Package 'cpgen'

September 15, 2015

Type Package

Title Parallelized Genomic Prediction and GWAS

Version 0.1

Date 2015-09-14

Author Claas Heuer

Maintainer Claas Heuer <cheuer@tierzucht.uni-kiel.de>

Description Frequently used methods in genomic applications with emphasis on parallel computing (OpenMP).
At its core, the package has a Gibbs Sampler that allows running univariate linear mixed models that have both, sparse and dense design matrices. The parallel sampling method in case of dense design matrices (e.g. Genotypes) allows running Ridge Regression or BayesA for a very large number of individuals. The Gibbs Sampler is capable of running Single Step Genomic Prediction models.
In addition, the package offers parallelized functions for common tasks like genome-wide association studies and cross validation in a memory efficient way.

License GPL (>= 2)

SystemRequirements C++11

Imports methods, stats

URL https://github.com/cheuerde/cpgen

Depends R(>= 3.1.0), Matrix(>= 1.0-5), pedigreemm(>= 0.3-3)

LinkingTo Rcpp, RcppEigen, RcppProgress

NeedsCompilation yes

Repository CRAN

Date/Publication 2015-09-15 08:35:30

R topics documented:

`cpgen-package` *cpgen - Parallel genomic evaluations*

Description

The package offers a variety of functions that are frequently being used in genomic prediction and genomewide association studies. The package is based on Rcpp and RcppEigen, hence all routines are implemented using the matrix algebra library Eigen. The main emphasis of the package lies in parallel computing which is realized by C++ functions making use of OpenMP.

Details

Package:	cpgen
Type:	Package
Version:	0.1
Date:	2014-07-10
License:	License: GPL (>= 2)

Author(s)

Claas Heuer

Maintainer: Claas Heuer <cheuer@tierzucht.uni-kiel.de>

References

Guennebaud, G., Jacob, B., et al.: "Eigen v3". http://eigen.tuxfamily.org (2010)

Dirk Eddelbuettel and Romain Francois (2011). "Rcpp: Seamless R and C++ Integration". Journal of Statistical Software, 40(8), 1-18. URL http://www.jstatsoft.org/v40/i08/.

Douglas Bates, Dirk Eddelbuettel (2013). "Fast and Elegant Numerical Linear Algebra Using the RcppEigen Package". Journal of Statistical Software, 52(5), 1-24. URL http://www.jstatsoft.org/v52/i05/.

`ccolmv` *Colwise means or variances*

Description

Computes the colwise means or variances of a matrix - internal use

Usage

```
ccolmv(X,compute_var=FALSE)
```

Arguments

`X`	matrix of type: `matrix` or `dgCMatrix`
`compute_var`	boolean, defines whether the colwise variances rather than the means will be returned

Value

Numeric Vector of colwise means or variances of X

Examples

```
X <- matrix(rnorm(1000*500),1000,500)
means <- ccolmv(X)
vars <- ccolmv(X,compute_var=TRUE)
```

ccov *ccov*

Description

Computation of covariance- or correlation-matrix. Shrinkage estimate through the use of 'lambda'. Weights for observations can be passed.

Usage

```
ccov(X,lambda=0, w=NULL, compute_cor=FALSE)
```

Arguments

X	matrix
lambda	numeric scalar, shrinkage parameter
w	numeric vector of weights with same lengths as rows in X
compute_cor	boolean - defines whether the functions returns a correlation- rather than a covariance matrix

Value

Covariance matrix with dimension ncol(X)

Examples

```
## Not run:
# generate random data
rand_data(500,5000)

# compute correlation matrix of t(M)
corM <- ccov(t(M),compute_cor=T)

## End(Not run)
```

ccross *ccross*

Description

Computation of the following matrix-product: $\mathbf{XDX}'$ Where $\mathbf{D}$ is a diagonal matrix, which is being passed to the function as a vector.

Usage

```
ccross(X,D=NULL)
```

Arguments

X	matrix
D	numeric vector, will be used as a weighting diagonal matrix of dimension ncol(X). If omitted an identity matrix will be assigned.

Value

Square matrix of dimension nrow(X)

Examples

```
# Computing the matrix-square-root of a positive definite square matrix:
## Not run:
# generate random data
rand_data(500,5000)

W <- ccross(M)

# this is the implementation of the matrix power-operator '%**%'
W_sqrt <- with(eigen(W), ccross(vectors,values**0.5))

## End(Not run)
```

---

cCV *Generate phenotype vectors for cross validation*

Description

This function takes a phenotype vector and generates `folds * reps` masked vectors for cross validation. Every vector has as many additional missing values as `length(y) / folds`.

Usage

```
cCV(y,folds=5,reps=1,matrix=FALSE,seed=NULL)
```

Arguments

y	vector of phenotypes - may already contain missing values
folds	integer, number of folds
reps	integer, number of replications
matrix	boolean, if TRUE function returns a matrix rather than a list
seed	numeric scalar, seed for `sample`

Value

List (matrix) with as many items (columns) as `folds * reps`

See Also

clmm, get_pred, get_cor

Examples

```
## Not run:
# generate random data
rand_data(500,5000)

y_CV <- cCV(y,folds=5,reps=20)

## End(Not run)
```

---

cGBLUP *Genomic BLUP*

Description

This function allows fitting a mixed model with one random effect besides the residual using clmm. The random effect $\mathbf{a}$ follows some covariance-structure $\mathbf{G}$

Usage

```
cGBLUP(y,G,X=NULL, scale_a = 0, df_a = -2, scale_e = 0, df_e = -2,
          niter = 10000, burnin = 5000, seed = NULL, verbose=TRUE)
```

Arguments

y	vector of phenotypes
G	Relationship matrix / covariance structure for random effects
X	Optional Design Matrix for fixed effects. If omitted a column-vector of ones will be assigned
scale_a	prior scale parameter for a
df_a	prior degrees of freedom for a
scale_e	prior scale parameter for e
df_e	prior degrees of freedom for e
niter	Number of iterations
burnin	Burnin
seed	Seed
verbose	Prints progress to the screen

Details

Kang et al. (2008):

$$\mathbf{y} = \mathbf{Xb} + \mathbf{a} + \mathbf{e} \text{ with: } \mathbf{a} \sim MVN(\mathbf{0}, \mathbf{G}\sigma_a^2)$$

By finding the decomposition: $\mathbf{G} = \mathbf{UDU}'$ and premultiplying the model equation by $\mathbf{U}'$ we get:

$$\mathbf{U}'\mathbf{y} = \mathbf{U}'\mathbf{Xb} + \mathbf{U}'\mathbf{a} + \mathbf{U}'\mathbf{e}$$

with:

$$Var(\mathbf{U}'\mathbf{y}) = \mathbf{U}'\mathbf{G}'\mathbf{U}\sigma_a^2 + \mathbf{U}'\mathbf{U}\sigma_e^2$$

$$\mathbf{U}'\mathbf{UDU}'\mathbf{U}\sigma_a^2 + \mathbf{I}\sigma_e^2$$

$$\mathbf{D}\sigma_a^2 + \mathbf{I}\sigma_e^2$$

After diagonalization of the variance-covariance structure the transformed model is being fitted by passing $\mathbf{D}^{1/2}$ as the design matrix for the random effects to `clmm`. The results are subsequently backtransformed and returned by the function.

Value

List of 6:

`var_e`	Posterior mean of the residual variance
`var_a`	Posterior mean of the random-effect variance
`b`	Posterior means of the fixed effects
`a`	Posterior means of the random effects
`posterior_var_e`	Posterior of the residual variance
`posterior_var_u`	Posterior of the random variance

Author(s)

Claas Heuer

References

Kang, H. M., N. A. Zaitlen, C. M. Wade, A. Kirby, D. Heckerman, M. J. Daly, and E. Eskin. "Efficient Control of Population Structure in Model Organism Association Mapping." Genetics 178, no. 3 (February 1, 2008): 1709-23. doi:10.1534/genetics.107.080101.

See Also

`clmm`, `cgrm`, `cGWAS.emmax`

Examples

```
## Not run:
# generate random data
rand_data(500,5000)

# compute a genomic relationship-matrix
G <- cgrm(M,lambda=0.01)

# run model
mod <- cGBLUP(y,G)

## End(Not run)
```

---

cgrm *Genomic Relationship Matrices*

Description

Based on a coefficient-matrix (i.e. marker matrix) $\mathbf{X}$ that will be scaled column-wise, a weight-vector $\mathbf{w}$ and a shrinkage parameter λ, `cgrm` returns the following similarity matrix:

$$\mathbf{G} = (1-\lambda)\frac{\mathbf{XDX}^{'}}{\sum \mathbf{w}} + \mathbf{I}\lambda$$

where $\mathbf{D} = diag(\mathbf{w})$. A weighted genomic relationship matrix allows running TA-BLUP as described in Zhang et al. (2010).

Usage

```
cgrm(X, w = NULL, lambda=0)
```

Arguments

X	coefficient matrix
w	numeric vector of weights for every column in X
lambda	numeric scalar, shrinkage parameter

Details

...

Value

Similarity matrix with dimension nrow(X)

Author(s)

Claas Heuer

References

de los Campos, G., Vazquez, A.I., Fernando, R., Klimentidis, Y.C., Sorensen, D., 2013. "Prediction of Complex Human Traits Using the Genomic Best Linear Unbiased Predictor". PLoS Genetics 9, e1003608. doi:10.1371/journal.pgen.1003608

Zhang Z, Liu J, Ding X, Bijma P, de Koning D-J, et al. (2010) "Best Linear Unbiased Prediction of Genomic Breeding Values Using a Trait-Specific Marker-Derived Relationship Matrix". PLoS ONE 5(9): e12648. doi:10.1371/journal.pone.0012648

See Also

cgrm.A, cgrm.D.

Examples

```
## Not run:
# generate random data
rand_data(500,5000)

weights <- (cor(M,y)**2)[,1]

G <- cgrm(M,weights,lambda=0.01)

## End(Not run)
```

---

cgrm.A *Additive Genomic Relationship Matrix*

Description

Based on a marker matrix $\mathbf{X}$ with {-1,0,1} - coding that will be centered column-wise and a shrinkage parameter λ, cgrm.A returns the following additive genomic relationship matrix according to VanRaden (2008):

$$\mathbf{G} = (1-\lambda)\frac{\mathbf{X}\mathbf{X}^{'}}{\sum_{i=1}^{n} 2p_i q_i} + \mathbf{I}\lambda$$

Usage

```
cgrm.A(X, lambda=0, yang=FALSE)
```

Arguments

X	marker matrix
lambda	numeric scalar, shrinkage parameter
yang	boolean, diagonal elements of A according to Yang et al. (2010)

Details

...

Value

Additive genomic relationship matrix with dimension nrow(X)

Author(s)

Claas Heuer

References

VanRaden, P.M. "Efficient Methods to Compute Genomic Predictions". Journal of Dairy Science 91, no. 11 (November 2008): 4414-23. doi:10.3168/jds.2007-0980.

Yang, Jian, Beben Benyamin, Brian P McEvoy, Scott Gordon, Anjali K Henders, Dale R Nyholt, Pamela A Madden, et al. "Common SNPs Explain a Large Proportion of the Heritability for Human Height". Nature Genetics 42, no. 7 (July 2010): 565-69. doi:10.1038/ng.608.

See Also

cgrm, cgrm.D

Examples

```
## Not run:
# generate random data
rand_data(500,5000)

### compute the additive genomic relationship matrix
A <- cgrm.A(M,lambda=0.01)

## End(Not run)
```

---

cgrm.D *Dominance Genomic Relationship Matrix*

Description

Based on a marker matrix $\mathbf{X}$ with {-1,0,1} - out of which a column-wise centered dominance coefficient matrix will be constructed and a shrinkage parameter λ, cgrm.D returns the following dominance genomic relationship matrix according to Su et al. (2012):

$$\mathbf{G} = (1-\lambda)\frac{\mathbf{X}\mathbf{X}^{'}}{\sum_{i=1}^{n} 2p_i q_i (1-2p_i q_i)} + \mathbf{I}\lambda$$

The additive marker coefficients will be used to compute dominance coefficients as: 1-abs(X)

Usage

```
cgrm.D(X, lambda=0)
```

Arguments

X	marker matrix
lambda	numeric scalar, shrinkage parameter

Details

...

Value

Dominance relationship matrix with dimension nrow(X)

Author(s)

Claas Heuer

References

Su G, Christensen OF, Ostersen T, Henryon M, Lund MS (2012) "Estimating Additive and Non-Additive Genetic Variances and Predicting Genetic Merits Using Genome-Wide Dense Single Nucleotide Polymorphism Markers". PLoS ONE 7(9): e45293. doi:10.1371/journal.pone.0045293

See Also

cgrm, cgrm.A.

Examples

```
## Not run:
# generate random data
rand_data(500,5000)

D <- cgrm.D(M,lambda=0.01)

## End(Not run)
```

cGWAS *Genomewide Association Study*

Description

This function runs GWAS for continuous traits. Population structure that can lead to false positive association signals can be accounted for by passing a Variance-covariance matrix of the phenotype vector (Kang et al., 2010). The GLS-solution for fixed effects is computed as:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}'\mathbf{V}^{-1}\mathbf{X})^{-1}\mathbf{X}'\mathbf{V}^{-1}\mathbf{y}$$

Equivalent solutions are obtained by premultiplying the design matrix $\mathbf{X}$ for fixed effects and the phenotype vector $\mathbf{y}$ by $\mathbf{V}^{-1/2}$:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^{*\prime}\mathbf{X}^{*})^{-1}\mathbf{X}^{*\prime}\mathbf{y}^{*}$$

with

$$\mathbf{X}^{*} = \mathbf{V}^{-1/2}\mathbf{X}$$

$$\mathbf{y}^{*} = \mathbf{V}^{-1/2}\mathbf{y}$$

Usage

```
cGWAS(y,M,X=NULL,V=NULL,dom=FALSE, verbose=TRUE)
```

Arguments

y	vector of phenotypes
M	Marker matrix
X	Optional Design Matrix for additional fixed effects. If omitted a column-vector of ones will be assigned
V	Inverse square root of the Variance-covariance matrix for the phenotype vector of type: `matrix` or `dgCMatrix`. Used for computing the GLS-solution of fixed effects. If omitted an identity-matrix will be assigned
dom	Defines whether to include an additional dominance coefficient for every marker. Note: only useful if the genotype-coding in `M` follows {-1,0,1} The dominance coefficient is computed as: `1-abs(M)`
verbose	prints progress to the screen

Details

...

Value

List of 3 vectors or matrices. If `dom=TRUE` every element of the list will be a matrix with two columns. First column additive, second dominance:

`p-value`	Vector of p-values for every marker
`beta`	GLS solution for fixed marker effects
`se`	Standard Errors for values in `beta`

Author(s)

Claas Heuer

References

Kang, Hyun Min, Jae Hoon Sul, Susan K Service, Noah A Zaitlen, Sit-yee Kong, Nelson B Freimer, Chiara Sabatti, and Eleazar Eskin. "Variance Component Model to Account for Sample Structure in Genome-Wide Association Studies." Nature Genetics 42, no. 4 (April 2010): 348-54. doi:10.1038/ng.548.

See Also

`cGWAS.emmax`

Examples

```
## Not run:
# generate random data
rand_data(500,5000)


### GWAS without accounting for population structure
mod <- cGWAS(y,M)

### GWAS - accounting for population structure
## Estimate variance covariance matrix of y

G <- cgrm.A(M,lambda=0.01)

fit <- cGBLUP(y,G,verbose=FALSE)

### construct V
V <- G*fit$var_a + diag(length(y))*fit$var_e

### get the inverse square root of V
V2inv <- V %**% -0.5

### run GWAS again
mod2 <- cGWAS(y,M,V=V2inv,verbose=TRUE)

## End(Not run)
```

cGWAS.emmax *Genomewide Association Study - EMMAX*

Description

This is a convenience function that uses the function cGWAS but estimates the variance-covariance matrix of the phenotype vector in advance using clmm. This method was termed EMMAX (Kang et al., 2010).

Usage

```
cGWAS.emmax(y,M,A=NULL,X=NULL,dom=FALSE,verbose=TRUE,scale_a = 0, df_a = -2,
   scale_e = 0, df_e = -2,niter=15000,burnin=7500,seed=NULL)
```

Arguments

y	vector of phenotypes
M	Marker matrix
A	Relationship matrix that is being used to estimate V - if omitted, A will be constructed using M and cgrm
X	Optional Design Matrix for additional fixed effects. If omitted a column-vector of ones will be assigned
dom	Defines whether to include an additional dominance coefficient for every marker. Note: only useful if the genotype-coding in M follows {-1,0,1} The dominance coefficient is computed as: 1-abs(M)
verbose	Prints progress to the screen
scale_a	prior scale parameter for a
df_a	prior degrees of freedom for a
scale_e	prior scale parameter for e
df_e	prior degrees of freedom for e
niter	Number of iterations used by clmm
burnin	Burnin for clmm
seed	Seed used by clmm

Details

...

Value

List of 3 vectors or matrices. If `dom=TRUE` every element of the list will be a matrix with two columns. First column additive, second dominance:

`p-value`	Vector of p-values for every marker
`beta`	GLS solution for fixed marker effects
`se`	Standard Errors for values in `beta`
`marker_variance`	Estimate of the marker variance reported by `clmm`
`residual_variance`	Estimate of the residual variance reported by `clmm`

Author(s)

Claas Heuer

References

Kang, H. M., N. A. Zaitlen, C. M. Wade, A. Kirby, D. Heckerman, M. J. Daly, and E. Eskin. "Efficient Control of Population Structure in Model Organism Association Mapping." Genetics 178, no. 3 (February 1, 2008): 1709-23. doi:10.1534/genetics.107.080101.

Kang, Hyun Min, Jae Hoon Sul, Susan K Service, Noah A Zaitlen, Sit-yee Kong, Nelson B Freimer, Chiara Sabatti, and Eleazar Eskin. "Variance Component Model to Account for Sample Structure in Genome-Wide Association Studies." Nature Genetics 42, no. 4 (April 2010): 348-54. doi:10.1038/ng.548.

See Also

`cGWAS`

Examples

```
## Not run:
# generate random data
rand_data(500,5000)

# run EMMAX
res <- cGWAS.emmax(y,M,verbose=TRUE)

## End(Not run)
```

check_openmp *Check OpenMP-support.*

Description

Checks whether the C++ binaries have been compiled with OpenMP-support.

Usage

```
check_openmp()
```

Value

Returns a message telling you whether OpenMP is available for the cpgen-functions or not.

See Also

set_num_threads, get_num_threads, get_max_threads

Examples

```
# check whether openmp is available or not
check_openmp()
```

clmm *Linear Mixed Models using Gibbs Sampling*

Description

This function runs linear mixed models of the following form:

$$\mathbf{y} = \mathbf{Xb} + \mathbf{Z}_1\mathbf{u}_1 + \mathbf{Z}_2\mathbf{u}_2 + \mathbf{Z}_3\mathbf{u}_3 + ... + \mathbf{Z}_k\mathbf{u}_k + \mathbf{e}$$

The function allows to include an arbitrary number of independent random effects with each of them being assumed to follow: $MVN(\mathbf{0}, \mathbf{I}\sigma^2_{u_k})$. If the covariance structure of one random effect is assumed to follow some $\mathbf{G}$ then the design matrix for that random effect can be constructed as described in Waldmann et al. (2008): $\mathbf{F} = \mathbf{ZG}^{1/2}$. Alternatively, the argument ginverse can be passed.

Usage

```
clmm(y, X = NULL, Z = NULL, ginverse = NULL, par_random = NULL,
niter=10000, burnin=5000, scale_e=0, df_e=-2, beta_posterior = FALSE,
verbose = TRUE, timings = FALSE, seed = NULL, use_BLAS=FALSE)
```

Arguments

y	vector or list of phenotypes
X	fixed effects design matrix of type: matrix or dgCMatrix. If omitted a column-vector of ones will be assigned
Z	list of design matrices for random effects - every element of the list represents one random effect and may be of type: matrix or dgCMatrix
ginverse	list of inverse covariance matrices for random effects in Z (e.g. Inverse of numerator relationship matrix). Every element of the list represents one random effect and may be of type: matrix or dgCMatrix. Note: If passed, ginverse must have as many items as Z. For no ginverse assign NULL for a particular random effect.
par_random	list of options for random effects. If passed, the list must have as many elements as random. Every element may be a list of 4: • scale - (vector of) scale parameters for the inverse chi-square prior • df - (vector of) degrees of freedom for the inverse chi-square prior • method - method to be used for the random effects, may be: ridge or BayesA • name - name for that effect • GWAS - list of options for conducting GWAS using window variance proportions (Fernando et al, 2013): – window_size - number of markers used to form a single window – threshold - window porportion of total variance, used to determine presents of association
niter	number of iterations
burnin	number of iterations to be discarded as burnin
verbose	prints progress to the screen
beta_posterior	save all posterior samples of regression coefficients
timings	prints time per iteration to the screen - sets verbose = FALSE
scale_e	scale parameter for the inverse chi-square prior for the residuals
df_e	degrees of freedom for the inverse chi-square prior for the residuals
seed	seed for the random number generator. If omitted, a seed will be generated based on machine and time
use_BLAS	use BLAS library instead of Eigen

Details

Single Model run

At this point the function allows to specify the method for any random term as: 'ridge' or 'BayesA'. In Ridge Regression the prior distribution of the random effects is assumed to be normal with a constant variance component, while in BayesA the marginal prior is a t-distribution, resulting from locus specific variances with inverse chi-square priors (Gianola et al., 2009). A wider range of methods is available in the excellent BGLR-package, which also allows phenotypes to be discrete (de los Campos et al. 2013).

The focus of this function is to allow solving high-dimensional problems that are mixtures of sparse and dense features in the design matrices. The computational expensive parts of the Gibbs Sampler are parallelized as described in Fernando et al. (2014). Note that the parallel performance highly depends on the number of observations and features present in the design matrices. It is highly recommended to set the number of threads for less than 10000 observations (length of phenotype vector) to 1 using: `set_num_threads(1)` before running a model. Even for larger sample sizes the parallel performance still depends on the dimension of the feature matrices. Good results in terms of parallel scaling were observed starting from 50000 observations and more than 10000 features (i.e. number of markers). Single threaded performance is very good thanks to smart computations during gibbs sampling (Fernando, 2013 (personal communication), de los Campos et al., 2009) and the use of efficient Eigen-methods for dense and sparse algebra.

Parallel Model runs

In the case of multiple phenotypes passed to the function as a list, the main advantage of the function is that several threads can access the very same data once assigned, which means that the design matrices only have to be allocated once. The parallel scaling of this function using multiple phenotypes is almost linear.

In C++:

For every element of the phenotype list a new instance of an MCMC-object will be created. All the memory allocation needed for running the model is done by the major thread. The function then iterates over all objects and runs the gibbs sampler. This step is parallelized, which means that as many models are being run at the same time as threads available. All MCMC-objects are totally independent from each other, they only share the same design-matrices. Every object has its own random-number generator with its own seed which allows perfectly reproducible results.

GWAS using genomic windows

The function allows to specify options to any random effect for conducting genomewide association studies using prediction vector variances of marker windows as described in Fernando et al. (2013). In every effective sample of the Gibbs Sampler the sampling variance of the genotypic value vector $\mathbf{g} = \mathbf{Zu}$ of the particular random effect is computed as: $\tilde{\sigma}^2_g = \left(\sum_{j=1}^{n}(g_j - \mu_g)^2\right)(n-1)^{-1}$, with μ_g being the mean of $\mathbf{g}$ and n the number of observations. Then for any window w the sampling variance of $\mathbf{g_w} = \mathbf{Z}_w\mathbf{u}_w$ is obtained as: $\tilde{\sigma}^2_{g_w} = \left(\sum_{j=1}^{n}(g_{w_j} - \mu_{g_w})^2\right)(n-1)^{-1}$, where w indicates the range over the columns of $\mathbf{Z}$ that forms the window w. The posterior probability that a window exceeds a specified proportion δ of the total variance is estimated by the number of samples in which $\frac{\tilde{\sigma}^2_{g_w}}{\tilde{\sigma}^2_g} > \delta$ divided by the total number of samples. It can be shown that among marker windows that have a posterior probability p or greater for having a variance greater than δ of the total variance, the proportion of false positive signals (PFP) are expected to be lower than $1-p$ (Fernando et al. 2004, Fernando et al., 2013).

Value

List of 4 + number of random effects:

`Residual_Variance`

List of 4:

- `Posterior_Mean` - Mean estimate of the residual variance
- `Posterior` - posterior samples of residual variance
- `scale_prior` - scale parameter that has been assigned

- df_prior - degrees of freedom that have been assigned

Predicted — numeric vector of predicted values

fixed_effects — List of 4:

- type - dense or sparse design matrix
- method - method that has been used = "fixed"
- posterior - list of 1 + 1 (if beta_posterior=TRUE)
  - estimates_mean - mean solutions for random effects
  - estimates - posterior samples of random effects

Susequently as many additional items as random effects of the following form

Effect_k — List of 4 + 1 (if GWAS options were specified):

- type - dense or sparse design matrix
- method - method that has been used
- scale_prior - scale parameter that has been assigned
- df_prior - degrees of freedom that have been assigned
- posterior - list of 3 + 1 (if beta_posterior=TRUE)
  - estimates_mean - mean solutions for random effects
  - variance_mean - mean variance
  - variance - posterior samples of variance
  - estimates - posterior samples of random effects
- GWAS - list of 9 (if specified)
  - window_size - number of features (markers) used to form a single window
  - threshold - window porportion of total variance, used to determine presents of association
  - mean_variance - mean variance of prediction vector using all windows
  - windows - identifier
  - start - starting column for window
  - end - ending column for window
  - window_variance - mean variance of prediction vector using this window
  - window_variance_proportion - mean window proportion of total variance
  - prob_window_var_bigger_threshold - mean probability that window variance exceeds threshold

mcmc — List of 4 + 1 (if timings=TRUE):

- niter - number of iterations
- burnin - number of samples discarded as burnin
- number_of_samples - number of samples used to estimate posterior means
- seed - seed used for the random number generator
- time_per_iter - average seconds per iteration

Author(s)

Claas Heuer

Credits: Xiaochen Sun (Iowa State University, Ames) gave strong assistance in the theoretical parts and contributed in the very first implementation of the Gibbs Sampler. Essential parts were adopted from the BayesC-implementation of Rohan Fernando and the BLR-package of Gustavo de los Campos. The idea of how to parallelize the single site Gibbs Sampler came from Rohan Fernando (2013).

References

Gianola, D., de Los Campos, G., Hill, W.G., Manfredi, E., Fernando, R.: "Additive genetic variability and the bayesian alphabet." Genetics 183(1), 347-363 (2009)

de los Campos, G., H. Naya, D. Gianola, J. Crossa, A. Legarra, E. Manfredi, K. Weigel, and J. M. Cotes. "Predicting Quantitative Traits With Regression Models for Dense Molecular Markers and Pedigree." Genetics 182, no. 1 (May 1, 2009): 375-85. doi:10.1534/genetics.109.101501.

Waldmann, Patrik, Jon Hallander, Fabian Hoti, and Mikko J. Sillanpaa. "Efficient Markov Chain Monte Carlo Implementation of Bayesian Analysis of Additive and Dominance Genetic Variances in Noninbred Pedigrees." Genetics 179, no. 2 (June 1, 2008): 1101-12. doi:10.1534/genetics.107.084160.

Meuwissen, T., B. J. Hayes, and M. E. Goddard. "Prediction of Total Genetic Value Using Genome-Wide Dense Marker Maps." Genetics 157, no. 4 (2001): 1819-29.

de los Campos, Gustavo, Paulino Perez Rodriguez, and Maintainer Paulino Perez Rodriguez. "Package 'BGLR,'" 2013. ftp://128.31.0.28/pub/CRAN/web/packages/BGLR/BGLR.pdf.

Fernando, R.L., Dekkers, J.C., Garrick, D.J. "A class of bayesian methods to combine large numbers of genotyped and non-genotyped animals for whole-genome analyses." Genetics Selection Evolution 46(1), 50 (2014)

Fernando, R., Nettleton, D., Southey, B., Dekkers, J., Rothschild, M., Soller, M. "Controlling the proportion of false positives in multiple dependent tests." Genetics 166(1), 611-619 (2004)

Fernando, Rohan L., and Dorian Garrick. "Bayesian methods applied to GWAS." Genome-Wide Association Studies and Genomic Prediction. Humana Press, 2013. 237-274.

See Also

cGBLUP, cSSBR, cGWAS.emmax

Examples

```
## Not run:

############################################################
### Running a model with an additive and dominance effect ###
############################################################

# generate random data
rand_data(500,5000)

### compute the relationship matrices
```

```
G.A <- cgrm.A(M,lambda=0.01)
G.D <- cgrm.D(M,lambda=0.01)

### generate the list of design matrices for clmm
Z_list = list(t(chol(G.A)),t(chol(G.D)))

### specify options
par_random = list(list(method="ridge",scale=var(y)/2 ,df=5, name="add"),
  list(method="ridge",scale=var(y)/10,df=5, name="dom"))

### run

set_num_threads(1)
fit <- clmm(y = y, Z = Z_list, par_random=par_random, niter=5000, burnin=2500)

### inspect results
str(fit)


########################
### Cross Validation ###
########################

### 4-fold cross-validation with one repetition:
# generate random data
rand_data(500,5000)

### compute the list of masked phenotype-vectors for CV
y_CV <- cCV(y, fold=4, reps=1)


### Cross Validation using GBLUP
G.A <- cgrm.A(M,lambda=0.01)


### generate the list of design matrices for clmm
Z_list = list(t(chol(G.A)))

### specify options
h2 = 0.3
scale = unlist(lapply(y_CV,function(x)var(x,na.rm=T))) * h2
df = rep(5,length(y_CV))
par_random = list(list(method="ridge",scale=scale,df=df, name="animal"))

### run model with 4 threads
set_num_threads(4)
fit <- clmm(y = y_CV, Z = Z_list, par_random=par_random, niter=5000, burnin=2500)

### inspect results
str(fit)

### obtain predictions
pred <- get_pred(fit)
```

```
### prediction accuracy
get_cor(pred,y_CV,y)

#########################################################
### GWAS using Bayesian Regression on marker windows ###
#########################################################

# generate random data
rand_data(500,5000)

### generate the list of design matrices for clmm
Z_list = list(M)

### specify options
h2 = 0.3
scale = var(y) * h2
df = 5
# specifying the model
# Here we use ridge regression on the marker covariates
# and define a window size of 100 and a threshold of 0.01
# which defines the proportion of genetic variance accounted
# for by a single window
par_random = list(list(method="ridge",scale=scale,df=df,
             GWAS=list(window_size=100, threshold=0.01), name="markers"))

### run
set_num_threads(1)
fit <- clmm(y =y, Z = Z_list, par_random=par_random, niter=2000, burnin=1000)

### inspect results
str(fit)

### extract GWAS part
gwas <- fit$markers$GWAS

# plot window variance proportions
plot(gwas$window_variance_proportion)

############################################################
### Sparse Animal Model using the pedigreemm milk data ###
############################################################

# cpgen offers two ways of running models with random effects that
# are assumed to follow some covariance structure.
# 1) Construct the Covariance matrix and pass the cholesky of that
#    as design matrix for that random effect
# 2) Construct the inverse of the covariance matrix (ginverse) and pass the design
#    matrix 'Z' that relates observations to factors in ginverse in conjunction
#    with the symmetric ginverse.

# This is approach 2) which is more convenient for pedigree based
# animal models, as ginverse (Inverse of numerator relationship matrix) is
```

```
# very sparse and can be obtained very efficiently

set_num_threads(1)

# load the data
data(milk)

# get Ainverse
# Ainv <- as(getAInv(pedCows),"dgCMatrix")

T_Inv <- as(pedCows, "sparseMatrix")
D_Inv <- Diagonal(x=1/Dmat(pedCows))
Ainv<-t(T_Inv)
dimnames(Ainv)[[1]]<-dimnames(Ainv)[[2]] <-pedCows@label
Ainv <- as(Ainv, "dgCMatrix")

# We need to construct the design matrix.
# Therefore we create a second id column with factor levels
# equal to the animals in the pedigree
milk$id2 <- factor(as.character(milk$id), levels = pedCows@label)

# set up the design matrix
Z <- sparse.model.matrix(~ -1 + id2, data = milk, drop.unused.levels=FALSE)

# run the model
niter = 5000
burnin = 2500

modAinv <- clmm(y = as.numeric(milk$milk), Z = list(Z), ginverse = list(Ainv),
                niter = niter, burnin = burnin)

# This is approach 1) run an equivalent model using the cholesky of A

# get L from A = LL'
L <- as(t(relfactor(pedCows)),"dgCMatrix")

# match with ids
ZL <- L[match(milk$id, pedCows@label),]

# run the model
modL <- clmm(as.numeric(milk$milk), Z= list(ZL),
             niter = niter, burnin = burnin)


### a more advanced model

# y = Xb + Zu + a + e
#
# u = permanent environment of animal
# a = additive genetic effect of animal

Zpe <- sparse.model.matrix(~ -1 + id2, drop.unused.levels = TRUE, data = milk)
```

```
# make X and account for lactation and herd
X <- sparse.model.matrix(~ 1 + lact + herd, data = milk)

niter = 10000
burnin = 2500

mod2 <- clmm(as.numeric(milk$milk), X = X, Z = list(Zpe,Z), ginverse = list(NULL, Ainv),
                        niter = niter, burnin = burnin)


# run all phenotypes in the milk dataset at once in parallel

Y <- list(as.numeric(milk$milk),as.numeric(milk$fat),as.numeric(milk$prot),as.numeric(milk$scs))

set_num_threads(4)

# ginverse version
model <- clmm(Y, X = X, Z = list(Zpe,Z), ginverse = list(NULL, Ainv),
              niter = niter, burnin = burnin)

# get heritabilities and repeatabilities with their standard deviations

heritabilities <- array(0, dim=c(length(Y),2))
colnames(heritabilities) <- c("h2","sd")

# only use post-burnin samples
range <- (burnin+1):niter

# h2
heritabilities[,1] <- unlist(lapply(model, function(x)
                                    mean(
                                         x$Effect_2$posterior$variance[range] /
                                         (x$Effect_1$posterior$variance[range] +
                                         x$Effect_2$posterior$variance[range] +
                                         x$Residual_Variance$Posterior[range]))
                                        )
                             )

# standard deviation of h2
heritabilities[,2] <- unlist(lapply(model, function(x)
                                    sd(
                                       x$Effect_2$posterior$variance[range] /
                                       (x$Effect_1$posterior$variance[range] +
                                       x$Effect_2$posterior$variance[range] +
                                       x$Residual_Variance$Posterior[range]))
                                      )
                             )


## End(Not run)
```

cmaf *cmaf*

Description

Computes the minor allele frequencies of a marker-matrix.

Usage

```
cmaf(X)
```

Arguments

X	Marker matrix with {-1,0,1} coding

Value

Numeric Vector of minor allele frequencies for every column in X

Examples

```
# generate random data
rand_data(500,5000)

# compute minor allele frequencies
mafs <- cmaf(M)
```

cscale_inplace *cscale_inplace*

Description

Center (and scale) a matrix 'inplace'. The function is meant for big matrices that shall be scaled inplace, hence without creating a copy

Usage

```
cscale_inplace(X, means = NULL, vars = NULL, scale=FALSE)
```

Arguments

X	numeric matrix
means	numeric vector, if ommitted will be computed using codeccolmv
vars	numeric vector, if ommitted will be computed using ccolmv
scale	boolean - scale the matrix

Value

nothing, function works 'inplace'

Examples

```
## Not run:
# generate random data
rand_data(500,5000)

# scale matrix
cscale_inplace(M,scale=TRUE)

## End(Not run)
```

---

cscanx *Read in a matrix from a file*

Description

Reads in a matrix from file (no header, no row-names, no NA's, space or tab-delimiter) and returns the according R-matrix. No Need to specify dimensions.

Usage

```
cscanx(path)
```

Arguments

path	character - location of the file to be read ("/path/to/file")

Value

Matrix shaped as in the file

Examples

```
# random matrix
X <- matrix(rnorm(10,5),10,5)

# write that matrix to a file
write.table(X,file="X",col.names=FALSE,row.names=FALSE,quote=FALSE)

# read in the matrix to object Z
Z <- cscanx("X")
```

csolve *csolve*

Description

This is a wrapper for the Cholesky-solvers 'LLT' (dense case) or 'Simplicial-LLT' (sparse case) from Eigen. The function computes the solution:

$$\mathbf{b} = \mathbf{X}^{-1}\mathbf{y}$$

If no vector y is passed, an identity matrix will be assigned and the function returns the inverse of $\mathbf{X}$. In the case of multiple right hand sides (as is the case when computing an inverse matrix) multiple threads will solve equal parts of it.

Usage

```
csolve(X,y=NULL)
```

Arguments

X	positive definite square matrix of type matrix or dgCMatrix
y	numeric vector of length equal to columns/rows of X

Value

Solution vector/matrix

Examples

```
# Least Squares Solving

# Generate random data

n = 1000
p = 500

M <- matrix(rnorm(n*p),n,p)
y <- rnorm(n)

# least squares solution:

b <- csolve(t(M) %c% M, t(M) %c% y)
```

cSSBR *Single Step Bayesian Regression*

Description

This function runs Single Step Bayesian Regression (SSBR) for the prediction of breeding values in a unified model that incorporates genotyped and non genotyped individuals (Fernando et al., 2014).

Usage

```
cSSBR(data, M, M.id, X=NULL, par_random=NULL, scale_e=0, df_e=0,
      niter=5000, burnin=2500, seed=NULL, verbose=TRUE)
```

Arguments

data	data.frame with four columns: id, sire, dam, y
M	Marker Matrix for genotyped individuals
M.id	Vector of length nrow(M) representing rownames for M
X	Fixed effects design matrix of type: matrix or dgCMatrix. If omitted a column-vector of ones will be assigned. Must have as many rows as data
par_random	as in clmm
niter	as in clmm
burnin	as in clmm
verbose	as in clmm
scale_e	as in clmm
df_e	as in clmm
seed	as in clmm

Details

The function sets up the following model using cSSBR.setup:

$$\mathbf{y} = \mathbf{Xb} + \mathbf{M}\alpha + \mathbf{Z}\epsilon + \mathbf{e}$$

The matrix $\mathbf{M}$ denotes a combined marker matrix consisting of actual and imputed marker covariates. Best linear predictions of gene content (Gengler et al., 2007) for the non-genotyped individuals are obtained using: $\mathbf{A}^{11}\hat{\mathbf{M}}_1 = -\mathbf{A}^{12}\mathbf{M}_2$ (Fernando et al., 2014). $\mathbf{A}^{11}$ and $\mathbf{A}^{12}$ are submatrices of the inverse of the numerator relationship matrix, which is easily obtained (Henderson, 1976). The subscripts 1 and 2 denote non genotyped and genotyped individuals respectively. The very sparse equation system is being solved using a sparse cholesky solver provided by the Eigen library. The residual imputation error has variance: $(\mathbf{A}^{11})^{-1}\sigma^2_\epsilon$.

Value

List of 4 + number of random effects as in clmm +

SSBR List of 7:

- ids - ids used in the model (ordered as in other model terms)
- y - phenotype vector
- X - Design matrix for fixed effects
- Marker_Matrix - Combined Marker Matrix including imputed and genotyped individuals
- Z_residual - Design Matrix used to model the residual error for the imputed individuals
- ginverse_residual - Submatrix of the inverse of the numerator relationship matrix. Used to model the residual error for the imputed individuals
- Breeding_Values - Predicted Breeding Values for all animals in data that have genotypes and/or phenotypes

Author(s)

Claas Heuer

References

Fernando, R.L., Dekkers, J.C., Garrick, D.J.: A class of bayesian methods to combine large numbers of genotyped and non-genotyped animals for whole-genome analyses. Genetics Selection Evolution 46(1), 50 (2014)

Gengler, N., Mayeres, P., Szydlowski, M.: A simple method to approximate gene content in large pedigree populations: application to the myostatin gene in dual-purpose belgian blue cattle. animal 1(01), 21 (2007)

Henderson, C.R.: A simple method for computing the inverse of a numerator relationship matrix used in prediction of breeding values. Biometrics 32(1), 69-83 (1976)

See Also

cSSBR.setup, clmm

Examples

```
# example dataset

id <- 1:6
sire <- c(rep(NA,3),rep(1,3))
dam <- c(rep(NA,3),2,2,3)

# phenotypes
y <- c(NA, 0.45, 0.87, 1.26, 1.03, 0.67)

dat <- data.frame(id=id,sire=sire,dam=dam,y=y)
```

```
# Marker genotypes
M <- rbind(c(1,2,1,1,0,0,1,2,1,0),
           c(2,1,1,1,2,0,1,1,1,1),
           c(0,1,0,0,2,1,2,1,1,1))

M.id <- 1:3

var_y <- var(y,na.rm=TRUE)
var_e <- (10*var_y / 21)
var_a <- var_e
var_m <- var_e / 10

# put emphasis on the prior
df = 500

par_random=list(list(method="ridge",scale=var_m,df = df),list(method="ridge",scale=var_a,df=df))

set_num_threads(1)
mod<-cSSBR(data = dat,
           M=M,
           M.id=M.id,
           par_random=par_random,
           scale_e = var_e,
           df_e=df,
           niter=50000,
           burnin=30000)

# check marker effects
print(round(mod[[4]]$posterior$estimates_mean,digits=2))

# check breeding value prediction:
print(round(mod$SSBR$Breeding_Values,digits=2))
```

`cSSBR.setup` *Preparing Model terms for Single Step Bayesian Regression*

Description

This function prepares all model terms for SSBR using pedigree and marker information. The function is particularly useful for using the reported model terms on multiple phenotypes, for cross validation (`clmm`), for genomewide association studies or to pass them to alternative software.

Usage

```
cSSBR.setup(data, M, M.id, verbose=TRUE)
```

Arguments

data	data.frame with four columns: id, sire, dam, y
M	Marker Matrix for genotyped individuals
M.id	Vector of length nrow(M) representing rownames for M
verbose	Prints progress to the screen

Details

...

Value

List of 5:

ids	ids for the model (ordered as in other model terms)
y	phenotype vector
Marker_Matrix	Combined Marker Matrix including imputed and genotyped individuals
Z_residual	Design Matrix used to model the residual error for the imputed individuals
ginverse_residual	Submatrix of the inverse of the numerator relationship matrix. Used to model the residual error for the imputed individuals

Author(s)

Claas Heuer

References

Fernando, R.L., Dekkers, J.C., Garrick, D.J.: A class of bayesian methods to combine large numbers of genotyped and non-genotyped animals for whole-genome analyses. Genetics Selection Evolution 46(1), 50 (2014)

See Also

cSSBR.setup, clmm

Examples

```
# example dataset

id <- 1:6
sire <- c(rep(NA,3),rep(1,3))
dam <- c(rep(NA,3),2,2,3)

# phenotypes
y <- c(NA, 0.45, 0.87, 1.26, 1.03, 0.67)
```

```
dat <- data.frame(id=id,sire=sire,dam=dam,y=y)


# Marker genotypes
M <- rbind(c(1,2,1,1,0,0,1,2,1,0),
           c(2,1,1,1,2,0,1,1,1,1),
           c(0,1,0,0,2,1,2,1,1,1))

M.id <- 1:3

model_terms <- cSSBR.setup(dat,M, M.id)

var_y <- var(y,na.rm=TRUE)
var_e <- (10*var_y / 21)
var_a <- var_e
var_m <- var_e / 10

# put emphasis on the prior
df = 500

par_random=list(list(method="ridge",scale=var_m,df = df),list(method="ridge",scale=var_a,df=df))

set_num_threads(1)

# passing model terms to 'clmm'
mod<-clmm(y=model_terms$y,
          Z=list(model_terms$Marker_Matrix,model_terms$Z_residual),
          ginverse = list(NULL, model_terms$ginverse_residual),
          par_random=par_random,
          scale_e = var_e,
          df_e=df,
          niter=50000,
          burnin=30000)

# check marker effects
print(round(mod[[4]]$posterior$estimates_mean,digits=2))
```

---

get_cor *Compute the prediction accuracy from Cross Validition*

Description

Takes a matrix of predictions returned by get_pred, a list of masked phenotypes returned by cCV and the original phenotype vector and returns the correlation between predicted and observed values

Usage

```
get_cor(predictions,cv_pheno,y)
```

Arguments

predictions	Prediction matrix returned by get_pred
cv_pheno	List of masked phenotypes returned by cCV
y	Original unmasked phenotype vector that has been used in cCV

Value

Numeric scalar - Mean prediction accuracy measured as correlation between predicted and observed phenotypes

See Also

clmm, get_pred, cCV

Examples

```
### Running a 4-fold cross-validation with one repetition:
## Not run:

# generate random data
rand_data(500,5000)

### compute the list of masked phenotype-vectors for CV
y_CV <- cCV(y,fold=4,reps=1)


### Cross Validation using GBLUP
G.A <- cgrm.A(M,lambda=0.01)


### generate the list of design matrices for clmm
Z_list = list(t(chol(G.A)))

### specify options
h2 = 0.3
scale = unlist(lapply(y_CV,function(x)var(x,na.rm=T))) * h2
df = rep(5,length(y_CV))
par_random = list(list(method="ridge",scale=scale,df=df))

### run
fit <- clmm(y_CV, Z=Z_list, par_random=par_random, niter=5000, burnin=2500)

### inspect results
str(fit)

### obtain predictions
pred <- get_pred(fit)

### prediction accuracy
get_cor(pred,y_CV,y)
```

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

`get_max_threads` *Get the maximum number of threads available*

Description

This is a wrapper for the OpenMP-function `omp_get_max_threads()`, hence the function will report the result of the according omp-function. Note: The returned value does not necessarily reflect the number of physical cores present but in most cases it will.

Usage

```
get_max_threads()
```

Value

Returns the value reported by `omp_get_max_threads()`

See Also

`set_num_threads`, `get_num_threads`, `check_openmp`

Examples

```
# set number of threads to the value reported by get_max_threads()
n_threads <- get_max_threads()
set_num_threads(n_threads)

# check
get_num_threads()
```

`get_num_threads` *Get the number of threads for* `cpgen`

Description

Check the variable that specifies the number of threads being used by `cpgen`-functions

Usage

```
get_num_threads()
```

Value

Returns the value of the global variable `cpgen.threads`

See Also

`set_num_threads`, `get_max_threads`, `check_openmp`

Examples

```
# set the number of threads to 1
set_num_threads(1)

# check
get_num_threads()

# set number of threads to the value reported by get_max_threads()
n_threads <- get_max_threads()
set_num_threads(n_threads)

# check
get_num_threads()
```

---

get_pred *Extract predictions vectors of an object returned by* `clmm` *using multpile phenotypes*

Description

Takes an object returned by `clmm` using multpile phenotypes and returns a matrix of predicted values from every model. Every column represents the prediction vector of one model

Usage

```
get_pred(mod)
```

Arguments

`mod`	List returned by `clmm` using multpile phenotypes

Value

Matrix of prediction vectors in columns

See Also

`clmm`, `get_cor`, `cCV`

Examples

```
### Running a 4-fold cross-validation with one repetition:
## Not run:

# generate random data
rand_data(500,5000)

### compute the list of masked phenotype-vectors for CV
y_CV <- cCV(y,fold=4,reps=1)


### Cross Validation using GBLUP
G.A <- cgrm.A(M,lambda=0.01)


### generate the list of design matrices for clmm
Z_list = list(t(chol(G.A)))

### specify options
h2 = 0.3
scale = unlist(lapply(y_CV,function(x)var(x,na.rm=T))) * h2
df = rep(5,length(y_CV))
par_random = list(list(method="ridge",scale=scale,df=df))

### run
fit <- clmm(y_CV, Z=Z_list, par_random=par_random, niter=5000, burnin=2500)

### inspect results
str(fit)

### obtain predictions
pred <- get_pred(fit)

### prediction accuracy
get_cor(pred,y_CV,y)

## End(Not run)
```

`Parallelization` *Multithreading using* `cpgen`

Description

The package `cpgen` makes use of shared memory multi-threading using OpenMP. R is of single-threaded nature, hence almost the entire package is written in C++. The package offers a variety of functions that lets you control and check the number of threads that are being used by the functions of the package. Internally every function uses the global variable `cpgen.threads` which is stored in `options()$cpgen.threads`. The value can be changed using the function

`set_num_threads()`. When the package is loaded in an R-session `cpgen.threads` will be set to the value returned by `get_max_threads()` which is a wrapper for the OpenMP-header function `omp_get_max_threads()`

Details

The following functions are multithreaded and access the variable `cpgen.threads`:

- cGWAS
- cGWAS.emmax
- clmm
- cGBLUP
- ccross
- %c%
- cgrm
- cgrm.A
- cgrm.D
- ccov
- csolve
- cSSBR.setup
- cSSBR

See Also

`set_num_threads`, `get_num_threads`, `get_max_threads`, `check_openmp`

`rand_data` *Generate random data for test purposes*

Description

Generates a random marker-matrix in {-1,0,1} coding and a phenotype vector. Phenotypic variance times h2 (variance explained by markers) is equally spread among all markers (sampled from uniform distribution).

Usage

```
rand_data(n=500,p_marker=10000,h2=0.3,prop_qtl=0.01,seed=NULL)
```

Arguments

`n`	Number of oberservations
`p_marker`	Number of markers
`h2`	Heritability of the trait
`prop_qtl`	Proportion of QTL of total number of markers
`seed`	Seed for RNG

Value

No return value. Generates two objects globally (M and y) that can be used after the execution of the function. M is the marker matrix and y the phenotype vector

Examples

```
# Generate random data with 100 observations and 500 markers
rand_data(100,500)

# check that objects have been created
str(M)
str(y)
```

---

set_num_threads *Set the number of OpenMP threads used by the functions of package* cpgen

---

Description

This function sets the value of the global variable stored in options()$cpgen.threads to the assigned integer. Note_1: The assigned value may exceed the number of physical cores present but that might lead to dramatical decrease in performance. Note_2: The function can override the global variable 'OMP_NUM_THREADS' (if global=TRUE and hence also other non-cpgen functions are affected by a call to set_num_threads().

Usage

```
set_num_threads(x,silent=FALSE, global=FALSE)
```

Arguments

x	Integer scalar that specifies the number of threads to be used by cpgen-functions
silent	boolean, controls whether to print a message
global	boolean, change openmp threads globally (might effect other libraries)

Value

Changes the global variable cpgen.threads to the value in x

See Also

get_num_threads, get_max_threads, check_openmp

Examples

```
# Control the number of threads being used in an R-session:

# set the number of threads to 1
## Not run:
set_num_threads(1)

#### Use a parallelized cpgen-function

# generate random data
rand_data(1000,10000)

# check single-threaded performance
system.time(W <- M%c%t(M))

# set number of threads to 2

set_num_threads(2)

# check multi-threaded performance
system.time(W <- M%c%t(M))

## End(Not run)
```

`%**%` *Square matrix power operator*

Description

This operator computes an arbitrary power of a positive definite square matrix using an Eigen-decomposition: $\mathbf{X}^p = \mathbf{U}\mathbf{D}^p\mathbf{U}'$

Usage

```
X %**% power
```

Arguments

`X`	Positive definite square matrix
`power`	numeric scalar - desired power of X

Value

Matrix `X` to the power `p`

Examples

```
## Not run:
# Inverse Square Root of a positive definite square matrix
X <- matrix(rnorm(100*5000),100,1000)

XX <- ccross(X)

XX_InvSqrt <- XX %**% -0.5

# check result: ((XX')^-0.5 (XX')^-0.5)^-1 = XX'
table(round(csolve(XX_InvSqrt %c% XX_InvSqrt),digits=2) == round(XX,digits=2) )

## End(Not run)
```

`%c%` *(Parallel) Matrix product operator*

Description

This operator computes the matrix-product between two matrices. It can be used as a replacement for `%*%` in many cases. The operator only accepts matrices of types: `matrix` or `dgCMatrix`. In the case of two dense matrices the operator will compute the crossproduct in parallel (Eigen + OpenMP)

Usage

```
X%c%Y
```

Arguments

`X`	Matrix or vector (treated as column-vector) of type: `matrix` or `dgCMatrix`
`Y`	as `X`

Value

Matrix of type: `matrix` or `dgCMatrix`

Examples

```
# Least Squares Solving

# Generate random data

n = 1000
p = 500

M <- matrix(rnorm(n*p),n,p)
y <- rnorm(n)
```

```
    # least squares solution:

    b <- csolve(t(M) %c% M, t(M) %c% y)
```

Index