Package ‘bayess’

March 6, 2024

Title Bayesian Essentials with R

Version 1.6

Date 2024-03-04

Depends stats, mnormt, gplots, combinat

Author Jean-Michel Marin [aut, cre],
Christian P. Robert [aut]

Maintainer Jean-Michel Marin <jean-michel.marin@umontpellier.fr>

Description Allows the reenactment of the R programs used in
the book Bayesian Essentials with R without further programming.
R code being available as well, they can be modified by the user
to conduct one's own simulations.
Marin J.-M. and Robert C. P. (2014) <doi:10.1007/978-1-4614-8687-9>.

URL https://www.r-project.org, https://github.com/jmm34/bayess

License GPL-2

NeedsCompilation no

Repository CRAN

Date/Publication 2024-03-06 14:30:05 UTC

R topics documented:

`ardipper`	*Accept-reject algorithm for the open population capture-recapture model*

Description

This function is associated with Chapter 5 on capture-recapture model. It simulates samples from the non-standard distribution on r_1, the number of individuals vanishing between the first and second experiments, as expressed in (5.4) in the book, conditional on r_2, the number of individuals vanishing between the second and third experiments.

Usage

```
ardipper(nsimu, n1, c2, c3, r2, q1)
```

Arguments

`nsimu`	number of simulations
`n1`	first capture sample size
`c2`	number of individuals recaptured during the second experiment
`c3`	number of individuals recaptured during the third experiment
`r2`	number of individuals vanishing between the second and third experiments
`q1`	probability of disappearing from the population

Value

A sample of `nsimu` integers

Examples

```
ardipper(10,11,3,1,0,.1)
```

`ARllog`	*log-likelihood associated with an AR(p) model defined either through its natural coefficients or through the roots of the associated lag-polynomial*

Description

This function is related to Chapter 6 on dynamical models. It returns the numerical value of the log-likelihood associated with a time series and an AR(p) model, along with the natural coefficients psi of the AR(p) model if it is defined via the roots `lr` and `lc` of the associated lag-polynomial. The function thus uses either the natural parameterisation of the AR(p) model

$$x_t - \mu + \sum_{i=1}^{p} \psi_i (x_{t-i} - \mu) = \varepsilon_t$$

or the parameterisation via the lag-polynomial roots

$$\prod_{i=1}^{p} (1 - \lambda_i B) x_t = \varepsilon_t$$

where $B^j x_t = x_{t-j}$.

Usage

```
ARllog(p,dat,pr, pc, lr, lc, mu, sig2, compsi = TRUE, pepsi = c(1, rep(0, p)))
```

Arguments

`p`	order of the AR(p) model
`dat`	time series modelled by the AR(p) model
`pr`	number of real roots
`pc`	number of non-conjugate complex roots
`lr`	real roots
`lc`	complex roots, stored as real part for odd indices and imaginary part for even indices
`mu`	drift coefficient μ such that $(x_t - \mu)_t$ is a standard AR(p) series
`sig2`	variance of the Gaussian white noise $(\varepsilon_t)_t$
`compsi`	boolean variable indicating whether the coefficients ψ_i need to be retrievedfrom the roots of the lag-polynomial, i.e. if the model is defined by `pepsi` (when `compsi` is `FALSE`) or by `lr` and `lc` (when `compsi` is `TRUE`).
`pepsi`	potential p+1 coefficients ψ_i if `compsi` is `FALSE`, with 1 asthe compulsory first value

Value

`ll`	value of the log-likelihood
`ps`	vector of the ψ_i's

See Also

`MAllog`,`ARmh`

Examples

```
ARllog(p=3,dat=faithful[,1],pr=3,pc=0,
lr=c(-.1,.5,.2),lc=0,mu=0,sig2=var(faithful[,1]),compsi=FALSE,pepsi=c(1,rep(.1,3)))
```

`ARmh` *Metropolis–Hastings evaluation of the posterior associated with an AR(p) model*

Description

This function is associated with Chapter 6 on dynamic models. It implements a Metropolis–Hastings algorithm on the coefficients of the AR(p) model resorting to a simulation of the real and complex roots of the model. It includes jumps between adjacent numbers of real and complex roots, as well as random modifications for a given number of real and complex roots.

Usage

```
ARmh(x, p = 1, W = 10^3)
```

Arguments

`x`	time series to be modelled as an AR(p) model
`p`	order of the AR(p) model
`W`	number of iterations

Details

Even though *Bayesian Essentials with R* does not cover the reversible jump MCMC techniques due to Green (1995), which allows to explore spaces of different dimensions at once, this function relies on a simple form of reversible jump MCMC when moving from one number of complex roots to the next.

Value

`psis`	matrix of simulated ψ_i's
`mus`	vector of simulated μ's
`sigs`	vector of simulated σ^2's
`llik`	vector of corresponding likelihood values (useful to check for convergence)
`pcomp`	vector of simulated numbers of complex roots

References

Green, P.J. (1995) Reversible jump MCMC computaton and Bayesian model choice. *Biometrika* **82**, 711–732.

See Also

ARllog

Examples

```
data(Eurostoxx50)
x=Eurostoxx50[, 4]
resAR5=ARmh(x=x,p=5,W=50)
plot(resAR5$mus,type="l",col="steelblue4",xlab="Iterations",ylab=expression(mu))
```

---

bank *bank dataset (Chapter 4)*

Description

The bank dataset we analyze in the first part of Chapter 3 comes from Flury and Riedwyl (1988) and is made of four measurements on 100 genuine Swiss banknotes and 100 counterfeit ones. The response variable y is thus the status of the banknote, where 0 stands for genuine and 1 stands for counterfeit, while the explanatory factors are bill measurements.

Usage

```
data(bank)
```

Format

A data frame with 200 observations on the following 5 variables.

- `x1` length of the bill (in mm)
- `x2` width of the left edge (in mm)
- `x3` width of the right edge (in mm)
- `x4` bottom margin width (in mm)
- `y` response variable

Source

Flury, B. and Riedwyl, H. (1988) *Multivariate Statistics. A Practical Approach*, Chapman and Hall, London-New York.

Examples

```
data(bank)
summary(bank)
```

`BayesReg` *Bayesian linear regression output*

Description

This function contains the R code for the implementation of Zellner's G-prior analysis of the regression model as described in Chapter 3. The purpose of `BayesRef` is dual: first, this R function shows how easily automated this approach can be. Second, it also illustrates how it is possible to get exactly the same type of output as the standard R function `summary(lm(y~X))`. In particular, it calculates the Bayes factors for variable selection, more precisely single variable exclusion.

Usage

```
BayesReg(y, X, g = length(y), betatilde = rep(0, dim(X)[2]), prt = TRUE)
```

Arguments

`y`	response variable
`X`	matrix of regressors
`g`	constant g for the G-prior
`betatilde`	prior mean on β
`prt`	boolean variable for printing out the standard output

Value

`postmeancoeff`	posterior mean of the regression coefficients
`postsqrtcoeff`	posterior standard deviation of the regression coefficients
`log10bf`	log-Bayes factors against the full model
`postmeansigma2`	posterior mean of the variance of the model
`postvarsigma2`	posterior variance of the variance of the model

Examples

```
data(faithful)
BayesReg(faithful[,1],faithful[,2])
```

caterpillar *Pine processionary caterpillar dataset*

Description

The caterpillar dataset is extracted from a 1973 study on pine processionary caterpillars. The response variable is the log transform of the number of nests per unit. There are $p = 8$ potential explanatory variables and $n = 33$ areas.

Usage

```
data(caterpillar)
```

Format

A data frame with 33 observations on the following 9 variables.

- x1 altitude (in meters)
- x2 slope (in degrees)
- x3 number of pine trees in the area
- x4 height (in meters) of the tree sampled at the center of the area
- x5 orientation of the area (from 1 if southbound to 2 otherwise)
- x6 height (in meters) of the dominant tree
- x7 number of vegetation strata
- x8 mix settlement index (from 1 if not mixed to 2 if mixed)
- y logarithmic transform of the average number of nests of caterpillars per tree

Details

This dataset is used in Chapter 3 on linear regression. It assesses the influence of some forest settlement characteristics on the development of caterpillar colonies. It was first published and studied in Tomassone et al. (1993). The response variable is the logarithmic transform of the average number of nests of caterpillars per tree in an area of 500 square meters (which corresponds to the last column in caterpillar). There are $p = 8$ potential explanatory variables defined on $n = 33$ areas.

Source

Tomassone, R., Dervin, C., and Masson, J.P. (1993) *Biometrie: modelisation de phenomenes biologiques*. Dunod, Paris.

Examples

```
data(caterpillar)
summary(caterpillar)
```

`datha` *Non-standardised Licence dataset*

Description

The dataset used in Chapter 6 is derived from an image of a license plate, called `license` and not provided in the package. The actual histogram of the grey levels is concentrated on 256 values because of the poor resolution of the image, but we transformed the original data as `datha.txt`.

Usage

```
data(datha)
```

Format

A data frame with 2625 observations on the following variable.

`x` Grey levels

Details

`datha.txt` was produced by the following R code:

```
> license=jitter(license,10)
> datha=log((license-min(license)+.01)/
+ (max(license)+.01-license))
> write.table(datha,"datha.txt",row.names=FALSE,col.names=FALSE)
```

where `jitter` is used to randomize the dataset and avoid repetitions

Examples

```
data(datha)
datha=as.matrix(datha)
range(datha)
```

`Dnadataset` *DNA sequence of an HIV genome*

Description

`Dnadataset` is a base sequence corresponding to a complete HIV (which stands for Human Immunodeficiency Virus) genome where A, C, G, and T have been recoded as 1,2,3,4. It is modelled as a hidden Markov chain and is used in Chapter 7.

Usage

```
data(Dnadataset)
```

Format

A data frame with 9718 rows and two columns, the first one corresponding to the row number and the second one to the amino-acid value coded from 1 to 4.

Examples

```
data(Dnadataset)
summary(Dnadataset)
```

---

eurodip — *European Dipper dataset*

Description

This capture-recapture dataset on the *European dipper* bird covers 7 years (1981-1987 inclusive) of observations of captures within one of three zones. It is used in Chapter 5.

Usage

```
data(eurodip)
```

Format

A data frame with 294 observations on the following 7 variables.

`t1` non-capture/location on year 1981

`t2` non-capture/location on year 1982

`t3` non-capture/location on year 1983

`t4` non-capture/location on year 1984

`t5` non-capture/location on year 1985

`t6` non-capture/location on year 1986

`t7` non-capture/location on year 1987

Details

The data consists of markings and recaptures of breeding adults each year during the breeding period from early March to early June. Birds were at least one year old when initially banded. In `eurodip`, each row gof seven digits corresponds to a capture-recapture story for a given dipper, 0 indicating an absence of capture that year and, in the case of a capture, 1, 2, or 3 representing the zone where the dipper is captured. This dataset corresponds to three geographical zones covering 200 square kilometers in eastern France. It was kindly provided to us by J.D. Lebreton.

References

Lebreton, J.-D., K. P. Burnham, J. Clobert, and D. R. Anderson. (1992) Modeling survival and testing biological hypotheses using marked animals: case studies and recent advances. *Ecol. Monogr.* **62**, 67-118.

Examples

```
data(eurodip)
summary(eurodip)
```

`Eurostoxx50` *Eurostoxx50 exerpt dataset*

Description

This dataset is a collection of four time series connected with the stock market. Those are the stock values of the four companies ABN Amro, Aegon, Ahold Kon., and Air Liquide, observed from January 1, 1998, to November 9, 2003.

Usage

```
data(Eurostoxx50)
```

Format

A data frame with 1486 observations on the following 5 variables.

`date` six-digit date

`Abn` value of the ABN Amro stock

`Aeg` value of the Aegon stock

`Aho` value of the Ahold Kon. stock

`AL` value of the Air Liquide stock

Details

Those four companies are the first stocks (in alphabetical order) to appear in the financial index Eurostoxx50.

Examples

```
data(Eurostoxx50)
summary(Eurostoxx50)
```

`gibbs` *Gibbs sampler and Chib's evidence approximation for a generic univariate mixture of normal distributions*

Description

This function implements a regular Gibbs sampling algorithm on the posterior distribution associated with a mixture of normal distributions, taking advantage of the missing data structure. It then runs an averaging of the simulations over all permutations of the component indices in order to avoid incomplete label switching and to validate Chib's representation of the evidence. This function reproduces `gibbsnorm` as its first stage, *however it may be much slower because of its second stage*.

Usage

```
gibbs(niter, datha, mix)
```

Arguments

`niter`	number of Gibbs iterations
`datha`	sample vector
`mix`	list made of `k`, number of components, `p`, `mu`, and `sig`, starting values of the paramerers, all of size `k` (see example below)

Value

`k`	number of components in the mixture (superfluous as it is invariant over the execution of the R code)
`mu`	matrix of the Gibbs samples on the μ_i parameters
`sig`	matrix of the Gibbs samples on the σ_i parameters
`prog`	matrix of the Gibbs samples on the mixture weights
`lolik`	vector of the observed log-likelihoods along iterations
`chibdeno`	denominator of Chib's approximation to the evidence (see example below)

References

Chib, S. (1995) Marginal likelihood from the Gibbs output. *J. American Statist. Associ.* **90**, 1313-1321.

See Also

`gibbsnorm`

Examples

```
faithdata=faithful[,1]
mu=rnorm(3,mean=mean(faithdata),sd=sd(faithdata)/10)
sig=1/rgamma(3,shape=10,scale=var(faithdata))
mix=list(k=3,p=rdirichlet(par=rep(1,3)),mu=mu,sig=sig)
resim3=gibbs(100,faithdata,mix)
lulu=order(resim3$lolik)[100]
lnum1=resim3$lolik[lulu]
lnum2=sum(dnorm(resim3$mu[lulu,],mean=mean(faithdata),sd=resim3$sig[lulu,],log=TRUE)+
dgamma(resim3$sig[lulu,],10,var(faithdata),log=TRUE)-2*log(resim3$sig[lulu,]))+
sum((rep(0.5,mix$k)-1)*log(resim3$p[lulu,]))+
lgamma(sum(rep(0.5,mix$k)))-sum(lgamma(rep(0.5,mix$k)))
lchibapprox3=lnum1+lnum2-log(resim3$deno)
```

---

gibbscap1 — *Gibbs sampler for the two-stage open population capture-recapture model*

Description

This function implements a regular Gibbs sampler associated with Chapter 5 for a two-stage capture recapture model with open populations, accounting for the possibility that some individuals vanish between two successive capture experiments.

Usage

```
gibbscap1(nsimu, n1, c2, c3, N0 = n1/runif(1), r10, r20)
```

Arguments

nsimu	number of simulated values in the sample
n1	first capture population size
c2	number of individuals recaptured during the second experiment
c3	number of individuals recaptured during the third experiment
N0	starting value for the population size
r10	starting value for the number of individuals who vanished between the first and second experiments
r20	starting value for the number of individuals who vanished between the second and third experiments

Value

N	Gibbs sample of the simulated population size
p	Gibbs sample of the probability of capture
q	Gibbs sample of the probability of leaving the population

r1	Gibbs sample of the number of individuals who vanished between the first and second experiments
r2	Gibbs sample of the number of individuals who vanished between the second and third experiments

Examples

```
res=gibbscap1(100,32,21,15,200,10,5)
plot(res$p,type="l",col="steelblue3",xlab="iterations",ylab="p")
```

---

gibbscap2 *Gibbs sampling for the Arnason-Schwarz capture-recapture model*

Description

In the Arnason-Schwarz capture-recapture model (see Chapter 5), individual histories are observed and missing steps can be inferred upon. For the dataset eurodip, the moves between regions can be reconstituted. This is the first instance of a hidden Markov model met in the book.

Usage

```
gibbscap2(nsimu, z)
```

Arguments

nsimu	numbed of simulation steps in the Gibbs sampler
z	data, capture history of each individual, with 0 coding non-capture

Value

p	Gibbs sample of capture probabilities across time
phi	Gibbs sample of survival probabilities across time and locations
psi	Gibbs sample of interstata movement probabilities across time and locations
late	Gibbs averages of completed histories

Examples

```
data(eurodip)
res=gibbscap2(10,eurodip[1:100,])
plot(res$p,type="l",col="steelblue3",xlab="iterations",ylab="p")
```

gibbsmean — *Gibbs sampler on a mixture posterior distribution with unknown means*

Description

This function implements a Gibbs sampler for a toy mixture problem (Chapter 6) with two Gaussian components and only the means unknown, so that likelihood and posterior surfaces can be drawn.

Usage

```
gibbsmean(p, datha, niter = 10^4)
```

Arguments

`p`	first component weight
`datha`	dataset to be modelled as a mixture
`niter`	number of Gibbs iterations

Value

Sample of μ's as a matrix of size `niter` x 2

See Also

`plotmix`

Examples

```
dat=plotmix(plottin=FALSE)$sample
simu=gibbsmean(0.7,dat,niter=100)
plot(simu,pch=19,cex=.5,col="sienna",xlab=expression(mu[1]),ylab=expression(mu[2]))
```

gibbsnorm — *Gibbs sampler for a generic mixture posterior distribution*

Description

This function implements the generic Gibbs sampler of Diebolt and Robert (1994) for producing a sample from the posterior distribution associated with a univariate mixture of k normal components with all $3k - 1$ parameters unknown.

Usage

```
gibbsnorm(niter, dat, mix)
```

Arguments

niter	number of iterations in the Gibbs sampler
dat	mixture sample
mix	list defined as mix=list(k=k,p=p,mu=mu,sig=sig), where k is an integer and the remaining entries are vectors of length k

Details

Under conjugate priors on the means (normal distributions), variances (inverse gamma distributions), and weights (Dirichlet distribution), the full conditional distributions given the latent variables are directly available and can be used in a straightforward Gibbs sampler. This function is only the first step of the function gibbs, but it may be much faster as it avoids the computation of the evidence via Chib's approach.

Value

k	number of components (superfluous)
mu	Gibbs sample of all mean parameters
sig	Gibbs sample of all variance parameters
p	Gibbs sample of all weight parameters
lopost	sequence of log-likelihood values along Gibbs iterations

References

Chib, S. (1995) Marginal likelihood from the Gibbs output. *J. American Statist. Associ.* **90**, 1313-1321.

Diebolt, J. and Robert, C.P. (1992) Estimation of finite mixture distributions by Bayesian sampling. *J. Royal Statist. Society* **56**, 363-375.

See Also

rdirichlet, gibbs

Examples

```
data(datha)
datha=as.matrix(datha)
mix=list(k=3,mu=mean(datha),sig=var(datha))
res=gibbsnorm(10,datha,mix)
plot(res$p[,1],type="l",col="steelblue3",xlab="iterations",ylab="p")
```

`hmflatlogit` *Metropolis-Hastings for the logit model under a flat prior*

Description

Under the assumption that the posterior distribution is well-defined, this Metropolis-Hastings algorithm produces a sample from the posterior distribution on the logit model coefficient β under a flat prior.

Usage

```
hmflatlogit(niter, y, X, scale)
```

Arguments

`niter`	number of iterations
`y`	binary response variable
`X`	matrix of covariates with the same number of rows as `y`
`scale`	scale of the Metropolis-Hastings random walk

Value

The function produces a sample of β's as a matrix of size `niter` x `p`, where `p` is the number of covariates.

See Also

`hmflatprobit`

Examples

```
data(bank)
bank=as.matrix(bank)
y=bank[,5]
X=bank[,1:4]
flatlogit=hmflatlogit(1000,y,X,1)
par(mfrow=c(1,3),mar=1+c(1.5,1.5,1.5,1.5))
plot(flatlogit[,1],type="l",xlab="Iterations",ylab=expression(beta[1]))
hist(flatlogit[101:1000,1],nclass=50,prob=TRUE,main="",xlab=expression(beta[1]))
acf(flatlogit[101:1000,1],lag=10,main="",ylab="Autocorrelation",ci=FALSE)
```

hmflatloglin *Metropolis-Hastings for the log-linear model under a flat prior*

Description

This version of hmflatlogit operates on the log-linear model, assuming that the posterior associated with the flat prior and the data is well-defined. The proposal is based on a random walk Metropolis-Hastings step.

Usage

```
hmflatloglin(niter, y, X, scale)
```

Arguments

niter	number of iterations
y	binary response variable
X	matrix of covariates with the same number of rows as y
scale	scale of the Metropolis-Hastings random walk

Value

The function produces a sample of β's as a matrix of size niter x p, where p is the number of covariates.

See Also

hmflatlogit

Examples

```
airqual=na.omit(airquality)
ozone=cut(airqual$Ozone,c(min(airqual$Ozone),median(airqual$Ozone),max(airqual$Ozone)),
include.lowest=TRUE)
month=as.factor(airqual$Month)
tempe=cut(airqual$Temp,c(min(airqual$Temp),median(airqual$Temp),max(airqual$Temp)),
include.lowest=TRUE)
counts=table(ozone,tempe,month)
counts=as.vector(counts)
ozo=gl(2,1,20)
temp=gl(2,2,20)
mon=gl(5,4,20)
x1=rep(1,20)
lulu=rep(0,20)
x2=x3=x4=x5=x6=x7=x8=x9=lulu
x2[ozo==2]=x3[temp==2]=x4[mon==2]=x5[mon==3]=x6[mon==4]=1
x7[mon==5]=x8[ozo==2 & temp==2]=x9[ozo==2 & mon==2]=1
x10=x11=x12=x13=x14=x15=x16=lulu
```

```
x10[ozo==2 & mon==3]=x11[ozo==2 & mon==4]=x12[ozo==2 & mon==5]=1
x13[temp==2 & mon==2]=x14[temp==2 & mon==3]=x15[temp==2 & mon==4]=1
x16[temp==2 & mon==5]=1
X=cbind(x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16)
flatloglin=hmflatloglin(1000,counts,X,0.5)
par(mfrow=c(4,4),mar=1+c(1.5,1.5,1.5,1.5),cex=0.8)
for (i in 1:16) plot(flatloglin[,i],type="l",ylab="",xlab="Iterations")
```

---

hmflatprobit *Metropolis-Hastings for the probit model under a flat prior*

Description

This random walk Metropolis-Hastings algorithm takes advantage of the availability of the maximum likelihood estimator (available via the glm function) to center and scale the random walk in an efficient manner.

Usage

```
hmflatprobit(niter, y, X, scale)
```

Arguments

niter	number of iterations
y	binary response variable
X	covariates
scale	scale of the random walk

Value

The function produces a sample of β's of size niter.

See Also

hmflatlogit

Examples

```
data(bank)
bank=as.matrix(bank)
y=bank[,5]
X=bank[,1:4]
flatprobit=hmflatprobit(1000,y,X,1)
mean(flatprobit[101:1000,1])
```

hmhmm	*Estimation of a hidden Markov model with 2 hidden and 4 observed states*

Description

This function implements a Metropolis within Gibbs algorithm that produces a sample on the parameters p_{ij} and q^i_j of the hidden Markov model (Chapter 7). It includes a function `likej` that computes the likelihood of the times series using a forward-backward algorithm.

Usage

```
hmhmm(M = 100, y)
```

Arguments

`M`	Number of Gibbs iterations
`y`	times series to be modelled by a hidden Markov model

Details

The Metropolis-within-Gibbs step involves Dirichlet proposals with a random choice of the scale between 1 and 1e5.

Value

`BigR`	matrix of the iterated values returned by the MCMC algorithm containing p_{11} and p_{22}, transition probabilities, and q^1 and q^2, vector of probabilities for both latent states
`olike`	sequence of the log-likelihoods produced by the MCMC sequence

Examples

```
res=hmhmm(M=500,y=sample(1:4,10,rep=TRUE))
plot(res$olike,type="l",main="log-likelihood",xlab="iterations",ylab="")
```

hmmeantemp	*Metropolis-Hastings with tempering steps for the mean mixture posterior model*

Description

This function provides another toy illustration of the capabilities of a tempered random walk Metropolis-Hastings algorithm applied to the posterior distribution associated with a two-component normal mixture with only its means unknown (Chapter 7). It shows how a decrease in the temperature leads to a proper exploration of the target density surface, despite the existence of two well-separated modes.

Usage

```
hmmeantemp(dat, niter, var = 1, alpha = 1)
```

Arguments

`dat`	set to be modelled as a mixture
`niter`	number of iterations
`var`	variance of the random walk
`alpha`	temperature, expressed as power of the likelihood

Details

When $\alpha = 1$ the function operates (and can be used) as a regular Metropolis-Hastings algorithm.

Value

sample of μ_i's as a matrix of size `niter` x 2

Examples

```
dat=plotmix(plot=FALSE)$sample
simu=hmmeantemp(dat,1000)
plot(simu,pch=19,cex=.5,col="sienna",xlab=expression(mu[1]),ylab=expression(mu[2]))
```

`hmnoinflogit` *Metropolis-Hastings for the logit model under a noninformative prior*

Description

This function runs a Metropolis-Hastings algorithm that produces a sample from the posterior distribution for the logit model (Chapter 4) coefficient β associated with a noninformative prior defined in the book.

Usage

```
hmnoinflogit(niter, y, X, scale)
```

Arguments

`niter`	number of iterations
`y`	binary response variable
`X`	matrix of covariates with the same number of rows as `y`
`scale`	scale of the random walk

Value

sample of β's as a matrix of size `niter` x p, where p is the number of covariates

See Also

hmnoinfprobit

Examples

```
data(bank)
bank=as.matrix(bank)
y=bank[,5]
X=bank[,1:4]
noinflogit=hmnoinflogit(1000,y,X,1)
par(mfrow=c(1,3),mar=1+c(1.5,1.5,1.5,1.5))
plot(noinflogit[,1],type="l",xlab="Iterations",ylab=expression(beta[1]))
hist(noinflogit[101:1000,1],nclass=50,prob=TRUE,main="",xlab=expression(beta[1]))
acf(noinflogit[101:1000,1],lag=10,main="",ylab="Autocorrelation",ci=FALSE)
```

---

hmnoinfloglin   *Metropolis-Hastings for the log-linear model under a noninformative prior*

Description

This function is a version of `hmnoinflogit` for the log-linear model, using a non-informative prior defined in Chapter 4 and a proposal based on a random walk Metropolis-Hastings step.

Usage

```
hmnoinfloglin(niter, y, X, scale)
```

Arguments

`niter`	number of iterations
`y`	binary response variable
`X`	matrix of covariates with the same number of rows as `y`
`scale`	scale of the random walk

Value

The function produces a sample of β's as a matrix of size `niter` x `p`, where `p` is the number of covariates.

See Also

hmflatloglin

Examples

```
airqual=na.omit(airquality)
ozone=cut(airqual$Ozone,c(min(airqual$Ozone),median(airqual$Ozone),max(airqual$Ozone)),
include.lowest=TRUE)
month=as.factor(airqual$Month)
tempe=cut(airqual$Temp,c(min(airqual$Temp),median(airqual$Temp),max(airqual$Temp)),
include.lowest=TRUE)
counts=table(ozone,tempe,month)
counts=as.vector(counts)
ozo=gl(2,1,20)
temp=gl(2,2,20)
mon=gl(5,4,20)
x1=rep(1,20)
lulu=rep(0,20)
x2=x3=x4=x5=x6=x7=x8=x9=lulu
x2[ozo==2]=x3[temp==2]=x4[mon==2]=x5[mon==3]=1
x6[mon==4]=x7[mon==5]=x8[ozo==2 & temp==2]=x9[ozo==2 & mon==2]=1
x10=x11=x12=x13=x14=x15=x16=lulu
x10[ozo==2 & mon==3]=x11[ozo==2 & mon==4]=x12[ozo==2 & mon==5]=x13[temp==2 & mon==2]=1
x14[temp==2 & mon==3]=x15[temp==2 & mon==4]=x16[temp==2 & mon==5]=1
X=cbind(x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16)
noinloglin=hmnoinfloglin(1000,counts,X,0.5)
par(mfrow=c(4,4),mar=1+c(1.5,1.5,1.5,1.5),cex=0.8)
for (i in 1:16) plot(noinloglin[,i],type="l",ylab="",xlab="Iterations")
```

---

hmnoinfprobit — *Metropolis-Hastings for the probit model under a noninformative prior*

Description

This function runs a Metropolis-Hastings algorithm that produces a sample from the posterior distribution for the probit model coefficient β associated with a noninformative prior defined in Chapter 4.

Usage

```
hmnoinfprobit(niter, y, X, scale)
```

Arguments

`niter`	number of iterations
`y`	binary response variable
`X`	matrix of covariates with the same number of rows as `y`
`scale`	scale of the random walk

Value

The function produces a sample of β's as a matrix of size `niter` x `p`, where `p` is the number of covariates.

See Also

`hmnoinflogit`, `hmflatprobit`

Examples

```
data(bank)
bank=as.matrix(bank)
y=bank[,5]
X=bank[,1:4]
noinfprobit=hmflatprobit(1000,y,X,1)
par(mfrow=c(1,3),mar=1+c(1.5,1.5,1.5,1.5))
plot(noinfprobit[,1],type="l",xlab="Iterations",ylab=expression(beta[1]))
hist(noinfprobit[101:1000,1],nclass=50,prob=TRUE,main="",xlab=expression(beta[1]))
acf(noinfprobit[101:1000,1],lag=10,main="",ylab="Autocorrelation",ci=FALSE)
```

---

isinghm *Metropolis-Hastings for the Ising model*

Description

This is the Metropolis-Hastings version of the original Gibbs algorithm on the Ising model (Chapter 8). Its basic step only proposes changes of values at selected pixels, avoiding the inefficient updates that do not modify the current value of `x`.

Usage

```
isinghm(niter, n, m=n,beta)
```

Arguments

niter	number of iterations of the algorithm
n	number of rows in the grid
m	number of columns in the grid
beta	Ising parameter

Value

`x`, a realisation from the Ising distribution as a `n` x `m` matrix of 0's and 1's

See Also

`isingibbs`

Examples

```
prepa=runif(1,0,2)
prop=isinghm(10,24,24,prepa)
image(1:24,1:24,prop)
```

`isingibbs` *Gibbs sampler for the Ising model*

Description

This is the original Geman and Geman (1984) Gibbs sampler on the Ising model that gave its name to the method. It simulates an $n \times m$ grid from the Ising distribution.

Usage

```
isingibbs(niter, n, m=n, beta)
```

Arguments

`niter`	number of iterations of the algorithm
`n`	number of rows in the grid
`m`	number of columns in the grid
`beta`	Ising parameter

Value

`x`, a realisation from the Ising distribution as a matrix of size `n x m`

References

Geman, S. and Geman, D. (1984) Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.*, **6**, 721–741.

See Also

`isinghm`

Examples

```
image(1:20,1:20,isingibbs(10,20,20,beta=0.3))
```

`Laichedata` *Laiche dataset*

Description

This dataset depicts the presence of plants (tufted sedges) in a part of a wetland. It is 25x25 matrix of zeroes and ones, used in Chapter 8.

Usage

```
data(Laichedata)
```

Format

A data frame corresponding to a 25x25 matrix of zeroes and ones.

Examples

```
data(Laichedata)
image(as.matrix(Laichedata))
```

`logitll` *Log-likelihood of the logit model*

Description

Direct computation of the logarithm of the likelihood of a standard logit model (Chapter 4)

$$P(y = 1|X, \beta) = \{1 + \exp(-\beta^T X)\}^{-1}.$$

Usage

```
logitll(beta, y, X)
```

Arguments

`beta`	coefficient of the logit model
`y`	vector of binary response variables
`X`	covariate matrix

Value

returns the logarithm of the logit likelihood for the data `y`, covariate matrix `X` and parameter vector `beta`

See Also

probitll

Examples

```
data(bank)
y=bank[,5]
X=as.matrix(bank[,-5])
logitll(runif(4),y,X)
```

---

logitnoinflpost *Log of the posterior distribution for the probit model under a noninformative prior*

Description

This function computes the logarithm of the posterior density associated with a logit model and the noninformative prior used in Chapter 4.

Usage

```
logitnoinflpost(beta, y, X)
```

Arguments

beta	parameter of the logit model
y	binary response variable
X	covariate matrix

Value

returns the logarithm of the logit likelihood for the data y, covariate matrix X and parameter beta

See Also

probitnoinflpost

Examples

```
data(bank)
y=bank[,5]
X=as.matrix(bank[,-5])
logitnoinflpost(runif(4),y,X)
```

loglinll — *Log of the likelihood of the log-linear model*

Description

This function provides a direct computation of the logarithm of the likelihood of a standard log-linear model, as defined in Chapter 4.

Usage

```
loglinll(beta, y, X)
```

Arguments

`beta`	coefficient of the logit model
`y`	vector of binary response variables
`X`	covariate matrix

Value

returns the logarithmic value of the logit likelihood for the data `y`, covariate matrix `X` and parameter vector `beta`

Examples

```
X=matrix(rnorm(20*3),ncol=3)
beta=c(3,-2,1)
y=rpois(20,exp(X%*%beta))
loglinll(beta, y, X)
```

loglinnoinflpost — *Log of the posterior density for the log-linear model under a noninformative prior*

Description

This function computes the logarithm of the posterior density associated with a log-linear model and the noninformative prior used in Chapter 4.

Usage

```
loglinnoinflpost(beta, y, X)
```

Arguments

`beta`	parameter of the log-linear model
`y`	binary response variable
`X`	covariate matrix

Details

This function does not test for coherence between the lengths of `y`, `X` and `beta`, hence may return an error message in case of incoherence.

Value

returns the logarithm of the logit posterior density for the data `y`, covariate matrix `X` and parameter vector `beta`

Examples

```
X=matrix(rnorm(20*3),ncol=3)
beta=c(3,-2,1)
y=rpois(20,exp(X%*%beta))
loglinnoinflpost(beta, y, X)
```

`MAllog` *log-likelihood associated with an MA(p) model*

Description

This function returns the numerical value of the log-likelihood associated with a time series and an MA(p) model in Chapter 7. It either uses the natural parameterisation of the MA(p) model

$$x_t - \mu = \varepsilon_t - \sum_{j=1}^{p} \psi_j \varepsilon_{t-j}$$

or the parameterisation via the lag-polynomial roots

$$x_t - \mu = \prod_{i=1}^{p} (1 - \lambda_i B) \varepsilon_t$$

where $B^j \varepsilon_t = \varepsilon_{t-j}$.

Usage

```
MAllog(p,dat,pr,pc,lr,lc,mu,sig2,compsi=T,pepsi=rep(0,p),eps=rnorm(p))
```

Arguments

`p`	order of the MA model
`dat`	time series modelled by the MA(p) model
`pr`	number of real roots in the lag polynomial
`pc`	number of complex roots in the lag polynomial, necessarily even
`lr`	real roots
`lc`	complex roots, stored as real part for odd indices and imaginary part for even indices. (`lc` is either 0 when `pc=0` or a vector of even length when `pc>0`.)
`mu`	drift parameter μ such that $(X_t - \mu)_t$ is a standard MA(p) series
`sig2`	variance of the Gaussian white noise $(\varepsilon_t)_t$
`compsi`	boolean variable indicating whether the coefficients ψ_i need to be retrieved from the roots of the lag-polynomial (if `TRUE`) or not (if `FALSE`)
`pepsi`	potential coefficients ψ_i, computed by the function if `compsi` is `TRUE`
`eps`	white noise terms $(\varepsilon_t)_{t\leq 0}$ with negative indices

Value

`ll`	value of the log-likelihood
`ps`	vector of the ψ_i's, similar to the entry if `compsi` is `FALSE`

See Also

`ARllog`, `MAmh`

Examples

```
MAllog(p=3,dat=faithful[,1],pr=3,pc=0,lr=rep(.1,3),lc=0,
mu=0,sig2=var(faithful[,1]),compsi=FALSE,pepsi=rep(.1,3),eps=rnorm(3))
```

MAmh *Metropolis–Hastings evaluation of the posterior associated with an MA(p) model*

Description

This function implements a Metropolis–Hastings algorithm on the coefficients of the MA(p) model, involving the simulation of the real and complex roots of the model. The algorithm includes jumps between adjacent numbers of real and complex roots, as well as random modifications for a given number of real and complex roots. It is thus a special case of a *reversible jump MCMC* algorithm (Green, 1995).

Usage

```
MAmh(x, p = 1, W = 10^3)
```

Arguments

`x`	time series to be modelled as an MA(p) model
`p`	order of the MA(p) model
`W`	number of iterations

Value

`psis`	matrix of simulated ψ_i's
`mus`	vector of simulated μ's
`sigs`	vector of simulated σ^2's
`llik`	vector of corresponding log-likelihood values (useful to check for convergence)
`pcomp`	vector of simulated numbers of complex roots

References

Green, P.J. (1995) Reversible jump MCMC computaton and Bayesian model choice. *Biometrika* **82**, 711–732.

See Also

`MAllog`

Examples

```
data(Eurostoxx50)
x=Eurostoxx50[1:350, 5]
resMA5=MAmh(x=x,p=5,W=50)
plot(resMA5$mus,type="l",col="steelblue4",xlab="Iterations",ylab=expression(mu))
```

`Menteith` *Grey-level image of the Lake of Menteith*

Description

This dataset is a 100x100 pixel satellite image of the lake of Menteith, near Stirling, Scotland. The purpose of analyzing this satellite dataset is to classify all pixels into one of six states in order to detect some homogeneous regions.

Usage

```
data(Menteith)
```

Format

data frame of a 100 x 100 image with 106 grey levels

See Also

reconstruct

Examples

```
data(Menteith)
image(1:100,1:100,as.matrix(Menteith),col=gray(256:1/256),xlab="",ylab="")
```

ModChoBayesReg *Bayesian model choice procedure for the linear model*

Description

This function computes the posterior probabilities of all (for less than 15 covariates) or the most probable (for more than 15 covariates) submodels obtained by eliminating some covariates.

Usage

```
ModChoBayesReg(y, X, g = length(y), betatilde = rep(0, dim(X)[2]),
niter = 1e+05, prt = TRUE)
```

Arguments

y	response variable
X	covariate matrix
g	constant in the g prior
betatilde	prior expectation of the regression coefficient β
niter	number of Gibbs iterations in the case there are more than 15 covariates
prt	boolean variable for printing the standard output

Details

When using a conjugate prior for the linear model such as the G prior, the marginal likelihood and hence the evidence are available in closed form. If the number of explanatory variables is less than 15, the exact derivation of the posterior probabilities for all submodels can be undertaken. Indeed, $2^{15} = 32768$ means that the problem remains tractable. When the number of explanatory variables gets larger, a random exploration of the collection of submodels becomes necessary, as explained in the book (Chapter 3). The proposal to change one variable indicator is made at random and accepting this move follows from a Metropolis–Hastings step.

Value

top10models	models with the ten largest posterior probabilities
postprobtop10	posterior probabilities of those ten most likely models

Examples

```
data(caterpillar)
y=log(caterpillar$y)
X=as.matrix(caterpillar[,1:8])
res2=ModChoBayesReg(y,X)
```

normaldata   *Normal dataset*

Description

This dataset is used as "the" normal dataset in Chapter 2. It is linked with the famous Michelson-Morley experiment that opened the way to Einstein's relativity theory in 1887. It corresponds to the more precise experiment of Illingworth in 1927. The datapoints are measurment of differences in the speeds of two light beams travelling the same distance in two orthogonal directions.

Usage

```
data(normaldata)
```

Format

A data frame with 64 observations on the following 2 variables.

`x1` index of the experiment

`x2` averaged fringe displacement in the experiment

Details

The 64 data points in this dataset are associated with session numbers, corresponding to two different times of the day, and they represent the averaged fringe displacement due to orientation taken over ten measurements made by Illingworth, who assumed a normal error model.

See Also

morley

Examples

```
data(normaldata)
shift=matrix(normaldata,ncol=2,byrow=TRUE)[,2]
hist(shift[[1]],nclass=10,col="steelblue",prob=TRUE,main="")
```

pbino *Posterior expectation for the binomial capture-recapture model*

Description

This function provides an estimation of the number of dippers by a posterior expectation, based on a uniform prior and the `eurodip` dataset, as described in Chapter 5.

Usage

```
pbino(nplus)
```

Arguments

nplus	number of different dippers captured

Value

returns a numerical value that estimates the number of dippers in the population

See Also

eurodip

Examples

```
data(eurodip)
year81=eurodip[,1]
nplus=sum(year81>0)
sum((1:400)*pbino(nplus))
```

pcapture *Posterior probabilities for the multiple stage capture-recapture model*

Description

This function computes the posterior expectation of the population size for a multiple stage capture-recapture experiment (Chapter 5) under a uniform prior on the range (0,400).

Usage

```
pcapture(T, nplus, nc)
```

Arguments

`T`	number of experiments
`nplus`	total number of captured animals
`nc`	total number of captures

Details

This analysis is based on the restrictive assumption that all dippers captured in the second year were already present in the population during the first year.

Value

numerical value of the posterior expectation

See Also

`pdarroch`

Examples

```
sum((1:400)*pcapture(2,70,81))
```

`pdarroch` *Posterior probabilities for the Darroch model*

Description

This function computes the posterior expectation of the population size for a two-stage Darroch capture-recapture experiment (Chapter 5) under a uniform prior on the range (0,400).

Usage

```
pdarroch(n1, n2, m2)
```

Arguments

`n1`	size of the first capture experiment
`n2`	size of the second capture experiment
`m2`	number of recaptured individuals

Details

This model can be seen as a conditional version of the two-stage model when conditioning on both sample sizes n_1 and n_2.

Value

numerical value of the posterior expectation

See Also

pcapture

Examples

```
for (i in 6:16) print(round(sum(pdarroch(22,43,i)*1:400)))
```

---

plotmix — *Graphical representation of a normal mixture log-likelihood*

Description

This function gives an image representation of the log-likelihood surface of a mixture (Chapter 6) of two normal densities with means μ_1 and μ_2 unknown. It first generates the random sample associated with the distribution.

Usage

```
plotmix(mu1 = 2.5, mu2 = 0, p = 0.7, n = 500, plottin = TRUE, nl = 50)
```

Arguments

mu1	first mean
mu2	second mean
p	weight of the first component
n	number of observations
plottin	boolean variable to plot the surface (or not)
nl	number of contours

Details

In this case, the parameters are identifiable: μ_1 and μ_2 cannot be confused when p is not 0.5. Nonetheless, the log-likelihood surface in this figure often exhibits two modes, one being close to the true value of the parameters used to simulate the dataset and one corresponding to a reflected separation of the dataset into two homogeneous groups.

Value

sample	the simulated sample
like	the discretised representation of the log-likelihood surface

See Also

gibbsmean, *hmmeantemp*

Examples

```
resumix=plotmix()
```

`pottsgibbs` *Gibbs sampler for the Potts model*

Description

This function produces one simulation of a square `numb` by `numb` grid from a Potts distribution with four colours and a four neighbour structure, relying on `niter` iterations of a standard Gibbs sampler.

Usage

```
pottsgibbs(niter, numb, beta)
```

Arguments

`niter`	number of Gibbs iterations
`numb`	size of the square grid
`beta`	parameter of the Potts model

Value

returns a random realisation from the Potts model

References

Geman, S. and Geman, D. (1984) Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images. *IEEE Trans. Pattern Anal. Mach. Intell.*, **6**, 721–741.

See Also

`pottshm`

Examples

```
ex=pottsgibbs(100,15,.4)
image(ex)
```

pottshm — *Metropolis-Hastings sampler for a Potts model with* `ncol` *classes*

Description

This function returns a simulation of a `n` by `m` grid from a Potts distribution with `ncol` colours and a four neighbour structure, using a Metropolis-Hastings step that avoids proposing a value identical to the current state of the Markov chain.

Usage

```
pottshm(ncol=2,niter=10^4,n,m=n,beta=0)
```

Arguments

`ncol`	number of colors
`niter`	number of Metropolis-Hastings iterations
`n`	number of rows in the image
`m`	number of columns in the image
`beta`	parameter of the Potts model

Value

returns a random realisation from the Potts model

See Also

`pottsgibbs`

Examples

```
ex=pottshm(niter=50,n=15,beta=.4)
hist(ex,prob=TRUE,col="steelblue",main="pottshm()")
```

probet — *Coverage of the interval* (a, b) *by the Beta cdf*

Description

This function computes the coverage of the interval (a, b) by the Beta $\mathrm{B}(\alpha, \alpha(1-c)/c)$ distribution.

Usage

```
probet(a, b, c, alpha)
```

Arguments

a	lower bound of the prior 95%~confidence interval
b	upper bound of the prior 95%~confidence interval
c	mean parameter of the prior distribution
alpha	scale parameter of the prior distribution

Value

numerical value between 0 and 1 corresponding to the coverage

See Also

solbeta

Examples

```
probet(.1,.5,.3,2)
```

---

probitll *Log-likelihood of the probit model*

Description

This function implements a direct computation of the logarithm of the likelihood of a standard probit model

$$P(y = 1|X, \beta) = \Phi(\beta^T X).$$

Usage

```
probitll(beta, y, X)
```

Arguments

beta	coefficient of the probit model
y	vector of binary response variables
X	covariate matrix

Value

returns the logarithm of the probit likelihood for the data `y`, covariate matrix `X` and parameter vector `beta`

See Also

logitll

Examples

```
data(bank)
y=bank[,5]
X=as.matrix(bank[,-5])
probitll(runif(4),y,X)
```

---

probitnoinflpost — *Log of the posterior density for the probit model under a non-informative model*

Description

This function computes the logarithm of the posterior density associated with a probit model and the non-informative prior used in Chapter 4.

Usage

```
probitnoinflpost(beta, y, X)
```

Arguments

beta	parameter of the probit model
y	binary response variable
X	covariate matrix

Value

returns the logarithm of the posterior density associated with a logit model for the data `y`, covariate matrix `X` and parameter `beta`

See Also

logitnoinflpost

Examples

```
data(bank)
y=bank[,5]
X=as.matrix(bank[,-5])
probitnoinflpost(runif(4),y,X)
```

rdirichlet — *Random generator for the Dirichlet distribution*

Description

This function simulates a sample from a Dirichlet distribution on the k dimensional simplex with arbitrary parameters. The simulation is based on a renormalised vector of gamma variates.

Usage

```
rdirichlet(n = 1, par = rep(1, 2))
```

Arguments

`n`	number of simulations
`par`	parameters of the Dirichlet distribution, whose length determines the value of `k`

Details

Surprisingly, there is no default Dirichlet distribution generator in the R base packages like `MASS` or `stats`. This function can be used in full generality, apart from the book (Chapter 6).

Value

returns a (n, k) matrix of Dirichlet simulations

Examples

```
apply(rdirichlet(10,rep(3,5)),2,mean)
```

reconstruct — *Image reconstruction for the Potts model with six classes*

Description

This function adresses the reconstruction of an image distributed from a Potts model based on a noisy version of this image. The purpose of image segmentation (Chapter 8) is to cluster pixels into homogeneous classes without supervision or preliminary definition of those classes, based only on the spatial coherence of the structure. The underlying algorithm is an hybrid Gibbs sampler.

Usage

```
reconstruct(niter, y)
```

Arguments

`niter`	number of Gibbs iterations
`y`	blurred image defined as a matrix

Details

Using a Potts model on the true image, and uniform priors on the genuine parameters of the model, the hybrid Gibbs sampler generates the image pixels and the other parameters one at a time, the *hybrid* stage being due to the Potts model parameter, since it implies using a numerical integration via `integrate`. The code includes (or rather excludes!) the numerical integration via the vector `dali`, which contains the values of the integration over a 21 point grid, since this numerical integration is extremely time-consuming.

Value

`beta`	MCMC chain for the parameter β of the Potts model
`mu`	MCMC chain for the mean parameter of the blurring model
`sigma`	MCMC chain for the variance parameter of the blurring model
`xcum`	frequencies of simulated colours at every pixel of the image

See Also

`Menteith`

Examples

```
## Not run: data(Menteith)
lm3=as.matrix(Menteith)
#warning, this step is a bit lengthy
titus=reconstruct(20,lm3)
#allocation function
affect=function(u) order(u)[6]
#
aff=apply(titus$xcum,1,affect)
aff=t(matrix(aff,100,100))
par(mfrow=c(2,1))
image(1:100,1:100,lm3,col=gray(256:1/256),xlab="",ylab="")
image(1:100,1:100,aff,col=gray(6:1/6),xlab="",ylab="")

## End(Not run)
```

solbeta — *Recursive resolution of beta prior calibration*

Description

In the capture-recapture experiment of Chapter 5, the prior information is represented by a prior expectation and prior confidence intervals. This function derives the corresponding beta $B(\alpha, \beta)$ prior distribution by a divide-and-conquer scheme.

Usage

```
solbeta(a, b, c, prec = 10^(-3))
```

Arguments

a	lower bound of the prior 95%~confidence interval
b	upper bound of the prior 95%~confidence interval
c	mean of the prior distribution
prec	maximal precision on the beta coefficient α

Details

Since the mean μ of the beta distribution is known, there is a single free parameter α to determine, since $\beta = \alpha(1-\mu)/\mu$. The function `solbeta` searches for the corresponding value of α, starting with a precision of 1 and stopping at the requested precision `prec`.

Value

alpha	first coefficient of the beta distribution
beta	second coefficient of the beta distribution

See Also

probet

Examples

```
solbeta(.1,.5,.3,10^(-4))
```

sumising — *Approximation by path sampling of the normalising constant for the Ising model*

Description

This function implements a path sampling approximation of the normalising constant of an Ising model with a four neighbour relation.

Usage

```
sumising(niter = 10^3, numb, beta)
```

Arguments

niter	number of iterations
numb	size of the square grid for the Ising model
beta	Ising model parameter

Value

returns a vector of 21 values for $Z(\beta)$ corresponding to a regular sequence of β's between 0 and 2

See Also

isingibbs,isinghm

Examples

```
Z=seq(0,2,length=21)
for (i in 1:21)
  Z[i]=sumising(5,numb=24,beta=Z[i])
lrcst=approxfun(seq(0,2,length=21),Z)
plot(seq(0,2,length=21),Z,xlab="",ylab="")
curve(lrcst,0,2,add=TRUE)
```

thresh — *Bound for the accept-reject algorithm in Chapter 5*

Description

This function is used in `ardipper` to determine the bound for the accept-reject algorithm simulating the non-standard conditional distribution of r_1.

Usage

```
thresh(k, n1, c2, c3, r2, q1)
```

Arguments

k	current proposal for the number of individuals vanishing between the first and second experiments
n1	first capture population size
c2	number of individuals recaptured during the second experiment
c3	number of individuals recaptured during the third experiment
r2	number of individuals vanishing between the second and third experiments
q1	probability of disappearing from the population

Details

This upper bound is equal to

$$\frac{\binom{n_1-c_2}{k}\binom{n_1-k}{c_3+r_2}}{\binom{\bar{r}}{k}}$$

Value

numerical value of the upper bound, to be compared with the uniform random draw

See Also

ardipper

Examples

```
## Not run: if (runif(1) < thresh(y,n1,c2,c3,r2,q1))
```

---

truncnorm — *Random simulator for the truncated normal distribution*

Description

This is a plain random generator for a normal variate $\mathcal{N}(\mu, \tau^2)$ truncated to (a, b), using the inverse cdf `qnorm`. It may thus be imprecise for extreme values of the bounds.

Usage

```
truncnorm(n, mu, tau2, a, b)
```

Arguments

n	number of simulated variates
mu	mean of the original normal
tau2	variance of the original normal
a	lower bound
b	upper bound

Value

a sample of real numbers over (a, b) with size n

See Also

reconstruct

Examples

```
x=truncnorm(10^3,1,2,3,4)
hist(x,nclass=123,col="wheat",prob=TRUE)
```

---

xneig4 *Number of neighbours with the same colour*

Description

This is a basis function used in simulation algorithms on the Ising and Potts models. It counts how many of the four neighbours of $x_{a,b}$ are of the same colour as this pixel.

Usage

```
xneig4(x, a, b, col)
```

Arguments

x	grid of coloured pixels
a	row index
b	column index
col	current or proposed colour

Value

integer between 0 and 4 giving the number of neighbours with the same colour

See Also

pottsgibbs, sumising

Examples

```
data(Laichedata)
xneig4(Laichedata,2,3,1)
xneig4(Laichedata,2,3,0)
```

Index