Package 'cpgen'

August 13, 2014

Type Package

Title Parallel genomic evaluations

Version 0.1

Date 2014-07-22

Author Claas Heuer

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

Description Frequently used methods in genomic applications with emphasis on parallel computing

License GPL (>= 2)

SystemRequirements C++11

Depends R(>= 3.1.0), Rcpp (>= 0.9.1), Matrix(>= 1.0-5)

LinkingTo Rcpp, RcppEigen

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,var=FALSE)
```

Arguments

X	Matrix
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,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, cor=FALSE)
```

Arguments

X	matrix
lambda	numeric scalar, shrinkage parameter
w	numeric vector of weights with same lengths as rows in X
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),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`

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. 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 used by clmm
burnin	Burnin for clmm
seed	Seed for clmm
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:

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

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, clmm.CV, 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):

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

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:

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

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

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

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 it it necessary to construct the design matrix for that random effect as described in Waldmann et al. (2008): $\mathbf{F} = \mathbf{ZG}^{1/2}$.

Usage

```
clmm(y, X = NULL , random = NULL, par_random = NULL, niter=10000,
burnin=5000,scale_e=0,df_e=-2, verbose = TRUE, seed = NULL)
```

Arguments

y	vector of phenotypes
X	Fixed effects design matrix of type: `matrix` or `dgCMatrix`. If omitted a column-vector of ones will be assigned
random	list of design matrices for random effects - every element of the list represents one random effect and may be of type: `matrix` or `dgCMatrix`
par_random	list of options for random effects. If passed, the list must have as many elements as `random`. Every element must be a list of 3: • scale - scale parameter for the inverse chi-square prior • df - degrees of freedom for the inverse chi-square prior • method - method to be used for the random effects, may be: `random` or `BayesA`
niter	number of iterations
burnin	number of iterations to be discarded as burnin
verbose	prints progress to the screen
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

Details

At this point the function allows to specify the method for any random term as: 'random' or 'BayesA'. 'random' assumes a common variance for all levels of the random effect, 'BayesA' assumes every level of the random effect to have its own distribution and variance as described in Meuwissen et al. (2001). 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.

Value

List of 3 + number of random effects:

`Residual_Variance`
: List of 4:
  - Posterior_Mean - Mean estimate of the residual variance
  - Posterior - Distribution 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

`Effect_1`
: List of 4:
  - type - dense or sparse design matrix
  - method - method that has been used = "fixed"
  - scale_prior - scale parameter that has been assigned
  - df_prior - degrees of freedom that have been assigned
  - posterior - list of 1 = mean of the solution for fixed effects

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

`Effect_k`
: List of 4:
  - 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
    - estimates_mean - mean solutions for random effects
    - variance_mean - mean variance
    - variance - distribution of variance

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

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.

See Also

clmm.CV, cGBLUP, cGWAS.emmax

Examples

```
### Running a model with an additive and dominance effect
## Not run:
# 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
random = list(t(chol(G.A)),t(chol(G.D)))

### specify options
par_random = list(list(method="random",scale=var(y)/2 * 7,df=5),
  list(method="random",scale=var(y)/10 * 7,df=5))

### run

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

### inspect results
str(fit)

## End(Not run)
```

---

clmm.CV *Cross Validation with Linear Mixed Models using Gibbs Sampling*

Description

This function allows running Cross Validation using clmm. All the formulation is equal to clmm except for the phenotypes, which are being passed as a list of equally sized vectors. 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 is almost linear.

Usage

```
clmm.CV(y, X = NULL , random = NULL, par_random = NULL, niter=10000,
burnin=5000,scale_e=0,df_e=-2, verbose = TRUE, seed = NULL)
```

Arguments

y	list of phenotype vectors
X	Fixed effects design matrix of type: `matrix` or `dgCMatrix`. If omitted a column-vector of ones will be assigned
random	list of design matrices for random effects - every element of the list represents one random effect and may be of type: `matrix` or `dgCMatrix`
par_random	list of options for random effects. If passed, the list must have as many elements as `random`. Every element must be a list of 3: • scale - scale parameter for the inverse chi-square prior • df - degrees of freedom for the inverse chi-square prior • method - method to be used for the random effects, may be: `random` or `BayesA`
niter	number of iterations
burnin	number of iterations to be discarded as burnin
verbose	prints progress to the screen
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

Details

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.

Value

List of `length(Y)` with elements equal to the output of `clmm`

Author(s)

Claas Heuer

See Also

`clmm`

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
random = list(t(chol(G.A)))

### specify options
par_random = list(list(method="random",scale=var(y)/2 * 7,df=5))

### run
fit <- clmm.CV(y_CV,random=random,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)
```

---

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)
```

---

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-solver 'llt' 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 functions returns the inverse of $\mathbf{X}$.

Usage

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

Arguments

X	positive definite square matrix
y	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)
```

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

Examples

```
# see example of clmm.CV
```

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.CV

Description

Takes an object returned by clmm.CV and returns a matrix of predicted values from every model. Every columns represents the prediction vector of one model

Usage

```
get_pred(mod)
```

Arguments

mod	List returned by clmm.CV

Value

Matrix of prediction vectors in columns

Examples

```
# see example of clmm.CV
```

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
- clmm.CV
- cGBLUP
- ccross
- %c%
- cgrm
- cgrm.A
- cgrm.D
- ccov

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. Both objects are independent and the only purpose is to test functions and show examples

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: The assigned value may exceed the number of physical cores present but that might lead to dramatical decrease in performance.

Usage

```
set_num_threads(x,silent=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

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) crossproduct operator*

Description

This operator computes the crossproduct between two matrices. It can be used as a replacement for `%*%` in most 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