 1.6e-5, 1.6e-5, 1.6e-5,
        1.6e-5), c0=10, d0=4500000,
  marginal.likelihood="Chib95", mcmc=10000)

BF <- BayesFactor(model1, model2, model3)
print(BF)

## End(Not run)
```

Description

This function generates a sample from the posterior distribution of a linear regression model with Gaussian errors in which the design matrix has been decomposed with singular value decomposition. The sampling is done via the Gibbs sampling algorithm. The user supplies data and priors, and a sample from the posterior distribution is returned as an `mcmc` object, which can be subsequently analyzed with functions provided in the `coda` package.

Usage

```
MCMCSVDreg(formula, data=parent.frame(), burnin = 1000, mcmc = 10000,
            thin=1, verbose = 0, seed = NA, tau2.start = 1,
            g0 = 0, a0 = 0.001, b0 = 0.001, c0=2, d0=2, w0=1,
            beta.samp=FALSE, intercept=TRUE, ...)
```

Arguments

<code>formula</code>	Model formula. Predictions are returned for elements of <code>y</code> that are coded as NA.
<code>data</code>	Data frame.
<code>burnin</code>	The number of burn-in iterations for the sampler.
<code>mcmc</code>	The number of MCMC iterations after burnin.
<code>thin</code>	The thinning interval used in the simulation. The number of MCMC iterations must be divisible by this value.
<code>verbose</code>	A switch which determines whether or not the progress of the sampler is printed to the screen. If <code>verbose</code> is greater than 0 the iteration number, the β vector, and the error variance are printed to the screen every <code>verbose</code> th iteration.
<code>seed</code>	The seed for the random number generator. If NA, the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first element of the list is the L'Ecuyer seed, which is a vector of length six or NA (if NA a default seed of <code>rep(12345, 6)</code> is used). The second element of list is a positive substream number. See the MCMCpack specification for more details.
<code>tau2.start</code>	The starting values for the τ^2 vector. Can be either a scalar or a vector. If a scalar is passed then that value will be the starting value for all elements of τ^2 .
<code>g0</code>	The prior mean of γ . This can either be a scalar or a column vector with dimension equal to the number of gammas. If this takes a scalar value, then that value will serve as the prior mean for all of the betas.
<code>a0</code>	$a_0/2$ is the shape parameter for the inverse Gamma prior on σ^2 (the variance of the disturbances). The amount of information in the inverse Gamma prior is something like that from a_0 pseudo-observations.
<code>b0</code>	$b_0/2$ is the scale parameter for the inverse Gamma prior on σ^2 (the variance of the disturbances). In constructing the inverse Gamma prior, b_0 acts like the sum of squared errors from the a_0 pseudo-observations.
<code>c0</code>	$c_0/2$ is the shape parameter for the inverse Gamma prior on τ_i^2 .
<code>d0</code>	$d_0/2$ is the scale parameter for the inverse Gamma prior on τ_i^2 .
<code>w0</code>	The prior probability that $\gamma_i = 0$. Can be either a scalar or an N vector where N is the number of observations.
<code>beta.samp</code>	Logical indicating whether the sampled elements of beta should be stored and returned.
<code>intercept</code>	Logical indicating whether the original design matrix should include a constant term.
<code>...</code>	further arguments to be passed

Details

The model takes the following form:

$$y = X\beta + \varepsilon$$

Where the errors are assumed to be iid Gaussian:

$$\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$$

.

Let N denote the number of rows of X and P the number of columns of X . Unlike the standard regression setup where $N \gg P$ here it is the case that $P \gg N$.

To deal with this problem a singular value decomposition of X' is performed: $X' = ADF$ and the regression model becomes

$$y = F'D\gamma + \varepsilon$$

where $\gamma = A'\beta$.

We assume the following priors:

$$\sigma^{-2} \sim \text{Gamma}(a_0/2, b_0/2)$$

$$\tau^{-2} \sim \text{Gamma}(c_0/2, d_0/2)$$

$$\gamma_i \sim w_0 \delta_0 + (1 - w_0) \mathcal{N}(g_0, \sigma^2 \tau_i^2 / d_i^2)$$

where δ_0 is a unit point mass at 0 and d_i is the i th diagonal element of D .

Value

An mcmc object that contains the posterior sample. This object can be summarized by functions provided by the coda package.

References

West, Mike; Josheph Nevins; Jeffrey Marks; Rainer Spang, and Harry Zuzan. 2000. "DNA MICROARRAY DATA ANALYSIS AND REGRESSION MODELING FOR GENETIC EXPRESSION PROFILING" Duke ISDS working paper.

Gottardo, Raphael, and Adrian Raftery. 2004. "Markov chain Monte Carlo with mixtures of singular distributions". Statistics department, University of Washington, Technical report 470.

Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.

Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. 2002. *Output Analysis and Diagnostics for MCMC (CODA)*. <http://www-fis.iarc.fr/coda/>.

See Also

[plot.mcmc](#), [summary.mcmc](#), [lm](#)

Examples

```
## Not run:
## End(Not run)
```

MCMCdynamicEI	<i>Markov Chain Monte Carlo for Quinn's Dynamic Ecological Inference Model</i>
---------------	--

Description

MCMCdynamicEI is used to fit Quinn's dynamic ecological inference model for partially observed 2×2 contingency tables.

Usage

```
MCMCdynamicEI(r0, r1, c0, c1, burnin=5000, mcmc=50000, thin=1,
               verbose=0, seed=NA, W=0, a0=0.825,
               b0=0.0105, a1=0.825, b1=0.0105, ...)
```

Arguments

r0	(<i>n</i> tables × 1) vector of row sums from row 0.
r1	(<i>n</i> tables × 1) vector of row sums from row 1.
c0	(<i>n</i> tables × 1) vector of column sums from column 0.
c1	(<i>n</i> tables × 1) vector of column sums from column 1.
burnin	The number of burn-in scans for the sampler.
mcmc	The number of mcmc scans to be saved.
thin	The thinning interval used in the simulation. The number of mcmc iterations must be divisible by this value.
verbose	A switch which determines whether or not the progress of the sampler is printed to the screen. If <i>verbose</i> is greater than 0 then every <i>verboseth</i> iteration will be printed to the screen.
seed	The seed for the random number generator. If NA, the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first element of the list is the L'Ecuyer seed, which is a vector of length six or NA (if NA a default seed of <code>rep(12345, 6)</code> is used). The second element of list is a positive substream number. See the MCMCpack specification for more details.
W	Weight (<i>not precision</i>) matrix structuring the temporal dependence among elements of θ_0 and θ_1 . The default value of 0 will construct a weight matrix that corresponds to random walk priors for θ_0 and θ_1 . The default assumes that the tables are equally spaced throughout time and that the elements of <i>r0</i> , <i>r1</i> , <i>c0</i> , and <i>c1</i> are temporally ordered.
a0	$a0/2$ is the shape parameter for the inverse-gamma prior on the σ_0^2 parameter.
b0	$b0/2$ is the scale parameter for the inverse-gamma prior on the σ_0^2 parameter.
a1	$a1/2$ is the shape parameter for the inverse-gamma prior on the σ_1^2 parameter.
b1	$b1/2$ is the scale parameter for the inverse-gamma prior on the σ_1^2 parameter.
...	further arguments to be passed

Details

Consider the following partially observed 2 by 2 contingency table for unit t where $t = 1, \dots, ntables$:

	$Y = 0$	$Y = 1$	
-----	-----	-----	-----
$X = 0$	Y_{0t}		r_{0t}
-----	-----	-----	-----
$X = 1$	Y_{1t}		r_{1t}
-----	-----	-----	-----
	c_{0t}	c_{1t}	N_t

Where r_{0t} , r_{1t} , c_{0t} , c_{1t} , and N_t are non-negative integers that are observed. The interior cell entries are not observed. It is assumed that $Y_{0t}|r_{0t} \sim \text{Binomial}(r_{0t}, p_{0t})$ and $Y_{1t}|r_{1t} \sim \text{Binomial}(r_{1t}, p_{1t})$. Let $\theta_{0t} = \log(p_{0t}/(1 - p_{0t}))$, and $\theta_{1t} = \log(p_{1t}/(1 - p_{1t}))$.

The following prior distributions are assumed:

$$p(\theta_0|\sigma_0^2) \propto \sigma_0^{-ntables} \exp\left(-\frac{1}{2\sigma_0^2}\theta_0'P\theta_0\right)$$

and

$$p(\theta_1|\sigma_1^2) \propto \sigma_1^{-ntables} \exp\left(-\frac{1}{2\sigma_1^2}\theta_1'P\theta_1\right)$$

where $P_{ts} = -W_{ts}$ for t not equal to s and $P_{tt} = \sum_{s \neq t} W_{ts}$. The θ_{0t} is assumed to be a priori independent of θ_{1t} for all t . In addition, the following hyperpriors are assumed: $\sigma_0^2 \sim \mathcal{IG}(a_0/2, b_0/2)$, and $\sigma_1^2 \sim \mathcal{IG}(a_1/2, b_1/2)$.

Inference centers on p_0 , p_1 , σ_0^2 , and σ_1^2 . Univariate slice sampling (Neal, 2003) together with Gibbs sampling is used to sample from the posterior distribution.

Value

An mcmc object that contains the sample from the posterior distribution. This object can be summarized by functions provided by the coda package.

References

- Kevin Quinn. 2004. "Ecological Inference in the Presence of Temporal Dependence." In *Ecological Inference: New Methodological Strategies*. Gary King, Ori Rosen, and Martin A. Tanner (eds.). New York: Cambridge University Press.
- Jonathan C. Wakefield. 2003. "Ecological inference for 2x2 tables." Read before the Royal Statistical Society, on November 12th, 2003.
- Radford Neal. 2003. "Slice Sampling" (with discussion). *Annals of Statistics*, 31: 705-767.
- Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.
- Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. 2002. *Output Analysis and Diagnostics for MCMC (CODA)*. <http://www-fis.iarc.fr/coda/>.

See Also

[MCMChierEI](#), [plot.mcmc](#), [summary.mcmc](#)

Examples

```

## Not run:
## simulated data example 1
set.seed(3920)
n <- 100
r0 <- rpois(n, 2000)
r1 <- round(runif(n, 100, 4000))
p0.true <- pnorm(-1.5 + 1:n/(n/2))
p1.true <- pnorm(1.0 - 1:n/(n/4))
y0 <- rbinom(n, r0, p0.true)
y1 <- rbinom(n, r1, p1.true)
c0 <- y0 + y1
c1 <- (r0+r1) - c0

## plot data
dtomogplot(r0, r1, c0, c1, delay=0.1)

## fit dynamic model
post1 <- MCMCdynamicEI(r0,r1,c0,c1, mcmc=40000, thin=5, verbose=100,
                      seed=list(NA, 1))

## fit exchangeable hierarchical model
post2 <- MCMChierEI(r0,r1,c0,c1, mcmc=40000, thin=5, verbose=100,
                   seed=list(NA, 2))

p0meanDyn <- colMeans(post1)[1:n]
p1meanDyn <- colMeans(post1)[(n+1):(2*n)]
p0meanHier <- colMeans(post2)[1:n]
p1meanHier <- colMeans(post2)[(n+1):(2*n)]

## plot truth and posterior means
pairs(cbind(p0.true, p0meanDyn, p0meanHier, p1.true, p1meanDyn, p1meanHier))

## simulated data example 2
set.seed(8722)
n <- 100
r0 <- rpois(n, 2000)
r1 <- round(runif(n, 100, 4000))
p0.true <- pnorm(-1.0 + sin(1:n/(n/4)))
p1.true <- pnorm(0.0 - 2*cos(1:n/(n/9)))
y0 <- rbinom(n, r0, p0.true)
y1 <- rbinom(n, r1, p1.true)
c0 <- y0 + y1
c1 <- (r0+r1) - c0

## plot data
dtomogplot(r0, r1, c0, c1, delay=0.1)

## fit dynamic model
post1 <- MCMCdynamicEI(r0,r1,c0,c1, mcmc=40000, thin=5, verbose=100,
                      seed=list(NA, 1))

## fit exchangeable hierarchical model
post2 <- MCMChierEI(r0,r1,c0,c1, mcmc=40000, thin=5, verbose=100,
                   seed=list(NA, 2))

```

```

p0meanDyn <- colMeans(post1)[1:n]
plmeanDyn <- colMeans(post1)[(n+1):(2*n)]
p0meanHier <- colMeans(post2)[1:n]
plmeanHier <- colMeans(post2)[(n+1):(2*n)]

## plot truth and posterior means
pairs(cbind(p0.true, p0meanDyn, p0meanHier, pl.true, plmeanDyn, plmeanHier))
## End(Not run)

```

MCMCfactanal

Markov Chain Monte Carlo for Normal Theory Factor Analysis Model

Description

This function generates a sample from the posterior distribution of a normal theory factor analysis model. Normal priors are assumed on the factor loadings and factor scores while inverse Gamma priors are assumed for the uniquenesses. The user supplies data and parameters for the prior distributions, and a sample from the posterior distribution is returned as an mcmc object, which can be subsequently analyzed with functions provided in the coda package.

Usage

```

MCMCfactanal(x, factors, lambda.constraints=list(),
             data=parent.frame(), burnin = 1000, mcmc = 20000,
             thin=1, verbose = 0, seed = NA,
             lambda.start = NA, psi.start = NA,
             l0=0, L0=0, a0=0.001, b0=0.001,
             store.scores = FALSE, std.var=TRUE, ... )

```

Arguments

<code>x</code>	Either a formula or a numeric matrix containing the manifest variables.
<code>factors</code>	The number of factors to be fitted.
<code>lambda.constraints</code>	List of lists specifying possible simple equality or inequality constraints on the factor loadings. A typical entry in the list has one of three forms: <code>varname=list(d,c)</code> which will constrain the <code>d</code> th loading for the variable named <code>varname</code> to be equal to <code>c</code> , <code>varname=list(d, "+")</code> which will constrain the <code>d</code> th loading for the variable named <code>varname</code> to be positive, and <code>varname=list(d, "-")</code> which will constrain the <code>d</code> th loading for the variable named <code>varname</code> to be negative. If <code>x</code> is a matrix without column names defaults names of "V1", "V2", ... , etc will be used.
<code>data</code>	A data frame.
<code>burnin</code>	The number of burn-in iterations for the sampler.
<code>mcmc</code>	The number of iterations for the sampler.
<code>thin</code>	The thinning interval used in the simulation. The number of iterations must be divisible by this value.
<code>verbose</code>	A switch which determines whether or not the progress of the sampler is printed to the screen. If <code>verbose</code> is greater than 0 the iteration number and the factor loadings and uniquenesses are printed to the screen every <code>verbose</code> th iteration.

<code>seed</code>	The seed for the random number generator. If NA, the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first element of the list is the L'Ecuyer seed, which is a vector of length six or NA (if NA a default seed of <code>rep(12345, 6)</code> is used). The second element of list is a positive substream number. See the MCMCpack specification for more details.
<code>lambda.start</code>	Starting values for the factor loading matrix Lambda. If <code>lambda.start</code> is set to a scalar the starting value for all unconstrained loadings will be set to that scalar. If <code>lambda.start</code> is a matrix of the same dimensions as Lambda then the <code>lambda.start</code> matrix is used as the starting values (except for equality-constrained elements). If <code>lambda.start</code> is set to NA (the default) then starting values for unconstrained elements are set to 0, and starting values for inequality constrained elements are set to either 0.5 or -0.5 depending on the nature of the constraints.
<code>psi.start</code>	Starting values for the uniquenesses. If <code>psi.start</code> is set to a scalar then the starting value for all diagonal elements of Psi are set to this value. If <code>psi.start</code> is a k -vector (where k is the number of manifest variables) then the starting value of Psi has <code>psi.start</code> on the main diagonal. If <code>psi.start</code> is set to NA (the default) the starting values of all the uniquenesses are set to 0.5.
<code>l0</code>	The means of the independent Normal prior on the factor loadings. Can be either a scalar or a matrix with the same dimensions as Lambda.
<code>L0</code>	The precisions (inverse variances) of the independent Normal prior on the factor loadings. Can be either a scalar or a matrix with the same dimensions as Lambda.
<code>a0</code>	Controls the shape of the inverse Gamma prior on the uniqueness. The actual shape parameter is set to <code>a0/2</code> . Can be either a scalar or a k -vector.
<code>b0</code>	Controls the scale of the inverse Gamma prior on the uniquenesses. The actual scale parameter is set to <code>b0/2</code> . Can be either a scalar or a k -vector.
<code>store.scores</code>	A switch that determines whether or not to store the factor scores for posterior analysis. <i>NOTE: This takes an enormous amount of memory, so should only be used if the chain is thinned heavily, or for applications with a small number of observations.</i> By default, the factor scores are not stored.
<code>std.var</code>	If TRUE (the default) the manifest variables are rescaled to have zero mean and unit variance. Otherwise, the manifest variables are rescaled to have zero mean but retain their observed variances.
<code>...</code>	further arguments to be passed

Details

The model takes the following form:

$$x_i = \Lambda\phi_i + \epsilon_i$$

$$\epsilon_i \sim \mathcal{N}(0, \Psi)$$

where x_i is the k -vector of observed variables specific to observation i , Λ is the $k \times d$ matrix of factor loadings, ϕ_i is the d -vector of latent factor scores, and Ψ is a diagonal, positive definite matrix. Traditional factor analysis texts refer to the diagonal elements of Ψ as uniquenesses.

The implementation used here assumes independent conjugate priors for each element of Λ , each ϕ_i , and each diagonal element of Ψ . More specifically we assume:

$$\Lambda_{ij} \sim \mathcal{N}(l_{0_{ij}}, L_{0_{ij}}^{-1}), i = 1, \dots, k, j = 1, \dots, d$$

$$\phi_i \sim \mathcal{N}(0, I), i = 1, \dots, n$$

$$\Psi_{ii} \sim \mathcal{IG}(a_{0_i}/2, b_{0_i}/2), i = 1, \dots, k$$

MCMCfactanal simulates from the posterior distribution using standard Gibbs sampling. The simulation proper is done in compiled C++ code to maximize efficiency. Please consult the coda documentation for a comprehensive list of functions that can be used to analyze the posterior sample.

As is the case with all measurement models, make sure that you have plenty of free memory, especially when storing the scores.

Value

An mcmc object that contains the sample from the posterior distribution. This object can be summarized by functions provided by the coda package.

References

Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.

Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. 2002. *Output Analysis and Diagnostics for MCMC (CODA)*. <http://www-fis.iarc.fr/coda/>.

See Also

[plot.mcmc,summary.mcmc,factanal](#)

Examples

```
## Not run:
### An example using the formula interface
data(swiss)
posterior <- MCMCfactanal(~Agriculture+Examination+Education+Catholic
  +Infant.Mortality, factors=2,
  lambda.constraints=list(Examination=list(1,"+"),
    Examination=list(2,"-"), Education=c(2,0),
    Infant.Mortality=c(1,0)),
  verbose=0, store.scores=FALSE, a0=1, b0=0.15,
  data=swiss, burnin=5000, mcmc=50000, thin=20)

plot(posterior)
summary(posterior)

### An example using the matrix interface
Y <- cbind(swiss$Agriculture, swiss$Examination,
  swiss$Education, swiss$Catholic,
  swiss$Infant.Mortality)
colnames(Y) <- c("Agriculture", "Examination", "Education", "Catholic",
```

```

                                "Infant.Mortality")
post <- MCMCfactanal(Y, factors=2,
                    lambda.constraints=list(Examination=list(1, "+"),
                                           Examination=list(2, "-"), Education=c(2,0),
                                           Infant.Mortality=c(1,0)),
                    verbose=0, store.scores=FALSE, a0=1, b0=0.15,
                    burnin=5000, mcmc=50000, thin=20)

## End(Not run)

```

MCMChierEI

Markov Chain Monte Carlo for Wakefield's Hierarchical Ecological Inference Model

Description

'MCMChierEI' is used to fit Wakefield's hierarchical ecological inference model for partially observed 2 x 2 contingency tables.

Usage

```

MCMChierEI(r0, r1, c0, c1, burnin=5000, mcmc=50000, thin=1,
           verbose=0, seed=NA,
           m0=0, M0=2.287656, m1=0, M1=2.287656, a0=0.825, b0=0.0105,
           a1=0.825, b1=0.0105, ...)

```

Arguments

r0	(<i>n</i> tables × 1) vector of row sums from row 0.
r1	(<i>n</i> tables × 1) vector of row sums from row 1.
c0	(<i>n</i> tables × 1) vector of column sums from column 0.
c1	(<i>n</i> tables × 1) vector of column sums from column 1.
burnin	The number of burn-in scans for the sampler.
mcmc	The number of mcmc scans to be saved.
thin	The thinning interval used in the simulation. The number of mcmc iterations must be divisible by this value.
verbose	A switch which determines whether or not the progress of the sampler is printed to the screen. If <i>verbose</i> is greater than 0 then every <i>verbose</i> th iteration will be printed to the screen.
seed	The seed for the random number generator. If NA, the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first element of the list is the L'Ecuyer seed, which is a vector of length six or NA (if NA a default seed of <code>rep(12345, 6)</code> is used). The second element of list is a positive substream number. See the MCMCpack specification for more details.
m0	Prior mean of the μ_0 parameter.
M0	Prior variance of the μ_0 parameter.

m1	Prior mean of the μ_1 parameter.
M1	Prior variance of the μ_1 parameter.
a0	$a_0/2$ is the shape parameter for the inverse-gamma prior on the σ_0^2 parameter.
b0	$b_0/2$ is the scale parameter for the inverse-gamma prior on the σ_0^2 parameter.
a1	$a_1/2$ is the shape parameter for the inverse-gamma prior on the σ_1^2 parameter.
b1	$b_1/2$ is the scale parameter for the inverse-gamma prior on the σ_1^2 parameter.
...	further arguments to be passed

Details

Consider the following partially observed 2 by 2 contingency table for unit t where $t = 1, \dots, ntables$:

	$Y = 0$	$Y = 1$	
-----	-----	-----	-----
$X = 0$	Y_{0t}		r_{0t}
-----	-----	-----	-----
$X = 1$	Y_{1t}		r_{1t}
-----	-----	-----	-----
	c_{0t}	c_{1t}	N_t

Where r_{0t} , r_{1t} , c_{0t} , c_{1t} , and N_t are non-negative integers that are observed. The interior cell entries are not observed. It is assumed that $Y_{0t}|r_{0t} \sim \text{Binomial}(r_{0t}, p_{0t})$ and $Y_{1t}|r_{1t} \sim \text{Binomial}(r_{1t}, p_{1t})$. Let $\theta_{0t} = \log(p_{0t}/(1 - p_{0t}))$, and $\theta_{1t} = \log(p_{1t}/(1 - p_{1t}))$.

The following prior distributions are assumed: $\theta_{0t} \sim \mathcal{N}(\mu_0, \sigma_0^2)$, $\theta_{1t} \sim \mathcal{N}(\mu_1, \sigma_1^2)$. θ_{0t} is assumed to be a priori independent of θ_{1t} for all t . In addition, we assume the following hyperpriors: $\mu_0 \sim \mathcal{N}(m_0, M_0)$, $\mu_1 \sim \mathcal{N}(m_1, M_1)$, $\sigma_0^2 \sim \text{IG}(a_0/2, b_0/2)$, and $\sigma_1^2 \sim \text{IG}(a_1/2, b_1/2)$.

The default priors have been chosen to make the implied prior distribution for p_0 and p_1 approximately uniform on (0,1).

Inference centers on p_0 , p_1 , μ_0 , μ_1 , σ_0^2 , and σ_1^2 . Univariate slice sampling (Neal, 2003) along with Gibbs sampling is used to sample from the posterior distribution.

See Section 5.4 of Wakefield (2003) for discussion of the priors used here. MCMChierEI departs from the Wakefield model in that the μ_0 and μ_1 are here assumed to be drawn from independent normal distributions whereas Wakefield assumes they are drawn from logistic distributions.

Value

An mcmc object that contains the sample from the posterior distribution. This object can be summarized by functions provided by the coda package.

References

- Jonathan C. Wakefield. 2003. "Ecological inference for 2x2 tables." Read before the Royal Statistical Society, on November 12th, 2003.
- Radford Neal. 2003. "Slice Sampling" (with discussion). *Annals of Statistics*, 31: 705-767.
- Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.

Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. 2002. *Output Analysis and Diagnostics for MCMC (CODA)*. <http://www-fis.iarc.fr/coda/>.

See Also

[MCMCdynamicEI](#), [plot.mcmc](#), [summary.mcmc](#)

Examples

```
## Not run:
## simulated data example
set.seed(3920)
n <- 100
r0 <- round(runif(n, 400, 1500))
r1 <- round(runif(n, 100, 4000))
p0.true <- pnorm(rnorm(n, m=0.5, s=0.25))
p1.true <- pnorm(rnorm(n, m=0.0, s=0.10))
y0 <- rbinom(n, r0, p0.true)
y1 <- rbinom(n, r1, p1.true)
c0 <- y0 + y1
c1 <- (r0+r1) - c0

## plot data
tomogplot(r0, r1, c0, c1)

## fit exchangeable hierarchical model
post <- MCMChierEI(r0,r1,c0,c1, mcmc=40000, thin=5, verbose=100,
                  seed=list(NA, 1))

p0meanHier <- colMeans(post)[1:n]
p1meanHier <- colMeans(post)[(n+1):(2*n)]

## plot truth and posterior means
pairs(cbind(p0.true, p0meanHier, p1.true, p1meanHier))
## End(Not run)
```

MCMCirt1d

Markov Chain Monte Carlo for One Dimensional Item Response Theory Model

Description

This function generates a sample from the posterior distribution of a one dimensional item response theory (IRT) model, with Normal priors on the subject abilities (ideal points), and multivariate Normal priors on the item parameters. The user supplies data and priors, and a sample from the posterior distribution is returned as an `mcmc` object, which can be subsequently analyzed with functions provided in the `coda` package.

If you are interested in fitting K-dimensional item response theory models, or would rather identify the model by placing constraints on the item parameters, please see [MCMCirtKd](#).

Usage

```
MCMCirt1d(datamatrix, theta.constraints=list(), burnin = 1000,
  mcmc = 20000, thin=1, verbose = 0, seed = NA, theta.start = NA,
  alpha.start = NA, beta.start = NA, t0 = 0, T0 = 1, ab0=0, AB0=.25,
  store.item = FALSE, store.ability = TRUE,
  drop.constant.items=TRUE, ... )
```

Arguments

- | | |
|--------------------------------|---|
| <code>datamatrix</code> | The matrix of data. Must be 0, 1, or missing values. The rows of <code>datamatrix</code> correspond to subjects and the columns correspond to items. |
| <code>theta.constraints</code> | A list specifying possible simple equality or inequality constraints on the ability parameters. A typical entry in the list has one of three forms: <code>varname=c</code> which will constrain the ability parameter for the subject named <code>varname</code> to be equal to <code>c</code> , <code>varname="+"</code> which will constrain the ability parameter for the subject named <code>varname</code> to be positive, and <code>varname="-"</code> which will constrain the ability parameter for the subject named <code>varname</code> to be negative. If <code>x</code> is a matrix without row names defaults names of "V1", "V2", ... , etc will be used. See Rivers (2003) for a thorough discussion of identification of IRT models. |
| <code>burnin</code> | The number of burn-in iterations for the sampler. |
| <code>mcmc</code> | The number of Gibbs iterations for the sampler. |
| <code>thin</code> | The thinning interval used in the simulation. The number of Gibbs iterations must be divisible by this value. |
| <code>verbose</code> | A switch which determines whether or not the progress of the sampler is printed to the screen. If <code>verbose</code> is greater than 0 then every <code>verboseth</code> iteration will be printed to the screen. |
| <code>seed</code> | The seed for the random number generator. If <code>NA</code> , the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first element of the list is the L'Ecuyer seed, which is a vector of length six or <code>NA</code> (if <code>NA</code> a default seed of <code>rep(12345, 6)</code> is used). The second element of list is a positive substream number. See the MCMCpack specification for more details. |
| <code>theta.start</code> | The starting values for the subject abilities (ideal points). This can either be a scalar or a column vector with dimension equal to the number of voters. If this takes a scalar value, then that value will serve as the starting value for all of the thetas. The default value of <code>NA</code> will choose the starting values based on an eigenvalue-eigenvector decomposition of the agreement score matrix formed from the <code>datamatrix</code> . |
| <code>alpha.start</code> | The starting values for the α difficulty parameters. This can either be a scalar or a column vector with dimension equal to the number of items. If this takes a scalar value, then that value will serve as the starting value for all of the alphas. The default value of <code>NA</code> will set the starting values based on a series of probit regressions that condition on the starting values of theta. |
| <code>beta.start</code> | The starting values for the β discrimination parameters. This can either be a scalar or a column vector with dimension equal to the number of items. If this takes a scalar value, then that value will serve as the starting value for all of the |

	betas. The default value of NA will set the starting values based on a series of probit regressions that condition on the starting values of theta.
t0	A scalar parameter giving the prior mean of the subject abilities (ideal points).
T0	A scalar parameter giving the prior precision (inverse variance) of the subject abilities (ideal points).
ab0	The prior mean of (alpha, beta). Can be either a scalar or a 2-vector. If a scalar both means will be set to the passed value. The prior mean is assumed to be the same across all items.
AB0	The prior precision of (alpha, beta). This can either be a scalar or a 2 by 2 matrix. If this takes a scalar value, then that value times an identity matrix serves as the prior precision. The prior precision is assumed to be the same across all items.
store.item	A switch that determines whether or not to store the item parameters for posterior analysis. <i>NOTE: In situations with many items storing the item parameters takes an enormous amount of memory, so store.item should only be FALSE if the chain is thinned heavily, or for applications with a small number of items.</i> By default, the item parameters are not stored.
store.ability	A switch that determines whether or not to store the ability parameters for posterior analysis. <i>NOTE: In situations with many individuals storing the ability parameters takes an enormous amount of memory, so store.ability should only be TRUE if the chain is thinned heavily, or for applications with a small number of individuals.</i> By default, the item parameters are stored.
drop.constant.items	A switch that determines whether or not items that have no variation should be deleted before fitting the model. Default = TRUE.
...	further arguments to be passed

Details

MCMCirt1d simulates from the posterior distribution using standard Gibbs sampling using data augmentation (a Normal draw for the subject abilities, a multivariate Normal draw for the item parameters, and a truncated Normal draw for the latent utilities). The simulation proper is done in compiled C++ code to maximize efficiency. Please consult the coda documentation for a comprehensive list of functions that can be used to analyze the posterior sample.

The model takes the following form. We assume that each subject has a subject ability (ideal point) denoted θ_j and that each item has a difficulty parameter α_i and discrimination parameter β_i . The observed choice by subject j on item i is the observed data matrix which is $(I \times J)$. We assume that the choice is dictated by an unobserved utility:

$$z_{i,j} = -\alpha_i + \beta_i \theta_j + \varepsilon_{i,j}$$

Where the errors are assumed to be distributed standard Normal. The parameters of interest are the subject abilities (ideal points) and the item parameters.

We assume the following priors. For the subject abilities (ideal points):

$$\theta_j \sim \mathcal{N}(t_0, T_0^{-1})$$

For the item parameters, the prior is:

$$[\alpha_i, \beta_i]' \sim \mathcal{N}_2(ab_0, AB_0^{-1})$$

The model is identified by the proper priors on the item parameters and constraints placed on the ability parameters.

As is the case with all measurement models, make sure that you have plenty of free memory, especially when storing the item parameters.

Value

An mcmc object that contains the sample from the posterior distribution. This object can be summarized by functions provided by the coda package.

References

- James H. Albert. 1992. "Bayesian Estimation of Normal Ogive Item Response Curves Using Gibbs Sampling." *Journal of Educational Statistics*. 17: 251-269.
- Joshua Clinton, Simon Jackman, and Douglas Rivers. 2004. "The Statistical Analysis of Roll Call Data." *American Political Science Review*. 98: 355-370.
- Valen E. Johnson and James H. Albert. 1999. "Ordinal Data Modeling." Springer: New York.
- Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.
- Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. 2002. *Output Analysis and Diagnostics for MCMC (CODA)*. <http://www-fis.iarc.fr/coda/>.
- Douglas Rivers. 2004. "Identification of Multidimensional Item-Response Models." Stanford University, typescript.

See Also

[plot.mcmc](#), [summary.mcmc](#), [MCMCirtKd](#)

Examples

```
## Not run:
## US Supreme Court Example with inequality constraints
data(SupremeCourt)
posterior1 <- MCMCirt1d(t(SupremeCourt),
  theta.constraints=list(Scalia="+", Ginsburg="-"),
  B0.alpha=.2, B0.beta=.2,
  burnin=500, mcmc=100000, thin=20, verbose=500,
  store.item=TRUE)
geweke.diag(posterior1)
plot(posterior1)
summary(posterior1)

## US Senate Example with equality constraints
data(Senate)
Sen.rollcalls <- Senate[,6:677]
posterior2 <- MCMCirt1d(Sen.rollcalls,
  theta.constraints=list(KENNEDY=-2, HELMS=2),
  burnin=2000, mcmc=100000, thin=20, verbose=500)
geweke.diag(posterior2)
plot(posterior2)
summary(posterior2)
## End(Not run)
```

MCMCirtKd

Markov Chain Monte Carlo for K-Dimensional Item Response Theory Model

Description

This function generates a sample from the posterior distribution of a K-dimensional item response theory (IRT) model, with standard normal priors on the subject abilities (ideal points), and normal priors on the item parameters. The user supplies data and priors, and a sample from the posterior distribution is returned as an mcmc object, which can be subsequently analyzed with functions provided in the coda package.

Usage

```
MCMCirtKd(datamatrix, dimensions, item.constraints=list(),
  burnin = 1000, mcmc = 10000, thin=1, verbose = 0, seed = NA,
  alphabeta.start = NA, b0 = 0, B0=0, store.item = FALSE,
  store.ability=TRUE, drop.constant.items=TRUE, ... )
```

Arguments

<code>datamatrix</code>	The matrix of data. Must be 0, 1, or missing values. It is of dimensionality subjects by items.
<code>dimensions</code>	The number of dimensions in the latent space.
<code>item.constraints</code>	List of lists specifying possible equality or simple inequality constraints on the item parameters. A typical entry in the list has one of three forms: <code>rowname=list(d, c)</code> which will constrain the <i>d</i> th item parameter for the item named <i>rowname</i> to be equal to <i>c</i> , <code>rowname=list(d, "+")</code> which will constrain the <i>d</i> th item parameter for the item named <i>rowname</i> to be positive, and <code>rowname=list(d, "-")</code> which will constrain the <i>d</i> th item parameter for the item named <i>rowname</i> to be negative. If <i>x</i> is a matrix without row names defaults names of "V1", "V2", ... , etc will be used. In a K dimensional model, the first item parameter for item <i>i</i> is the difficulty parameter (α_i), the second item parameter is the discrimination parameter on dimension 1 ($\beta_{i,1}$), the third item parameter is the discrimination parameter on dimension 2 ($\beta_{i,2}$), ..., and the (K+1)th item parameter is the discrimination parameter on dimension K ($\beta_{i,1}$). The item difficulty parameters (α) should generally not be constrained.
<code>burnin</code>	The number of burn-in iterations for the sampler.
<code>mcmc</code>	The number of iterations for the sampler.
<code>thin</code>	The thinning interval used in the simulation. The number of iterations must be divisible by this value.
<code>verbose</code>	A switch which determines whether or not the progress of the sampler is printed to the screen. If <code>verbose</code> is greater than 0 then every <code>verbose</code> th iteration will be printed to the screen.
<code>seed</code>	The seed for the random number generator. If NA, the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first

element of the list is the L'Ecuyer seed, which is a vector of length six or NA (if NA a default seed of `rep(12345, 6)` is used). The second element of list is a positive substream number. See the MCMCpack specification for more details.

<code>alphabeta.start</code>	The starting values for the α and β difficulty and discrimination parameters. If <code>alphabeta.start</code> is set to a scalar the starting value for all unconstrained item parameters will be set to that scalar. If <code>alphabeta.start</code> is a matrix of dimension $(K+1) \times items$ then the <code>alphabeta.start</code> matrix is used as the starting values (except for equality-constrained elements). If <code>alphabeta.start</code> is set to NA (the default) then starting values for unconstrained elements are set to values generated from a series of proportional odds logistic regression fits, and starting values for inequality constrained elements are set to either 1.0 or -1.0 depending on the nature of the constraints.
<code>b0</code>	The prior means of the α and β difficulty and discrimination parameters, stacked for all items. If a scalar is passed, it is used as the prior mean for all items.
<code>B0</code>	The prior precisions (inverse variances) of the independent normal prior on the item parameters. Can be either a scalar or a matrix of dimension $(K+1) \times items$.
<code>store.item</code>	A switch that determines whether or not to store the item parameters for posterior analysis. <i>NOTE: In applications with many items this takes an enormous amount of memory. If you have many items and want to want to store the item parameters you may want to thin the chain heavily.</i> By default, the item parameters are not stored.
<code>store.ability</code>	A switch that determines whether or not to store the subject abilities for posterior analysis. <i>NOTE: In applications with many subjects this takes an enormous amount of memory. If you have many subjects and want to want to store the ability parameters you may want to thin the chain heavily.</i> By default, the ability parameters are all stored.
<code>drop.constant.items</code>	A switch that determines whether or not items that have no variation should be deleted before fitting the model. Default = TRUE.
<code>...</code>	further arguments to be passed

Details

MCMCirtKd simulates from the posterior distribution using standard Gibbs sampling using data augmentation (a normal draw for the subject abilities, a multivariate normal draw for the item parameters, and a truncated normal draw for the latent utilities). The simulation proper is done in compiled C++ code to maximize efficiency. Please consult the coda documentation for a comprehensive list of functions that can be used to analyze the posterior sample.

The default number of burnin and mcmc iterations is much smaller than the typical default values in MCMCpack. This is because fitting this model is extremely computationally expensive. It does not mean that this small of a number of scans will yield good estimates. The priors of this model need to be proper for identification purposes. The user is asked to provide prior means and precisions (*not variances*) for the item parameters and the subject parameters.

The model takes the following form. We assume that each subject has an ability (ideal point) denoted θ_j ($K \times 1$), and that each item has a difficulty parameter α_i and discrimination parameter β_i ($K \times 1$). The observed choice by subject j on item i is the observed data matrix which is $(I \times J)$. We assume that the choice is dictated by an unobserved utility:

$$z_{i,j} = -\alpha_i + \beta_i' \theta_j + \varepsilon_{i,j}$$

Where the $\varepsilon_{i,j}$ s are assumed to be distributed standard normal. The parameters of interest are the subject abilities (ideal points) and the item parameters.

We assume the following priors. For the subject abilities (ideal points) we assume independent standard normal priors:

$$\theta_{j,k} \sim \mathcal{N}(0, 1)$$

These cannot be changed by the user. For each item parameter, we assume independent normal priors:

$$[\alpha_i, \beta_i]' \sim \mathcal{N}_{(K+1)}(b_{0,i}, B_{0,i})$$

Where $B_{0,i}$ is a diagonal matrix. One can specify a separate prior mean and precision for each item parameter.

The model is identified by the constraints on the item parameters (see Jackman 2001). The user cannot place constraints on the subject abilities. This identification scheme differs from that in `MCMCirt1d`, which uses constraints on the subject abilities to identify the model. In our experience, using subject ability constraints for models in greater than one dimension does not work particularly well.

As is the case with all measurement models, make sure that you have plenty of free memory, especially when storing the item parameters.

Value

An `mcmc` object that contains the posterior sample. This object can be summarized by functions provided by the `coda` package.

References

- James H. Albert. 1992. "Bayesian Estimation of Normal Ogive Item Response Curves Using Gibbs Sampling." *Journal of Educational Statistics*. 17: 251-269.
- Joshua Clinton, Simon Jackman, and Douglas Rivers. 2000. "The Statistical Analysis of Legislative Behavior: A Unified Approach." Paper presented at the Annual Meeting of the Political Methodology Society.
- Simon Jackman. 2001. "Multidimensional Analysis of Roll Call Data via Bayesian Simulation." *Political Analysis*. 9: 227-241.
- Valen E. Johnson and James H. Albert. 1999. "Ordinal Data Modeling." Springer: New York.
- Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.
- Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. 2002. *Output Analysis and Diagnostics for MCMC (CODA)*. <http://www-fis.iarc.fr/coda/>.
- Douglas Rivers. 2004. "Identification of Multidimensional Item-Response Models." Stanford University, typescript.

See Also

[plot.mcmc](#), [summary.mcmc](#), [MCMCirt1d](#), [MCMCordfactanal](#)

Examples

```
## Not run:
data(SupremeCourt)
# note that the rownames (the item names) are "1", "2", etc
posterior1 <- MCMCirtKd(t(SupremeCourt), dimensions=1,
```

```

        burnin=5000, mcmc=50000, thin=10,
        B0=.25, store.item=TRUE,
        item.constraints=list("1"=list(2, "-"))
plot (posterior1)
summary (posterior1)

data (Senate)
Sen.rollcalls <- Senate[,6:677]
posterior2 <- MCMCirtKd(Sen.rollcalls, dimensions=2,
        burnin=5000, mcmc=50000, thin=10,
        item.constraints=list(rc2=list(2, "-"), rc2=c(3,0),
                             rc3=list(3, "-")),
        B0=.25)
plot (posterior2)
summary (posterior2)
## End(Not run)

```

MCMCirtKdRob

Markov Chain Monte Carlo for Robust K-Dimensional Item Response Theory Model

Description

This function generates a posterior sample from a Robust K-dimensional item response theory (IRT) model with logistic link, independent standard normal priors on the subject abilities (ideal points), and independent normal priors on the item parameters. The user supplies data and priors, and a sample from the posterior distribution is returned as an mcmc object, which can be subsequently analyzed with functions provided in the coda package.

Usage

```

MCMCirtKdRob(datamatrix, dimensions, item.constraints=list(),
  ability.constraints=list(), burnin = 500, mcmc = 5000, thin=1,
  interval.method="step", theta.w=0.5, theta.mp=4,
  alphabeta.w=1.0, alphabeta.mp=4, delta0.w=NA, delta0.mp=3,
  delta1.w=NA, delta1.mp=3, verbose = FALSE, seed = NA,
  theta.start = NA, alphabeta.start = NA,
  delta0.start = NA, delta1.start = NA,
  b0 = 0, B0=0, k0=.1, k1=.1, c0=1, d0=1,
  c1=1, d1=1, store.item=TRUE, store.ability=FALSE,
  drop.constant.items=TRUE, ... )

```

Arguments

- `datamatrix` The matrix of data. Must be 0, 1, or missing values. It is of dimensionality subjects by items.
- `dimensions` The number of dimensions in the latent space.
- `item.constraints` List of lists specifying possible equality or simple inequality constraints on the item parameters. A typical entry in the list has one of three forms: `rowname=list(d, c)` which will constrain the `d`th item parameter for the item named `rowname` to be

equal to `c`, `rowname=list(d, "+")` which will constrain the `dth` item parameter for the item named `rowname` to be positive, and `rowname=list(d, "-")` which will constrain the `dth` item parameter for the item named `rowname` to be negative. If `datamatrix` is a matrix without row names defaults names of "V1", "V2", ... , etc will be used. In a K -dimensional model, the first item parameter for item i is the difficulty parameter (α_i), the second item parameter is the discrimination parameter on dimension 1 ($\beta_{i,1}$), the third item parameter is the discrimination parameter on dimension 2 ($\beta_{i,2}$), ..., and the $(K + 1)$ th item parameter is the discrimination parameter on dimension K ($\beta_{i,K}$). The item difficulty parameters (α) should generally not be constrained.

`ability.constraints`

List of lists specifying possible equality or simple inequality constraints on the ability parameters. A typical entry in the list has one of three forms: `colname=list(d, c)` which will constrain the `dth` ability parameter for the subject named `colname` to be equal to `c`, `colname=list(d, "+")` which will constrain the `dth` ability parameter for the subject named `colname` to be positive, and `colname=list(d, "-")` which will constrain the `dth` ability parameter for the subject named `colname` to be negative. If `datamatrix` is a matrix without column names defaults names of "V1", "V2", ... , etc will be used.

`burnin`

The number of burn-in iterations for the sampler.

`mcmc`

The number of iterations for the sampler after burn-in.

`thin`

The thinning interval used in the simulation. The number of iterations must be divisible by this value.

`interval.method`

Method for finding the slicing interval. Can be equal to either `step` in which case the stepping out algorithm of Neal (2003) is used or `doubling` in which case the doubling procedure of Neal (2003) is used. The stepping out method tends to be faster on a per-iteration basis as it typically requires few function calls. The doubling method expands the initial interval more quickly which makes the Markov chain mix somewhat more quickly– at least in some situations.

`theta.w`

The initial width of the slice sampling interval for each ability parameter (the elements of θ)

`theta.mp`

The parameter governing the maximum possible width of the slice interval for each ability parameter (the elements of θ). If `interval.method="step"` then the maximum width is `theta.w * theta.mp`.

If `interval.method="doubling"` then the maximum width is `theta.w * 2^theta.mp`.

`alphabetaw`

The initial width of the slice sampling interval for each item parameter (the elements of α and β)

`alphabeta.mp`

The parameter governing the maximum possible width of the slice interval for each item parameters (the elements of α and β). If `interval.method="step"` then the maximum width is `alphabeta.w * alphabeta.mp`.

If `interval.method="doubling"` then the maximum width is `alphabeta.w * 2^alphabeta.mp`.

`delta0.w`

The initial width of the slice sampling interval for δ_0

`delta0.mp`

The parameter governing the maximum possible width of the slice interval for δ_0 . If `interval.method="step"` then the maximum width is `delta0.w * delta0.mp`. If `interval.method="doubling"` then the maximum width is `delta0.w * 2^delta0.mp`.

<code>delta1.w</code>	The initial width of the slice sampling interval for δ_1
<code>delta1.mp</code>	The parameter governing the maximum possible width of the slice interval for δ_1 . If <code>interval.method="step"</code> then the maximum width is <code>delta1.w * delta1.mp</code> . If <code>interval.method="doubling"</code> then the maximum width is <code>delta1.w * 2^delta1.mp</code> .
<code>verbose</code>	A switch which determines whether or not the progress of the sampler is printed to the screen. If <code>verbose > 0</code> , the iteration number will be printed to the screen every <code>verbose</code> 'th iteration.
<code>seed</code>	The seed for the random number generator. If NA, the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first element of the list is the L'Ecuyer seed, which is a vector of length six or NA (if NA a default seed of <code>rep(12345, 6)</code> is used). The second element of list is a positive substream number. See the MCMCpack specification for more details.
<code>theta.start</code>	The starting values for the ability parameters θ . Can be either a scalar or a matrix with number of rows equal to the number of subjects and number of columns equal to the dimension K of the latent space. If <code>theta.start=NA</code> then starting values will be chosen that are 0 for unconstrained subjects, -0.5 for subjects with negative inequality constraints and 0.5 for subjects with positive inequality constraints.
<code>alphabeta.start</code>	The starting values for the α and β difficulty and discrimination parameters. If <code>alphabeta.start</code> is set to a scalar the starting value for all unconstrained item parameters will be set to that scalar. If <code>alphabeta.start</code> is a matrix of dimension $(K + 1) \times items$ then the <code>alphabeta.start</code> matrix is used as the starting values (except for equality-constrained elements). If <code>alphabeta.start</code> is set to NA (the default) then starting values for unconstrained elements are set to values generated from a series of proportional odds logistic regression fits, and starting values for inequality constrained elements are set to either 1.0 or -1.0 depending on the nature of the constraints.
<code>delta0.start</code>	The starting value for the δ_0 parameter.
<code>delta1.start</code>	The starting value for the δ_1 parameter.
<code>b0</code>	The prior means of the α and β difficulty and discrimination parameters, stacked for all items. If a scalar is passed, it is used as the prior mean for all items.
<code>B0</code>	The prior precisions (inverse variances) of the independent Normal prior on the item parameters. Can be either a scalar or a matrix of dimension $(K + 1) \times items$.
<code>k0</code>	δ_0 is constrained to lie in the interval between 0 and <code>k0</code> .
<code>k1</code>	δ_1 is constrained to lie in the interval between 0 and <code>k1</code> .
<code>c0</code>	Parameter governing the prior for δ_0 . δ_0 divided by <code>k0</code> is assumed to follow a beta distribution with first parameter <code>c0</code> .
<code>d0</code>	Parameter governing the prior for δ_0 . δ_0 divided by <code>k0</code> is assumed to follow a beta distribution with second parameter <code>d0</code> .
<code>c1</code>	Parameter governing the prior for δ_1 . δ_1 divided by <code>k1</code> is assumed to follow a beta distribution with first parameter <code>c1</code> .
<code>d1</code>	Parameter governing the prior for δ_1 . δ_1 divided by <code>k1</code> is assumed to follow a beta distribution with second parameter <code>d1</code> .

`store.item` A switch that determines whether or not to store the item parameters for posterior analysis. *NOTE: This typically takes an enormous amount of memory, so should only be used if the chain is thinned heavily, or for applications with a small number of items.* By default, the item parameters are not stored.

`store.ability` A switch that determines whether or not to store the subject abilities for posterior analysis. By default, the item parameters are all stored.

`drop.constant.items` A switch that determines whether or not items that have no variation should be deleted before fitting the model. Default = TRUE.

... further arguments to be passed

Details

MCMCirtKd simulates from the posterior using the slice sampling algorithm of Neal (2003). The simulation proper is done in compiled C++ code to maximize efficiency. Please consult the coda documentation for a comprehensive list of functions that can be used to analyze the posterior sample.

The model takes the following form. We assume that each subject has an subject ability (ideal point) denoted θ_j ($K \times 1$), and that each item has a scalar difficulty parameter α_i and discrimination parameter β_i ($K \times 1$). The observed choice by subject j on item i is the observed data matrix which is ($I \times J$).

The probability that subject j answers item i correctly is assumed to be:

$$\pi_{ij} = \delta_0 + (1 - \delta_0 - \delta_1) / (1 + \exp(\alpha_i - \beta_i \theta_j))$$

This model was discussed in Bafumi et al. (2005).

We assume the following priors. For the subject abilities (ideal points) we assume independent standard Normal priors:

$$\theta_{j,k} \sim \mathcal{N}(0, 1)$$

These cannot be changed by the user. For each item parameter, we assume independent Normal priors:

$$[\alpha_i, \beta_i]' \sim \mathcal{N}_{(K+1)}(b_{0,i}, B_{0,i})$$

Where $B_{0,i}$ is a diagonal matrix. One can specify a separate prior mean and precision for each item parameter. We also assume $\delta_0/k_0 \sim \text{Beta}(c_0, d_0)$ and $\delta_1/k_1 \sim \text{Beta}(c_1, d_1)$.

The model is identified by constraints on the item parameters and / or ability parameters. See Rivers (2004) for a discussion of identification of IRT models.

As is the case with all measurement models, make sure that you have plenty of free memory, especially when storing the item parameters.

Value

An `mcmc` object that contains the posterior sample. This object can be summarized by functions provided by the coda package.

References

- James H. Albert. 1992. "Bayesian Estimation of Normal Ogive Item Response Curves Using Gibbs Sampling." *Journal of Educational Statistics*. 17: 251-269.
- Joseph Bafumi, Andrew Gelman, David K. Park, and Noah Kaplan. 2005. "Practical Issues in Implementing and Understanding Bayesian Ideal Point Estimation." *Political Analysis*.
- Joshua Clinton, Simon Jackman, and Douglas Rivers. 2000. "The Statistical Analysis of Legislative Behavior: A Unified Approach." Paper presented at the Annual Meeting of the Political Methodology Society.
- Simon Jackman. 2001. "Multidimensional Analysis of Roll Call Data via Bayesian Simulation." *Political Analysis*. 9: 227-241.
- Valen E. Johnson and James H. Albert. 1999. "Ordinal Data Modeling." Springer: New York.
- Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.
- Radford Neal. 2003. "Slice Sampling" (with discussion). *Annals of Statistics*, 31: 705-767.
- Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. 2002. *Output Analysis and Diagnostics for MCMC (CODA)*. <http://www-fis.iarc.fr/coda/>.
- Douglas Rivers. 2004. "Identification of Multidimensional Item-Response Models." Stanford University, typescript.

See Also

[plot.mcmc](#), [summary.mcmc](#), [MCMCirt1d](#), [MCMCirtKd](#)

Examples

```
## Not run:
## Court example with ability (ideal point) and
## item (case) constraints
data(SupremeCourt)
post1 <- MCMCirtKdRob(t(SupremeCourt), dimensions=1,
                    burnin=500, mcmc=5000, thin=1,
                    B0=.25, store.item=TRUE, store.ability=TRUE,
                    ability.constraints=list("Thomas"=list(1, "+"),
                    "Stevens"=list(1, -4)),
                    item.constraints=list("1"=list(2, "-")),
                    verbose=50)

plot(post1)
summary(post1)

## Senate example with ability (ideal point) constraints
data(Senate)
namestring <- as.character(Senate$member)
namestring[78] <- "CHAFEE1"
namestring[79] <- "CHAFEE2"
namestring[59] <- "SMITH.NH"
namestring[74] <- "SMITH.OR"
rownames(Senate) <- namestring
post2 <- MCMCirtKdRob(Senate[,6:677], dimensions=1,
                    burnin=1000, mcmc=5000, thin=1,
                    ability.constraints=list("KENNEDY"=list(1, -4),
                    "HELMS"=list(1, 4), "ASHCROFT"=list(1, "+"),
                    "BOXER"=list(1, "-"), "KERRY"=list(1, "-)),
```

```

                                "HATCH"=list(1, "+")),
                                B0=0.1, store.ability=TRUE, store.item=FALSE,
                                verbose=5, k0=0.15, k1=0.15,
                                delta0.start=0.13, delta1.start=0.13)

plot(post2)
summary(post2)
## End(Not run)

```

MCMClogit

Markov Chain Monte Carlo for Logistic Regression

Description

This function generates a sample from the posterior distribution of a logistic regression model using a random walk Metropolis algorithm. The user supplies data and priors, and a sample from the posterior distribution is returned as an mcmc object, which can be subsequently analyzed with functions provided in the coda package.

Usage

```

MCMClogit(formula, data = parent.frame(), burnin = 1000, mcmc = 10000,
           thin=1, tune=1.1, verbose = 0, seed = NA, beta.start = NA,
           b0 = 0, B0 = 0, user.prior.density=NULL, logfun=TRUE,
           marginal.likelihood=c("none", "Laplace"), ...)

```

Arguments

formula	Model formula.
data	Data frame.
burnin	The number of burn-in iterations for the sampler.
mcmc	The number of Metropolis iterations for the sampler.
thin	The thinning interval used in the simulation. The number of mcmc iterations must be divisible by this value.
tune	Metropolis tuning parameter. Can be either a positive scalar or a k -vector, where k is the length of β . Make sure that the acceptance rate is satisfactory (typically between 0.20 and 0.5) before using the posterior sample for inference.
verbose	A switch which determines whether or not the progress of the sampler is printed to the screen. If <code>verbose</code> is greater than 0 the iteration number, the current beta vector, and the Metropolis acceptance rate are printed to the screen every <code>verbose</code> th iteration.
seed	The seed for the random number generator. If NA, the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first element of the list is the L'Ecuyer seed, which is a vector of length six or NA (if NA a default seed of <code>rep(12345, 6)</code> is used). The second element of list is a positive substream number. See the MCMCpack specification for more details.

<code>beta.start</code>	The starting value for the β vector. This can either be a scalar or a column vector with dimension equal to the number of betas. If this takes a scalar value, then that value will serve as the starting value for all of the betas. The default value of NA will use the maximum likelihood estimate of β as the starting value.
<code>b0</code>	If <code>user.prior.density==NULL</code> <code>b0</code> is the prior mean of β under a multivariate normal prior. This can either be a scalar or a column vector with dimension equal to the number of betas. If this takes a scalar value, then that value will serve as the prior mean for all of the betas.
<code>B0</code>	If <code>user.prior.density==NULL</code> <code>B0</code> is the prior precision of β under a multivariate normal prior. This can either be a scalar or a square matrix with dimensions equal to the number of betas. If this takes a scalar value, then that value times an identity matrix serves as the prior precision of β . Default value of 0 is equivalent to an improper uniform prior for beta.
<code>user.prior.density</code>	If non-NULL, the prior (log)density up to a constant of proportionality. This must be a function defined in R whose first argument is a continuous (possibly vector) variable. This first argument is the point in the state space at which the prior (log)density is to be evaluated. Additional arguments can be passed to <code>user.prior.density()</code> by inserting them in the call to <code>MCMClogit()</code> . See the Details section and the examples below for more information.
<code>logfun</code>	Logical indicating whether <code>user.prior.density()</code> returns the natural log of a density function (TRUE) or a density (FALSE).
<code>marginal.likelihood</code>	How should the marginal likelihood be calculated? Options are: <code>none</code> in which case the marginal likelihood will not be calculated or <code>Laplace</code> in which case the Laplace approximation (see Kass and Raftery, 1995) is used.
<code>...</code>	further arguments to be passed

Details

`MCMClogit` simulates from the posterior distribution of a logistic regression model using a random walk Metropolis algorithm. The simulation proper is done in compiled C++ code to maximize efficiency. Please consult the coda documentation for a comprehensive list of functions that can be used to analyze the posterior sample.

The model takes the following form:

$$y_i \sim \text{Bernoulli}(\pi_i)$$

Where the inverse link function:

$$\pi_i = \frac{\exp(x_i' \beta)}{1 + \exp(x_i' \beta)}$$

By default, we assume a multivariate Normal prior on β :

$$\beta \sim \mathcal{N}(b_0, B_0^{-1})$$

Additionally, arbitrary user-defined priors can be specified with the `user.prior.density` argument.

If the default multivariate normal prior is used, the Metropolis proposal distribution is centered at the current value of β and has variance-covariance $V = T(B_0 + C^{-1})^{-1}T$, where T is a the diagonal positive definite matrix formed from the `tune`, B_0 is the prior precision, and C is the large sample variance-covariance matrix of the MLEs. This last calculation is done via an initial call to `glm`.


```

plot (posterior)
summary (posterior)

## End (Not run)

```

MCMCmetrop1R

*Metropolis Sampling from User-Written R function***Description**

This function allows a user to construct a sample from a user-defined continuous distribution using a random walk Metropolis algorithm.

Usage

```

MCMCmetrop1R (fun, theta.init, burnin = 500, mcmc = 20000, thin = 1,
              tune = 1, verbose = 0, seed=NA, logfun = TRUE,
              force.samp=FALSE, optim.trace = 0, optim.REPORT = 10,
              optim.maxit = 500, ...)

```

Arguments

fun	The unnormalized (log)density of the distribution from which to take a sample. This must be a function defined in R whose first argument is a continuous (possibly vector) variable. This first argument is the point in the state space at which the (log)density is to be evaluated. Additional arguments can be passed to <code>fun()</code> by inserting them in the call to <code>MCMCmetrop1R()</code> . See the Details section and the examples below for more information.
theta.init	Starting values for the sampling. Must be of the appropriate dimension. It must also be the case that <code>fun(theta.init, ...)</code> is greater than $-\text{Inf}$ if <code>fun()</code> is a logdensity or greater than 0 if <code>fun()</code> is a density.
burnin	The number of burn-in iterations for the sampler.
mcmc	The number of MCMC iterations after burnin.
thin	The thinning interval used in the simulation. The number of MCMC iterations must be divisible by this value.
tune	The tuning parameter for the Metropolis sampling. Can be either a positive scalar or a k -vector, where k is the length of θ .
verbose	A switch which determines whether or not the progress of the sampler is printed to the screen. If <code>verbose</code> is greater than 0 the iteration number, the θ vector, the function value, and the Metropolis acceptance rate are sent to the screen every <code>verbose</code> th iteration.
seed	The seed for the random number generator. If NA, the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first element of the list is the L'Ecuyer seed, which is a vector of length six or NA (if NA a default seed of <code>rep(12345, 6)</code> is used). The second element of list is a positive substream number. See the MCMCpack specification for more details.

<code>logfun</code>	Logical indicating whether <code>fun</code> returns the natural log of a density function (TRUE) or a density (FALSE).
<code>force.samp</code>	Logical indicating whether the sampling should proceed if the Hessian calculated from the initial call to <code>optim</code> routine to maximize the (log)density is not negative definite. If <code>force.samp==TRUE</code> and the Hessian from <code>optim</code> is non-negative definite, the Hessian is rescaled by subtracting small values from its main diagonal until it is negative definite. Sampling proceeds using this rescaled Hessian in place of the original Hessian from <code>optim</code> . By default, if <code>force.samp==FALSE</code> and the Hessian from <code>optim</code> is non-negative definite, an error message is printed and the call to <code>MCMCmetrop1R</code> is terminated. <i>Please note that a non-negative Hessian at the mode is often an indication that the function of interest is not a proper density. Thus, <code>force.samp</code> should only be set equal to <code>TRUE</code> with great caution.</i>
<code>optim.trace</code>	The value of the <code>trace</code> parameter sent to <code>optim</code> during an initial maximization of <code>fun</code> .
<code>optim.REPORT</code>	The value of the <code>REPORT</code> parameter sent to <code>optim</code> during an initial maximization of <code>fun</code> .
<code>optim.maxit</code>	The value of the <code>maxit</code> parameter sent to <code>optim</code> during an initial maximization of <code>fun</code> .
<code>...</code>	Additional arguments.

Details

`MCMCmetrop1R` produces a sample from a user-defined distribution using a random walk Metropolis algorithm with multivariate normal proposal distribution. See Gelman et al. (2003) and Robert & Casella (2004) for details of the random walk Metropolis algorithm.

The proposal distribution is centered at the current value of θ and has variance-covariance $V = T(-1 \cdot H)^{-1}T$, where T is a the diagonal positive definite matrix formed from the `tune` and H is the approximate Hessian of `fun` evaluated at its mode. This last calculation is done via an initial call to `optim`.

Value

An `mcmc` object that contains the posterior density sample. This object can be summarized by functions provided by the `coda` package.

References

- Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. 2003. *Bayesian Data Analysis*. 2nd Edition. Boca Raton: Chapman & Hall/CRC.
- Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.
- Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. 2002. *Output Analysis and Diagnostics for MCMC (CODA)*. <http://www-fis.iarc.fr/coda/>.
- Christian P. Robert and George Casella. 2004. *Monte Carlo Statistical Methods*. 2nd Edition. New York: Springer.

See Also

[plot.mcmc](#), [summary.mcmc](#), [optim](#), [metrop](#)

Examples

```
## Not run:

## logistic regression with an improper uniform prior
## X and y are passed as args to MCMCmetrop1R

logitfun <- function(beta, y, X){
  eta <- X %*% beta
  p <- 1.0/(1.0+exp(-eta))
  sum( y * log(p) + (1-y)*log(1-p) )
}

x1 <- rnorm(1000)
x2 <- rnorm(1000)
Xdata <- cbind(1,x1,x2)
p <- exp(.5 - x1 + x2)/(1+exp(.5 - x1 + x2))
yvector <- rbinom(1000, 1, p)

post.samp <- MCMCmetrop1R(logitfun, theta.init=c(0,0,0),
                          X=Xdata, y=yvector,
                          thin=1, mcmc=40000, burnin=500,
                          tune=c(1.5, 1.5, 1.5),
                          verbose=500, logfun=TRUE, optim.maxit=100)

raftery.diag(post.samp)
plot(post.samp)
summary(post.samp)
## #####

## negative binomial regression with an improper uniform prior
## X and y are passed as args to MCMCmetrop1R
negbinfun <- function(theta, y, X){
  k <- length(theta)
  beta <- theta[1:(k-1)]
  alpha <- exp(theta[k])
  mu <- exp(X %*% beta)
  log.like <- sum(
    lgamma(y+alpha) - lfactorial(y) - lgamma(alpha) +
    alpha * log(alpha/(alpha+mu)) +
    y * log(mu/(alpha+mu))
  )
}

n <- 1000
x1 <- rnorm(n)
x2 <- rnorm(n)
XX <- cbind(1,x1,x2)
mu <- exp(1.5+x1+2*x2)*rgamma(n,1)
yy <- rpois(n, mu)

post.samp <- MCMCmetrop1R(negbinfun, theta.init=c(0,0,0,0), y=yy, X=XX,
                          thin=1, mcmc=35000, burnin=1000,
                          tune=1.5, verbose=500, logfun=TRUE,
                          optim.maxit=500, seed=list(NA,1))

raftery.diag(post.samp)
```

```

plot(post.samp)
summary(post.samp)
## #####

## sample from a univariate normal distribution with
## mean 5 and standard deviation 0.1
##
## (MCMC obviously not necessary here and this should
## really be done with the logdensity for better
## numerical accuracy-- this is just an illustration of how
## MCMCmetrop1R works with a density rather than logdensity)

post.samp <- MCMCmetrop1R(dnorm, theta.init=5.3, mean=5, sd=0.1,
                          thin=1, mcmc=50000, burnin=500,
                          tune=2.0, verbose=5000, logfun=FALSE)

summary(post.samp)

## End(Not run)

```

MCMCmixfactanal *Markov Chain Monte Carlo for Mixed Data Factor Analysis Model*

Description

This function generates a sample from the posterior distribution of a mixed data (both continuous and ordinal) factor analysis model. Normal priors are assumed on the factor loadings and factor scores, improper uniform priors are assumed on the cutpoints, and inverse gamma priors are assumed for the error variances (uniquenesses). The user supplies data and parameters for the prior distributions, and a sample from the posterior distribution is returned as an mcmc object, which can be subsequently analyzed with functions provided in the coda package.

Usage

```

MCMCmixfactanal(x, factors, lambda.constraints=list(),
                data=parent.frame(), burnin = 1000, mcmc = 20000,
                thin=1, tune=NA, verbose = 0, seed = NA,
                lambda.start = NA, psi.start=NA,
                l0=0, L0=0, a0=0.001, b0=0.001,
                store.lambda=TRUE, store.scores=FALSE,
                std.mean=TRUE, std.var=TRUE, ... )

```

Arguments

x	A one-sided formula containing the manifest variables. Ordinal (including dichotomous) variables must be coded as ordered factors. Each level of these ordered factors must be present in the data passed to the function. NOTE: data input is different in MCMCmixfactanal than in either MCMCfactanal or MCMCordfactanal.
factors	The number of factors to be fitted.

<code>lambda.constraints</code>	List of lists specifying possible equality or simple inequality constraints on the factor loadings. A typical entry in the list has one of three forms: <code>varname=list(d, c)</code> which will constrain the <code>d</code> th loading for the variable named <code>varname</code> to be equal to <code>c</code> , <code>varname=list(d, "+")</code> which will constrain the <code>d</code> th loading for the variable named <code>varname</code> to be positive, and <code>varname=list(d, "-")</code> which will constrain the <code>d</code> th loading for the variable named <code>varname</code> to be negative. If <code>x</code> is a matrix without column names defaults names of "V1", "V2", ... , etc will be used. Note that, unlike <code>MCMCfactanal</code> , the Λ matrix used here has <code>factors+1</code> columns. The first column of Λ corresponds to negative item difficulty parameters for ordinal manifest variables and mean parameters for continuous manifest variables and should generally not be constrained directly by the user.
<code>data</code>	A data frame.
<code>burnin</code>	The number of burn-in iterations for the sampler.
<code>mcmc</code>	The number of iterations for the sampler.
<code>thin</code>	The thinning interval used in the simulation. The number of iterations must be divisible by this value.
<code>tune</code>	The tuning parameter for the Metropolis-Hastings sampling. Can be either a scalar or a k -vector (where k is the number of manifest variables). <code>tune</code> must be strictly positive.
<code>verbose</code>	A switch which determines whether or not the progress of the sampler is printed to the screen. If <code>verbose</code> is great than 0 the iteration number and the Metropolis-Hastings acceptance rate are printed to the screen every <code>verbose</code> th iteration.
<code>seed</code>	The seed for the random number generator. If NA, the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first element of the list is the L'Ecuyer seed, which is a vector of length six or NA (if NA a default seed of <code>rep(12345, 6)</code> is used). The second element of list is a positive substream number. See the <code>MCMCpack</code> specification for more details.
<code>lambda.start</code>	Starting values for the factor loading matrix <code>Lambda</code> . If <code>lambda.start</code> is set to a scalar the starting value for all unconstrained loadings will be set to that scalar. If <code>lambda.start</code> is a matrix of the same dimensions as <code>Lambda</code> then the <code>lambda.start</code> matrix is used as the starting values (except for equality-constrained elements). If <code>lambda.start</code> is set to NA (the default) then starting values for unconstrained elements in the first column of <code>Lambda</code> are based on the observed response pattern, the remaining unconstrained elements of <code>Lambda</code> are set to 0, and starting values for inequality constrained elements are set to either 1.0 or -1.0 depending on the nature of the constraints.
<code>psi.start</code>	Starting values for the error variance (uniqueness) matrix. If <code>psi.start</code> is set to a scalar then the starting value for all diagonal elements of <code>Psi</code> that represent error variances for continuous variables are set to this value. If <code>psi.start</code> is a k -vector (where k is the number of manifest variables) then the starting value of <code>Psi</code> has <code>psi.start</code> on the main diagonal with the exception that entries corresponding to error variances for ordinal variables are set to 1. If <code>psi.start</code> is set to NA (the default) the starting values of all the continuous variable uniquenesses are set to 0.5. Error variances for ordinal response variables are always constrained (regardless of the value of <code>psi.start</code> to have an error variance of 1 in order to achieve identification.

l0	The means of the independent Normal prior on the factor loadings. Can be either a scalar or a matrix with the same dimensions as <code>Lambda</code> .
L0	The precisions (inverse variances) of the independent Normal prior on the factor loadings. Can be either a scalar or a matrix with the same dimensions as <code>Lambda</code> .
a0	Controls the shape of the inverse Gamma prior on the uniqueness. The actual shape parameter is set to <code>a0/2</code> . Can be either a scalar or a k -vector.
b0	Controls the scale of the inverse Gamma prior on the uniquenesses. The actual scale parameter is set to <code>b0/2</code> . Can be either a scalar or a k -vector.
<code>store.lambda</code>	A switch that determines whether or not to store the factor loadings for posterior analysis. By default, the factor loadings are all stored.
<code>store.scores</code>	A switch that determines whether or not to store the factor scores for posterior analysis. <i>NOTE: This takes an enormous amount of memory, so should only be used if the chain is thinned heavily, or for applications with a small number of observations.</i> By default, the factor scores are not stored.
<code>std.mean</code>	If <code>TRUE</code> (the default) the continuous manifest variables are rescaled to have zero mean.
<code>std.var</code>	If <code>TRUE</code> (the default) the continuous manifest variables are rescaled to have unit variance.
<code>...</code>	further arguments to be passed

Details

The model takes the following form:

Let $i = 1, \dots, N$ index observations and $j = 1, \dots, K$ index response variables within an observation. An observed variable x_{ij} can be either ordinal with a total of C_j categories or continuous. The distribution of X is governed by a $N \times K$ matrix of latent variables X^* and a series of cutpoints γ . X^* is assumed to be generated according to:

$$x_i^* = \Lambda \phi_i + \epsilon_i$$

$$\epsilon_i \sim \mathcal{N}(0, \Psi)$$

where x_i^* is the k -vector of latent variables specific to observation i , Λ is the $k \times d$ matrix of factor loadings, and ϕ_i is the d -vector of latent factor scores. It is assumed that the first element of ϕ_i is equal to 1 for all i .

If the j th variable is ordinal, the probability that it takes the value c in observation i is:

$$\pi_{ijc} = \Phi(\gamma_{jc} - \Lambda'_j \phi_i) - \Phi(\gamma_{j(c-1)} - \Lambda'_j \phi_i)$$

If the j th variable is continuous, it is assumed that $x_{ij}^* = x_{ij}$ for all i .

The implementation used here assumes independent conjugate priors for each element of Λ and each ϕ_i . More specifically we assume:

$$\Lambda_{ij} \sim \mathcal{N}(l_{0ij}, L_{0ij}^{-1}), i = 1, \dots, k, j = 1, \dots, d$$

$$\phi_{i(2:d)} \sim \mathcal{N}(0, I), i = 1, \dots, n$$


```

      RPM, Length, Wheelbase, Width, Weight, Origin)
rownames(new.cars) <- paste(Manufacturer, Model)
detach(Cars93)

# drop obs 57 (Mazda RX 7) b/c it has a rotary engine
new.cars <- new.cars[-57,]
# drop 3 cylinder cars
new.cars <- new.cars[new.cars$Cylinders!=3,]
# drop 5 cylinder cars
new.cars <- new.cars[new.cars$Cylinders!=5,]

new.cars$log.Price <- log(new.cars$Price)
new.cars$log.MPG.city <- log(new.cars$MPG.city)
new.cars$log.MPG.highway <- log(new.cars$MPG.highway)
new.cars$log.EngineSize <- log(new.cars$EngineSize)
new.cars$log.Horsepower <- log(new.cars$Horsepower)

new.cars$Cylinders <- ordered(new.cars$Cylinders)
new.cars$Origin <- ordered(new.cars$Origin)

post <- MCMCmixfactanal(~log.Price+log.MPG.city+
  log.MPG.highway+Cylinders+log.EngineSize+
  log.Horsepower+RPM+Length+
  Wheelbase+Width+Weight+Origin, data=new.cars,
  lambda.constraints=list(log.Horsepower=list(2,"+"),
  log.Horsepower=c(3,0), weight=list(3,"+")),
  factors=2,
  burnin=5000, mcmc=500000, thin=100, verbose=500,
  L0=.25, tune=3.0)

plot(post)
summary(post)

## End(Not run)

```

Description

This function generates a sample from the posterior distribution of a multinomial logistic regression model using either a random walk Metropolis algorithm or a slice sampler. The user supplies data and priors, and a sample from the posterior distribution is returned as an mcmc object, which can be subsequently analyzed with functions provided in the coda package.

Usage

```

MCMCmnl(formula, baseline = NULL, data = parent.frame(),
  burnin = 1000, mcmc = 10000, thin = 1,
  mcmc.method = "MH", tune = 1.1, verbose = 0,
  seed = NA, beta.start = NA, b0 = 0, B0 = 0, ...)

```

Arguments

<code>formula</code>	<p>Model formula.</p> <p>If the choicetypes do not vary across individuals, the y variable should be a factor or numeric variable that gives the observed choice of each individual. If the choicetypes do vary across individuals, y should be a $n \times p$ matrix where n is the number of individuals and p is the maximum number of choices in any choicetype. Here each column of y corresponds to a particular observed choice and the elements of y should be either 0 (not chosen but available), 1 (chosen), or -999 (not available).</p> <p>Choice-specific covariates have to be entered using the syntax: <code>choicevar(cvar, "var", "choice")</code> where <code>cvar</code> is the name of a variable in <code>data</code>, <code>"var"</code> is the name of the new variable to be created, and <code>"choice"</code> is the level of y that <code>cvar</code> corresponds to. Specifying each choice-specific covariate will typically require p calls to the <code>choicevar</code> function in the formula.</p> <p>Individual specific covariates can be entered into the formula normally.</p> <p>See the examples section below to see the syntax used to fit various models.</p>
<code>baseline</code>	<p>The baseline category of the response variable. <code>baseline</code> should be set equal to one of the observed levels of the response variable. If left equal to <code>NULL</code> then the baseline level is set to the alpha-numerically first element of the response variable. If the choicetypes vary across individuals, the baseline choice must be in the choicetype of each individual.</p>
<code>data</code>	<p>The data frame used for the analysis. Each row of the dataframe should correspond to an individual who is making a choice.</p>
<code>burnin</code>	<p>The number of burn-in iterations for the sampler.</p>
<code>mcmc</code>	<p>The number of iterations to run the sampler past burn-in.</p>
<code>thin</code>	<p>The thinning interval used in the simulation. The number of <code>mcmc</code> iterations must be divisible by this value.</p>
<code>mcmc.method</code>	<p>Can be set to either "MH" (default) or "slice" to perform random walk Metropolis sampling or slice sampling respectively.</p>
<code>tune</code>	<p>Metropolis tuning parameter. Can be either a positive scalar or a k-vector, where k is the length of β. Make sure that the acceptance rate is satisfactory (typically between 0.20 and 0.5) before using the posterior sample for inference.</p>
<code>verbose</code>	<p>A switch which determines whether or not the progress of the sampler is printed to the screen. If <code>verbose</code> is greater than 0 the iteration number, the current beta vector, and the Metropolis acceptance rate are printed to the screen every <code>verbose</code>th iteration.</p>
<code>seed</code>	<p>The seed for the random number generator. If <code>NA</code>, the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first element of the list is the L'Ecuyer seed, which is a vector of length six or <code>NA</code> (if <code>NA</code> a default seed of <code>rep(12345, 6)</code> is used). The second element of list is a positive substream number. See the MCMCpack specification for more details.</p>
<code>beta.start</code>	<p>The starting value for the β vector. This can either be a scalar or a column vector with dimension equal to the number of betas. If this takes a scalar value, then that value will serve as the starting value for all of the betas. The default value of <code>NA</code> will use the maximum likelihood estimate of β as the starting value.</p>

b0	The prior mean of β . This can either be a scalar or a column vector with dimension equal to the number of betas. If this takes a scalar value, then that value will serve as the prior mean for all of the betas.
B0	The prior precision of β . This can either be a scalar or a square matrix with dimensions equal to the number of betas. If this takes a scalar value, then that value times an identity matrix serves as the prior precision of β . Default value of 0 is equivalent to an improper uniform prior for beta.
...	Further arguments to be passed.

Details

MCMCmnl simulates from the posterior distribution of a multinomial logistic regression model using either a random walk Metropolis algorithm or a univariate slice sampler. The simulation proper is done in compiled C++ code to maximize efficiency. Please consult the coda documentation for a comprehensive list of functions that can be used to analyze the posterior sample.

The model takes the following form:

$$y_i \sim \text{Multinomial}(\pi_i)$$

where:

$$\pi_{ij} = \frac{\exp(x'_{ij}\beta)}{\sum_{k=1}^p \exp(x'_{ik}\beta)}$$

We assume a multivariate Normal prior on β :

$$\beta \sim \mathcal{N}(b_0, B_0^{-1})$$

The Metropolis proposal distribution is centered at the current value of β and has variance-covariance $V = T(B_0 + C^{-1})^{-1}T$, where T is a the diagonal positive definite matrix formed from the `tune`, B_0 is the prior precision, and C is the large sample variance-covariance matrix of the MLEs. This last calculation is done via an initial call to `optim`.

Value

An `mcmc` object that contains the posterior sample. This object can be summarized by functions provided by the coda package.

References

- Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.
- Radford Neal. 2003. "Slice Sampling" (with discussion). *Annals of Statistics*, 31: 705-767.
- Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. 2002. *Output Analysis and Diagnostics for MCMC (CODA)*. <http://www-fis.iarc.fr/coda/>.

See Also

[plot.mcmc](#), [summary.mcmc](#), [multinom](#)

Examples

```

## Not run:
data(Nethvote)

## just a choice-specific X var
post1 <- MCMCmnl(vote ~
  choicevar(distD66, "sqdist", "D66") +
  choicevar(distPvdA, "sqdist", "PvdA") +
  choicevar(distVVD, "sqdist", "VVD") +
  choicevar(distCDA, "sqdist", "CDA"),
  baseline="D66", mcmc.method="MH", B0=0,
  verbose=500, mcmc=100000, thin=10, tune=1.0,
  data=Nethvote)

plot(post1)
summary(post1)

## just individual-specific X vars
post2<- MCMCmnl(vote ~
  relig + class + income + educ + age + urban,
  baseline="D66", mcmc.method="MH", B0=0,
  verbose=500, mcmc=100000, thin=10, tune=0.5,
  data=Nethvote)

plot(post2)
summary(post2)

## both choice-specific and individual-specific X vars
post3 <- MCMCmnl(vote ~
  choicevar(distD66, "sqdist", "D66") +
  choicevar(distPvdA, "sqdist", "PvdA") +
  choicevar(distVVD, "sqdist", "VVD") +
  choicevar(distCDA, "sqdist", "CDA") +
  relig + class + income + educ + age + urban,
  baseline="D66", mcmc.method="MH", B0=0,
  verbose=500, mcmc=100000, thin=10, tune=0.5,
  data=Nethvote)

plot(post3)
summary(post3)

## numeric y variable
nethvote$vote <- as.numeric(nethvote$vote)
post4 <- MCMCmnl(vote ~
  choicevar(distD66, "sqdist", "2") +
  choicevar(distPvdA, "sqdist", "3") +
  choicevar(distVVD, "sqdist", "4") +
  choicevar(distCDA, "sqdist", "1") +
  relig + class + income + educ + age + urban,
  baseline="2", mcmc.method="MH", B0=0,
  verbose=500, mcmc=100000, thin=10, tune=0.5,
  data=Nethvote)

plot(post4)

```

```

summary(post4)

## Simulated data example with nonconstant choicest
n <- 1000
y <- matrix(0, n, 4)
colnames(y) <- c("a", "b", "c", "d")
xa <- rnorm(n)
xb <- rnorm(n)
xc <- rnorm(n)
xd <- rnorm(n)
xchoice <- cbind(xa, xb, xc, xd)
z <- rnorm(n)
for (i in 1:n){
  ## randomly determine choicest (c is always in choicest)
  choicest <- c(3, sample(c(1,2,4), 2, replace=FALSE))
  numer <- matrix(0, 4, 1)
  for (j in choicest){
    if (j == 3){
      numer[j] <- exp(xchoice[i, j] )
    }
    else {
      numer[j] <- exp(xchoice[i, j] - z[i] )
    }
  }
  p <- numer / sum(numer)
  y[i,] <- rmultinom(1, 1, p)
  y[i,-choicest] <- -999
}

post5 <- MCMCmnl(y~choicevar(xa, "x", "a") +
  choicevar(xb, "x", "b") +
  choicevar(xc, "x", "c") +
  choicevar(xd, "x", "d") + z,
  baseline="c", verbose=500,
  mcmc=100000, thin=10, tune=.85)

plot(post5)
summary(post5)

## End(Not run)

```

Description

This function generates a sample from the posterior distribution of an ordered probit regression model using the data augmentation approach of Cowles (1996). The user supplies data and priors, and a sample from the posterior distribution is returned as an mcmc object, which can be subsequently analyzed with functions provided in the coda package.

Usage

```
MCMCoprobit(formula, data = parent.frame(), burnin = 1000, mcmc = 10000,
  thin=1, tune = NA, verbose = 0, seed = NA, beta.start = NA,
  b0 = 0, B0 = 0, ...)
```

Arguments

formula	Model formula.
data	Data frame.
burnin	The number of burn-in iterations for the sampler.
mcmc	The number of MCMC iterations for the sampler.
thin	The thinning interval used in the simulation. The number of Gibbs iterations must be divisible by this value.
tune	The tuning parameter for the Metropolis-Hastings step. Default of NA corresponds to a choice of 0.05 divided by the number of categories in the response variable.
verbose	A switch which determines whether or not the progress of the sampler is printed to the screen. If <code>verbose</code> is greater than 0 the iteration number, the beta vector, and the Metropolis-Hastings acceptance rate are printed to the screen every <code>verbose</code> th iteration.
seed	The seed for the random number generator. If NA, the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first element of the list is the L'Ecuyer seed, which is a vector of length six or NA (if NA a default seed of <code>rep(12345, 6)</code> is used). The second element of list is a positive substream number. See the MCMCpack specification for more details.
beta.start	The starting value for the β vector. This can either be a scalar or a column vector with dimension equal to the number of betas. If this takes a scalar value, then that value will serve as the starting value for all of the betas. The default value of NA will use rescaled estimates from an ordered logit model.
b0	The prior mean of β . This can either be a scalar or a column vector with dimension equal to the number of betas. If this takes a scalar value, then that value will serve as the prior mean for all of the betas.
B0	The prior precision of β . This can either be a scalar or a square matrix with dimensions equal to the number of betas. If this takes a scalar value, then that value times an identity matrix serves as the prior precision of β . Default value of 0 is equivalent to an improper uniform prior on β .
...	further arguments to be passed

Details

MCMCoprobit simulates from the posterior distribution of an ordered probit regression model using data augmentation. The simulation proper is done in compiled C++ code to maximize efficiency. Please consult the coda documentation for a comprehensive list of functions that can be used to analyze the posterior sample.

The observed variable y_i is ordinal with a total of C categories, with distribution governed by a latent variable:

$$z_i = x_i' \beta + \varepsilon_i$$

The errors are assumed to be from a standard Normal distribution. The probabilities of observing each outcome is governed by this latent variable and $C - 1$ estimable cutpoints, which are denoted γ_c . The probability that individual i is in category c is computed by:

$$\pi_{ic} = \Phi(\gamma_c - x'_i\beta) - \Phi(\gamma_{c-1} - x'_i\beta)$$

These probabilities are used to form the multinomial distribution that defines the likelihoods.

The algorithm employed is discussed in depth by Cowles (1996). Note that the model does include a constant in the data matrix. Thus, the first element γ_1 is normalized to zero, and is not returned in the mcmc object.

Value

An mcmc object that contains the posterior sample. This object can be summarized by functions provided by the coda package.

References

- M. K. Cowles. 1996. "Accelerating Monte Carlo Markov Chain Convergence for Cumulative-link Generalized Linear Models." *Statistics and Computing*. 6: 101-110.
- Valen E. Johnson and James H. Albert. 1999. "Ordinal Data Modeling." Springer: New York.
- Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.
- Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. 2002. *Output Analysis and Diagnostics for MCMC (CODA)*. <http://www-fis.iarc.fr/coda/>

See Also

[plot.mcmc,summary.mcmc](#)

Examples

```
## Not run:
x1 <- rnorm(100); x2 <- rnorm(100);
z <- 1.0 + x1*0.1 - x2*0.5 + rnorm(100);
y <- z; y[z < 0] <- 0; y[z >= 0 & z < 1] <- 1;
y[z >= 1 & z < 1.5] <- 2; y[z >= 1.5] <- 3;
posterior <- MCMCoprobit(y ~ x1 + x2, tune=0.3, mcmc=20000)
plot(posterior)
summary(posterior)

## End(Not run)
```

MCMCordfactanal

Markov Chain Monte Carlo for Ordinal Data Factor Analysis Model

Description

This function generates a sample from the posterior distribution of an ordinal data factor analysis model. Normal priors are assumed on the factor loadings and factor scores while improper uniform priors are assumed on the cutpoints. The user supplies data and parameters for the prior distributions, and a sample from the posterior distribution is returned as an mcmc object, which can be subsequently analyzed with functions provided in the coda package.

Usage

```
MCMCordfactanal(x, factors, lambda.constraints=list(),
                data=parent.frame(), burnin = 1000, mcmc = 20000,
                thin=1, tune=NA, verbose = 0, seed = NA,
                lambda.start = NA, l0=0, L0=0,
                store.lambda=TRUE, store.scores=FALSE,
                drop.constantvars=TRUE, ... )
```

Arguments

- | | |
|--------------------|--|
| x | Either a formula or a numeric matrix containing the manifest variables. |
| factors | The number of factors to be fitted. |
| lambda.constraints | List of lists specifying possible equality or simple inequality constraints on the factor loadings. A typical entry in the list has one of three forms: <code>varname=list(d, c)</code> which will constrain the <i>d</i> th loading for the variable named <code>varname</code> to be equal to <code>c</code> , <code>varname=list(d, "+")</code> which will constrain the <i>d</i> th loading for the variable named <code>varname</code> to be positive, and <code>varname=list(d, "-")</code> which will constrain the <i>d</i> th loading for the variable named <code>varname</code> to be negative. If <code>x</code> is a matrix without column names defaults names of "V1", "V2", ... , etc will be used. Note that, unlike <code>MCMCfactanal</code> , the Λ matrix used here has <code>factors+1</code> columns. The first column of Λ corresponds to negative item difficulty parameters and should generally not be constrained. |
| data | A data frame. |
| burnin | The number of burn-in iterations for the sampler. |
| mcmc | The number of iterations for the sampler. |
| thin | The thinning interval used in the simulation. The number of iterations must be divisible by this value. |
| tune | The tuning parameter for the Metropolis-Hastings sampling. Can be either a scalar or a <i>k</i> -vector. Must be strictly positive. |
| verbose | A switch which determines whether or not the progress of the sampler is printed to the screen. If <code>verbose</code> is greater than 0 the iteration number and the Metropolis-Hastings acceptance rate are printed to the screen every <code>verbose</code> th iteration. |
| seed | The seed for the random number generator. If NA, the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first element of the list is the L'Ecuyer seed, which is a vector of length six or NA (if NA a default seed of <code>rep(12345, 6)</code> is used). The second element of list is a positive substream number. See the MCMCpack specification for more details. |
| lambda.start | Starting values for the factor loading matrix Lambda. If <code>lambda.start</code> is set to a scalar the starting value for all unconstrained loadings will be set to that scalar. If <code>lambda.start</code> is a matrix of the same dimensions as Lambda then the <code>lambda.start</code> matrix is used as the starting values (except for equality-constrained elements). If <code>lambda.start</code> is set to NA (the default) then starting values for unconstrained elements in the first column of Lambda are based on the observed response pattern, the remaining unconstrained elements of Lambda are set to 0, and starting values for inequality constrained elements are set to either 1.0 or -1.0 depending on the nature of the constraints. |

l0	The means of the independent Normal prior on the factor loadings. Can be either a scalar or a matrix with the same dimensions as <code>Lambda</code> .
L0	The precisions (inverse variances) of the independent Normal prior on the factor loadings. Can be either a scalar or a matrix with the same dimensions as <code>Lambda</code> .
<code>store.lambda</code>	A switch that determines whether or not to store the factor loadings for posterior analysis. By default, the factor loadings are all stored.
<code>store.scores</code>	A switch that determines whether or not to store the factor scores for posterior analysis. <i>NOTE: This takes an enormous amount of memory, so should only be used if the chain is thinned heavily, or for applications with a small number of observations.</i> By default, the factor scores are not stored.
<code>drop.constantvars</code>	A switch that determines whether or not manifest variables that have no variation should be deleted before fitting the model. Default = TRUE.
...	further arguments to be passed

Details

The model takes the following form:

Let $i = 1, \dots, N$ index observations and $j = 1, \dots, K$ index response variables within an observation. The typical observed variable x_{ij} is ordinal with a total of C_j categories. The distribution of X is governed by a $N \times K$ matrix of latent variables X^* and a series of cutpoints γ . X^* is assumed to be generated according to:

$$x_i^* = \Lambda \phi_i + \epsilon_i$$

$$\epsilon_i \sim \mathcal{N}(0, I)$$

where x_i^* is the k -vector of latent variables specific to observation i , Λ is the $k \times d$ matrix of factor loadings, and ϕ_i is the d -vector of latent factor scores. It is assumed that the first element of ϕ_i is equal to 1 for all i .

The probability that the j th variable in observation i takes the value c is:

$$\pi_{ijc} = \Phi(\gamma_{jc} - \Lambda'_j \phi_i) - \Phi(\gamma_{j(c-1)} - \Lambda'_j \phi_i)$$

The implementation used here assumes independent conjugate priors for each element of Λ and each ϕ_i . More specifically we assume:

$$\Lambda_{ij} \sim \mathcal{N}(l_{0_{ij}}, L_{0_{ij}}^{-1}), i = 1, \dots, k, j = 1, \dots, d$$

$$\phi_{i(2:d)} \sim \mathcal{N}(0, I), i = 1, \dots, n$$

The standard two-parameter item response theory model with probit link is a special case of the model sketched above.

`MCMCordfactanal` simulates from the posterior distribution using a Metropolis-Hastings within Gibbs sampling algorithm. The algorithm employed is based on work by Cowles (1996). Note that the first element of ϕ_i is a 1. As a result, the first column of Λ can be interpreted as item difficulty parameters. Further, the first element γ_1 is normalized to zero, and thus not returned in the `mcmc` object. The simulation proper is done in compiled C++ code to maximize efficiency.

Please consult the coda documentation for a comprehensive list of functions that can be used to analyze the posterior sample.

As is the case with all measurement models, make sure that you have plenty of free memory, especially when storing the scores.

Value

An mcmc object that contains the posterior sample. This object can be summarized by functions provided by the coda package.

References

Shawn Treier and Simon Jackman. 2003. "Democracy as a Latent Variable." Paper presented at the Midwest Political Science Association Annual Meeting.

M. K. Cowles. 1996. "Accelerating Monte Carlo Markov Chain Convergence for Cumulative-link Generalized Linear Models." *Statistics and Computing*. 6: 101-110.

Valen E. Johnson and James H. Albert. 1999. "Ordinal Data Modeling." Springer: New York.

Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.

Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. 2002. *Output Analysis and Diagnostics for MCMC (CODA)*. <http://www-fis.iarc.fr/coda/>.

See Also

[plot.mcmc](#), [summary.mcmc](#), [factanal](#), [MCMCfactanal](#), [MCMCirt1d](#), [MCMCirtKd](#)

Examples

```
## Not run:
data(painters)
new.painters <- painters[,1:4]
cuts <- apply(new.painters, 2, quantile, c(.25, .50, .75))
for (i in 1:4){
  new.painters[new.painters[,i]<cuts[1,i],i] <- 100
  new.painters[new.painters[,i]<cuts[2,i],i] <- 200
  new.painters[new.painters[,i]<cuts[3,i],i] <- 300
  new.painters[new.painters[,i]<100,i] <- 400
}

posterior <- MCMCordfactanal(~Composition+Drawing+Colour+Expression,
                             data=new.painters, factors=1,
                             lambda.constraints=list(Drawing=list(2, "+")),
                             burnin=5000, mcmc=500000, thin=200, verbose=500,
                             L0=0.5, store.lambda=TRUE,
                             store.scores=TRUE, tune=1.2)

plot(posterior)
summary(posterior)
## End(Not run)
```

Description

MCMCpanel generates a sample from the posterior distribution of a General Linear Panel Model using Algorithm 2 of Chib and Carlin (1999). This model uses a multivariate Normal prior for the fixed effects parameters, a Wishart prior on the random effects precision matrix, and a Gamma prior on the conditional error precision. The user supplies data and priors, and a sample from the posterior distribution is returned as an mcmc object, which can be subsequently analyzed with functions provided in the coda package.

Usage

```
MCMCpanel(obs, Y, X, W, burnin = 1000, mcmc = 10000, thin = 5,
           verbose = 0, seed = NA, sigma2.start = NA,
           D.start = NA, b0 = 0, B0 = 1, eta0, R0, nu0 = 0.001,
           delta0 = 0.001, ...)
```

Arguments

obs	An ($nk \times 1$) vector that contains unique observation numbers for each subject.
Y	An ($nk \times 1$) vector of response variables, stacked across all subjects.
X	An ($nk \times p$) matrix of fixed effects covariates, stacked across all subjects.
W	An ($nk \times q$) matrix of random effects covariates, stacked across all subjects.
burnin	The number of burnin iterations for the sampler.
mcmc	The number of Gibbs iterations for the sampler.
thin	The thinning interval used in the simulation. The number of mcmc iterations must be divisible by this value.
seed	The seed for the random number generator. If NA, the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first element of the list is the L'Ecuyer seed, which is a vector of length six or NA (if NA a default seed of <code>rep(12345, 6)</code> is used). The second element of list is a positive substream number. See the MCMCpack specification for more details.
verbose	A switch which determines whether or not the progress of the sampler is printed to the screen. If <code>verbose</code> is greater than 0 the iteration number and parameters are printed to the screen every <code>verboseth</code> iteration.
sigma2.start	The starting value for the conditional error variance. Default value of NA uses the least squares estimates.
D.start	The starting value for precision matrix of the random effects. This can either be a scalar or square matrix with dimension equal to the number of random effects. If this takes a scalar value, then that value multiplied by an identity matrix will be the starting value. Default value of NA uses an identity matrix multiplied by 0.5 the OLS σ^2 estimate.

b0	The prior mean of β . This can either be a scalar or a column vector with dimension equal to the number of betas. If this takes a scalar value, then that value will serve as the prior mean for all of the betas.
B0	The prior precision of β . This can either be a scalar or a square matrix with dimensions equal to the number of betas. If this takes a scalar value, then that value times an identity matrix serves as the prior precision of beta. Default value of 0 is equivalent to an improper uniform prior for beta.
eta0	The shape parameter for the Wishart prior on precision matrix for the random effects.
R0	The scale matrix for the Wishart prior on precision matrix for the random effects.
nu0	The shape parameter for the Gamma prior on the conditional error precision.
delta0	The scale parameter for the Gamma prior on the conditional error precision.
...	further arguments to be passed

Details

MCMCpanel simulates from the posterior distribution sample using the blocked Gibbs sampler of Chib and Carlin (1999), Algorithm 2. The simulation proper is done in compiled C++ code to maximize efficiency. Please consult the coda documentation for a comprehensive list of functions that can be used to analyze the posterior sample.

The model takes the following form:

$$y_i = X_i\beta + W_i b_i + \varepsilon_i$$

Where the random effects:

$$b_i \sim \mathcal{N}_q(0, D)$$

And the errors:

$$\varepsilon_i \sim \mathcal{N}(0, \sigma^2 I_k)$$

We assume standard, conjugate priors:

$$\beta \sim \mathcal{N}(b_0, B_0^{-1})$$

And:

$$\sigma^{-2} \sim \mathcal{Gamma}(\nu_0/2, \delta_0/2)$$

And:

$$D^{-1} \sim \mathcal{Wishart}(\eta_0, R_0^{-1})$$

See Chib and Carlin (1999) or Martin and Saunders (2002) for more details.

NOTE: Unlike most models in MCMCpack, we do not provide default parameters for the priors on the precision matrix for the random effects. When fitting one of these models, it is of utmost importance to choose a prior that reflects your prior beliefs about the random effects. Using the `dwish` and `rwish` functions might be useful in choosing these values. Also, the user is not allowed to specify a starting value for the β parameters, as they are simulated in the first block of the sampler.

Value

An `mcmc` object that contains the posterior sample. This object can be summarized by functions provided by the coda package.

References

- Siddhartha Chib and Bradley P. Carlin. 1999. "On MCMC Sampling in Hierarchical Longitudinal Models." *Statistics and Computing*. 9: 17-26.
- Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.
- Andrew D. Martin and Kyle L. Saunders. 2002. "Bayesian Inference for Political Science Panel Data." Paper presented at the 2002 Annual Meeting of the American Political Science Association.
- Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. 2002. *Output Analysis and Diagnostics for MCMC (CODA)*. <http://www-fis.iarc.fr/coda/>.

See Also

[plot.mcmc,summary.mcmc](#)

MCMCpoisson

Markov Chain Monte Carlo for Poisson Regression

Description

This function generates a sample from the posterior distribution of a Poisson regression model using a random walk Metropolis algorithm. The user supplies data and priors, and a sample from the posterior distribution is returned as an mcmc object, which can be subsequently analyzed with functions provided in the coda package.

Usage

```
MCMCpoisson(formula, data = parent.frame(), burnin = 1000, mcmc = 10000,
             thin = 1, tune = 1.1, verbose = 0, seed = NA, beta.start = NA,
             b0 = 0, B0 = 0, marginal.likelihood = c("none", "Laplace"), ...)
```

Arguments

formula	Model formula.
data	Data frame.
burnin	The number of burn-in iterations for the sampler.
mcmc	The number of Metropolis iterations for the sampler.
thin	The thinning interval used in the simulation. The number of mcmc iterations must be divisible by this value.
tune	Metropolis tuning parameter. Can be either a positive scalar or a k -vector, where k is the length of β . Make sure that the acceptance rate is satisfactory (typically between 0.20 and 0.5) before using the posterior sample for inference.
verbose	A switch which determines whether or not the progress of the sampler is printed to the screen. If <code>verbose</code> is greater than 0 the iteration number, the current beta vector, and the Metropolis acceptance rate are printed to the screen every <code>verbose</code> th iteration.

<code>seed</code>	The seed for the random number generator. If NA, the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first element of the list is the L'Ecuyer seed, which is a vector of length six or NA (if NA a default seed of <code>rep(12345, 6)</code> is used). The second element of list is a positive substream number. See the MCMCpack specification for more details.
<code>beta.start</code>	The starting value for the β vector. This can either be a scalar or a column vector with dimension equal to the number of betas. If this takes a scalar value, then that value will serve as the starting value for all of the betas. The default value of NA will use the maximum likelihood estimate of β as the starting value.
<code>b0</code>	The prior mean of β . This can either be a scalar or a column vector with dimension equal to the number of betas. If this takes a scalar value, then that value will serve as the prior mean for all of the betas.
<code>B0</code>	The prior precision of β . This can either be a scalar or a square matrix with dimensions equal to the number of betas. If this takes a scalar value, then that value times an identity matrix serves as the prior precision of β . Default value of 0 is equivalent to an improper uniform prior for beta.
<code>marginal.likelihood</code>	How should the marginal likelihood be calculated? Options are: <code>none</code> in which case the marginal likelihood will not be calculated or <code>Laplace</code> in which case the Laplace approximation (see Kass and Raftery, 1995) is used.
<code>...</code>	further arguments to be passed

Details

MCMCpoisson simulates from the posterior distribution of a Poisson regression model using a random walk Metropolis algorithm. The simulation proper is done in compiled C++ code to maximize efficiency. Please consult the coda documentation for a comprehensive list of functions that can be used to analyze the posterior sample.

The model takes the following form:

$$y_i \sim \text{Poisson}(\mu_i)$$

Where the inverse link function:

$$\mu_i = \exp(x_i' \beta)$$

We assume a multivariate Normal prior on β :

$$\beta \sim \mathcal{N}(b_0, B_0^{-1})$$

The Metropolis proposal distribution is centered at the current value of θ and has variance-covariance $V = T(B_0 + C^{-1})^{-1}T$, where T is a the diagonal positive definite matrix formed from the `tune`, B_0 is the prior precision, and C is the large sample variance-covariance matrix of the MLEs. This last calculation is done via an initial call to `glm`.

Value

An `mcmc` object that contains the posterior sample. This object can be summarized by functions provided by the coda package.

References

Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.

Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. 2002. *Output Analysis and Diagnostics for MCMC (CODA)*. <http://www-fis.iarc.fr/coda/>.

See Also

`plot.mcmc`, `summary.mcmc`, `glm`

Examples

```
## Not run:
counts <- c(18,17,15,20,10,20,25,13,12)
outcome <- gl(3,1,9)
treatment <- gl(3,3)
posterior <- MCMCpoisson(counts ~ outcome + treatment)
plot(posterior)
summary(posterior)

## End(Not run)
```

MCMCprobit

Markov Chain Monte Carlo for Probit Regression

Description

This function generates a sample from the posterior distribution of a probit regression model using the data augmentation approach of Albert and Chib (1993). The user supplies data and priors, and a sample from the posterior distribution is returned as an `mcmc` object, which can be subsequently analyzed with functions provided in the `coda` package.

Usage

```
MCMCprobit(formula, data = parent.frame(), burnin = 1000, mcmc = 10000,
  thin = 1, verbose = 0, seed = NA, beta.start = NA,
  b0 = 0, B0 = 0, bayes.resid = FALSE,
  marginal.likelihood=c("none", "Laplace"), ...)
```

Arguments

<code>formula</code>	Model formula.
<code>data</code>	Data frame.
<code>burnin</code>	The number of burn-in iterations for the sampler.
<code>mcmc</code>	The number of Gibbs iterations for the sampler.
<code>thin</code>	The thinning interval used in the simulation. The number of Gibbs iterations must be divisible by this value.
<code>verbose</code>	A switch which determines whether or not the progress of the sampler is printed to the screen. If <code>verbose</code> is greater than 0 the iteration number and the betas are printed to the screen every <code>verboseth</code> iteration.

<code>seed</code>	The seed for the random number generator. If NA, the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first element of the list is the L'Ecuyer seed, which is a vector of length six or NA (if NA a default seed of <code>rep(12345, 6)</code> is used). The second element of list is a positive substream number. See the MCMCpack specification for more details.
<code>beta.start</code>	The starting value for the β vector. This can either be a scalar or a column vector with dimension equal to the number of betas. If this takes a scalar value, then that value will serve as the starting value for all of the betas. The default value of NA will use the maximum likelihood estimate of β as the starting value.
<code>b0</code>	The prior mean of β . This can either be a scalar or a column vector with dimension equal to the number of betas. If this takes a scalar value, then that value will serve as the prior mean for all of the betas.
<code>B0</code>	The prior precision of β . This can either be a scalar or a square matrix with dimensions equal to the number of betas. If this takes a scalar value, then that value times an identity matrix serves as the prior precision of β . Default value of 0 is equivalent to an improper uniform prior on β .
<code>bayes.resid</code>	Should latent Bayesian residuals (Albert and Chib, 1995) be returned? Default is FALSE meaning no residuals should be returned. Alternatively, the user can specify an array of integers giving the observation numbers for which latent residuals should be calculated and returned. TRUE will return draws of latent residuals for all observations.
<code>marginal.likelihood</code>	How should the marginal likelihood be calculated? Options are: <code>none</code> in which case the marginal likelihood will not be calculated or <code>Laplace</code> in which case the Laplace approximation (see Kass and Raftery, 1995) is used.
<code>...</code>	further arguments to be passed

Details

MCMCprobit simulates from the posterior distribution of a probit regression model using data augmentation. The simulation proper is done in compiled C++ code to maximize efficiency. Please consult the coda documentation for a comprehensive list of functions that can be used to analyze the posterior sample.

The model takes the following form:

$$y_i \sim \text{Bernoulli}(\pi_i)$$

Where the inverse link function:

$$\pi_i = \Phi(x_i' \beta)$$

We assume a multivariate Normal prior on β :

$$\beta \sim \mathcal{N}(b_0, B_0^{-1})$$

See Albert and Chib (1993) for estimation details.

Value

An `mcmc` object that contains the posterior sample. This object can be summarized by functions provided by the coda package.

References

- Albert, J. H. and S. Chib. 1993. "Bayesian Analysis of Binary and Polychotomous Response Data." *J. Amer. Statist. Assoc.* 88, 669-679
- Albert, J. H. and S. Chib. 1995. "Bayesian Residual Analysis for Binary Response Regression Models." *Biometrika*. 82, 747-759.
- Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.
- Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. 2002. *Output Analysis and Diagnostics for MCMC (CODA)*. <http://www-fis.iarc.fr/coda/>.

See Also

`plot.mcmc`, `summary.mcmc`, `glm`

Examples

```
## Not run:
data(birthwt)
posterior <- MCMCprobit(low~age+as.factor(race)+smoke, data=birthwt)
plot(posterior)
summary(posterior)

## End(Not run)
```

MCMCregress

Markov Chain Monte Carlo for Gaussian Linear Regression

Description

This function generates a sample from the posterior distribution of a linear regression model with Gaussian errors using Gibbs sampling (with a multivariate Gaussian prior on the beta vector, and an inverse Gamma prior on the conditional error variance). The user supplies data and priors, and a sample from the posterior distribution is returned as an `mcmc` object, which can be subsequently analyzed with functions provided in the `coda` package.

Usage

```
MCMCregress(formula, data = parent.frame(), burnin = 1000, mcmc = 10000,
  thin = 1, verbose = 0, seed = NA, beta.start = NA,
  b0 = 0, B0 = 0, c0 = 0.001, d0 = 0.001,
  marginal.likelihood = c("none", "Laplace", "Chib95"), ...)
```

Arguments

<code>formula</code>	Model formula.
<code>data</code>	Data frame.
<code>burnin</code>	The number of burn-in iterations for the sampler.
<code>mcmc</code>	The number of MCMC iterations after burnin.
<code>thin</code>	The thinning interval used in the simulation. The number of MCMC iterations must be divisible by this value.

<code>verbose</code>	A switch which determines whether or not the progress of the sampler is printed to the screen. If <code>verbose</code> is greater than 0 the iteration number, the β vector, and the error variance are printed to the screen every <code>verboseth</code> iteration.
<code>seed</code>	The seed for the random number generator. If NA, the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first element of the list is the L'Ecuyer seed, which is a vector of length six or NA (if NA a default seed of <code>rep(12345, 6)</code> is used). The second element of list is a positive substream number. See the MCMCpack specification for more details.
<code>beta.start</code>	The starting values for the β vector. This can either be a scalar or a column vector with dimension equal to the number of betas. The default value of NA will use the OLS estimate of β as the starting value. If this is a scalar, that value will serve as the starting value mean for all of the betas.
<code>b0</code>	The prior mean of β . This can either be a scalar or a column vector with dimension equal to the number of betas. If this takes a scalar value, then that value will serve as the prior mean for all of the betas.
<code>B0</code>	The prior precision of β . This can either be a scalar or a square matrix with dimensions equal to the number of betas. If this takes a scalar value, then that value times an identity matrix serves as the prior precision of beta. Default value of 0 is equivalent to an improper uniform prior for beta.
<code>c0</code>	$c_0/2$ is the shape parameter for the inverse Gamma prior on σ^2 (the variance of the disturbances). The amount of information in the inverse Gamma prior is something like that from c_0 pseudo-observations.
<code>d0</code>	$d_0/2$ is the scale parameter for the inverse Gamma prior on σ^2 (the variance of the disturbances). In constructing the inverse Gamma prior, d_0 acts like the sum of squared errors from the c_0 pseudo-observations.
<code>marginal.likelihood</code>	How should the marginal likelihood be calculated? Options are: <code>none</code> in which case the marginal likelihood will not be calculated, <code>Laplace</code> in which case the Laplace approximation (see Kass and Raftery, 1995) is used, and <code>Chib95</code> in which case the method of Chib (1995) is used.
<code>...</code>	further arguments to be passed

Details

MCMCregress simulates from the posterior distribution using standard Gibbs sampling (a multivariate Normal draw for the betas, and an inverse Gamma draw for the conditional error variance). The simulation proper is done in compiled C++ code to maximize efficiency. Please consult the coda documentation for a comprehensive list of functions that can be used to analyze the posterior sample.

The model takes the following form:

$$y_i = x_i' \beta + \varepsilon_i$$

Where the errors are assumed to be Gaussian:

$$\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$$

We assume standard, semi-conjugate priors:

$$\beta \sim \mathcal{N}(b_0, B_0^{-1})$$

And:

$$\sigma^{-2} \sim \text{Gamma}(c_0/2, d_0/2)$$

Where β and σ^{-2} are assumed *a priori* independent. Note that only starting values for β are allowed because simulation is done using Gibbs sampling with the conditional error variance as the first block in the sampler.

Value

An mcmc object that contains the posterior sample. This object can be summarized by functions provided by the coda package.

References

Siddhartha Chib. 1995. "Marginal Likelihood from the Gibbs Output." *Journal of the American Statistical Association*. 90: 1313-1321.

Robert E. Kass and Adrian E. Raftery. 1995. "Bayes Factors." *Journal of the American Statistical Association*. 90: 773-795.

Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.

Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. 2002. *Output Analysis and Diagnostics for MCMC (CODA)*. <http://www-fis.iarc.fr/coda/>.

See Also

[plot.mcmc](#), [summary.mcmc](#), [lm](#)

Examples

```
## Not run:
line <- list(X = c(-2,-1,0,1,2), Y = c(1,3,3,3,5))
posterior <- MCMCregress(Y~X, data=line, verbose=1000)
plot(posterior)
raftery.diag(posterior)
summary(posterior)
## End(Not run)
```

MCMCtobit

Markov Chain Monte Carlo for Gaussian Linear Regression with a Censored Dependent Variable

Description

This function generates a sample from the posterior distribution of a linear regression model with Gaussian errors using Gibbs sampling (with a multivariate Gaussian prior on the beta vector, and an inverse Gamma prior on the conditional error variance). The dependent variable may be censored from below, from above, or both. The user supplies data and priors, and a sample from the posterior distribution is returned as an mcmc object, which can be subsequently analyzed with functions provided in the coda package.

Usage

```
MCMCtobit(formula, data = parent.frame(), below = 0, above = Inf,
  burnin = 1000, mcmc = 10000, thin = 1, verbose = 0, seed = NA,
  beta.start = NA, b0 = 0, B0 = 0, c0 = 0.001, d0 = 0.001, ...)
```

Arguments

formula	A model formula.
data	A dataframe.
below	The point at which the dependent variable is censored from below. The default is zero. To censor from above only, specify that below = -Inf.
above	The point at which the dependent variable is censored from above. To censor from below only, use the default value of Inf.
burnin	The number of burn-in iterations for the sampler.
mcmc	The number of MCMC iterations after burnin.
thin	The thinning interval used in the simulation. The number of MCMC iterations must be divisible by this value.
verbose	A switch which determines whether or not the progress of the sampler is printed to the screen. If verbose is greater than 0 the iteration number, the β vector, and the error variance is printed to the screen every <code>verboseth</code> iteration.
seed	The seed for the random number generator. If NA, the Mersenne Twister generator is used with default seed 12345; if an integer is passed it is used to seed the Mersenne twister. The user can also pass a list of length two to use the L'Ecuyer random number generator, which is suitable for parallel computation. The first element of the list is the L'Ecuyer seed, which is a vector of length six or NA (if NA a default seed of <code>rep(12345, 6)</code> is used). The second element of list is a positive substream number. See the MCMCpack specification for more details.
beta.start	The starting values for the β vector. This can either be a scalar or a column vector with dimension equal to the number of betas. The default value of NA will use the OLS estimate of β as the starting value. If this is a scalar, that value will serve as the starting value mean for all of the betas.
b0	The prior mean of β . This can either be a scalar or a column vector with dimension equal to the number of betas. If this takes a scalar value, then that value will serve as the prior mean for all of the betas.
B0	The prior precision of β . This can either be a scalar or a square matrix with dimensions equal to the number of betas. If this takes a scalar value, then that value times an identity matrix serves as the prior precision of beta. Default value of 0 is equivalent to an improper uniform prior for beta.
c0	$c_0/2$ is the shape parameter for the inverse Gamma prior on σ^2 (the variance of the disturbances). The amount of information in the inverse Gamma prior is something like that from c_0 pseudo-observations.
d0	$d_0/2$ is the scale parameter for the inverse Gamma prior on σ^2 (the variance of the disturbances). In constructing the inverse Gamma prior, d_0 acts like the sum of squared errors from the c_0 pseudo-observations.
...	further arguments to be passed

Details

MCMCtobit simulates from the posterior distribution using standard Gibbs sampling (a multivariate Normal draw for the betas, and an inverse Gamma draw for the conditional error variance). MCMCtobit differs from MCMCregress in that the dependent variable may be censored from below, from above, or both. The simulation proper is done in compiled C++ code to maximize efficiency. Please consult the coda documentation for a comprehensive list of functions that can be used to analyze the posterior sample.

The model takes the following form:

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

where the errors are assumed to be Gaussian:

$$\varepsilon_i \sim \mathcal{N}(0, \sigma^2).$$

Let c_1 and c_2 be the two censoring points, and let y_i^* be the partially observed dependent variable. Then,

$$y_i = y_i^* \quad \text{if } c_1 < y_i^* < c_2,$$

$$y_i = c_1 \quad \text{if } c_1 \geq y_i^*,$$

$$y_i = c_2 \quad \text{if } c_2 \leq y_i^*.$$

We assume standard, semi-conjugate priors:

$$\beta \sim \mathcal{N}(b_0, B_0^{-1}),$$

and:

$$\sigma^{-2} \sim \mathcal{Gamma}(c_0/2, d_0/2),$$

where β and σ^{-2} are assumed *a priori* independent. Note that only starting values for β are allowed because simulation is done using Gibbs sampling with the conditional error variance as the first block in the sampler.

Value

An mcmc object that contains the posterior sample. This object can be summarized by functions provided by the coda package.

Author(s)

Ben Goodrich

References

Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.

Martyn Plummer, Nicky Best, Kate Cowles, and Karen Vines. 2002. *Output Analysis and Diagnostics for MCMC (CODA)*. <http://www-fis.iarc.fr/coda/>.

James Tobin. 1958. "Estimation of relationships for limited dependent variables." *Econometrica.*, 26:24-36.

See Also

[plot.mcmc](#), [summary.mcmc](#), [survreg](#), [MCMCregress](#)

Examples

```
## Not run:
library(survival)
example(tobin)
summary(tfit)
tfit.mcmc <- MCMCtobit(durable ~ age + quant, data=tobin, mcmc=30000,
                      verbose=1000)

plot(tfit.mcmc)
raftery.diag(tfit.mcmc)
summary(tfit.mcmc)
## End(Not run)
```

MCbinomialbeta

*Monte Carlo Simulation from a Binomial Likelihood with a Beta Prior***Description**

This function generates a sample from the posterior distribution of a binomial likelihood with a Beta prior.

Usage

```
MCbinomialbeta(y, n, alpha=1, beta=1, mc=1000, ...)
```

Arguments

<code>y</code>	The number of successes in the independent Bernoulli trials.
<code>n</code>	The number of independent Bernoulli trials.
<code>alpha</code>	Beta prior distribution alpha parameter.
<code>beta</code>	Beta prior distribution beta parameter.
<code>mc</code>	The number of Monte Carlo draws to make.
<code>...</code>	further arguments to be passed

Details

MCbinomialbeta directly simulates from the posterior distribution. This model is designed primarily for instructional use. π is the probability of success for each independent Bernoulli trial. We assume a conjugate Beta prior:

$$\pi \sim \text{Beta}(\alpha, \beta)$$

y is the number of successes in n trials. By default, a uniform prior is used.

Value

An mcmc object that contains the posterior sample. This object can be summarized by functions provided by the coda package.

See Also

[plot.mcmc](#), [summary.mcmc](#)

Examples

```
## Not run:
posterior <- MChinomialbeta(3,12,mc=5000)
summary(posterior)
plot(posterior)
grid <- seq(0,1,0.01)
plot(grid, dbeta(grid, 1, 1), type="l", col="red", lwd=3, ylim=c(0,3.6),
      xlab="pi", ylab="density")
lines(density(posterior), col="blue", lwd=3)
legend(.75, 3.6, c("prior", "posterior"), lwd=3, col=c("red", "blue"))
## End(Not run)
```

MCmultinomdirichlet

Monte Carlo Simulation from a Multinomial Likelihood with a Dirichlet Prior

Description

This function generates a sample from the posterior distribution of a multinomial likelihood with a Dirichlet prior.

Usage

```
MCmultinomdirichlet(y, alpha0, mc=1000, ...)
```

Arguments

<code>y</code>	A vector of data (number of successes for each category).
<code>alpha0</code>	The vector of parameters of the Dirichlet prior.
<code>mc</code>	The number of Monte Carlo draws to make.
<code>...</code>	further arguments to be passed

Details

MCmultinomdirichlet directly simulates from the posterior distribution. This model is designed primarily for instructional use. π is the parameter of interest of the multinomial distribution. It is of dimension $(d \times 1)$. We assume a conjugate Dirichlet prior:

$$\pi \sim \text{Dirichlet}(\alpha_0)$$

y is a $(d \times 1)$ vector of observed data.

Value

An mcmc object that contains the posterior sample. This object can be summarized by functions provided by the coda package.

See Also

[plot.mcmc](#), [summary.mcmc](#)

Examples

```
## Not run:
## Example from Gelman, et. al. (1995, p. 78)
posterior <- MCMultinomDirichlet(c(727,583,137), c(1,1,1), mc=10000)
bush.dukakis.diff <- posterior[,1] - posterior[,2]
cat("Pr(Bush > Dukakis): ",
    sum(bush.dukakis.diff > 0) / length(bush.dukakis.diff), "\n")
hist(bush.dukakis.diff)
## End(Not run)
```

MCnormalnormal	<i>Monte Carlo Simulation from a Normal Likelihood (with known variance) with a Normal Prior</i>
----------------	--

Description

This function generates a sample from the posterior distribution of a Normal likelihood (with known variance) with a Normal prior.

Usage

```
MCnormalnormal(y, sigma2, mu0, tau20, mc=1000, ...)
```

Arguments

<code>y</code>	The data.
<code>sigma2</code>	The known variance of <code>y</code> .
<code>mu0</code>	The prior mean of <code>mu</code> .
<code>tau20</code>	The prior variance of <code>mu</code> .
<code>mc</code>	The number of Monte Carlo draws to make.
<code>...</code>	further arguments to be passed

Details

MCnormalnormal directly simulates from the posterior distribution. This model is designed primarily for instructional use. μ is the parameter of interest of the Normal distribution. We assume a conjugate normal prior:

$$\mu \sim \mathcal{N}(\mu_0, \tau_0^2)$$

`y` is a vector of observed data.

Value

An mcmc object that contains the posterior sample. This object can be summarized by functions provided by the coda package.

See Also

[plot.mcmc](#), [summary.mcmc](#)

Examples

```
## Not run:
y <- c(2.65, 1.80, 2.29, 2.11, 2.27, 2.61, 2.49, 0.96, 1.72, 2.40)
posterior <- MCMCpack::MCnormalnormal(y, 1, 0, 1, 5000)
summary(posterior)
plot(posterior)
grid <- seq(-3, 3, 0.01)
plot(grid, dnorm(grid, 0, 1), type="l", col="red", lwd=3, ylim=c(0,1.4),
      xlab="mu", ylab="density")
lines(density(posterior), col="blue", lwd=3)
legend(-3, 1.4, c("prior", "posterior"), lwd=3, col=c("red", "blue"))
## End(Not run)
```

MCpoissongamma

Monte Carlo Simulation from a Poisson Likelihood with a Gamma Prior

Description

This function generates a sample from the posterior distribution of a Poisson likelihood with a Gamma prior.

Usage

```
MCpoissongamma(y, alpha, beta, mc=1000, ...)
```

Arguments

<code>y</code>	A vector of counts (must be non-negative).
<code>alpha</code>	Gamma prior distribution shape parameter.
<code>beta</code>	Gamma prior distribution scale parameter.
<code>mc</code>	The number of Monte Carlo draws to make.
<code>...</code>	further arguments to be passed

Details

MCpoissongamma directly simulates from the posterior distribution. This model is designed primarily for instructional use. λ is the parameter of interest of the Poisson distribution. We assume a conjugate Gamma prior:

$$\lambda \sim \text{Gamma}(\alpha, \beta)$$

y is a vector of counts.

Value

An mcmc object that contains the posterior sample. This object can be summarized by functions provided by the coda package.

See Also

[plot.mcmc](#), [summary.mcmc](#)

Examples

```
## Not run:
data(quine)
posterior <- MCPoissonGamma(quine$Days, 15, 1, 5000)
summary(posterior)
plot(posterior)
grid <- seq(14,18,0.01)
plot(grid, dgamma(grid, 15, 1), type="l", col="red", lwd=3, ylim=c(0,1.3),
      xlab="lambda", ylab="density")
lines(density(posterior), col="blue", lwd=3)
legend(17, 1.3, c("prior", "posterior"), lwd=3, col=c("red", "blue"))
## End(Not run)
```

Nethvote

*Dutch Voting Behavior in 1989***Description**

Dutch Voting Behavior in 1989.

Usage

```
data(Nethvote)
```

Format

A data frame with 1754 observations and 11 variables from the 1989 Dutch Parliamentary Election Study (Anker and Oppenheim, 1993). Each observation is a survey respondent. These data are a subset of one of five multiply imputed datasets used in Quinn and Martin (2002). For more information see Quinn and Martin (2002).

vote A factor giving the self-reported vote choice of each respondent. The levels are CDA (Christen Democratisch Appel), D66 (Democraten 66), PvdA (Partij van de Arbeid), and VVD (Volkspartij voor Vrijheid en Democratie).

distD66 A numeric variable giving the squared ideological distance between the respondent and the D66. Larger values indicate ideological dissimilarity between the respondent and the party.

distPvdA A numeric variable giving the squared ideological distance between the respondent and the PvdA. Larger values indicate ideological dissimilarity between the respondent and the party.

distVVD A numeric variable giving the squared ideological distance between the respondent and the VVD. Larger values indicate ideological dissimilarity between the respondent and the party.

distCDA A numeric variable giving the squared ideological distance between the respondent and the CDA. Larger values indicate ideological dissimilarity between the respondent and the party.

relig An indicator variable equal to 0 if the respondent is not religious and 1 if the respondent is religious.

class Social class of respondent. 0 is the lowest social class, 4 is the highest social class.

income Income of respondent. 0 is lowest and 6 is highest.

educ Education of respondent. 0 is lowest and 4 is highest.

age Age category of respondent. 0 is lowest and 12 is highest.

urban Indicator variable equal to 0 if the respondent is not a resident of an urban area and 1 if the respondent is a resident of an urban area.

Source

H. Anker and E.V. Oppenhuis. 1993. "Dutch Parliamentary Election Study." (computer file). Dutch Electoral Research Foundation and Netherlands Central Bureau of Statistics, Amsterdam.

References

K.M. Quinn and A.M. Martin. 2002. "An Integrated Computational Model of Multiparty Electoral Competition." *Statistical Science*. 17: 405-419.

PErisk

Political Economic Risk Data from 62 Countries in 1987

Description

Political Economic Risk Data from 62 Countries in 1987.

Usage

`data(PErisk)`

Format

A data frame with 62 observations on the following 9 variables. All data points are from 1987. See Quinn (2004) for more details.

country a factor with levels Argentina Australia Austria Bangladesh Belgium Bolivia Botswana Brazil Burma Cameroon Canada Chile Colombia Congo-Kinshasa Costa Rica Cote d'Ivoire Denmark Dominican Republic Ecuador Finland Gambia, The Ghana Greece Hungary India Indonesia Iran Ireland Israel Italy Japan Kenya Korea, South Malawi Malaysia Mexico Morocco New Zealand Nigeria Norway Papua New Guinea Paraguay Philippines Poland Portugal Sierra Leone Singapore South Africa Spain Sri Lanka Sweden Switzerland Syria Thailand Togo Tunisia Turkey United Kingdom Uruguay Venezuela Zambia Zimbabwe

courts an ordered factor with levels 0 < 1. **courts** is an indicator of whether the country in question is judged to have an independent judiciary. From Henisz (2002).

barb2 a numeric vector giving the natural log of the black market premium in each country. The black market premium is coded as the black market exchange rate (local currency per dollar) divided by the official exchange rate minus 1. From Marshall, Gurr, and Harff (2002).

prsexp2 an ordered factor with levels 0 < 1 < 2 < 3 < 4 < 5, giving the lack of expropriation risk. From Marshall, Gurr, and Harff (2002).

prscorr2 an ordered factor with levels 0 < 1 < 2 < 3 < 4 < 5, measuring the lack of corruption. From Marshall, Gurr, and Harff (2002).

gdpw2 a numeric vector giving the natural log of real GDP per worker in 1985 international prices. From Alvarez et al. (1999).

Source

Mike Alvarez, Jose Antonio Cheibub, Fernando Limongi, and Adam Przeworski. 1999. "ACLP Political and Economic Database." <http://www.ssc.upenn.edu/~cheibub/data/>.

Witold J. Henisz. 2002. "The Political Constraint Index (POLCON) Dataset." <http://www-management.wharton.upenn.edu/henisz/POLCON/ContactInfo.html>.

Monty G. Marshall, Ted Robert Gurr, and Barbara Harff. 2002. "State Failure Task Force Problem Set." <http://www.cidcm.umd.edu/inscr/stfail/index.htm>.

References

Kevin M. Quinn. 2004. "Bayesian Factor Analysis for Mixed Ordinal and Continuous Response." *Political Analysis*. 12: 338-353.

PostProbMod	<i>Calculate Posterior Probability of Model</i>
-------------	---

Description

This function takes an object of class `BayesFactor` and calculates the posterior probability that each model under study is correct given that one of the models under study is correct.

Usage

```
PostProbMod(BF, prior.probs=1)
```

Arguments

<code>BF</code>	An object of class <code>BayesFactor</code> .
<code>prior.probs</code>	The prior probabilities that each model is correct. Can be either a scalar or array. Must be positive. If the sum of the prior probabilities is not equal to 1 <code>prior.probs</code> will be normalized so that it does sum to unity.

Value

An array holding the posterior probabilities that each model under study is correct given that one of the models under study is correct.

See Also

[MCMCregress](#)

Examples

```
## Not run:
data(birthwt)

post1 <- MCMCregress(bwt~age+lwt+as.factor(race) + smoke + ht,
                    data=birthwt, b0=c(2700, 0, 0, -500, -500,
                                         -500, -500),
                    B0=c(1e-6, .01, .01, 1.6e-5, 1.6e-5, 1.6e-5,
                        1.6e-5), c0=10, d0=4500000,
```

```

      marginal.likelihood="Chib95", mcmc=10000)

post2 <- MCMCregress(bwt~age+lwt+as.factor(race) + smoke,
  data=birthwt, b0=c(2700, 0, 0, -500, -500,
                    -500),
  B0=c(1e-6, .01, .01, 1.6e-5, 1.6e-5, 1.6e-5),
  c0=10, d0=4500000,
  marginal.likelihood="Chib95", mcmc=10000)

post3 <- MCMCregress(bwt~as.factor(race) + smoke + ht,
  data=birthwt, b0=c(2700, -500, -500, -500, -500),
  B0=c(1e-6, 1.6e-5, 1.6e-5, 1.6e-5, 1.6e-5,
        1.6e-5), c0=10, d0=4500000,
  marginal.likelihood="Chib95", mcmc=10000)

BF <- BayesFactor(post1, post2, post3)
mod.probs <- PostProbMod(BF)
print(mod.probs)
## End(Not run)

```

 Senate

106th U.S. Senate Roll Call Vote Matrix

Description

This dataframe contains a matrix of votes cast by U.S. Senators in the 106th Congress.

Usage

```
data(Senate)
```

Format

The dataframe contains roll call data for all Senators in the 106th Senate. The first column (id) is the ICPSR member ID number, the second column (statecode) is the ICPSR state code, the third column (party) is the member's state name, and the fourth column (member) is the member's name. This is followed by all roll call votes (including unanimous ones) in the 106th. Nay votes are coded 0, yea votes are coded 1, and NAs are missing votes.

Source

Keith Poole. 2005. *106th Roll Call Vote Data*. <http://voteview.uh.edu/>.

SupremeCourt	<i>U.S. Supreme Court Vote Matrix</i>
--------------	---------------------------------------

Description

This dataframe contains a matrix votes cast by U.S. Supreme Court justices in all cases in the 2000 term.

Usage

```
data(SupremeCourt)
```

Format

The dataframe has contains data for justices Rehnquist, Stevens, O'Connor, Scalia, Kennedy, Souter, Thomas, Ginsburg, and Breyer for the 2000 term of the U.S. Supreme Court. It contains data from 43 non-unanimous cases. The votes are coded liberal (1) and conservative (0) using the protocol of Spaeth (2003). The unit of analysis is the case citation (ANALU=0). We are concerned with formally decided cases issued with written opinions, after full oral argument and cases decided by an equally divided vote (DECTYPE=1,5,6,7).

Source

Harold J. Spaeth. 2005. *Original United States Supreme Court Database: 1953-2004 Terms*. <http://www.as.uky.edu/polisci/ulmerproject/sctdata.htm>.

choicevar	<i>Handle Choice-Specific Covariates in Multinomial Choice Models</i>
-----------	---

Description

This function handles choice-specific covariates in multinomial choice models. See the example for an example of useage.

Usage

```
choicevar(var, varname, choicelevel)
```

Arguments

var	The is the name of the variable in the dataframe.
varname	The name of the new variable to be created.
choicelevel	The level of y that the variable corresponds to.

Value

The new variable used by the `MCMCmnl()` function.

See Also

[MCMCmnl](#)

Description

Density function and random generation from the Dirichlet distribution.

Usage

```
ddirichlet(x, alpha)
rdirichlet(n, alpha)
```

Arguments

x	A vector containing a single deviate or matrix containing one random deviate per row.
n	Number of random vectors to generate.
alpha	Vector of shape parameters, or matrix of shape parameters corresponding to the number of draw.

Details

The Dirichlet distribution is the multidimensional generalization of the beta distribution.

Value

`ddirichlet` gives the density. `rdirichlet` returns a matrix with `n` rows, each containing a single Dirichlet random deviate.

Author(s)

Code is taken from Greg's Miscellaneous Functions (`gregmisc`). His code was based on code posted by Ben Bolker to R-News on 15 Dec 2000.

See Also

[Beta](#)

Examples

```
density <- ddirichlet(c(.1, .2, .7), c(1,1,1))
draws <- rdirichlet(20, c(1,1,1) )
```

dtomogplot

*Dynamic Tomography Plot***Description**

dtomogplot is used to produce a tomography plot (see King, 1997) for a series of temporally ordered, partially observed 2 x 2 contingency tables.

Usage

```
dtomogplot(r0, r1, c0, c1, time.vec=NA, delay=0,
           xlab="fraction of r0 in c0 (p0)",
           ylab="fraction of r1 in c0 (p1)",
           color.palette=heat.colors, bgcol="black", ...)
```

Arguments

r0	An ($n_{tables} \times 1$) vector of row sums from row 0.
r1	An ($n_{tables} \times 1$) vector of row sums from row 1.
c0	An ($n_{tables} \times 1$) vector of column sums from column 0.
c1	An ($n_{tables} \times 1$) vector of column sums from column 1.
time.vec	Vector of time periods that correspond to the elements of r_0 , r_1 , c_0 , and c_1 .
delay	Time delay in seconds between the plotting of the tomography lines. Setting a positive delay is useful for visualizing temporal dependence.
xlab	The x axis label for the plot.
ylab	The y axis label for the plot.
color.palette	Color palette to be used to encode temporal patterns.
bgcol	The background color for the plot.
...	further arguments to be passed

Details

Consider the following partially observed 2 by 2 contingency table:

	$Y = 0$	$Y = 1$	
-----	-----	-----	-----
$X = 0$	Y_0		r_0
-----	-----	-----	-----
$X = 1$	Y_1		r_1
-----	-----	-----	-----
	c_0	c_1	N

where r_0 , r_1 , c_0 , c_1 , and N are non-negative integers that are observed. The interior cell entries are not observed. It is assumed that $Y_0|r_0 \sim \text{Binomial}(r_0, p_0)$ and $Y_1|r_1 \sim \text{Binomial}(r_1, p_1)$.

This function plots the bounds on the maximum likelihood estimates for (p_0, p_1) and color codes them by the elements of `time.vec`.

References

Gary King, 1997. *A Solution to the Ecological Inference Problem*. Princeton: Princeton University Press.

Jonathan Wakefield. 2001. "Ecological Inference for 2 x 2 Tables," Center for Statistics and the Social Sciences Working Paper no. 12. University of Washington.

Kevin M. Quinn. 2002. "Ecological Inference in the Presence of Temporal Dependence." Paper prepared for Ecological Inference Conference, Harvard University, June 17-18, 2002.

See Also

[MCMChierEI](#), [MCMCdynamicEI](#), [tomogplot](#)

Examples

```
## Not run:
## simulated data example 1
set.seed(3920)
n <- 100
r0 <- rpois(n, 2000)
r1 <- round(runif(n, 100, 4000))
p0.true <- pnorm(-1.5 + 1:n/(n/2))
p1.true <- pnorm(1.0 - 1:n/(n/4))
y0 <- rbinom(n, r0, p0.true)
y1 <- rbinom(n, r1, p1.true)
c0 <- y0 + y1
c1 <- (r0+r1) - c0

## plot data
dtomogplot(r0, r1, c0, c1, delay=0.1)

## simulated data example 2
set.seed(8722)
n <- 100
r0 <- rpois(n, 2000)
r1 <- round(runif(n, 100, 4000))
p0.true <- pnorm(-1.0 + sin(1:n/(n/4)))
p1.true <- pnorm(0.0 - 2*cos(1:n/(n/9)))
y0 <- rbinom(n, r0, p0.true)
y1 <- rbinom(n, r1, p1.true)
c0 <- y0 + y1
c1 <- (r0+r1) - c0

## plot data
dtomogplot(r0, r1, c0, c1, delay=0.1)
## End(Not run)
```

 InvGamma

The Inverse Gamma Distribution

Description

Density function and random generation from the inverse gamma distribution.

Usage

```
rinvgamma(n, shape, scale = 1)
dinvgamma(x, shape, scale = 1)
```

Arguments

x	Scalar location to evaluate density.
n	Number of draws from the distribution.
shape	Scalar shape parameter.
scale	Scalar scale parameter (default value one).

Details

An inverse gamma random variable with shape a and scale b has mean $\frac{b}{a-1}$ (assuming $a > 1$) and variance $\frac{b^2}{(a-1)^2(a-2)}$ (assuming $a > 2$).

Value

`dinvgamma` evaluates the density at `x`. `rinvgamma` takes `n` draws from the inverse Gamma distribution. The parameterization is consistent with the Gamma Distribution in the stats package.

See Also

[GammaDist](#)

Examples

```
density <- dinvgamma(4.2, 1.1)
draws <- rinvgamma(10, 3.2)
```

 InvWishart

The Inverse Wishart Distribution

Description

Density function and random generation from the Inverse Wishart distribution.

Usage

```
diwish(W, v, S)
riwish(v, S)
```

Arguments

W	Positive definite matrix W ($p \times p$).
v	Degrees of freedom (scalar).
S	Scale matrix ($p \times p$).

Details

The mean of an inverse Wishart random variable with v degrees of freedom and scale matrix S is $(v - p - 1)^{-1}S$.

Value

`diwish` evaluates the density at positive definite matrix W . `riwish` generates one random draw from the distribution.

Examples

```
density <- diwish(matrix(c(2,-.3,-.3,4),2,2), 3, matrix(c(1,.3,.3,1),2,2))
draw <- riwish(3, matrix(c(1,.3,.3,1),2,2))
```

NoncenHypergeom *The Noncentral Hypergeometric Distribution*

Description

Evaluates the density at a single point or all points, and generate random draws from the Noncentral Hypergeometric distribution.

Usage

```
dnoncenhypergeom(x=NA, n1, n2, m1, psi)
rnoncenhypergeom(n, n1, n2, m1, psi)
```

Arguments

x	The location to evaluate the density. If <code>x</code> is NA, then a matrix is returned with the density evaluated at all possible points.
n	The number of draws to make from the distribution.
n1	The size of group one.
n2	The size of group two.
m1	The observed number of positive outcomes (in both groups).
psi	Odds ratio.

Details

The Noncentral Hypergeometric is particularly useful for conditional inference for (2×2) tables. We use the parameterization and algorithms of Liao and Rosen (2001). The underlying R code is based on their published code. See their article for details of the parameterization.

Value

`dnoncenhypergeom` evaluates the density at point `x`, or a matrix with the first column containing the possible values of the random variable, and the second column containing the probabilities. `rnoncenhypergeom` returns a list of `n` random draws from the distribution.

Source

J. G. Liao and Ori Rosen. 2001. "Fast and Stable Algorithms for Computing and Sampling From the Noncentral Hypergeometric Distribution." *The American Statistician*. 55: 366-369.

Examples

```
density <- dnoncenhypergeom(NA, 500, 500, 500, 6.0)
draws <- rnoncenhypergeom(10, 500, 500, 500, 6.0)
```

procrustes

Procrustes Transformation

Description

This function performs a Procrustes transformation on a matrix `X` to minimize the squared distance between `X` and another matrix `Xstar`.

Usage

```
procrustes(X, Xstar, translation=FALSE, dilation=FALSE)
```

Arguments

<code>X</code>	The matrix to be transformed.
<code>Xstar</code>	The target matrix.
<code>translation</code>	logical value indicating whether <code>X</code> should be translated.
<code>dilation</code>	logical value indicating whether <code>X</code> should be dilated.

Details

`R`, `tt`, and `s` are chosen so that

$$sXR + 1tt' \approx X^*$$

.

`X.new` is given by:

$$X_{new} = sXR + 1tt'$$

.

Value

A list containing: `X.new` the matrix that is the Procrustes transformed version of `X`, `R` the rotation matrix, `tt` the translation vector, and `s` the scale factor.

References

Borg and Groenen. 1997. *Modern Multidimensional Scaling*. New York: Springer. pp. 340-342.

See Also

[MCMCirtKd](#)

read.Scythe

Read a Matrix from a File written by Scythe

Description

This function reads a matrix from an ASCII file in the form produced by the Scythe Statistical Library. Scythe output files contain the number of rows and columns in the first row, followed by the data.

Usage

```
read.Scythe(infile=NA)
```

Arguments

`infile` The file to be read. This can include path information.

Value

A matrix containing the data stored in the read file.

References

Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.

See Also

[write.Scythe](#)

Examples

```
## Not run: mymatrix <- read.Scythe("myfile.txt")
```

tomogplot	<i>Tomography Plot</i>
-----------	------------------------

Description

tomogplot is used to produce a tomography plot (see King, 1997) for a series of partially observed 2×2 contingency tables.

Usage

```
tomogplot(r0, r1, c0, c1, xlab="fraction of r0 in c0 (p0)",
          ylab="fraction of r1 in c0 (p1)", bgcol="white", ...)
```

Arguments

r0	An ($n_{tables} \times 1$) vector of row sums from row 0.
r1	An ($n_{tables} \times 1$) vector of row sums from row 1.
c0	An ($n_{tables} \times 1$) vector of column sums from column 0.
c1	An ($n_{tables} \times 1$) vector of column sums from column 1.
xlab	The x axis label for the plot.
ylab	The y axis label for the plot.
bgcol	The background color for the plot.
...	further arguments to be passed

Details

Consider the following partially observed 2 by 2 contingency table:

	$Y = 0$	$Y = 1$	
-----	-----	-----	-----
$X = 0$	Y_0		r_0
-----	-----	-----	-----
$X = 1$	Y_1		r_1
-----	-----	-----	-----
	c_0	c_1	N

where r_0 , r_1 , c_0 , c_1 , and N are non-negative integers that are observed. The interior cell entries are not observed. It is assumed that $Y_0|r_0 \sim \text{Binomial}(r_0, p_0)$ and $Y_1|r_1 \sim \text{Binomial}(r_1, p_1)$.

This function plots the bounds on the maximum likelihood estimates for (p_0, p_1) .

References

Gary King, 1997. *A Solution to the Ecological Inference Problem*. Princeton: Princeton University Press.

Jonathan Wakefield. 2001. "Ecological Inference for 2 x 2 Tables," Center for Statistics and the Social Sciences Working Paper no. 12. University of Washington.

See Also

[MCMChierEI](#), [MCMCdynamicEI](#), [dtomogplot](#)

Examples

```
r0 <- rpois(100, 500)
r1 <- rpois(100, 200)
c0 <- rpois(100, 100)
c1 <- (r0 + r1) - c0
tomogplot(r0, r1, c0, c1)
```

vech

Extract Lower Triangular Elements from a Symmetric Matrix

Description

This function takes a symmetric matrix and extracts a list of all lower triangular elements.

Usage

```
vech(x)
```

Arguments

x A symmetric matrix.

Details

This function checks to make sure the matrix is square, but it does not check for symmetry (it just pulls the lower triangular elements). The elements are stored in column major order. The original matrix can be restored using the `xpnd` command.

Value

A list of the lower triangular elements.

See Also

[xpnd](#)

Examples

```
symmat <- matrix(c(1,2,3,4,2,4,5,6,3,5,7,8,4,6,8,9),4,4)
vech(symmat)
```

Description

Density function and random generation from the Wishart distribution.

Usage

```
dwish(W, v, S)
rwish(v, S)
```

Arguments

W	Positive definite matrix W ($p \times p$).
v	Degrees of freedom (scalar).
S	Inverse scale matrix ($p \times p$).

Details

The mean of a Wishart random variable with v degrees of freedom and inverse scale matrix S is vS .

Value

`dwish` evaluates the density at positive definite matrix W . `rwish` generates one random draw from the distribution.

Examples

```
density <- dwish(matrix(c(2,-.3,-.3,4),2,2), 3, matrix(c(1,.3,.3,1),2,2))
draw <- rwish(3, matrix(c(1,.3,.3,1),2,2))
```

Description

This function writes a matrix to an ASCII file that can be read by the Sycthe Statistical Library. Scythe requires that input files contain the number of rows and columns in the first row, followed by the data.

Usage

```
write.Scythe(outmatrix, outfile=NA, overwrite=FALSE)
```

Arguments

<code>outmatrix</code>	The matrix to be written to a file.
<code>outfile</code>	The file to be written. This can include path information.
<code>overwrite</code>	A logical that determines whether an existing file should be over-written. By default, it protects the user from over-writing existing files.

Value

A zero if the file is properly written.

References

Andrew D. Martin, Kevin M. Quinn, and Daniel Pemstein. 2004. *Scythe Statistical Library 1.0*. <http://scythe.wustl.edu>.

See Also

[write.Scythe](#)

Examples

```
## Not run: write.Scythe(mymatrix, "myfile.txt")
```

xpnd

Expand a Vector into a Symmetric Matrix

Description

This function takes a vector of appropriate length (typically created using `vech`) and creates a symmetric matrix.

Usage

```
xpnd(x, nrow)
```

Arguments

<code>x</code>	A list of elements to expand into symmetric matrix.
<code>nrow</code>	The number of rows (and columns) in the returned matrix. Look into the details.

Details

This function is particularly useful when dealing with variance covariance matrices. Note that R stores matrices in column major order, and that the items in `x` will be recycled to fill the matrix if need be.

The number of rows can be specified or automatically computed from the number of elements in a given object via $(-1 + \sqrt{1 + 8 * length(x)})/2$.

Value

An ($nrows \times nrows$) symmetric matrix.

See Also

[vech](#)

Examples

```
xpnd(c(1, 2, 3, 4, 4, 5, 6, 7, 8, 9), 4)  
xpnd(c(1, 2, 3, 4, 4, 5, 6, 7, 8, 9))
```

Index

*Topic **datasets**

Nethvote, 60
PErisk, 61
Senate, 63
SupremeCourt, 64

*Topic **distribution**

Dirichlet, 65
InvGamma, 68
InvWishart, 69
NoncenHypergeom, 69
Wishart, 74

*Topic **file**

read.Scythe, 71
write.Scythe, 74

*Topic **hplot**

dtomogplot, 66
tomogplot, 72

*Topic **manip**

choicevar, 65
procrustes, 70
vech, 73
xpnd, 75

*Topic **models**

BayesFactor, 1
MCbinomialbeta, 56
MCMCdynamicEI, 5
MCMCfactanal, 8
MCMChierEI, 11
MCMCirt1d, 14
MCMCirtKd, 17
MCMCirtKdRob, 20
MCMClogit, 25
MCMCmetrop1R, 28
MCMCmixfactanal, 31
MCMCmnl, 36
MCMCoprobit, 40
MCMCordfactanal, 42
MCMCpanel, 45
MCMCpoisson, 47
MCMCprobit, 49
MCMCregress, 51
MCMCSVDreg, 3
MCMCtobit, 54

MCmultinomdirichlet, 57
MCnormalnormal, 58
MCpoissongamma, 59
PostProbMod, 62

BayesFactor, 1
Beta, 66

choicevar, 65

ddirichlet (*Dirichlet*), 65
dinvgamma (*InvGamma*), 68
Dirichlet, 65
diwish (*InvWishart*), 69
dnoncenhypergeom
 (*NoncenHypergeom*), 69
dtomogplot, 66, 73
dwish (*Wishart*), 74

factanal, 10, 34, 44

GammaDist, 68
glm, 27, 49, 51

InvGamma, 68
InvWishart, 69
is.BayesFactor (*BayesFactor*), 1
lm, 5, 53

MCbinomialbeta, 56
MCMCdynamicEI, 5, 13, 67, 73
MCMCfactanal, 8, 34, 44
MCMChierEI, 7, 11, 67, 73
MCMCirt1d, 14, 20, 24, 34, 44
MCMCirtKd, 14, 16, 17, 24, 34, 44, 71
MCMCirtKdRob, 20
MCMClogit, 25
MCMCmetrop1R, 28
MCMCmixfactanal, 31
MCMCmnl, 36, 65
MCMCoprobit, 40
MCMCordfactanal, 20, 34, 42
MCMCpanel, 45
MCMCpoisson, 47

MCMCprobit, 49
MCMCregress, 2, 51, 56, 63
MCMCSVDreg, 3
MCMCtobit, 54
MCmultinomdirichlet, 57
MCnormalnormal, 58
MCpoissongamma, 59
metrop, 30
multinom, 38

Nethvote, 60
NoncenHypergeom, 69

optim, 30

PERisk, 61
plot.mcmc, 5, 7, 10, 13, 16, 20, 24, 27, 30,
34, 38, 41, 44, 47, 49, 51, 53, 56–60
PostProbMod, 62
procrustes, 70

rdirichlet (*Dirichlet*), 65
read.Scythe, 71
rinvgamma (*InvGamma*), 68
riwish (*InvWishart*), 69
rnoncenhypergeom
(*NoncenHypergeom*), 69
rwish (*Wishart*), 74

Senate, 63
summary.mcmc, 5, 7, 10, 13, 16, 20, 24, 27,
30, 34, 38, 41, 44, 47, 49, 51, 53,
56–60
SupremeCourt, 64
survreg, 56

tomogplot, 67, 72

vech, 73, 76

Wishart, 74
write.Scythe, 72, 74, 75

xpnd, 73, 75