Package ‘ptycho’

November 12, 2015

Type Package

Title Bayesian Variable Selection with Hierarchical Priors

Version 1.1-4

Author Laurel Stell and Chiara Sabatti

Maintainer Laurel Stell <lstell@stanford.edu>

URL web.stanford.edu/~lstell/ptycho/

Description Bayesian variable selection for linear regression models using hierarchical priors. There is a prior that combines information across responses and one that combines information across covariates, as well as a standard spike and slab prior for comparison. An MCMC samples from the marginal posterior distribution for the 0-1 variables indicating if each covariate belongs to the model for each response.

License GPL (>= 2)

Depends R (>= 3.0.0)

Imports coda, plyr, reshape2

Suggests foreach, doRNG

NeedsCompilation no

Repository CRAN

Date/Publication 2015-11-12 19:09:21

R topics documented:

`ptycho-package` *Bayesian Variable Selection with Hierarchical Priors*

Description

Bayesian variable selection for linear regression models using hierarchical priors. There is a prior that combines information across responses and one that combines information across covariates, as well as a standard spike and slab prior for comparison. An MCMC samples from the marginal posterior distribution for the 0-1 variables indicating if each covariate belongs to the model for each response.

Details

This package provides functions to carry out Bayesian model selection combining different layers of information: across multiple traits or across multiple variants in the same gene. The priors are described by Stell and Sabatti (2015). To sample the posterior distribution for specified genotype and phenotype matrices, use `ptycho`.

This package also provides functions to generate simulated data as in Stell and Sabatti (2015); see `createData` and `web.stanford.edu/~lstell/ptycho/`. Those datasets are not included in this package because they have images about 20 MB or larger. Instead small data objects are included for examples; see Data.

Functions for post-processing `ptycho` objects are described at `checkConvergence` and PosteriorStatistics.

Author(s)

Laurel Stell and Chiara Sabatti
Maintainer: Laurel Stell <lstell@stanford.edu>

References

Stell, L. and Sabatti, C. (2015) Genetic variant selection: learning across traits and sites, arXiv:1504.00946.

`checkConvergence` *Compute Differences Between MCMC Chains*

Description

Compute the differences between the chains in a `ptycho` object of the means of τ and the indicator variables.

Usage

```
checkConvergence(obj, doLastIterOnly=TRUE)
```

Arguments

`obj`	A `ptycho` object
`doLastIterOnly`	Logical specifying whether to compute differences only for the last MCMC iteration in the input object or for all iterations

Details

For τ and each indicator variable in the input `ptycho` object, compute the difference between the maximum and the minimum mean for each chain. If `doLastIterOnly` is `TRUE`, then the differences are only computed for the last iteration in each chain; otherwise, the differences are computed at each iteration in the input object.

Value

A data frame with the following columns:

`iter` MCMC iteration number

`type` Factor specifying the type of the variable; one of “tau”, “var” for variant indicator variable, or “grp” for second-level indicator variable

`index` Number specifying the pertinent column in the design matrix (for `type` equal to “var” or for `type` equal to “grp” when *Across Traits* prior was used) or the variant group index (for `type` equal to “grp” when *Across Sites* prior was used); equal to 1 for `type` equal to “tau”

`y` Factor specifying the name of the response; empty for `type` equal to “tau” or for `type` equal to “grp” when *Across Traits* prior was used

`range` Difference between maximum and minimum across chains

Author(s)

Laurel Stell and Chiara Sabatti
Maintainer: Laurel Stell <lstell@stanford.edu>

See Also

`ptycho`; also `ptychoOut` for example below

Examples

```
data(ptychoOut)
cvg <- checkConvergence(ptychoOut, doLastIterOnly=FALSE)
reshape2::dcast(cvg, ... ~ iter, value.var="range")
```

`createData` *Simulate Data*

Description

Create a data object suitable for `ptycho.all`.

Usage

```
createData(X, y, omega = NULL, beta = NULL)
createDataBayesModel(mode = c("exchange","pleiotropy","gene"), n, p, q,
                       nreps, tau.min, tau.max, G)
createPubData(mode = c("tinysim","ptychoIn",
                         "exchange","pleiotropy","gene",
                         "actualGeno","actualPheno","corTest",
                         "fixedOmega","uniformEffects"),
                X=NULL, y=NULL, var.detail=NULL, variants=NULL)
```

Arguments

`X`	Design matrix or alist specifying how to generate such a matrix. If a list, the first entry is a function name and the second is a list of arguments to the function. In `createPubData`, `X` is ignored unless `mode` is “actualGeno”, “actualPheno”, or `corTest`.
`y`	Numeric vector or matrix or list with the following components: `nreps` Number of replicates to simulate `q` Number of responses to generate for each replicate `sd` Standard deviation of the simulated noise In `createPubData`, `y` is ignored unless `mode` is “actualPheno”.
`omega`	Numeric vector or matrix or list specifying how to generate a list with the component `omega`; see Details for its meaning. If this is a list, the first entry is a function name and the second is a list of arguments to the function, which will be prepended by the number of rows in output `X` and the number of columns in output `y`. Only used if `y` is a list.
`beta`	List specifying how to generate a matrix of effect sizes. The first entry of the list is a function name and the second is a list of arguments to the function, which will be prepended by a matrix specifying the variables selected and `y$sd`. Only used if `y` is a list.
`n`	Number of observations to simulate
`p`	Number of covariates to simulate
`q`	Number of responses to simulate for each replicate
`nreps`	Number of replicates to simulate
`mode`	String specifying type of dataset to create:

`tinysim` Simulated data included with this package; equivalent to mode `pleiotropy` except that the dataset is tiny, with `n=100`, `p=10`, `q=5`, and `nreps=10`

`ptychoIn` Simulated data included with this package; equivalent to mode `gene` except that the dataset is tiny, with `n=3000`, `p=10`, `q=1`, and `nreps=1`

`exchange` Create orthogonal `X` and exchangeable variants; `n=5000`, `p=50`, `q=5`, and `nreps=100`

`pleiotropy` Create orthogonal `X`, and several variants have nonzero effects on multiple responses; `n=5000`, `p=50`, `q=5`, and `nreps=100`

`gene` Create orthogonal `X`, and each group of variants typically has either several or no variants that effect a response; `n=5000`, `p=50`, `q=5`, and `nreps=100`

`actualGeno` Simulate responses for input `X`

`corTest` Simulate `q=2` responses for input `X`. There will be 10 replicates with the first variant in argument `variants` causal for both responses, 10 with the second variant causal, and 20 with variant `i` causal for response `i`. No other variant will be causal.

`actualPheno` Put input `X` and `y` into data object

`fixedOmega` Create orthogonal `X`, and each variant has a certain probability of a nonzero effect size

`uniformEffects` Same as mode `fixedOmega` except that effect sizes are uniformly rather than normally distributed

For `createDataBayesModel`, `mode` must be one of “exchange”, “pleiotropy”, or “gene”.

`tau.min, tau.max`	Endpoints of uniform distribution from which to draw `tau`
`G`	Number of groups of covariates; unused if `mode` is not “gene”
`var.detail`	Data frame with row names same as column names of `X`; must have columns “MAF” and “GENE”. Ignored unless `mode` is “actualGeno”.
`variants`	Character vector containing names of two columns of `X`; ignored unless `mode` is “corTest”.

Details

We describe `createData` and then describe its wrappers `createDataBayesModel` and `createPubData`.

Although `createData` can form the data object required by `ptycho.all` when `X` and `y` are input, it primarily exists to simplify simulating data from $Y = X\beta + \epsilon$, where ϵ is normal with mean zero and specified standard deviation and β is sparse with entries simulated as specified.

The function generates a specified number of replicates, all of which use the same design matrix X. If this matrix is not input, then its argument must specify a function call to generate it. In either case, suppose X has n rows and p columns.

If the input `y` is numeric, then it will be used for the lone replicate. If it is a matrix, it must have n rows; let q be its number of columns. If input `y` is a numeric vector, it must have n entries and will be cast as a matrix with $q = 1$ column. Otherwise, input `y` is a list specifying, along with the arguments `omega` and `beta`, how to simulate the response(s). Because it is useful in analysis of the estimation of the marginal posterior distribution, the returned object always contains, regardless of how `X` and `y` are specified, a matrix `eta2` with (j, k) entry equal to $\mathbf{x}_j^T\mathbf{y}_k/(n\mathbf{y}_k^T\mathbf{y}_k)$

If `y` is to be simulated, the first step is to choose the probability that each covariate is associated with each reponse as specified by the input argument `omega`. If this argument is a matrix, it must have size p-by-q. If it is not a matrix but is numeric, it will be passed to `matrix` to create a matrix of the correct size. Otherwise, the matrix for each replicate will be generated by calling the function whose name is given by `omega[[1]]` with argument list `(p, q, omega[[2]])`. This function must return a list with component `omega` set to a p-by-q matrix; the list may also contain additional components. The package contains several functions whose names start with "createOmega" that might guide users in writing their own functions.

The next step is to draw a p-by-q matrix `indic.var` whose (j, k) entry is equal to one with probability `omega[j,k]` and zero otherwise. This matrix will be drawn until all column sums are positive.

For each entry in `indic.var` that is equal to one, the effect size must be drawn. This is done by calling the function whose name is given by `beta[[1]]` with argument list `(indic.var, y$sd, beta[[2]])`. This function must return a list with component `beta` set to a p-by-q matrix; the list may also contain additional components. If `indic.var[j,k]` is zero, then `beta[j,k]` should be zero. The package contains functions whose names start with "createBeta" that might guide users in writing their own functions.

Finally, an n-by-q matrix of noise is drawn from $N(0, \sigma^2)$, where σ is the input `noise.sd`, and added to $X\beta$ to obtain `y`. The column names of each response matrix generated will be `y1`, `y2`, and so forth.

The function `createPubData` generates the data sets used in Stell and Sabatti (2015). For `mode` equal to "exchange", "pleiotropy", or "geno", it calls `createData` via `createDataBayesModel`; otherwise, it calls `createData` directly. These functions also serve as additional examples of the use of `createData`. For reproducibility, `createPubData` first sets the random seed to 1234, except that it is set to 4 when `mode` equals "ptycholn" and it does not set it when `mode` equals "corTest".

In `createDataBayesModel`, if `mode` is "exchange", then one $\omega \sim \text{Beta}(12, 48)$ is drawn independently for each trait. If `mode` is "pleiotropy", then one probability of association for a trait is drawn from Beta(16,55) for each data set, that probability is used to draw `indic.grp` for each variant, and then the probability of nonzero `indic.var[j,k]` is drawn from Beta(48,12) for each nonzero `indic.grp[j]`. Finally, if `mode` is "gene", the process is analogous to pleiotropy except that each trait is simulated independently.

Value

List containing:

`X`	Design matrix
`q`	Number of columns in each response
`noise.sd`	Standard deviation of the simulated noise; `NULL` if input `y` is numeric
`omega`	Input `omega`
`beta`	Input `beta`
`replicates`	List of length `y$nreps` (length 1 if `y` is numeric), each entry of which is a list with the following components:

`omega` Matrix containing probabilities of association between covariates and responses; row names are `colnames(X)` and column names are `colnames(y)`; `NULL` if input `y` is numeric `indic.var` Matrix containing ones for associations and zeros otherwise; row and column names are same as for `omega`; `NULL` if input `y` is numeric

`beta` Matrix of effect sizes; row and column names are same as for `omega`; `NULL` if input `y` is numeric

`y` Response matrix

`eta2` Matrix with row names equal to `colnames(X)` and column names equal to `colnames(y)`

For `createDataBayesModel` with `mode` that uses a second level of indicator variables, each entry in the `replicate` list also has components `omega.grp` and `indic.grp` containing the intermediate steps of drawing the second-level indicator variable before drawing `omega`. If the argument `beta` to `createData` is "createBetaNormal" (which it is when called by `createDataBayesModel`), then each replicate will also have a component `tau` giving the value drawn by a call to `runif(1, tau.min, tau.max)`.

Author(s)

Laurel Stell and Chiara Sabatti
Maintainer: Laurel Stell <lstell@stanford.edu>

References

Stell, L. and Sabatti, C. (2015) Genetic variant selection: learning across traits and sites, arXiv:1504.00946.

See Also

`createOrthogonalX`, `createGroupsSim`; also Data describes `tinysim` in example below as well as another object output by `createData`

Examples

```
### EXAMPLE 1
data(tinysim)
# Data generated with mode equal to pleiotropy, so indic.grp exists and
# has an entry for each column in X.
colnames(tinysim$X)
tinysim$replicates[[5]]$indic.grp
# X4, X6, and X9 are associated with some responses.
tinysim$replicates[[5]]$indic.var

### EXAMPLE 2
# Generate miniature data set with information shared across covariates.
set.seed(1234)
tiny1 <- createDataBayesModel(mode="gene", n=100, p=10, q=5, nreps=10,
                              tau.min=0.045, tau.max=0.063, G=2)
# A covariate can only have indic.var=1 if the group it belongs to has
# indic.grp=1.  For example,indic.grp[1,4]=0 implies
# indic.var[groups$group2var[1],4]=0.
tiny1$replicates[[1]]$indic.grp
tiny1$omega[[2]]$groups$group2var[1]
```

```
tiny1$replicates[[1]]$indic.var

### EXAMPLE 3
# Alternatively, call createData directly
groups <- createGroupsSim(G=2, p=10)
omegaargs <- list(indic.grp.shape1=16, indic.grp.shape2=55,
                  shape1=48, shape2=12, groups=groups)
betaargs <- list(tau.min=0.045, tau.max=0.063)
set.seed(1234)
tiny2 <- createData(X=list("createOrthogonalX", list(n=100, p=10)),
                    y=list(nreps=10, q=5, sd=1),
                    omega=list("createOmegaCrossVars", omegaargs),
                    beta=list("createBetaNormal", betaargs))
identical(tiny1, tiny2)
### SEE THE CODE FOR createPubData FOR MORE EXAMPLES.
```

`createGroupsSim` *Create Groups of Covariates*

Description

Create an object specifying groups of covariates as needed for some of the simulations.

Usage

```
createGroupsSim(G, p)
```

Arguments

`G`	Number of groups
`p`	Number of covariates

Details

If `G` divides `p`, then each group will have `p/G` consecutive covariates. If `G` does not divide `p`, then the last group will have fewer covariates.

Value

List containing the following components:

`var2group` Integer vector of length `p`, with entry j being the index of the group containing covariate j

`group2var` List of length `G`, each entry of which is an integer vector containing the indices of the covariates belonging to that group

`sizes` Vector of length `G` containing the number of covariates in each group

Author(s)

Laurel Stell and Chiara Sabatti
Maintainer: Laurel Stell <lstell@stanford.edu>

See Also

`createData`

Examples

```
grp <- createGroupsSim(G=3, p=15)
# Which covariates are in group 2?  Two ways to find out:
which(grp$var2group == 2)
grp$group2var[[2]]
```

`createOrthogonalX` *Create Design Matrix With Orthogonal Columns*

Description

Create a design matrix whose columns are orthogonal to each other.

Usage

```
createOrthogonalX(n, p)
```

Arguments

`n`	Number of rows in X
`p`	Number of columns in X

Details

First create $\hat{X} = (I_p \;\; I_p \;\; \cdots \;\; I_p)^T$, where I_p, the identity matrix of size p, is repeated `ceiling(n/p)` times. If `p` does not divide `n`, remove rows at the bottom so that $\hat{X}$ has `n` rows. Divide by the root mean square of the columns of $\hat{X}$.

Value

Matrix with `n` rows, `p` columns, and column names `X1`, `X2`, and so forth.

Author(s)

Laurel Stell and Chiara Sabatti
Maintainer: Laurel Stell <lstell@stanford.edu>

See Also

createData

Examples

```
n <- 50; p <- 5
X <- createOrthogonalX(n, p)
XtX <- t(X) %*% X
D <- diag(n-1, nrow=p)
# XtX and D are not quite equal due to roundoff error
range(XtX - D)
```

Data *Sample Data*

Description

Data objects used in examples in this package.

Usage

```
data(tinysim)
data(ptychoIn)
data(ptychoOut)
```

Format

The object `tinysim` is an object returned by `createDataBayesModel` with `mode` equal to “pleiotropy”.

The object `ptychoIn` is an object returned by `createDataBayesModel` with `mode` equal to “gene”.

The object `ptychoOut` is an object returned by `ptycho` applied to the data in `ptychoIn` using the *Across Sites* prior.

Details

These data objects are constructed to illustrate certain features in the examples while still being small enough not to be burdensome.

The object `tinysim` contains simulated data. Its design matrix is 100-by-10. It has 10 replicates, each with a 100-by-5 response matrix. It is generated by `createPubData`.

The object `ptychoIn` also contains simulated data generated by `createPubData`. Its design matrix is 3000-by-10 because, for its effect sizes, `n` must be about that large to distinguish signal from noise as explained in the supplemental text to Stell and Sabatti (2015). To keep the object small, it has only 1 replicate, which has only one response.

The object `ptychoOut` is generated by

```
G <- 2; p <- ncol(ptychoIn$X)
groups <- createGroupsSim(G, p)
state <- list(list(indic.grp=rep(FALSE,G),
                   indic.var=matrix(FALSE,nrow=p,ncol=1), tau=1),
              list(indic.grp=rep(TRUE,G),
                   indic.var=matrix(TRUE,nrow=p,ncol=1), tau=1))
ptychoOut <- ptycho(X=ptychoIn$X, y=ptychoIn$replicates[[1]]$y,
                    groups=groups, initStates=state,
                    only.means=10000*seq_len(5), random.seed=12345)
```

Source

Stell, L. and Sabatti, C. (2015) Genetic variant selection: learning across traits and sites, arXiv:1504.00946.

See Also

createPubData, ptycho

PosteriorStatistics *Extract Posterior Statistics*

Description

Extract posterior statistics from a ptycho object.

Usage

```
meanTau(obj)
varTau(obj)
meanIndicators(obj)
meanVarIndicators(obj)
meanGrpIndicators(obj)
```

Arguments

obj	A ptycho object

Details

A ptycho object contains means for many different variables. If multiple chains were run, it has separate means for each, and it may have running means from different points within each chain. The functions described here simplify extracting from the input object certain statistics of the posterior distribution sampled by ptycho.

The function meanTau identifies the last iteration saved in the input ptycho object and computes the mean of τ at that iteration across all chains. The function varTau is analogous, computing var(τ).

Similarly, meanIndicators returns the mean across all chains of each indicator variable. The functions meanVarIndicators and meanGrpIndicators compute the means only of the indicators of variants or only of second-level indicator variables, respectively.

Value

Both `meanTau` and `varTau` return a scalar.

The other functions, which extract means of indicator variables, return vectors with names copied from the column names of the input `obj`.

Author(s)

Laurel Stell and Chiara Sabatti
Maintainer: Laurel Stell <lstell@stanford.edu>

See Also

`ptycho`, WhichCols; also Data describes `ptychoIn` and `ptychoOut` in example below

Examples

```
data(ptychoIn)
data(ptychoOut)
# Compare averages of sampled group indicator variables to truth.
cbind(ptychoIn$replicates[[1]]$indic.grp,
      meanGrpIndicators(ptychoOut))
# Compare averages of sampled covariate indicator variables to truth.
cbind(ptychoIn$replicates[[1]]$indic.var,
      meanVarIndicators(ptychoOut))
# Compare average of sampled values of tau to truth.
ptychoIn$replicates[[1]]$tau
meanTau(ptychoOut)
# Variance of sampled values of tau is reasonable because sampled model
# is usually NOT empty.
varTau(ptychoOut)
```

`print.ptycho` *Print ptycho Object*

Description

Print the sample means in a `ptycho` object; do not print the attributes.

Usage

```
## S3 method for class 'ptycho'
print(x, ...)
```

Arguments

`x`	Object of class `ptycho`
`...`	Additional print arguments

Author(s)

Laurel Stell and Chiara Sabatti
Maintainer: Laurel Stell <lstell@stanford.edu>

See Also

`ptycho`

`ptycho` *Sample From Posterior Distributions*

Description

Generate MCMC samples from posterior distribution. Two interfaces are provided: `ptycho` generates samples for one design matrix and response matrix while `ptycho.all` runs in batch an object generated by `createData`.

Usage

```
ptycho(X, y, initStates, groups = NULL,
       tau.min = 0.01, tau.max = 10, tau.sd = (tau.max - tau.min)/4,
       doGPrior = TRUE, doDetPrior = FALSE, prob.varadd = 0.5,
       isOmegaFixed = FALSE, omega = NULL, omega.grp = NULL,
       probs.grp = NULL, rho.alpha = 10, rho.lambda = rho.alpha,
       only.means = FALSE, nburn = 0, nthin = 1, nSavePerChain,
       parallel.chains=FALSE, random.seed=NULL)
ptycho.all(data, across=c("none","traits","sites"), doGrpIndicator,
           dir.out, nreplicates=NULL, parallel.replicates=FALSE,
           doSetSeed=TRUE, ncolumns=NULL, ...)
```

Arguments

`X`	n-by-p design matrix
`y`	n-by-q matrix containing response(s)
`initStates`	List containing initial states for chains. Each state is a list with components: `indic.var` p-by-q logical matrix. If (j, k) entry is `TRUE`, then covariate j is initially in the model for response k. `tau` Scalar `indic.grp` Logical vector of length equal to the number of groups; analogous to `indic.var`; `NULL` to use priors that do not incorporate a second-level indicator variable
`groups`	To combine information across variants, list containing `var2group` Integer vector of length p, with entry j being the index of the group containing covariate j

	`group2var` List of length G, each entry of which is an integer vector containing the indices of the covariates belonging to that group `sizes` Vector of length G containing the number of covariates in each group Otherwise, `NULL`.
`tau.min, tau.max`	Endpoints of uniform prior distribution on `tau`
`tau.sd`	Standard deviation of the Metropolis-Hastings proposal distribution for `tau`
`doGPrior`	Logical indicating whether to use the g-prior for effect sizes
`doDetPrior`	Unsupported; use default value
`prob.varadd`	If `initStates[[1]]$indic.grp` is `NULL`, the probability that the Metropolis-Hastings proposal changes one entry of `indic.var` from `FALSE` to `TRUE`. Otherwise, the probability of this event given that the proposal does not change `indic.grp`.
`isOmegaFixed`	Logical indicating whether `omega` is known
`omega`	If `isOmegaFixed` is `TRUE`, a p-by-q matrix containing the known probabilities. Otherwise, a matrix containing the parameters for the Beta prior distribution on `omega`. Such a matrix has columns “A” and “B”; the number of rows should be: • 1 if $q = 1$ and `initStates[[1]]$indic.grp` is `NULL`, • `length(groups$group2var)` if that is nonzero, or • p otherwise. If `omega` is `NULL` and `isOmegaFixed` is `FALSE`, defaults to uniform priors.
`omega.grp`	If `isOmegaFixed` is `TRUE`, the known probability that entries in `indic.grp` are `TRUE`. Otherwise, a vector with names “A” and “B” containing the parameters for the Beta prior distribution on `omega.grp`. If `NULL`, defaults to uniform priors. Unused if `initStates[[1]]$indic.grp` is `NULL`.
`probs.grp`	Vector containing the probabilities that the Metropolis-Hastings proposal will add, leave unchanged, or remove, respectively, a group. If `NULL`, defaults to `c(0.25,0.5,0.25)`. Unused if `initStates[[1]]$indic.grp` is `NULL`.
`rho.alpha, rho.lambda`	Parameters for the Gamma prior distribution on ρ, which is the precision of the noise. Here, the Gamma(α, λ) distribution has density function proportional to $x^\alpha e^{-\lambda x}$.
`only.means`	If logical, specifies whether to return samples or the running means of the samples. Can also be a vector containing the iterations (after the burn-in interval) at which to save the means.
`nburn`	Number of MCMC samples to make before starting to save samples or to compute means
`nthin`	Interval between saved samples; default value 1 saves all samples. Unused if `only.means` is `TRUE` or a vector.
`nSavePerChain`	If `only.means` is `FALSE`, number of MCMC samples to return from each chain, which means a total of `nthin * nSavePerChain + nburn` samples are drawn per chain. If `only.means` is `TRUE`, then `nSavePerChain + nburn` samples are drawn, and only the averages of the last `nSavePerChain` samples are returned. Unused if `only.means` is not a logical.

`parallel.chains`	Logical indicating whether to run chains in parallel; see Details.
`random.seed`	Random seed to pass to chain iterator; if `NULL`, the random seed is not set. See Details.
`data`	Data in format output by `createData`
`across`	Whether to combine information across traits, sites, or neither
`doGrpIndicator`	Whether to use priors that incorporate `indic.grp`
`dir.out`	Directory to which to `save` samples or means
`nreplicates`	Vector of replicates to run; if `NULL`, all will be run
`parallel.replicates`	Logical indicating whether to run replicates in parallel; see Details.
`doSetSeed`	If `TRUE`, call `set.seed(n.repl)` before running samples.
`ncolumns`	Scalar. If `across` is "none" or "sites", each of the first `ncolumns` of `repl$y` will be used in turn, running all columns by default. Ignored if `across` is "sites".
`...`	Additional arguments passed to `ptycho`

Details

These functions run MCMC sampling from the posterior of the linear regression models using hierarchical priors described in Stell and Sabatti (2015). The function `ptycho.all` is a wrapper of `ptycho` to simplify running the simulation experiments over many replicates. These functions determine which priors to use as follows:

- Standard spike and slab priors that do not combine information (basic)
  For `ptycho.all`, argument `across` is "none" and `doGrpIndicator` is `FALSE`.
  For `ptycho`, argument `y` has one column, `groups` is `NULL`, and `indic.grp` is `NULL` or missing in each entry of `initStates`.
- Combine information across traits (*Across Traits*)
  For `ptycho.all`, argument `across` is "traits" and `doGrpIndicator` is `TRUE`.
  For `ptycho`, argument `y` has $p > 1$ columns, `groups` is `NULL`, and `indic.grp` is a logical vector of length p in each entry of `initStates`.
- Combine information across variants (*Across Sites*)
  For `ptycho.all`, argument `across` is "sites" and `doGrpIndicator` is `TRUE`.
  For `ptycho`, argument `y` has one column, `groups` specifies how to combine information, and `indic.grp` in each entry of `initStates` is a logical vector of the same length as `groups$group2var`.
- Combine information across traits *incorrectly* (*Unadjusted*)
  For `ptycho.all`, argument `across` is "traits" and `doGrpIndicator` is `FALSE`.
  For `ptycho`, argument `y` has $p > 1$ columns, `groups` is `NULL`, and `indic.grp` is `NULL` in each entry of `initStates`.
  This prior does not properly correct for multiple hypothesis testing and is only included because it is needed to reproduce results in Stell and Sabatti (2015).

Combining information across both phenotypes and variants is planned for a future release. These functions perform some checks for compatibility of `X`, `y`, `groups`, and `initStates`; but invalid input could lead to unpredictable behavior. Singular X can result in an error; even strongly correlated covariates can cause difficulties as described by Stell and Sabatti (2015).

The simplest way to run the simulations in Stell and Sabatti (2015) is, for example,

```
data <- createPubData("pleiotropy")
ptycho.all(data=data, across="traits", doGrpIndicator=TRUE,
           dir.out="/path/to/output/dir/",
           only.means=50000*(1:10), nburn=10000)
ptycho.all(data=data, across="sites", doGrpIndicator=TRUE,
           dir.out="/path/to/another/dir/",
           groups=createGroupsSim(G=10, ncol(data$X)),
           only.means=50000*(1:10), nburn=10000)
```

With these calls, the replicates run sequentially and so do the chains of the MCMC sampler; the results will be reproducible because the random seed is set for each replicate.

Parallelization is implemented via the **foreach** package. The user must not only have it installed but also an appropriate parallel backend, which must be registered. To run chains in parallel using the **doMC**, for example,

```
data(ptychoIn)
G <- 2; p <- ncol(ptychoIn$X)
groups <- createGroupsSim(G, p)
state <- list(list(indic.grp=rep(FALSE,G),
                   indic.var=matrix(FALSE,nrow=p,ncol=1), tau=1),
              list(indic.grp=rep(TRUE,G),
                   indic.var=matrix(TRUE,nrow=p,ncol=1), tau=1))
require(doMC)
registerDoMC(length(state))
ptychoOut <- ptycho(X=ptychoIn$X, y=ptychoIn$replicates[[1]]$y,
                    groups=groups, initStates=state,
                    only.means=100*seq_len(5), parallel.chains=TRUE)
```

The results would not be reproducible, however, even if one set the random seed before calling `ptycho`. For reproducible results, pass the random seed in the call to `ptycho`, which requires that the **doRNG** package is also installed. Running the chains in parallel when calling `ptycho.all` also requires the option `parallel.chains=TRUE`, which uses **doRNG** unless `doSetSeed=FALSE`. By default, one of the chains starts with all variants in the model, so that chain takes much longer to run than do the other chains. Consequently, when running multiple replicates via `ptycho.all`, much greater time savings can be achieved by running the replicates in parallel with, for example,

```
data <- createPubData("pleiotropy")
require(doMC)
registerDoMC(8)
ptycho.all(data=data, across="traits", doGrpIndicator=TRUE,
           dir.out="/path/to/output/dir/",
           only.means=50000*(1:10), nburn=10000)
```

In this case, the default behavior of reproducible results does *not* require **doRNG** because the seed is set after each parallel worker is created.

We conclude this description with a discussion of the running time of the MCMC sampler. Our actual data has 5335 subjects, 764 variants and three traits. An `mcmc.list` containing 50,000 samples for each of four chains can take about 5~GB. Running chains in parallel, it takes less than an hour (on a Linux computer with 2.6 GHz processors) to perform 510,000 samples per chain. The run time depends primarily on the number of entries that are TRUE in the sampled indic.var matrices; increasing this will increase run times. A chain that initially has all entries of indic.var set to TRUE will take longer than one where the model is initially empty. Priors that inflate the posterior expectation of indic.var[j,k] (such as combining information across responses without using indic.grp) will also take longer.

Value

The results of ptycho.all are written to files by save. For priors that use only one response, the output for replicate r and column c will be written to 'rpl<r>col<c>.Rdata' in the directory specified by dir.out. For priors that use multiple responses, ptycho is called only once for each replicate, and the file name will be 'rpl<r>col1.Rdata'. The object in each such file has the name smpl and is the value of a call to ptycho. The format of these objects depends upon the argument only.means. In all cases, however, it has attribute params set to a list containing most of the arguments in the call to ptycho.

If only.means is FALSE, then ptycho returns an mcmc.list whose length is the same as the length of initStates. Each entry in this list is an mcmc object with nSavePerChain rows and a column for each entry of indic.var and indic.grp plus a column for tau.

Otherwise, ptycho returns an object of class ptycho, which is actually a matrix. The matrix has a column for each sampled indicator variable, for tau and its square (so that its variance can be computed), and for the chain and iteration numbers. If only.means is TRUE, then each row contains the means of the samples in one chain and there will be length(initStates) * nSavePerChain rows. If only.means is a vector, then there will be length(initStates) * length(only.means) rows.

Author(s)

Laurel Stell and Chiara Sabatti
Maintainer: Laurel Stell <lstell@stanford.edu>

References

Stell, L. and Sabatti, C. (2015) Genetic variant selection: learning across traits and sites, arXiv:1504.00946.

See Also

createData for simulating input data.

checkConvergence and PosteriorStatistics for analyzing output of ptycho.

Data describes tinysim in example below as well as an object created with ptycho.

Examples

```
data(tinysim)
# Use replicate 4.
X <- tinysim$X; p <- ncol(X); nr <- 4
# COMBINE INFORMATION ACROSS RESPONSES
Y <- tinysim$replicates[[nr]]$y; q <- ncol(Y)
# Run 2 chains.
```

```
state <- list(list(indic.grp=rep(FALSE,p),
                   indic.var=matrix(FALSE,nrow=p,ncol=q), tau=1),
              list(indic.grp=rep(TRUE,p),
                   indic.var=matrix(TRUE,nrow=p,ncol=q), tau=1))
# In each chain, discard first 10 burn-in samples, then generate
# 100 samples and save running means after every 20 samples.
smpl.ph <- ptycho(X=X, y=Y, initStates=state, only.means=20*(1:5),
                  nburn=10)
# COMBINE INFORMATION ACROSS VARIANTS
# Use two groups of variants.
G <- 2; groups <- createGroupsSim(G, p)
# Run 2 chains.
state <- list(list(indic.grp=rep(FALSE,G),
                   indic.var=matrix(FALSE,nrow=p,ncol=1), tau=1),
              list(indic.grp=rep(TRUE,G),
                   indic.var=matrix(TRUE,nrow=p,ncol=1), tau=1))
# Use response 3.
y <- tinysim$replicates[[nr]]$y[,3,drop=FALSE]
smpl.var <- ptycho(X=X, y=y, groups=groups, initStates=state,
                   only.means=c(20*(1:5)), nburn=10, nthin=1)
```

WhichCols *Identify Columns Containing Indicator Variables*

Description

Determine which columns contain `indic.var` or `indic.grp` in an object returned by `ptycho`.

Usage

```
indicVarCols(obj)
indicGrpCols(obj)
```

Arguments

`obj`	Object output by `ptycho` or any numeric object

Value

If the input object is a numeric vector, returns the indices of its entries that have names starting with "indic.var" or "indic.grp", respectively.

Otherwise, it returns the indices of `colnames(obj)` that start with "indic.var" or "indic.grp", respectively.

Author(s)

Laurel Stell and Chiara Sabatti
Maintainer: Laurel Stell <lstell@stanford.edu>

See Also

ptycho; also ptychoOut and PosteriorStatistics for example below

Examples

```
data(ptychoOut)
colnames(ptychoOut)[indicVarCols(ptychoOut)]
# Can also apply these functions to output of meanIndicators ...
mi <- meanIndicators(ptychoOut)
mi[indicGrpCols(mi)]
# ... instead of using meanGrpIndicators or meanVarIndicators
meanGrpIndicators(ptychoOut)
```

Index