Package ‘miraculix’

April 17, 2020

Version 0.9.20

Title Algebraic and Statistical Functions for Genetics

Author Martin Schlather [aut, cre], Malena Erbe [aut, cre], Florian Skene [aut], Alexander Freudenberg [ctr]

Description This is a collection of fast tools for application in quantitative genetics. For instance, the SNP matrix can be stored in a minimum of memory and the calculation of the genomic relationship matrix is based on a rapid algorithm. It also contains the window scanning approach by Kabluchko and Spodarev (2009), <doi:10.1239/aap/1240319575> to detect anomalous genomic areas <doi:10.1186/s12864-018-5009-y>. Furthermore, the package is used in the Modular Breeding Program Simulator (MoBPS, <https://github.com/tpook92/MoBPS>, <http://www.mobps.de/>). The tools are based on SIMD (Single Instruction Multiple Data, <https://en.wikipedia.org/wiki/SIMD>) and OMP (Open Multi-Processing, <https://de.wikipedia.org/wiki/OpenMP>).

Maintainer Martin Schlather <`schlather@math.uni-mannheim.de`>

LinkingTo RandomFieldsUtils

Depends R (>= 3.0), RandomFieldsUtils (>= 0.5)

Imports methods, graphics

Suggests

License GPL (>= 3)

Biarch true

URL `http://ms.math.uni-mannheim.de/de/publications/software/miraculix`

NeedsCompilation yes

Repository CRAN

Date/Publication 2020-04-17 12:50:02 UTC

R topics documented:

`miraculix-package` *MIRACULIX*

Description

Various functions used in quantitative genetics

Details

1. Very fast calculation of genomic relationship matrix for 0-1-coded haplotypes and 0-1-2-coded genotypes; Matrix should be in the RAM
   (a) `relationshipMatrix` fast calculation of $(M - P)(M - P)^T/\sigma^2$
   (b) `crossprodx` fast implementation of `crossprod` for SNP matrices
2. further commands
   (a) `haplomatrix` compresses haplotype data
   (b) `as.matrix` uncompresses `genomicmatrix` or `haplomatrix`
   (c) `genomicmatrix` transformation to a compressed genotype from a usual matrix or a compressed haplotype
   (d) `genomicmatrix`,`fillGeno` creating a compressed matrix and filling it with uncompressed data. These two functions make sense if the SNP matrix is too large to be kept in the RAM.
   (e) `solveRelMat` calculates the inverse of a relatioship matrix and also solves equations
   (f) `allele_freq` calculates the allele frequencies of a SNP matrix that might have been compressed by `genomicmatrix`, for instance.
   (g) `genoVector`, `vectorGeno` multiplication of vector onto a compressed SNP matrix from the right and left, respectively.
   (h) `vectorGeno` etc. fast calculation of 012 matrix with an arbitrary vector
   (i) `matrixvector012` etc. fast calculation of an arbitrary matrix with a 012 vector
3. Functions related to the package **MoBPs** by Torsten Pook.

(a) `codeOrigins`,`decodeOrigins` compressed data representation of breeding relevant information of an individuum
(b) `computeSNPS` calculates the genome of an individuum from the coding in the population tree
(c) `compute` concatenation of `computeSNPS`, `relationshipMatrix`, and `solveRelMat`

Support

This package was partially developed at the Department of Animal Breeding and Genetics and CiBreed, University of Goettingen.

Author(s)

Martin Schlather, `<schlather@math.uni-mannheim.de>` `http://ms.math.uni-mannheim.de`;

Malena Erbe

Examples

```
indiv <- 5
snps <- indiv * 10
M <- matrix(ncol=indiv, sample(0:2, indiv * snps, replace=TRUE))
print(M)
print(relationshipMatrix(M))
```

`genomicmatrix` *Transform a Matrix to a Compressed Matrix*

Description

Coerce to or create a compressed genomic matrix

Usage

```
genomicmatrix(snps, individuals, file.type,
               coding, header, IndividualsPerColumn,
      DoubledIndividuals, leadingcolumns, loading,
               ...)
## S3 method for class 'genomicmatrix'
as(object, ...)
```

Arguments

`object, snps`	integer, matrix, vector, a haplomatrix or file name. See Details.
`individuals`	integer. See Details
`file.type`	if `object` is a filename then the precise coding of preceding headers, preceding columns, and the coding of the data can be very different. Instead of giving all the arguments `coding`, ..., `leadingcolumns`, the `file.type` can be given:

'beagle' i.e.`coding="AB? "`

'plink' i.e. `coding="AB? "`

'plink2' i.e. `coding="12? "`,

'plinkbinary' i.e. `coding="12345"`

`coding` if `object` is a filename then `coding` is a string of 4 or 5 characters.

In case of 5 characters, a file with genomic data is assumed and the characters have the following meaning:

1st code for 0

2nd code for 1

3rd code for 2

4th code for NA

5th the field separator character

In case of 4 characters, a file with haplotype information is assumed and the characters have the following meaning:

1st code for 0

2nd code for 1

3rd code for NA

4th the field separator character

The haplotype data is turned into genomic data.

`header` integer. If `object` is a filename then `header` has the following meaning

positive `header`**:** `header` gives the number of preceding lines in the file that will be ignored. An ASCII file is assumed in this case.

negative `header`**:** a binary file is assumed and $-$`header` gives the number of preceding characters that will be jumped.

`IndividualsPerColumn`

Logical. If `IndividualsPerColumn=TRUE` then the first argument indicates a (SNPs $\times$ Individ) matrix. Otherwise, the first argument indicates a (Individ $\times$ SNPs) matrix, which will be transposed before storage.

`DoubledIndividuals`

Logical. If `DoubledIndividuals=TRUE` the haplotype information for the second chromosome is given in direction of the individuals, i.e. if additionally `IndividualsPerColumn=TRUE`, the number of columns are doubled. Otherwise, the information is given in the other direction. The information at one locus is always given back-to-back. If `object` is a filename, coding has 4 characters (i.e. it is a haplo file)

`leadingcolumns` Integer. If `object` is a filename then `leadingcolumns` gives the number of first columns in the file that are ignored.

`loading` logical. If `object` is a filename then `loading` decides whether the file contents is read into RAM. Otherwise the file is read on the fly whenever possible. `loading` is `TRUE` for `genomicmatrix` and `FALSE` otherwise.

`...` options, see `RFoptions`

Details

genomicmatrix creates a compressed matrix according to the coding scheme given by RFoptions()$genetics$snpcoding.

In case snps is a string, i.e., a file name, the extension of the file name predefines the file.type:

'.txt' ='beagle'

'.bgl' ='beagle'

'.phased' ='plink'

'.tped' ='plink2'

'.ped' ='plink2'

'.bed' ='plinkbinary'

The definition can be overwritten by file.type. The latter can be overwritten by all other options (except individuals).

If individuals is given, genomicmatrix creates a snps $\times$ individuals compressed data matrix filled with zeros. The matrix can be modified afterwards by fillGeno.

If a haplomatrix is given, it is transformed into a genomicmatrix.

If genomicmatrix is given, the matrix is returned as is and a warning is given.

Both functions, genomicmatrix and as have exactly the same behavior execept for loading which is TRUE for genomicmatrix by default and fixed to be FALSE for as.genomicmatrix.

Value

an object of class genomatrix

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, http://ms.math.uni-mannheim.de

See Also

haplomatrix as.matrix

Examples

```
set.seed(0)
snps <- 100
indiv <- 10
M <- matrix(sample(0:2, snps * indiv, replace=TRUE), nrow = snps)
(GM <- genomicmatrix(M))
stopifnot(all(as.matrix(GM) == M))


## There is a difference between genomicmatrix and as.genomicmatrix
## in case of files: 'as.genomicmatrix' creates only a pointer to
## the file, while 'genomicmatrix' reads the file
file <- "miraculix"
```

```
if (interactive() && !file.exists(paste0(file, ".bgl"))) {
  f <- rhaplo(indiv=100, loci=1000, file=file, file.type="beagle")
  print(f)
  print(G <- as.genomicmatrix(f))
  print(g <- genomicmatrix(f))
  Print(object.size(G), object.size(g)) ## g needs much more memory
  file.remove(f)
}
```

`genomicmatrix-class` *Class* `genomicmatrix`

Description

Class representing a genomic matrix

Usage

```
## S3 method for class 'genomicmatrix'
print(x, ...)
## S3 method for class 'genomicmatrix'
str(object, ...)
## S3 method for class 'genomicmatrix'
as.matrix(x, ...)
```

Arguments

`x,object`	a compressed (SNP x Individuals) matrix
`...`	see `print`, `str` for options; see section 'Details' for `as.matrix`.

Details

Since the genomic matrix has only the values 0,1,2, `genomicmatrix` uses a two bit compressed storing mode in case `RFoptions(snpcoding = TwoBit)` or `snpcoding = Shuffle`, for instance, see `RFoptions` for more information and further options.

The options `...` for `as.matrix` are

N vector of integers, which gives the selected rows. If missing all rows are selected.

do.centering logical. If `TRUE` the value of `RFoptions()$genetics$centering` is considered.

- `TRUE` centering by `rowMeans`.
- `FALSE` no centering is performed (although `do.centering = TRUE`!)
- `is.numeric` the values given by the user are substracted.

Author(s)

Martin Schlather, `<schlather@math.uni-mannheim.de>`, `http://ms.math.uni-mannheim.de`

See Also

genomicmatrix haplomatrix-class

Examples

```
set.seed(0)
snps <- 100
indiv <- 10
M <- matrix(sample(0:2, snps * indiv, replace=TRUE), nrow = snps)
GM <- genomicmatrix(M)
print(GM)
str(GM)
stopifnot(all(as.matrix(GM) == M))
```

---

haplomatrix *Transform a Haplotype Vector to a Compressed Haplotype Vector*

Description

Coerce a matrix to a compressed haplotype matrix

Usage

```
haplomatrix(M, IndividualsPerColumn=TRUE, DoubledIndividuals=TRUE)
## S3 method for class 'haplomatrix'
as(object, ...)
```

Arguments

`M,object`	matrix of two rows containing only the values 0 and 1
`IndividualsPerColumn`	Logical. If `IndividualsPerColumn=TRUE` then the first argument indicates a (SNPs $\times$ Individ) matrix. Otherwise, the first argument indicates a (Individ $\times$ SNPs) matrix, which will be transposed before storage.
`DoubledIndividuals`	Logical. If `DoubledIndividuals=TRUE` the haplotype information for the second chromosome is given in direction of the individuals, i.e. if additionally `IndividualsPerColumn=TRUE`, the number of columns are doubled. Otherwise, the information is given in the other direction. The information at one locus is always given back-to-back.
`...`	All arguments of `haplomatrix` except `M`

Value

an object of class genomicmatrix

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, http://ms.math.uni-mannheim.de

See Also

Note that a haplotype file can be read in by `genomicmatrix`.

`as.matrix` transforms a `genomicmatrix` to a human readable matrix.

Examples

```
set.seed(0)
snps <- 100
cols <- 2
M <- matrix(sample(0:1, snps * cols, replace=TRUE), ncol = snps)
Print(M)
print(GM <- haplomatrix(M))
stopifnot(all(as.matrix(GM) == M))
```

---

haplomatrix-class *Class* haplomatrix

Description

Class representing a haplo matrix

Usage

```
## S3 method for class 'haplomatrix'
print(x, ...)
## S3 method for class 'haplomatrix'
str(object, ...)
## S3 method for class 'haplomatrix'
as.matrix(x, ...)
```

Arguments

`x,object`	a compressed (SNP x Individuals) matrix
`...`	see `print`, `str` for their options. The command `as.matrix` has the following options `indiv` vector of integer, indicating individuals to be extracted `sets` value `1`, `2` or `1:2`. Indicates the chromosome set to be returned. Default:1:2

`IndividualsPerColumn` Logical. If `IndividualsPerColumn=TRUE` then the first argument indicates a (SNPs $\times$ Individ) matrix. Otherwise, the first argument indicates a (Individ $\times$ SNPs) matrix, which will be transposed before storage. Default: `TRUE`

`DoubledIndividuals` Logical. If `DoubledIndividuals=TRUE` the haplotype information for the second chromosome is given in direction of the individuals, i.e. if additionally `IndividualsPerColumn=TRUE`, the number of columns are doubled. Otherwise, the information is given in the other direction. The information at one locus is always given back-to-back. Default: `TRUE`

Details

Since the haplo matrix takes only the values 0 and 1, `haplomatrix` uses a one bit compressed storing mode. A haplomatrix can quickly be transformed into a `genomicmatrix` (by exactly this command) in case of the default two-bit coding, e.g. `RFoptions(snpcoding=Shuffle)`.

Author(s)

Martin Schlather, `<schlather@math.uni-mannheim.de>`, `http://ms.math.uni-mannheim.de`

See Also

`genomicmatrix-class`

Examples

```
set.seed(0)
indiv <- 5
loci <- 4
M <- matrix(sample(0:1, 2 * indiv * loci, replace=TRUE), nrow = loci)
str(M)

GM <- haplomatrix(M)
print(GM)
str(GM)
print(as.matrix(GM))
print(as.matrix(GM, indiv=2:4, sets=1))
stopifnot(sum(abs(as.matrix(GM) - M)) == 0)
```

`Instruction Set` *CPU instruction set*

Description

The function checks whether a certain instruction is available under the current compilation of the package.

Usage

```
has.instruction.set(which=c("SSE2", "SSSE3",  "AVX", "AVX2"))
```

Arguments

`which`	character vector.

Value

logical vector of length `which`. An element is `TRUE` if the instruction set is recognized by the package.

Author(s)

Martin Schlather, `<schlather@math.uni-mannheim.de>`, `http://ms.math.uni-mannheim.de`

Examples

```
has.instruction.set("AVX2")
```

`Manipulate` *Manipulating Compressed Matrices*

Description

`copyGeno` copies a coded SNP matrix

`zeroNthGeno` writes zeros into selected rows of a coded SNP matrix

`fillGeno` allows to fill (or replace) colums of a compressed (snps $\times$ indiv) matrix.

Usage

```
fillGeno(SNPxIndiv, indiv, values, IndividualsPerColumn=TRUE,
         DoubledIndividuals=TRUE)
copyGeno(SNPxIndiv)
zeroNthGeno(SNPxIndiv, snps)
```

Arguments

`SNPxIndiv`	a compressed SNP (genotype) vector or matrix, obtained from `genomicmatrix` or `haplomatrix`
`indiv`	integer vector. It gives the columns of the (SNP $\times$ Indiv) matrix that has to be filled with `values`
`values`	coded or uncoded vector or matrix of haplotype or genotypes.
`snps`	vector of integers, which gives the selected rows. If missing all rows are selected.

IndividualsPerColumn
: Logical. If IndividualsPerColumn=TRUE then the first argument indicates a (SNPs $\times$ Individ) matrix. Otherwise, the first argument indicates a (Individ $\times$ SNPs) matrix, which will be transposed before storage.

DoubledIndividuals
: Logical. If DoubledIndividuals=TRUE the haplotype information for the second chromosome is given in direction of the individuals, i.e. if additionally IndividualsPerColumn=TRUE, the number of columns are doubled. Otherwise, the information is given in the other direction. The information at one locus is always given back-to-back.

Value

All functions return a compressed SNP matrix of class genomicmatrix.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, http://ms.math.uni-mannheim.de

See Also

genomicmatrix-class

vectorGeno for multiplying a vector from the left

genoVector for multiplying a vector from the right

Examples

```
require(RandomFieldsUtils)
set.seed(0)

indiv <- sample(1000, 1)
snps <- indiv * 2^sample(7,1)
M <- matrix(nrow = snps, sample(0:2, snps * indiv, replace=TRUE))
storage.mode(M) <- sample(c("integer", "double"), 1)
CM <- genomicmatrix(M)
str(CM)
Z <- as.matrix(CM)
Print(M, CM, Z)
stopifnot(all(M == Z))

N <- sample(snps, snps / 4)
Z1 <- as.matrix(CM, snps=N)
stopifnot(all(M[N, ] == Z1))
```

MoPBS *Functions designed for the R package* **MoBPS**

Description

The functions below have been written mainly for use in the package **MoBPS** written by Torsten Pook.

`codeOrigins` compresses information about generation of introduced new genes, sex, number of individuals and haplotype in a single 32 Bit integer value.

`decodeOrigins` make the compressed data human readable again.

`computeSNPS` extracts from a coded, complete breeding scheme an individuum defined by its generation, sex and number within its cohort.

`compute` essentially concatenates (efficiently) the three commands `computeSNPS`, `relationshipMatrix`, `solveRelMat`

Usage

```
codeOrigins(M)
decodeOrigins(CM, row)
computeSNPS(population, gen, sex, nr, from_p = 1, to_p = Inf,
           output_compressed=FALSE, select = NULL, what = c("geno", "haplo"))
compute(population, gen, sex, nr, tau, vec, betahat, select = NULL,
        matrix.return=FALSE)
```

Arguments

`M`	matrix with information on generation of introduced new genes, sex, number of individual and haplotype on each line. the generation takes values in 1...2^6, sex values in 1...2^1, individual values in 1...2^22 and the haplotype values in 1...2^3
`CM`	a vector obtained from coding a matrix by `codeOrigins`
`row`	integer. Row number of the matrix `M` or `CM` to be decoded.
`population`	list of list, as described in package **MoBPs**, which contains the whole information of all generations of a breeding scheme
`gen,sex,nr`	information specifying an individuum; instead of the three argument, only gen might be given, which is matrix of three columns then.
`from_p, to_p`	loci between which the genomic information of the specified individuum is extracted. Default: whole genomic information
`output_compressed`	logical. If `FALSE` the output is human readable
`select`	integer vector. List of loci that should be returned; the loci might be further restricted by `from_P` and `to_p`.
`what`	The type of information that should be extracted and returned

`tau,vec,betahat`	see `solveRelMat`
`matrix.return`	logical. If `TRUE` also the relationship matrix is returned.

Value

`codeOrigins` : a vector with length equal to the number of rows of `M`.

`decodeOrigins` : an integer vector of 4 components.

`computeSNPS` : vector of integers with either human readable values or compressed data depending on the argument `what`.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, http://ms.math.uni-mannheim.de

Examples

```
set.seed(0)
n <- sample(1000, 1)
M <- cbind(sample(1:2^6, n, replace=TRUE),
           sample(1:2^1, n, replace=TRUE),
           sample(1:2^22, n, replace=TRUE),
           sample(1:2^3, n, replace=TRUE))
print(head(M))
Z <- matrix(NA, ncol=ncol(M), nrow=nrow(M))
CM <- codeOrigins(M)
print(head(CM))
for (i in 1:nrow(M))  Z[i, ] <- decodeOrigins(CM, i)
stopifnot(all(M == Z))
```

`Random Haplotype Values`

Generation of Random Haplotype Matrix

Description

A random haplotype matrix is generated according to some given frequencies.

Usage

```
rhaplo(freq, indiv, loci, freq2, file,
       file.type = c("beagle", "plink", "plink2"),
       debugging = FALSE)
```

Arguments

`freq`	vector of probabilities which gives the allele frequencies for one or both haplotypes; if not given, a half is assumed and `loci` must be given.
`indiv`	number of individuals
`loci`	if not given, the number of loci equals the length of `freq`, otherwise `freq` is recycled to reach the given nnumber of loci
`freq2`	optional. Frequencies for the second chromosome. The vector `freq2` may have a different length than `freq` if `loci` is given or `freq2` is a scalar. The vector `freq2` may contain `NA`s. Then, the value of the second chromosome at this locus is taken over from the first chromosome.
`file, file.type`	string. If given, a file is written that mimics the `file.type` style. An extension is appended to `file` according to the `file.type` style.
`debugging`	logical. Mainly for internal purposes. If `TRUE` the genomic matrix is appended as an attribute to the return value.

Value

If `missing(file)` an object of class genomicmatrix is returned, else the file name with appended extension according to `file.type`

Author(s)

Martin Schlather, `<schlather@math.uni-mannheim.de>`, `http://ms.math.uni-mannheim.de`

See Also

A haplotype file can be read in by `genomicmatrix`.

`as.matrix` transforms a `genomicmatrix` to a human readable matrix.

Examples

```
as.matrix(rhaplo(seq(0, 1, len=10), indiv=5))

## note that the next examples write a file on the current directory
file <- "miraculix"
if (interactive() && !file.exists(paste0(file, ".bgl"))) {
  f <- rhaplo(freq = c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6),
              freq2 = c(0.6, 0.4, 0.5, 0.3, 0.0, 1.0),
              indiv=5, file=file, file.type="beagle",
              debugging = TRUE)
  print(f)
  print(as.genomicmatrix(f))
  print(g <- genomicmatrix(f))
  print(as.matrix(g))

  stopifnot(all(as.matrix(g) == attr(f, "M")))
  file.remove(f)
}
```

relationshipMatrix — *Fast calculation of the Genomic Relationship Matrix and its derivatives*

Description

relationshipMatrix calculates the relationship matrix $A = (M-P)^T(M-P)/\sigma^2$ from the SNP matrix M where $P = p(1,\ldots,1)$ with $p = M\% * \%(1,\ldots,1)^T/n$. Furthermore, $sigma^2$ equals $\sigma^2 = p^T(1-p/2) \in [0,\infty)$.

crossprodx calculates the cross-product of SNPxIndiv, i.e. it is identical to call relationshipMatrix with optional argument, centered=FALSE, cf. RFoptions

allele_freq calculates $p/2$.

SNPeffect calculates $M(A+\tau I)^{-1}v$

solveRelMat calculates

$$(A+\tau I)^{-1}v$$

and

$$A(A+\tau I)^{-1}v+\beta$$

where A is the relationship matrix.

Usage

```
relationshipMatrix(SNPxIndiv, ...)
crossprodx(SNPxIndiv)

solveRelMat(A, tau, vec, betahat=NULL, destroy_A=FALSE)
SNPeffect(SNPxIndiv, vec, centered=TRUE, tau=0)
allele_freq(SNPxIndiv)
```

Arguments

SNPxIndiv	$\{0,1\,2\}$-valued (snps $\times$ indiv) matrix or the result of genomicmatrix.
...	see RFoptions – better use RFoptions. The main two options are: centered: see below normalized:logical. if FALSE then the division by $sigma^2$ is not performed
centered	if FALSE then P is not substracted.
A	a symmetric, positive definite matrix, which is a relationship matrix
tau	non-negative scalar
vec	the vector v
betahat	scalar or NULL. See also section value.
destroy_A	logical. If TRUE the values of the matrix A will be overwritten during the calculations (leading to a faster execution with less memory needs).

Details

Let $p = M\% * \%(1, \ldots, 1)^T/n$ where n is the number of individuals. Then, the matrix P equals $P = p(1, \ldots, 1)$.

The constant $sigma^2$ equals $\sigma^2 = p^T(1 - p/2)$.

`solveRelMat` has a speed and memory advantage in comparison to the direct implementation of the above formulae.

Value

`relationsshipMatrix` returns a (Indiv $\times$ Indiv) numerical matrix.

The return value of `solveRelMat` depends on `betahat`. If the latter is `NULL`, only the vector $(A + \tau I)^{-1}v$ is returned. Else, a list of 2 elements is returned. First element equals the vector

$$(A + \tau I)^{-1}v,$$

the second element equals

$$A(A + \tau I)^{-1}v + \beta.$$

Benchmarks

Computing times for the relationship matrix in comparison to 'crossprod' in standard implementation on Intel(R) Core(TM) i7-8550U CPU @ 1.80GHz, R version 3.6.0 (Linux) with `indiv = 1000` and `snps = 5e5` are:

Shuffle256 : 48 x faster (AVX2; 16x compressed)
Packed256 : 36 x faster (AVX2; 16x compressed)
Shuffle : 35 x faster (SSSE3; 16x compressed)
Multiply256 : 29 x faster (AVX2; 16x compressed)
Packed : 28 x faster (SSE2; 16x compressed)
Hamming2 : 24 x faster (SSE2; 4x compressed)
Hamming3 : 21 x faster (SSSE3; 4x compressed)
Multiply : 17 x faster (SSE2; 16x compressed)
ThreeBit : 17 x faster (uint64_t; 10x compressed)
TwoBit : 15 x faster (uint64_t; 16x compressed)
NoSNPcoding : 4 x faster (int, AVX2; not compressed)
NoSNPcodingAVX: 2 x faster (double, AVX; not compressed)
NoSNPcodingR : calls crossprod

In parantheses, first the instruction set or s the main data type is given, then the data compression with respect to 32 bit integer.

The following code was used:

```
RFoptions(cores = 1)
indiv <- 1000
snps <- 5e5 ## may cause memory allocation problems in R; better use 5e4 !!
methods <- c(NoSNPcodingR, NoSNPcodingAVX,
             FirstGenuineMethod:LastGenuineMethod)
M <- matrix(ncol=indiv, sample(0:2, indiv * snps, replace=TRUE))
for (storageMode in c("integer", "double")){
    storage.mode(M) <- storageMode
  cat("\n\n")
  print(S <- system.time(C <- crossprod(M)))
  p <- rowMeans(M)
  P <- p %*% t(rep(1, indiv))
  sigma2 <- sum(p * (1- p/2))
  A <- crossprod(M-P) / sigma2
  print(S <- system.time(C <- crossprod(M)))
  for (method in methods) {
    RFoptions(snpcoding = method)
    cat("\nstorage=", storageMode, "  method=", SNPCODING_NAMES[method + 1],
    "\n")
    S0 <- system.time(G <- genomicmatrix(M))
    print(S1 <- system.time(C1 <- crossprodx(M)))
    print(S2 <- system.time(C2 <- crossprodx(G)))
    stopifnot(all(C == C1))
    stopifnot(all(C == C2))
    R1 <- S / S1
    R2 <- S / S2
    print(0.5 * (R1 + R2))
    print(S3 <- system.time(C3 <- relationshipMatrix(M)))
    print(S4 <- system.time(C4 <- relationshipMatrix(G)))
    R3 <- S / S3
    R4 <- S / S4
    print(0.5 * (R3 + R4))
    stopifnot(all.equal(as.double(A), as.double(C3)))
    stopifnot(all.equal(as.double(A), as.double(C4)))
    gc()
  }
}
```

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, http://ms.math.uni-mannheim.de

Examples

```
require(RandomFieldsUtils)
set.seed(0)
snpcodes <- c(TwoBit, ThreeBit)
if (has.instruction.set("SSE2")) snpcodes <- c(snpcodes, Hamming2)
if (has.instruction.set("SSSE3")) snpcodes <- c(snpcodes, Hamming3, Shuffle)
if (has.instruction.set("AVX2")) snpcodes <- c(snpcodes, Shuffle256)

Print(snpcodes)

indiv <- 1 + sample(100:500, 1)
snps <- indiv * 2^sample(1:if (interactive()) 7 else 5, 1)
RFoptions(snpcoding=sample(snpcodes, 1))
```

```
M <- matrix(ncol=indiv, sample(0:2, indiv * snps, replace=TRUE))
print(system.time(G <- genomicmatrix(M)))
print(G)

## crossprodx vs crossprod: about 10x faster
Print(system.time(C <- crossprodx(M)))
print(system.time(C2 <- crossprod(M)))
stopifnot(all(C == C2))

## allele_freq vs rowMeans: about equally fast
Print(system.time(af <- allele_freq(M)))
print(system.time(alleleFreq <- 0.5 * rowMeans(M)))
stopifnot(all.equal(as.double(alleleFreq), as.double(af)))

## relationshipMatrix vs crossprod: about 10x faster
Print(system.time(R <- relationshipMatrix(M)))
print(system.time(R <- relationshipMatrix(G)))
print(system.time({
  sigma2 <- 2 * sum(alleleFreq * (1 - alleleFreq))
  R2 <- crossprod(M - 2 * alleleFreq) / sigma2
}))
stopifnot(all.equal(as.double(R), as.double(R2)))


### solveRelMat vs. solve: about equally fast
tau <- 0.0001
vec <- runif(indiv)
beta <- runif(1)
Print(system.time(S <- solveRelMat(R, tau=tau, vec=vec, betahat=beta)))
print(system.time({r <- solve(R + diag(indiv) * tau, vec);
                   y <- as.vector(R %*% r + beta)}))
stopifnot(all.equal(S$rest, r))
stopifnot(all.equal(S$yhat, y))
```

`RFoptions` *Setting control arguments*

Description

`RFoptions` sets and returns control arguments for diverse packages (**miraculix**, **RandomFields**).

`RFoptions` should not be used within parallelizing R commands such as `mclapply` in package **parallel**.

Details

The specific parameters for **miraculix** are the following. See `RFoptions` in **RandomFieldsUtils** for further options.

any2bit logical. If `TRUE` then always the most time efficient code is used among

- `TwoBit` (no SIMD needed)
- `Packed` (SSE2 needed)
- `Shuffle` (SSSE3 needed)
- `Shuffle256` (AVX2 needed)

whatever is available.

Default : `FALSE`. This value might change to `TRUE` in future.

centered logical or numerical. If `TRUE` the P matrix is substracted before the crossproduct of the the SNP matrix is calculated, see `relationshipMatrix` for the P matrix.

If numeric, then the length of this vector must equal the number of SNPs per individual. Then this vector is substracted for each individual. Furthermore, `normalized` is FALSE. As the size of `centered` can be large, this vector is never returned by `RFoption()`; instead `NA` is returned.

Note that `centered` also sets the value of `normalized`.

Default : `TRUE`

`cores` Number of cores for multicore algorithms.

digits OBSOLETE. scalar. If `digits` is negative no rounding is performed. Else the matrix P when calculating the relationsship matrix $(M-P)^T(M-P)$ is rounded to the given number of absolute (not significant) digits.

Default : `3.0`.

normalized logical. If `TRUE` the relationship matrix is normalized by σ^2, see `relationshipMatrix`.

Its value is set to the value of `centered` whenever the value of `centered` is changed. So `normalized` must be set always after `centered`, e.g. `RFoptions(centered=TRUE,normalized=FALSE)`, but not `RFoptions(normalized=FALSE,centered=TRUE)`.

Default : `TRUE`

snpcoding integer. Possible values are

`Shuffle`	two bit mini hash table based on SSSE3
`Shuffle256`	two bit mini hash table based on AVX2
`Packed`	4-bit integer arithmetic based on SSE2
`Packed256`	4-bit integer arithmetic based on AVX2
`Multiply`	16-bit integer arithmetic based on SSE2
`Multiply256`	16-bit integer arithmetic based on AVX2
`Twobit`	two bit hash table
`Threebit`	three bit hash table
`Hamming2`	method used in PLINK
`Hamming3`	method used in PLINK
`AutoCoding`	method is chosen by the programme itself
`NoSNPcoding`	no coding, i.e. 32 bit integer
`NoSNPcodingR`	No coding: 32 bit integer, R code. Only for testing purposes.
`NoSNPcodingAVX`	No coding: AVX implementation if available (double precision or integer).

In `for` loops that run through all available methods the constants `FirstGenuineMethod` and `LastGenuineMethod` might be useful.

In case of the package **MoPBS** or if interest is in the 2 bit methods only, use the constants `FirstMoBPSmethod` and `LastMoBPSmethod`.

In case the names of the method is needed, use `SNPCODING_NAMES[snp_coding + 1]`.

Default : `Shuffle`

returnsigma logical. Whether σ^2 shall be also returned when the relationship matrix is calculated.

Value

NULL if any argument is given, and the full list of arguments, otherwise.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de> http://ms.math.uni-mannheim.de/de/publications/software

See Also

RFoptions,

Examples

```
RFoptions()$genetics
```

---

scanning *Scan Statistics*

Description

The function implements the scan statistics method of Kabluchko and Spodarev (2009), Theorem 3.1.

Usage

```
scanning(pos, freq, file, tuningUnits, alpha = 0.1, coarsening = 1,
         minscans=0, maxscans = 0, sumscan = FALSE, perSNP = TRUE,
         colname , n, threshold, collect=!old.def, old.def=FALSE,
         max.intervals = length(alpha) * 100000,
         max.basepair.distance = 50000, exclude.negative.at.boundary = TRUE,
         maximum = TRUE, mean.freq, sd.freq, mean.n)

scan.statistics(file, tuningUnits, alpha=c(0.05, 0.01), repet=1000,
                coarsening = 1,
                minscans=0, maxscans=0, sumscan = FALSE, perSNP=TRUE,
                colname, n, return.simu = FALSE,
                debug = FALSE, formula = FALSE,
                old.def=FALSE,
                max.intervals = length(alpha) * 100000,
                max.basepair.distance = 50000,
                exclude.negative.at.boundary = TRUE,
                pos, freq)
```

Arguments

`pos, freq`	alternatively to the `file` name, two vectors, `pos` and `freq` might be given.
`file`	filename or list. The rda file must contain the variables `pos`, `freq`, `colname`, and `n`. Or it is a list with the same named elements. If the extension of the filename is 'bed', the behaviour of the programme is different, see the details
`tuningUnits`	real number. The value 0 codes the case of Theorem 3.1 in Kabluchko and Spordarev (2009). A positive value codes the case of Theorem 2.1 (which is very much preferred). The case of Theorem 3.2 does not suit, hence is not coded. Good values for `tuningUnits` seem to be around 0.85. Note that first, the frequencies are standardized. Then `tuningUnits`$*$`mean`$(n)/n$ is substracted.
`alpha`	level(s) of testing. The levels should decrease.
`coarsening`	integer. If the value is larger than 1 then the data are first windower'ed by `length=coarsening`. This is important to do if the data are fine scaled!
`repet`	The number of simulation to determine the threshold(s) for testing in `scan.statistics`; see also `formula`. Should be at least 100 better 1000.
`minscans,maxscans`	integers. The minimunm and maximum length of the window, respectively. If non-positive the window sizes are not restricted from below or above, respectively.
`sumscan`	logical. If `TRUE` the old style picture appears. Otherwise the relative number of significant intervals containing a certain point is shown.
`perSNP`	logical. If `TRUE` then the test is based on SNPs as units. If `FALSE` the test is based on basepairs (not programmed yet).
`colname`	the column of the data frame that gives the relative frequencies. The default name (i.e., if missing) is "HeterAB". Alternatively `colname` is a number indicating the respective column. In case the extension of the filename equals 'bed', the behaviour is different, see Details.
`n`	The number of individuals, the data are based on. Usually that number is determined automatically, but might be given for safety explicitely
`return.simu`	logical. to do
`debug`	logical or 2. If not `FALSE` important data are saved on the disk. If `debug == 2` pictures of each simulation are shown. [to do in more detail]
`threshold`	`scanning` counts the number of intervals found above the given `threshold`. `threshold` is an alternative to `alpha` and is used instead of `alpha` if both are given. This threshold is applied to the standardized frequency data. A value around 0.8 seems to be appropriate for Christian's data whereas values around 18 are appropriate for Amanda's data.
`collect`	`scanning` can be used in two ways. If `collect=FALSE` essentially only the scan statistic is determined. If `collect=TRUE` then also all the intervalls are determined that are considered to be significant at the given `alpha` levels.

`old.def` logical. If `TRUE` all the tiny snippets that have not been agglutinated yet, are also returned. If `TRUE` it takes a lot of memory.

Further, if `TRUE`, negative (modified) values are allowed at the borders of an interval.

Finally, if `TRUE` the parameters `max.intervals`, `max.basepair.distance`, `exclude.negative.at.bour` are not considered.

`max.intervals` [only if `old.def=FALSE`]

As the number of intervals is determined dynamically, the total number of significant intervals cannot be determined in advance. To economise a lot of copying, an upper threshold is given by the user. 100000 for each level should be large enough. If not, please contact the author.

`max.basepair.distance`

[only if `old.def=FALSE`] if a basepair distance is larger than `max.basepair.distance` then the significant areas are considered as two separate areas.

`exclude.negative.at.boundary`

logical. If `TRUE` negative values at boundaries are not allowed. I.e. each significant area starts and ends with a positive modified frequency.

`maximum` logical. MISSING DOC

`mean.freq` If given, `mean.freq` overwrites `mean(freq)`

`sd.freq` If given, `sd.freq` overwrites `sd(freq)`

`mean.n` If given, `mean.n` overwrites `mean(n)`

`formula` if `formula=TRUE` then the formula of Kabluchko and Spodarev (2009) is used in `scan.statistics`. Otherwise, a `repet` number of simulations under the null hypothesis are performed to get the threshold right.

Details

The ideas for the code are taken from Kabluchko and Spordarev (2009) although the values are not calculated from the respective theorems. Instead, values are obtained by simulation in a procedure similar to Bootstrapping.

In case the file is a bed-file, the following differences to the standard behaviour appears:

1. `colname` must be of the form `c(pos=,freq=,n=)` with default value `c(pos=3,freq=4,n=5)`
2. the sign of the frequency is changed
3. it is not checked whether the frequencies * n equals an integer number

Value

`scanning` returns invisibly a list that contains always

file, pos, freq, tuningUnits, alpha, n, maxscans, perSNP the input data

above.threshold the number of intervals showing a total sum larger than the given `threshold`.

threshold corresponding to alpha, if not given explicitely

maximum the maximum value reached scanning over all windows

if `collect=TRUE` then the list also contains

areas matrix of three rows containing information of all the (overlapping) intervals where the sums exceeds the `thresholds`. Each interval is given by a column. First row: left end point of the interval. Second row: right end point of the interval. Third interval: maximum number of threshold that are passed.

values the sums that correspond to the maxima in `areas`

significant.areas list of matrices. For each `threshold`, all the overlapping intervals are joined that overlap, so that non-overlapping intervals are finally obtained.

Message whether the null hypothesis is rejected at the lowest `alpha` level.

`scan.statistics` returns invisibly a list containing all elements of `scanning` for `collect=TRUE`. Additionally, it contains

maxima the maxima of `repet` simulated data if `formula=FALSE`

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>

References

Kabluchko, Z. and Spodarev, E. (2009) Scan statistics of Levy noises and marked empirical processes. *Adv. Appl. Probab.* **41**, 13-37.

Examples

```
if (interactive()) {
  n <- 30
  loci <- 9000
  positions <- 25000:15000000
} else {
  n <- 3
  loci <- 900
  positions <- 2500:1500000
}
pos <- sort(sample(positions, loci))
freq <- rpois(loci, lambda=0.3) / n

alpha <- c(0.1, 0.05, 0.01)
s <- scan.statistics(n=n, pos=pos, freq=freq, repet=100,
                     tuningUnits=0.65, alpha=alpha)
str(s)
```

`vector012matrix` *multiplication from left of 012 vector with a matrix*

Description

`vector012matrix` and `matrixvector012` multiply a real-valued matrix from left and right with a vector that contains only the values 0,1,2, respectively. For larger matrices (greater than 25×25) the functions are 3 to 10 times faster than the matrix multiplication `%*%`.

This function is not based on `RFoptions()$genetics$snpcoding`.

Usage

```
vector012matrix(v, M)
matrixvector012(M, v)
```

Arguments

`v`	an integer valued with values 0,1,2 only. Anything different from 1 and 2 is treated as 0.
`M`	a real-valued matrix whose size matches `v`

Value

The two function `vector012matrix` and `matrixvector012` return a vector of length `ncol(M)` and `nrow(M)`, respectively.

Author(s)

Martin Schlather, <`schlather@math.uni-mannheim.de`>

See Also

`vectorGeno`

`relationshipMatrix`

Examples

```
set.seed(0)

n <- 800
m <- 800

vl <- sample(0:2, m, replace = TRUE)
vr <- sample(0:2, n, replace = TRUE)
M <- matrix(1 : (n * m), ncol=n) + 0.0

## v1 and v2 are the same
v1 <- M %*% vr
v2 <- matrixvector012(M, vr)
stopifnot(all(v1 == v2))

## v1 and v2 are the same
v1 <- vl %*% M
v2 <- vector012matrix(vl, M)
```

```
stopifnot(all(v1 == v2))

## matrixvector012 is 3 to 15 times faster for larger matrices
N <- 1 + as.integer(100000000 / n^2)
print(system.time( for (i in 1:N) M %*% vr ))
print(system.time( for (i in 1:N) matrixvector012(M, vr) )) # much faster




## vector012matrix is 3 to 10 times faster for larger matrices
print(system.time(for (i in 1:N) vl %*% M ))
print(system.time( for (i in 1:N) vector012matrix(vl, M) )) # much faster
```

---

vectorGeno — *Multiplication of a vector to a compressed SNP matrix*

Description

vectorGeno multiplies a vector from the left onto a compressed SNP matrix.

genoVector does it from the right.

Usage

```
vectorGeno(V, SNPxIndiv, do.centering=FALSE, decode=TRUE)
genoVector(SNPxIndiv, V, do.centering=FALSE)
```

Arguments

SNPxIndiv	a compressed SNP (genotype) vector or matrix obtained from genomicmatrix. Uncoded SNP matrix is also possible.
do.centering	not programmed yet.
decode	Logical. This option only applies when RFoptions()$genetics$snpcoding equals Shuffle256, Shuffle, Packed256, Packed, Multiply, or TwoBit. If TRUE the matrix is decoded and standard matrix multiplication performed afterwards. This is currently faster than to operate on the coded version (decode=FALSE), but takes (considerably) more memory.
V	numerical vector

Details

Let G be a (SNP$\times$ Indiv) matrix. vectorGeno and genoVector return VG and GV, respectively.

Value

vector of length nrow(SNPxIndiv) and ncol(SNPxIndiv) for vectorGeno and genoVector, respectively.

Author(s)

Martin Schlather, <schlather@math.uni-mannheim.de>, http://ms.math.uni-mannheim.de

Examples

```
require(RandomFieldsUtils)
set.seed(0)

indiv <- 1 + sample(500, 1)
snps <- indiv * 2^sample(7, 1)
snps <- indiv * 100
M <- matrix(ncol=indiv, sample(0:2, indiv * snps, replace=TRUE))
print(system.time(CM <- genomicmatrix(M)))


## V %*% G
Vl <- runif(snps)
print(system.time(VM1 <- vectorGeno(Vl, CM))) # 1.2x slower than '%*%'
print(system.time(VM <- as.vector(Vl %*% M)))
stopifnot(all.equal(as.double(VM), as.double(VM1)))

## G %*% V
Vr <- runif(indiv)
print(system.time(MV1 <- genoVector(CM, Vr))) ## 3x faster than '%*%'
print(system.time(MV <- as.vector(M %*% Vr)))
stopifnot(all.equal(as.double(MV), as.double(MV1)))
```

Windower *Windower*

Description

averages over running windows

Usage

```
windower(data, length=20000, step=length/2, start=0, n.min=0, na.rm=TRUE,
                       what=c("mean", "var", "sd", "min", "max", "median",
                               "sum"))
```

Arguments

`data`	data frame from a '.bed' file. The first column indicates the chromosome. The second and the third row give starting and end point [in base pairs]. The 4th column gives the values. All the other columns will be ignored
`length`	length in base pairs of the window
`step`	positive integer. shift of the window by `step` base pairs
`start`	the base pair position where the very first window starts.
`n.min`	the required minimal number of SNPs in the window. If there are less SNPs inside, this window is not reported.
`na.rm`	logical. if `TRUE` then na.rm are just ignored.
`what`	string. Name of the function that should be 'windowed'; `"mean"` is standard.

Value

It returns a matrix with 4 columns: the first and the second column contain the starting and end point of the window in '.bed' coding. The third column gives the mean (or variance etc). The 4th column gives the number of values the mean is based on.

Author(s)

Martin Schlather, `<schlather@math.uni-mannheim.de>`

Examples

```
loci <- 10000
pos <- sort(sample(10^4:10^6, loci))
pos2 <- pos + 1
freq <- runif(loci)^5
data <- data.frame(V1=rep(1, loci), V2=pos, V3=pos2,  V4=freq)


win.mean <- windower(data, n.min=25)
head(win.mean)

win.var <- windower(data, n.min=25, what="var")
head(win.var)

win.sd <- windower(data, n.min=25, what="sd")
head(win.sd)

win.min <- windower(data, n.min=0, what="min")
head(win.min)

win.max <- windower(data, n.min=0, what="max")
head(win.max)

win.median <- windower(data, n.min=0, what="median")
head(win.median)
```

Index