Package 's525'

April 23, 2018

Type Package

Title Implementation of common Bayesian models

Version 1.0

Date 2017-11-26

Author Daniel Bourgeois

Maintainer Daniel Bourgeois <dcb10@rice.edu>

Description This package contains implementations of common Bayesian models.

Imports truncnorm, MASS, coda

License MIT License

R topics documented:

`s525-package` *Implementation of common Bayesian models*

Description

This package contains implementations of common Bayesian models.

Details

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.

Author(s)

Daniel Bourgeois

Maintainer: Daniel Bourgeois <dcb10@rice.edu>

Examples

```
# simple examples of the most important functions
```

`factor_analysis` *Runs gibbs sampler for a factor model.*

Description

The model is as follows:

$$y_i = \Lambda \eta_i + \epsilon_i$$

$$\epsilon_i \sim N_p(0, \Sigma)$$

where $\Sigma = \text{diag}(\sigma_1^2, ..., \sigma_p^2)$.

See Joyee Ghosh & David B. Dunson (2009) Default Prior Distributions and Efficient Posterior Computation in Bayesian Factor Analysis, Journal of Computational and Graphical Statistics, 18:2, 306-320, DOI: 10.1198/jcgs.2009.07145 for conditional posteriors.

Usage

```
factor_analysis <- function(
  Y,
  k,
  niter = 1000,
  shape_psi = 1/2,
  rate_psi = 1/2,
  shape_sigma2 = 1,
  rate_sigma2 = 0.2,
  nonzero_structure = NULL)
```

Arguments

`Y`	n by p matrix
`k`	number of factors
`niter`	number of iterations for the gibbs sampler to run.
`shape_psi`	shape parameter for psi. Can be a scalar or a k vector
`rate_psi`	rate parameter for psi. Can be a k vector
`shape_sigma2`	shape parameter for sigma2. Can be a p vector
`rate_sigma2`	rate parameter for sigma2. Can be a p vector
`nonzero_structure`	A boolean p x k matrix. If the i, jth spot is TRUE, then λ_{ij} is free. If the i, jth spot is FALSE, then λ_{ij} is zero. If not set, then a lower triangular matrix is used.

Value

`sigma`	An niter x p matrix of posterior values
`eta`	An niter x n x k array of posterior values
`lambda`	An niter x p x k array of posterior values

`factor_analysis_with_regression`

Runs gibbs sampler for a factor model with regression on the factors.

Description

The model is as follows:

$$y_i = \alpha + \Lambda\eta_i + \epsilon_i,$$
$$\eta_i = Bx_i + \delta_i$$
$$\epsilon_i \sim N_p(0, \tau^{-1})$$
$$\delta_i \sim N_k(0, I)$$

where τ = diag($\tau_1, ..., \tau_p$).

See Joyee Ghosh & David B. Dunson (2009) Default Prior Distributions and Efficient Posterior Computation in Bayesian Factor Analysis, Journal of Computational and Graphical Statistics, 18:2, 306-320, DOI: 10.1198/jcgs.2009.07145.

Usage

```
factor_analysis_with_regression <- function(
  Y,
  X,
  k,
  niter = 1,
  shape_psi = 1/2,
  rate_psi = 1/2,
  shape_tau = 1,
  rate_tau = 0.2,
  coef_multiplier = 10,
  nonzero_structure = NULL)
```

Arguments

`Y`	n by p matrix
`X`	n by f matrix
`k`	number of factors
`niter`	number of iterations for the gibbs sampler to run.
`shape_psi`	shape parameter for psi. Can be a scalar or a k vector
`rate_psi`	rate parameter for psi. Can be a k vector
`shape_tau`	shape parameter for sigma2. Can be a p vector
`rate_tau`	rate parameter for sigma2. Can be a p vector
`nonzero_structure`	A boolean p x k matrix. If the i, jth spot is TRUE, then λ_{ij} is free. If the i, jth spot is FALSE, then λ_{ij} is zero. If not set, then a lower triangular matrix is used.

Value

`alpha`	An niter x p matrix of posterior values
`lambda`	An niter x p x k array of posterior values
`tau`	An niter x p matrix of posterior values
`eta`	An niter x n x k array of posterior values
`B`	An niter x k x f array of posterior values

`factor_infinite` *Runs gibbs sampler for a factor model with potentially infinite k*

Description

The model is as follows:

$$y_i = \Lambda \eta_i + \epsilon_i$$

$$\epsilon_i \sim N_p(0, \Sigma)$$

where Σ = diag($\sigma_1^2, ..., \sigma_p^2$).

See Bhattacharya, Anirban, and David B. Dunson. 'Sparse Bayesian infinite factor models.' Biometrika (2011): 291-306.

Usage

```
factor_infinite <- function(
  Y,
  k_star,
  niter = 1000,
  a_sig = 1,
  b_sig = 1,
  rho = 3,
  a1 = 1,
  a2 = 3)
```

Arguments

`Y`	n by p matrix
`k_star`	number of factors to cut off at
`niter`	number of iterations for the gibbs sampler to run.
`a_sig`	shape hyper parameter for the σ^2
`b_sig`	shape hyper parameter for the σ^2
`rho`	hyper parameter
`a1`	hyper parameter
`a2`	hyper parameter

Value

`sigma2`	An niter x p matrix of posterior values
`lambda`	An niter x p x k_star array of posterior values

`gibbs` *Gibbs Sampler Skeleton*

Description

This function provides the common skeleton used in all gibbs sampler: set initial values and then for as many iterations as needed, sample conditional posteriors of each of the parameters

Usage

```
gibbs <- function(
  niter,
  init,
  hypers,
  known_data,
  conditional_samplers,
  iter_argname = "iter",
  ignore = c())
```

Arguments

`niter`	number of iterations to run sampler
`init`	A list of initial values to start the sampler. The list must be named and the names must correspond to the names in conditional_samplers.
`hypers`	A list of hyper values. This is passed into the conditional samplers as needed.
`known_data`	A list of data that is passed into the conditional samplers.
`conditional_samplers`	A list of functions. Each function is a sampler for each of the parameters. The list must be named and the names must correspond to the names in init.
`iter_argname`	The conditional samplers may choose to take iter_argname as a function argument. For example, if the conditional sampler is only to perform a sample every so many iterations.
`ignore`	A vector of parameter names to not store and return.
`asmcmc`	Whether or not to make the return value a mcmc object.

Details

This function returns a list of arrays where each element is the set of samples generated for the corresponding parameter. In addition, the amount of running time used to calculate each variable is returned.

Examples

```
# Do a gibbs sampling for an AR(1) model:
#   y_t = phi*y_{t-1} + eps_t
#   eps_t ~ normal(0, s2a)
#   phi ~ I_[-1,1](phi)*N(0,1)
#   s2a ~ inv_gamma(a, b)

# doing a grid sample along n_grid points; the prior
# is a truncated normal(0, 1)
sample_phi = function(y, s2a, n_grid = 1000)
{
  T = length(y)
  log_prob = function(phi)
  {
    - 0.5/s2a*(
    sum((y[2:T] - phi*y[1:(T-1)])^2)) - 0.5*phi^2
  }

  xs = seq(-0.999, 0.999, length.out = n_grid)
  prob = exp(sapply(xs, log_prob))
  sample(xs, 1, prob = prob)
}

# s2a has a inv gamma (a, b) prior; using conjugacy
sample_s2a = function(y, phi, a, b)
{
  T = length(y)

  1/rgamma(1,
    shape = a + T/2,
    rate = b + 0.5*sum((y[2:T] - c(phi)*y[1:(T-1)])^2))
}

n = 1000
niter = 500
ret = gibbs(
  niter,
  init                 = list(s2a = 1, phi = 0),
  hypers               = list(a = 0, b = 0),
  known_data           = list(y = rnorm(n)),
  conditional_samplers = list(s2a = sample_s2a, phi = sample_phi),
  ignore               = "s2a")

plot(ret$phi)
```

`horseshoe_regression` *Gibbs Sampler for Horseshoe Regression*

Description

The horseshoe regression model is given by

$$[y_i|x_i, \beta, \sigma] \sim N(x_i^t\beta, \sigma^2), i = 1, ..., n,$$

$$[\beta_j|\sigma, \lambda_j] \sim N(0, \sigma^2\lambda_j^2),$$

$$[\lambda_j|A] \sim C^+(0, A),$$

$$A \sim Uniform(0, 10),$$

$$p(\sigma^2) \propto \frac{1}{\sigma^2}.$$

The half-Cauchy parameter expansion is used; given by

$$[\eta_j|\gamma_j] \sim Gamma(\frac{1}{2}, \gamma_j),$$

$$[\gamma_j] \sim Gamma(\frac{1}{2}, \frac{1}{A^2}).$$

Let $\eta_j = \lambda_j^{-2}$, $\tau_A = A^{-2}$, $\tau = \frac{1}{\sigma^2}$ and $\Lambda = diag(\eta_1, ..., \eta_p)$. The full conditionals are given by:

$$[\beta|Y, X, \eta, \tau] \sim \mathcal{N}((X'X + \Lambda)^{-1}X'Y, \tau^{-1}(X'X + \Lambda)^{-1}),$$

$$[\eta_j|\beta_j, \gamma_j, \tau] \sim \exp(\frac{\tau\beta_j^2}{2} + \gamma_j),$$

$$[\gamma_j|\eta_j, \tau_A] \sim \exp(\eta_j + \tau_A),$$

$$[\tau_A|\gamma] \sim \text{Gamma}(\frac{p-1}{2}, \sum \gamma_i)\text{I}_{(\frac{1}{100}, \infty)},$$

$$[\tau|Y, X, \beta, \eta] \sim \text{Gamma}(\frac{n+p}{2}, \frac{(y - X\beta)'(y - X\beta) + \beta'\Lambda\beta}{2}).$$

Usage

```
horseshoe_regression <- function(
  Y,
  X,
  niter = 10000)
```

Arguments

`Y`	n by 1
`X`	n by p predictor matrix
`niter`	number of gibbs sampling iterations

Details

This function returns the generated parameters from the gibbs sampling markov chain.

Value

`beta`	An niter x p matrix
`lambda`	An niter x p matrix
`sigma`	An niter x 1 matrix

Examples

```
# Load the data
 prostate.data = "https://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.data"
prostate = read.table(file = prostate.data, sep="", header = TRUE)
# Training data:
prostate_train = prostate[which(prostate$train),-10]
# Testing data:
prostate_test = prostate[which(!prostate$train),-10]
# Response:
y = prostate_train$lpsa
# Center and scale the data:
y = scale(y)
# And the predictors
X = scale(prostate_train[,names(prostate_train) != "lpsa"])

gibbs_hs <- horseshoe_regression(y, X, niter=10000)
shrinkage_regression_plot(gibbs_hs$beta, y, X)
```

`hpdAlong` *hpdAlong*

Description

Take the hpd interval using replicates along the along dimension.

Usage

```
  hpdAlong <- function(x, along, prob = 0.95)
```

Arguments

`x`	An n dimensional array
`along`	the dimension of x to use replicates along
`prob`	the target probability content of the intervals

Details

This function returns two n - 1 dimensional arrays that is the result of taking the HPDinterval along each of the vectors.

`mat_apply` *mat apply*

Description

Apply a function to each element in an input and return an arbitrary dimensioned array.

Usage

```
mat_apply <- function(vec, fun)
```

Arguments

`vec`	vector of inputs fed into fun
`fun`	a function to call for each input of vec

Details

This function returns an array with first dimension of length(vec) provided length(vec) > 1–the first dimension indexes over the length(vec) outputs.

Examples

```
# returns a 10 x 3 x 3 array. ret[i,,] contains i*diag(3).
ret <- mat_apply(1:10, function(i){ i*diag(3) })
```

`meanAlong` *meanAlong*

Description

Take the mean along a chosen dimension of an arbitrarily dimensioned array.

Usage

```
meanAlong <- function(x, along)
```

Arguments

`x`	An n dimensional array
`along`	the dimension of x to sum along

Details

# The method of the source code came from Hadley Wickham: # https://stackoverflow.com/questions/14500707/select-along-one-of-n-dimensions-in-array

This function returns an n - 1 dimensional array that is the result of the mean along the along dimension.

Examples

```
nx = 3
ny = 3
nz = 3
x = array(1:(nx*ny*nz), dim = c(nx, ny, nz))

print(x)
meanAlong(x, 3)
```

`metropolis` *Metropolis MCMC*

Description

The metropolis algorithm is a special case of the metropolis-hastings algorithm, namely where the proposal distribution is symmetric.

Usage

```
metropolis <- function(
  rproposal,
  prob,
  niter,
  init,
  log_prob = FALSE)
```

Arguments

`rproposal`	rproposal(previous_val) generates a value from the proposal distribution
`prob`	prob(val_proposed)/prob(val_t_minus_1) forms the acceptance probability
`niter`	number of iterations to perform
`init`	vector of initial values
`log_prob`	whether or not prob function specifies log probabilities

Details

This function returns a niter x d matrix of values where d is the dimension of init and the dimension of each element from rproposal. The returned matrix contains all generated values of the metropolis walk.

Examples

```
vals <- metropolis(
  rproposal = function(val){ rnorm(1, mean = val, sd = 1){} },
  prob = posterior_prob,
  niter = 10000,
  init = 0)

library(coda)
plot(as.mcmc(vals))
```

`plackett_luce` *Gibbs Sampler for posterior plackett luce parameters*

Description

The prior on each parameter is given by a gamma distribution.

The parameter expansion is given by Caron, Francois, and Arnaud Doucet. "Efficient Bayesian inference for generalized Bradley<bf>Terry models." Journal of Computational and Graphical Statistics 21.1 (2012): 174-196.

Usage

```
plackett_luce <- function(
  rank_matrix,
  shape = 1,
  rate = 1,
  niter = 1000)
```

Arguments

`rank_matrix`	n by p matrix of integers ranging from 1,...,k where k = max(rank_matrix, na.rm=TRUE). Each row may contain NA values provided that at least 2 values of each row are not NA. The same integer can appear in each row of the rank matrix as well.
`shape`	scalar or k vector specifying prior shape parameter over pl parameters
`rate`	scalar or k vector specifying prior rate parameter over pl parameters
`niter`	number of gibbs sampling iterations

Details

This function returns a niter x k matrix of the gibbs sampler.

Examples

```
library(coda)

vals <- rplackett_luce(100, vs = c(100, 1, 1))
post_vals <- plackett_luce(vals)
plot(as.mcmc(post_vals))
```

`probit_horseshoe_regression` *Gibbs Sampler for Probit Regression with Horseshoe Prior*

Description

The probit regression model with horseshoe prior is given by

$$y_i|\pi_i \sim Bernoulli(\pi_i),$$
$$\pi_i = \Phi(x_i^t\beta),$$
$$[\beta_j|\lambda_j] \sim N(0, \lambda_j^2), j = 2, ..., p,$$
$$p(\beta_1) \propto 1,$$
$$[\lambda_j|A] \sim C^+(0, A), j = 2, ..., p,$$
$$A \sim Uniform(0, 10).$$

where Φ is given by the gaussian CDF.

The implemented parameter-expanded model is given by

$$y_i^* = x_i^t\beta + \epsilon_i,$$
$$\epsilon_i \sim N(0, 1),$$
$$y_i = I(y_i^* > 0).$$

The half-Cauchy parameter expansion is also used; given by

$$[\eta_j|\gamma_j] \sim Gamma(\frac{1}{2}, \gamma_j),$$
$$[\gamma_j] \sim Gamma(\frac{1}{2}, \frac{1}{A^2})$$

and $\eta_j = \lambda_j^{-2}$, $\tau_A = A^{-2}$, The full conditionals are given by:

$$[y_i^*|y_i, \beta, X] \sim sgn(y_i, y_i^*)N(x_i^t\beta, 1)$$

where sqn is 1 if both arguments are of the same sign and zero otherwise,

$$[\beta|Y^*, X, \eta] \sim \mathcal{N}(Q^{-1}l, Q^{-1})$$

where $Q = X'X + diag(0, 1/\eta_2, ..., 1/\eta_p)$ and $l = X'Y^*$,

$$[\eta_j|\beta_j, \gamma_j] \sim \exp(\frac{\beta_j^2}{2} + \gamma_j),$$
$$[\gamma_j|\eta_j, \tau_A] \sim \exp(\eta_j + \tau_A),$$
$$[\tau_A|\gamma] \sim \text{Gamma}(\frac{p-2}{2}, \sum \gamma_i)\text{I}_{(\frac{1}{100}, \infty)}.$$

Usage

```
probit_horseshoe_regression <- function(
  Y,
  X,
  niter,
  init = NULL)
```

Arguments

`Y`	n by 1 vector of ones and zeros
`X`	n by p predictor matrix, where p > 1 and the first column of X is all 1.
`niter`	number of gibbs sampling iterations
`init`	Initial starting values for beta. If NULL, beta is set to zero.

Details

This function returns a niter x p matrix of values where p is the second dimension of the predictor matrix X. The returned matrix contains all generated values of the gibbs sampling markov chain.

Examples

```
print("TODO")
```

`probit_regression` *Gibbs Sampler for Probit Regression*

Description

The probit regression model is given by

$$y_i|\pi_i \sim Bernoulli(\pi_i),$$

$$\pi_i = \Phi(x_i^t\beta),$$

$$\beta_j \sim N(0, A^2).$$

where Φ is given by the gaussian CDF.

The implemented parameter-expanded model is given by

$$y_i^* = x_i^t\beta + \epsilon_i,$$

$$\epsilon_i \sim N(0, 1),$$

$$y_i = I(y_i^* > 0).$$

The full conditional distributions are given by

$$[\beta|y, y^*, X] \sim N(Q^{-1}l, Q^{-1})$$

where $Q = X^t X + A^{-2} I$ and $l = X^t y^*$,

$$[y_i^* | y_i, \beta, X] \sim sgn(y_i, y_i^*) N(x_i^t \beta, 1)$$

where sqn is 1 if both arguments are of the same sign and zero otherwise.

If A is a vector, $Q = X^t X + diag(A)$. A flat prior on the intercept at position 1 is thus given by A = c(Inf, rep(3, p)).

Usage

```
probit_regression <- function(
  Y,
  X,
  niter,
  A = 3,
  init = NULL)
```

Arguments

`Y`	n by 1 vector of ones and zeros
`X`	n by p predictor matrix
`niter`	number of gibbs sampling iterations
`A`	A parameter. The default value is chosen to provide a reasonable range of π. If A is a vector, then different variances are given for each intercept.
`init`	Initial starting values for beta. If NULL, beta is set to zero.

Details

This function returns a niter x p matrix of values where p is the second dimension of the predictor matrix X. The returned matrix contains all generated values of the gibbs sampling markov chain.

Examples

```
library(LearnBayes)
library(s525)

data(donner)
y = donner$survival
X = cbind(1, donner$age, donner$male)
niter <- 2000

gibbs_results <- probit_regression(y, X, niter)
```

`rdirichlet` *Generate from a dirichlet distribution*

Description

Generate from a dirichlet distribution

Usage

```
rdirichlet <- function(n, shape)
```

Arguments

`n`	number of permutations to generate
`shape`	shape parameter of length k

Details

This function returns a n x k matrix where each row is a generated value from the dirichlet distribution with the given shape parameter.

`ridge_regression` *Gibbs Sampler for Ridge Regression*

Description

The ridge regression model coeffecients are given by

$$\arg\min_{\beta} \parallel Y - X\beta \parallel^2 + \lambda \parallel \beta \parallel^2 .$$

This is the implementation of the bayesian interpretation. Namely,

$$[y_i|x_i, \beta, \tau] \sim N(x_i^t\beta, \tau^{-1}), i = 1, ..., n,$$
$$[\beta|\eta] \sim N(0, \eta^{-1}I),$$
$$\tau \sim Gamma(\frac{1}{100}, \frac{1}{100}),$$
$$\sigma_\beta \sim unif(0, A)$$

where $A = 1000$ and $\tau = \frac{1}{\sigma_\beta^2}$.

Usage

```
ridge_regression <- function(
  Y,
  X,
  niter = 10000)
```

Arguments

`Y`	n by 1
`X`	n by p predictor matrix
`niter`	number of gibbs sampling iterations

Details

This function returns the generated parameters from the gibbs sampling markov chain.

Value

`beta`	An niter by p matrix
`sigma`	An niter by 1 matrix
`sigma_beta`	An niter by 1 matrix

Examples

```
# Load the data
 prostate.data = "https://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.data"
prostate = read.table(file = "prostate.data", sep="", header = TRUE)
# Training data:
prostate_train = prostate[which(prostate$train),-10]
# Testing data:
prostate_test = prostate[which(!prostate$train),-10]
# Response:
y = prostate_train$lpsa
# Center and scale the data:
y = scale(y)
# And the predictors
X = scale(prostate_train[,names(prostate_train) != "lpsa"])

gibbs_ridge <- ridge_regression(y, X, niter=10000)
shrinkage_regression_plot(gibbs_ridge$beta, y, X, main = "Ridge Prior")
```

`rnorm_cork` *Sample from multivariate normal distribution with potential inequality constraints and potential equality constraints*

Description

Sample X from a multivariate normal distribution with mean μ, covariance matrix Σ, subject to the constraints that $M^t X \leq m$ and $A^t X \leq a$.

Usage

```
rnorm_cork = function(
  n,
  mean.vec, #   Sample X ~ normal with mean.vec, cov.mat subject
  cov.mat,  #   to the constraint that MX <= m, AX = a
  ineq.mat = NULL, #M
  ineq.vec = NULL, #m
  eq.mat = NULL,   #A
  eq.vec = NULL)   #a
```

Arguments

`n`	number of elements to generate
`mean.vec`	Mean before constraints
`cov.mat`	Covariance matrix before constrainnts
`ineq.mat`	Inequality matrix. If NULL, no inequality constraints are computed
`ineq.vec`	Inequality vector. If NULL, no inequality constraints are computed
`eq.mat`	Equality matrix. If NULL, no equality constraints are computed
`eq.vec`	Equality vector. If NULL, no equality constraints are computed

Details

See Schmidt, Mikkel. "Linearly constrained bayesian matrix factorization for blind source separation." Advances in neural information processing systems. 2009.

Examples

```
M = rbind(c(-1,1,0,0),c(0,0,-1,1))
xmin = 0
xmax = 1
ymin = 0
ymax = 1
m = c(-xmin, xmax, -ymin, ymax)
n = 5000

A = array(c(1,-1), dim = c(2,1))
a = 0

rho = -0.9
mean.vec = c(2,2)
cov.mat = cbind(c(1, rho), c(rho, 1))

par(mfrow = c(2,2))

# no constraints
rr = rnorm_cork(
  n,
  mean.vec,
  cov.mat)
```

```
plot(rr[,1], rr[,2])

# just inequality constraints
rr = rnorm_cork(
  n,
  mean.vec,
  cov.mat,
  ineq.mat = M,
  ineq.vec = m)
plot(rr[,1], rr[,2])

# just equality constraints
rr = rnorm_cork(
  n,
  mean.vec,
  cov.mat,
  eq.mat = A,
  eq.vec = a)
plot(rr[,1], rr[,2])

# both equality and inequality constraints
rr = rnorm_cork(
  n,
  mean.vec,
  cov.mat,
  ineq.mat = M,
  ineq.vec = m,
  eq.mat = A,
  eq.vec = a)
plot(rr[,1], rr[,2])

# Another
nN = 10
mean.vec = rep(0, nN)
cov.mat = diag(nN)

M = array(0, c(nN - 1, nN))
diag(M[,1:(nN-1)]) = 1
diag(M[,2:(nN-0)]) = -1
m = rep(0, nrow(M))

# set first value to zero and last value to one
A = array(0, c(2, nN))
A[1,1] = 1
A[2,nN] = 1
a = c(0, 1)

# case 0:
# simulate null, null
rho = 0
oo = rnorm_cork(100, c(1, 1), cbind(c(1, rho), c(rho, 1)))
plot(oo[,1], oo[,2])
```

```
# case 1:
# simulate with inequalities
oo = rnorm_cork(100, mean.vec, cov.mat, t(M), m)
stopifnot(all(sapply(2:nN, function(idx){ all(oo[,idx-1] < oo[,idx]) })))
xs = range(oo)
plot(xs, xs, type = "l")
points(oo[,nN-1], oo[,nN])

# case 2:
# simulate with equalities
oo = rnorm_cork(100, mean.vec, cov.mat, eq.mat = t(A), eq.vec = a)
stopifnot(all(oo[,1] == 0 & oo[,nN] == 1))
par(mfrow = c(2,2))
plot(oo[1,])
plot(oo[2,])
plot(oo[3,])
plot(oo[4,])

# case 3:
# simulate with inequalities and equalities
oo = rnorm_cork(100, mean.vec, cov.mat, t(M), m, t(A), a)
stopifnot(all(sapply(2:nN, function(idx){ all(oo[,idx-1] < oo[,idx]) })))
stopifnot(all(oo[,1] == 0 & oo[,nN] == 1))
par(mfrow = c(2,2))
plot(oo[1,])
plot(oo[2,])
plot(oo[3,])
plot(oo[4,])
```

rnorm_qinv_l *Sample from multivariate normal distribution*

Description

Sample from multivariate normal distribution with mean $Q^{-1}l$ and covariance matrix Q^{-1}.

Usage

```
rnorm_qinv_l <- function(
  n,
  Q,
  l,
  L)
```

Arguments

n	number of elements to generate
Q	p by p precision matrix.
l	p by 1 vector
L	NULL by default. If not null, Q is ignored and assumed to be LL^t.

Details

The algorithm is as follows

1. Cholesky decomposition of Q into LL^t. (This step is skipped if L is passed in).
2. Sample z from rnorm(p). Let $y = Lz + l$.
3. Solve for x in $LL^t x = y$ and return.

Value

`x`	An p x 1 vector if $n = 1$ otherwise a n by p matrix

`rnorm_qinv_l_chol` *Sample from multivariate normal distribution*

Description

Sample from multivariate normal distribution with mean $Q^{-1}l$ and covariance matrix Q^{-1} where $Q = LL^t$.

Usage

```
rnorm_qinv_l_chol <- function(
  n,
  L,
  l)
```

Arguments

`n`	number of elements to generate
`L`	NULL by default. If not null, Q is ignored and assumed to be LL^t.
`l`	p by 1 vector

Details

The algorithm is as follows

1. Cholesky decomposition of Q into LL^t. (This step is skipped if L is passed in).
2. Sample z from rnorm(p). Let $y = Lz + l$.
3. Solve for x in $LL^t x = y$ and return.

Value

`x`	An p x 1 vector if $n = 1$ otherwise a n by p matrix

rnorm_qinv_l_eigen — *Sample from multivariate normal distribution given symmetric eigen decomposition*

Description

Sample from multivariate normal distribution with mean $Q^{-1}l$ and covariance matrix Q^{-1} where $Q = UDU^t$

Usage

```
rnorm_qinv_l_eigen <- function(
  n,
  U,
  d,
  l)
```

Arguments

n	number of elements to generate
U	orthogonal matrix such that $Q = UDU^t$
d	p by 1 vector, $D = diag(d)$ and $Q = UDU^t$
l	p by 1 vector

Details

This function is useful to sample from $N((Q + sI)^{-1}l, (Q + sI)^{-1})$ given the eigen decomposition of Q

Value

x	An p x 1 vector if $n = 1$ otherwise a n by p matrix

Examples

```
W = cbind(c(10,1), c(1,10))
ee = eigen(W, symmetric = TRUE)
U = ee$vectors
d = ee$values
crossprod(U)

solve(W)
U

l = c(50,100)
m = U
ret = rnorm_qinv_l_eigen(50000, U, d, l)
mean(ret[,1] - m[1])
mean(ret[,2] - m[2])
sum(cov(ret) - (U
```

`rplackett_luce` *Generate random permutations from a plackett luce distribution*

Description

The plackett luce model assigns a probability distribution over permutations.

Usage

```
rplackett_luce <- function(n, vs)
```

Arguments

`n`	number of permutations to generate
`vs`	k vector of probabilities; need not be scaled to sum to 1

Details

This function returns a n x k matrix where each row is a permutation generated from a plackett luce distribution with parameters given by vs.

Examples

```
library(coda)

vals <- rplackett_luce(100, vs = c(100, 1, 1))
post_vals <- plackett_luce(vals)
plot(as.mcmc(post_vals))
```

`rpost_regression_coef` *Generate posterior for regression coefecients*

Description

Generate a $N(\mu, \Sigma)$ random variable where

$$\Sigma = (X^t X + D^{-1})^{-1},$$

$$\mu = \Sigma X^t \alpha.$$

The algorithm is $O(np^2)$; for large p it performs fast.

Usage

```
rpost_regression_coef <- function(
  X,
  D,
  alpha,
  u = NULL)
```

Arguments

X	n by p matrix
D	p by p matrix
alpha	n by 1 vector
u	Optional. If specified, don't generate $u \sim N(0, D)$.

Details

The algorithm is from

Bhattacharya, Anirban, Antik Chakraborty, and Bani K. Mallick. "Fast sampling with Gaussian scale mixture priors in high-dimensional regression." Biometrika (2016): asw042.

The algorithm is as follows:

1. Sample $u \sim N(0, D), d \sim N(0, I_n)$
2. Set $v = Xu + d$
3. Solve $(XDX^t + I_n)w = (\alpha - v)$
4. Return $\beta = u + DX^t w$

Value

beta	An p x 1 vector

---

rstnorm_ineq_cork — *Sample from multivariate standard normal distribution with inequality constraints*

Description

Sample from multivariate normal distribution with mean 0 and covariance matrix I_d with the constraint that $M^t z \leq m$ where M is given by ineq.mat and m is given by ineq.vec

Usage

```
rstnorm_ineq_cork = function(
  n,
  ineq.mat,
  ineq.vec,
  thin = 1,
  burn = 0,
  init = NULL)
```

Arguments

`n`	number of elements to generate
`ineq.mat`	Inequality matrix
`ineq.vec`	Inequality vector
`thin`	Take every thinth value from the gibbs sampler
`burn`	Discard the first burn elements generated
`init`	initial z value. If not given, an SVD of ineq.mat happens.

Details

See Schmidt, Mikkel. "Linearly constrained bayesian matrix factorization for blind source separation." Advances in neural information processing systems. 2009.

Examples

```
# sample on a cube!
M = rbind(c(-1,1,0,0),c(0,0,-1,1))
xmin = 1
xmax = 100
ymin = 1
ymax = 100
m = c(-xmin, xmax, -ymin, ymax)
n = 1000
rr = rstnorm_ineq_cork(n, M, m)
plot(rr[,1], rr[,2])

# sample on a triangle and have the third dimension truncated positive
M = cbind(c(-1, 0, 0), c(0, -1, 0), c(1, 1, 0), c(0, 0, -1))
m = c(0, 0, 3, 0)
n = 10000
rr = rstnorm_ineq_cork(n, M, m)
plot(rr[,1], rr[,2])
hist(rr[,3])
```

`shrinkage_regression_plot`
Plot the MC chain of regression coeffecients compared to OLS.

Description

Plot the MC chain of regression coeffecients comparet to OLS using 95% HPD confidence intervals.

Usage

```
horseshoe_regression_plot <- function(
  beta,
  Y,
  X,
  main = "Horseshoe Prior",
  ylim = NULL)
```

Arguments

`beta`	n x p MCMC chain of beta coeffecients.
`Y`	Y values used to generate beta; used as input to OLS regression
`X`	X values used to generate beta; used as input to OLS regression
`main`	Title of the plot
`ylim`	y range of plot

Examples

```
# Load the data
 prostate.data = "https://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.data"
prostate = read.table(file = "prostate.data", sep="", header = TRUE)
# Training data:
prostate_train = prostate[which(prostate$train),-10]
# Testing data:
prostate_test = prostate[which(!prostate$train),-10]
# Response:
y = prostate_train$lpsa
# Center and scale the data:
y = scale(y)
# And the predictors
X = scale(prostate_train[,names(prostate_train) != "lpsa"])

gibbs_hs <- horseshoe_regression(y, X, niter=10000)
shrinkage_regression_plot(gibbs_hs$beta, y, X)
```

`sumAlong` *sumAlong*

Description

Sum along a chosen dimension of an arbitrarily dimensioned array.

Usage

```
  sumAlong <- function(x, along)
```

Arguments

`x`	An n dimensional array
`along`	the dimension of x to sum along

Details

# The method of the source code came from Hadley Wickham: # https://stackoverflow.com/questions/14500707/select-along-one-of-n-dimensions-in-array

This function returns an n - 1 dimensional array that is the result of summing along the along dimension.

Examples

```
nx = 3
ny = 3
nz = 3
x = array(1:(nx*ny*nz), dim = c(nx, ny, nz))

print(x)
sumAlong(x, 3)
```

Index