GWA likelihood methods

Generated by Doxygen 1.8.20

Chapter 1

Main Page

This is an R package to perform genome-wide association studies on replicated data. It implements the EMMAX method, including cases where each genotype has more than one observation. In addition, if there are several traits it pays to analyze them together. While the GWA itself is done separately for each trait, some components of the mixed model and the SNP regression are common among traits and thus some time savings can be achieved by considering traits from the same data set together.

The SNP regression portion is multi-threaded. This is implemented so as not to interfere with multi-threading of linear algebra libraries used in R. More details can be found in the documentation.

To install, make sure you have the `devtools` package on your system, and then run `install_github("tonymugen/GWAlikeMeth")`. It should work under any Unix-like system, but has not been tested under Windows.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 6

Module Documentation

6.1 Arithmetic operators

Functions

- Matrix BayesicSpace::operator∗ (const double &scal, const Matrix &m)

  Scalar-matrix product.
- Matrix BayesicSpace::operator+ (const double &scal, const Matrix &m)

  Scalar-matrix addition.

6.1.1 Detailed Description

Scalar by matrix addition and multiplication operators to maintain commutativity.

6.1.2 Function Documentation

6.1.2.1 operator∗()

```
Matrix BayesicSpace::operator* (
            const double & scal,
            const Matrix & m )
```

Scalar-matrix product.

Parameters

in	*scal*	scalar
in	*m*	matrix

Returns

Matrix result

6.1.2.2 operator+()

```
Matrix BayesicSpace::operator+ (
            const double & scal,
            const Matrix & m )
```

Scalar-matrix addition.

Parameters

in	*scal*	scalar
in	*m*	matrix

Returns

Matrix result

Chapter 7

Class Documentation

7.1 BayesicSpace::EmmREML Class Reference

EMMA REML functor class.

`#include <likeMeth.hpp>`

Public Member Functions

- EmmREML ()
  Default constructor.
- EmmREML (const Matrix &etaSq, const vector< double > &lambda, const size_t &jCol)
  Constructor.
- ~EmmREML ()
  Destructor.
- EmmREML (const EmmREML &)=delete
  Copy constructor (not implemented)
- EmmREML & operator= (const EmmREML &)=delete
  Assignement operator (not implemented)
- double operator() (const double &delta)
  Function operator.
- void setColID (const size_t &j)
  Set column index.

7.1.1 Detailed Description

EMMA REML functor class.

Class that implements the REML function from Kang *et al* (2008). Used in the likelihood maximization step.

7.1.2 Constructor & Destructor Documentation

7.1.2.1 EmmREML()

```
BayesicSpace::EmmREML::EmmREML (
            const Matrix & etaSq,
            const vector< double > & lambda,
            const size_t & jCol )  [inline]
```

Constructor.

Sets up the object.

Parameters

in	*etaSq*	address of a Matrix object with η^2
in	*lambda*	address of a Matrix object with λ
in	*jCol*	phenotype column index

7.1.3 Member Function Documentation

7.1.3.1 operator()()

```
double EmmREML::operator() (
            const double & delta )
```

Function operator.

Does the restricted likelihood calculation for each value of $\delta = \frac{\sigma_e^2}{\sigma_g^2}$ and the given columns of the η^2 and λ matrices.

Parameters

in	*delta*	δ value

7.1.3.2 setColID()

```
void BayesicSpace::EmmREML::setColID (
            const size_t & j )  [inline]
```

Set column index.

Parameters

in	*j*	column index

The documentation for this class was generated from the following files:

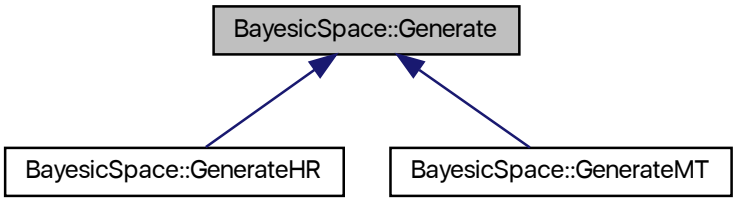
- src/likeMeth.hpp
- src/likeMeth.cpp

7.2 BayesicSpace::Generate Class Reference

Abstract base random number class.

`#include <random.hpp>`

Inheritance diagram for BayesicSpace::Generate:



Public Member Functions

- virtual ~Generate ()
  Destructor.
- virtual uint64_t ranInt () const =0
  Generate a (pseudo-)random 64-bit unsigned integer.

Protected Member Functions

- Generate ()
  Protected default constructor.
- Generate (const Generate &old)=default
  Protected copy constructor.
- Generate (Generate &&old)=default
  Protected move constructor.
- Generate & operator= (const Generate &old)=default
  Protected copy assignment operator.
- Generate & operator= (Generate &&old)=default
  Protected move assignment.

7.2.1 Detailed Description

Abstract base random number class.

Provides the interface for random or pseudorandom (depending on derived class) generation. For internal use by the RanDraw interface class.

7.2.2 Constructor & Destructor Documentation

7.2.2.1 Generate() [1/2]

```
BayesicSpace::Generate::Generate (
            const Generate & old )  [protected], [default]
```

Protected copy constructor.

Parameters

in	*old*	object to copy

7.2.2.2 Generate() [2/2]

```
BayesicSpace::Generate::Generate (
            Generate && old )  [protected], [default]
```

Protected move constructor.

Parameters

in	*old*	object to move

7.2.3 Member Function Documentation

7.2.3.1 operator=() [1/2]

```
Generate& BayesicSpace::Generate::operator= (
            const Generate & old )  [protected], [default]
```

Protected copy assignment operator.

Parameters

in	*old*	object to copy

7.2.3.2 operator=() [2/2]

```
Generate& BayesicSpace::Generate::operator= (
            Generate && old )  [protected], [default]
```

Protected move assignment.

Parameters

in	*old*	object to move

7.2.3.3 ranInt()

```
virtual uint64_t BayesicSpace::Generate::ranInt ( ) const  [pure virtual]
```

Generate a (pseudo-)random 64-bit unsigned integer.

Returns

random or pseudo-random 64-bit unsigned integer

Implemented in BayesicSpace::GenerateMT, and BayesicSpace::GenerateHR.

The documentation for this class was generated from the following file:

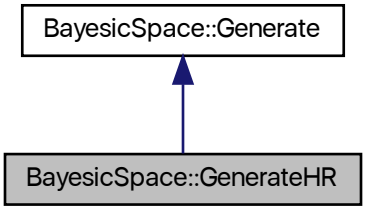
- src/random.hpp

7.3 BayesicSpace::GenerateHR Class Reference

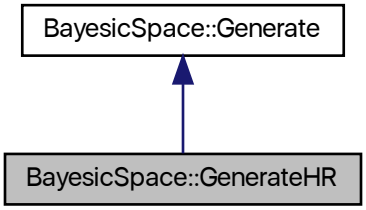
Hardware random number generating class.

`#include <random.hpp>`

Inheritance diagram for BayesicSpace::GenerateHR:



Collaboration diagram for BayesicSpace::GenerateHR:



Public Member Functions

- GenerateHR ()

  Default constructor.
- ~GenerateHR ()

  Destructor.
- GenerateHR (const GenerateHR &old)=default

  Copy constructor.
- GenerateHR (GenerateHR &&old)=default

  Move constructor.

- GenerateHR & operator= (const GenerateHR &old)=default

  Copy assignment operator.
- GenerateHR & operator= (GenerateHR &&old)=default

  Move assignment.
- uint64_t ranInt () const

  Generate a random 64-bit unsigned integer.

Additional Inherited Members

7.3.1 Detailed Description

Hardware random number generating class.

Generates random deviates from a number of distributions, using hardware random numbers (*RDRAND* processor instruction). Health of the RDRAND generator is tested every time a new number is required. Throws a `string` object "RDRAND_failed" if the test fails. The implementation of random 64-bit integer generation follows `Intel's suggestions`.

7.3.2 Constructor & Destructor Documentation

7.3.2.1 GenerateHR() [1/2]

```
BayesicSpace::GenerateHR::GenerateHR (
            const GenerateHR & old )  [default]
```

Copy constructor.

Parameters

in	*old*	object to copy

7.3.2.2 GenerateHR() [2/2]

```
BayesicSpace::GenerateHR::GenerateHR (
            GenerateHR && old )  [default]
```

Move constructor.

Parameters

in	*old*	object to move

7.3.3 Member Function Documentation

7.3.3.1 operator=() [1/2]

```
GenerateHR& BayesicSpace::GenerateHR::operator= (
            const GenerateHR & old )  [default]
```

Copy assignment operator.

Parameters

in	*old*	object to copy

7.3.3.2 operator=() [2/2]

```
GenerateHR& BayesicSpace::GenerateHR::operator= (
            GenerateHR && old )  [default]
```

Move assignment.

Parameters

in	*old*	object to move

7.3.3.3 ranInt()

```
uint64_t GenerateHR::ranInt ( ) const  [virtual]
```

Generate a random 64-bit unsigned integer.

Monitors the health of the CPU random number generator and throws a `string` object "RDRAND_failed" if a failure is detected after ten tries.

Returns

digital random 64-bit unsigned integer

Implements BayesicSpace::Generate.

The documentation for this class was generated from the following files:

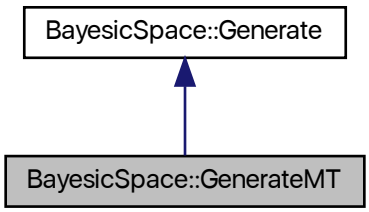
- src/random.hpp
- src/random.cpp

7.4 BayesicSpace::GenerateMT Class Reference

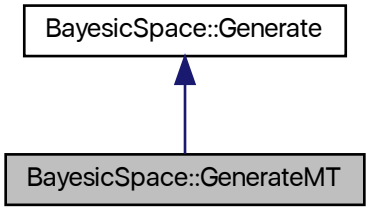
Pseudo-random number generator.

`#include <random.hpp>`

Inheritance diagram for BayesicSpace::GenerateMT:



Collaboration diagram for BayesicSpace::GenerateMT:



Public Member Functions

- GenerateMT ()

  Default constructor.
- ~GenerateMT ()

  Protected destructor.
- GenerateMT (const GenerateMT &old)=default

  Copy constructor.
- GenerateMT (GenerateMT &&old)=default

  Move constructor.
- GenerateMT & operator= (const GenerateMT &old)=default

  Copy assignment operator.
- GenerateMT & operator= (GenerateMT &&old)=default

  Move assignment.
- uint64_t ranInt () const

  Generate a pseudo-random 64-bit unsigned integer.

Protected Attributes

- uint64_t mt_ [312]

  Generator state array.
- size_t mti_

  State of the array index.
- uint64_t x_

  Current state.

Static Protected Attributes

- static const uint16_t n_ = 312

  Degree of recurrence.
- static const uint16_t m_ = 156

  Middle word.
- static const uint64_t um_ = static_cast<uint64_t>(0x7FFFFFFF)

  Most significant 33 bits.
- static const uint64_t lm_ = static_cast<uint64_t>(0xFFFFFFFF80000000)

  Least significant 31 bits.
- static const uint64_t b_ = static_cast<uint64_t>(0x71D67FFFEDA60000)

  Tempering bitmask.
- static const uint64_t c_ = static_cast<uint64_t>(0xFFF7EEE000000000)

  Tempering bitmask.
- static const uint64_t d_ = static_cast<uint64_t>(0x5555555555555555)

  Tempering bitmask.
- static const uint32_t l_ = 43

  Tempering shift.
- static const uint32_t s_ = 17

  Tempering shift.

- static const uint32_t t_ = 37

  Tempering shift.
- static const uint32_t u_ = 29

  Tempering shift.
- static const uint64_t alt_ [2] = {static_cast<uint64_t>(0), static_cast<uint64_t>(0xB5026F5AA96619E9)}

  Array of alternative values for the twist.

Additional Inherited Members

7.4.1 Detailed Description

Pseudo-random number generator.

An implementaiton of the 64-bit MT19937 ("Mersenne Twister") **[matsumoto98a]** pseudo-random number generator (PRNG). The constructor automatically seeds the PRNG. The implementation was guided by the reference code posted by the authors.

7.4.2 Constructor & Destructor Documentation

7.4.2.1 GenerateMT() [1/3]

```
GenerateMT::GenerateMT ( )
```

Default constructor.

Seeds the PRNG with a call to the *RDTSC* instruction.

7.4.2.2 GenerateMT() [2/3]

```
BayesicSpace::GenerateMT::GenerateMT (
            const GenerateMT & old )  [default]
```

Copy constructor.

Parameters

in	*old*	object to copy

7.4.2.3 GenerateMT() [3/3]

```
BayesicSpace::GenerateMT::GenerateMT (
            GenerateMT && old )  [default]
```

Move constructor.

Parameters

in	*old*	object to move

7.4.3 Member Function Documentation

7.4.3.1 operator=() [1/2]

```
GenerateMT& BayesicSpace::GenerateMT::operator= (
            const GenerateMT & old )  [default]
```

Copy assignment operator.

Parameters

in	*old*	object to copy

7.4.3.2 operator=() [2/2]

```
GenerateMT& BayesicSpace::GenerateMT::operator= (
            GenerateMT && old )  [default]
```

Move assignment.

Parameters

in	*old*	object to move

7.4.3.3 ranInt()

```
uint64_t GenerateMT::ranInt ( ) const  [virtual]
```

Generate a pseudo-random 64-bit unsigned integer.

Returns

pseudo-random 64-bit unsigned integer

Implements BayesicSpace::Generate.

The documentation for this class was generated from the following files:

- src/random.hpp
- src/random.cpp

7.5 BayesicSpace::Matrix Class Reference

Test matrix class.

```
#include <locMatrix.hpp>
```

Public Member Functions

- Matrix ()
  Default constructor.
- Matrix (const size_t &nrow, const size_t &ncol)
  Allocating constructor.
- Matrix (const double &val, const size_t &nrow, const size_t &ncol)
  Initializing constructor.
- Matrix (const double inArr[ ], const size_t &nrow, const size_t &ncol)
  Constructor from C array.
- Matrix (const vector< double > &inVec, const size_t &nrow, const size_t &ncol)
  Constructor from C++ vector.
- Matrix (const string &fileName, const char &delim)
  Constructor from file.
- ~Matrix ()
  Destructor.
- Matrix (const Matrix &inMat)
  Copy constructor.
- Matrix & operator= (const Matrix &inMat)
  Copy assignment operator.
- Matrix (Matrix &&inMat)
  Move constructor.

- Matrix & operator= (Matrix &&inMat)

  Move assignment operator.
- size_t getNrows () const

  Access to number of rows.
- size_t getNcols () const

  Access to number of columns.
- double getElem (const size_t &iRow, const size_t &jCol) const

  Access to an element.
- void setElem (const size_t &iRow, const size_t &jCol, const double &input)

  Set element to a value.
- void setCol (const size_t jCol, const vector< double > data)

  Copy data from a vector to a column.
- void resize (const size_t &nrow, const size_t &ncol)

  Resize matrix.
- void save (const string &outFileName) const

  Save matrix contents to a tab-delimited file.
- void vectorize (vector< double > &out) const

  Vectorize the matrix.
- void chol ()

  In-place Cholesky decomposition.
- void chol (Matrix &out) const

  Copy Cholesky decomposition.
- void cholInv ()

  In-place Cholesky inverse.
- void cholInv (Matrix &out) const

  Copy Cholesky inverse.
- void svd (Matrix &U, vector< double > &s)

  Perform SVD.
- void svdSafe (Matrix &U, vector< double > &s) const

  Perform "safe" SVD.
- void eigen (const char &tri, Matrix &U, vector< double > &lam)

  All eigenvalues and vectors of a symmetric matrix.
- void eigen (const char &tri, const size_t &n, Matrix &U, vector< double > &lam)

  Some eigenvalues and vectors of a symmetric matrix.
- void eigenSafe (const char &tri, Matrix &U, vector< double > &lam)

  All eigenvalues and vectors of a symmetric matrix ("safe")
- void eigenSafe (const char &tri, const size_t &n, Matrix &U, vector< double > &lam)

  Some eigenvalues and vectors of a symmetric matrix ("safe")
- void premultZ (const Matrix &Z)

  In-place multiply by a design matrix from the left.
- void premultZ (const Matrix &Z, Matrix &out) const

  Multiply by a design matrix from the left and copy result.
- void premultZt (const Matrix &Z)

  In-place multiply by the transpose of a design matrix from the left.
- void premultZt (const Matrix &Z, Matrix &out) const

  Multiply by the transpose of a design matrix from the left and copy result.
- void postmultZ (const Matrix &Z)

In-place multiply by a design matrix from the right.

- void postmultZ (const Matrix &Z, Matrix &out) const

  Multiply by a design matrix from the right and copy result.
- void postmultZt (const Matrix &Z)

  In-place multiply by the transpose of a design matrix from the right.
- void postmultZt (const Matrix &Z, Matrix &out) const

  Multiply by the transpose of a design matrix from the right and copy result.
- void syrk (const char &tri, const double &alpha, const double &beta, Matrix &C) const

  Inner self crossproduct.
- void tsyrk (const char &tri, const double &alpha, const double &beta, Matrix &C) const

  Outer self crossproduct.
- void symm (const char &tri, const char &side, const double &alpha, const Matrix &symA, const double &beta, Matrix &C) const

  Multiply by symmetric matrix.
- void symc (const char &tri, const double &alpha, const Matrix &X, const size_t &xCol, const double &beta, vector< double > &y) const
- void gemm (const bool &transA, const double &alpha, const Matrix &A, const bool &transB, const double &beta, Matrix &C) const

  General matrix multiplication.
- void gemc (const bool &trans, const double &alpha, const Matrix &X, const size_t &xCol, const double &beta, vector< double > &y) const

  Multiply a general matrix by a column of another matrix.
- Matrix colShuffle () const

  Shuffle columns.
- Matrix rowShuffle () const

  Shuffle rows.
- Matrix operator* (const Matrix &m) const

  Hadamard matrix product.
- Matrix operator* (const double &scal) const

  Matrix-scalar product.
- Matrix operator/ (const Matrix &m) const

  Entrywise matrix division.
- Matrix operator/ (const double &scal) const

  Matrix-scalar division.
- Matrix operator+ (const Matrix &m) const

  Entrywise matrix addition.
- Matrix operator+ (const double &scal) const

  Matrix-scalar addition.
- Matrix operator- (const Matrix &m) const

  Entrywise matrix subtraction.
- Matrix operator- (const double &scal) const

  Matrix-scalar subtraction.
- Matrix & operator+= (const double &scal)

  Matrix-scalar compound addition.
- Matrix & operator*= (const double &scal)

  Matrix-scalar compound product.
- Matrix & operator-= (const double &scal)

  Matrix-scalar compound subtraction.

- Matrix & operator/= (const double &scal)

  Matrix-scalar compound division.
- void rowMeans (vector< double > &means) const

  Row means.
- void colMeans (vector< double > &means) const

  Column means.
- void rowSums (vector< double > &sums) const

  Row sums.
- void colSums (vector< double > &sums) const

  Column sums.
- void rowMultiply (const vector< double > &scalars)

  Multiply rows by a vector.
- void rowMultiply (const double &scalar, const size_t &iRow)

  Multiply a row by a scalar.
- void colMultiply (const vector< double > &scalars)

  Multiply columns by a vector.
- void colMultiply (const double &scalar, const size_t &jCol)

  Multiply a column by a scalar.
- void rowDivide (const vector< double > &scalars)

  Divide rows by a vector.
- void rowDivide (const double &scalar, const size_t &iRow)

  Divide a row by a scalar.
- void colDivide (const vector< double > &scalars)

  Divide columns by a vector.
- void colDivide (const double &scalar, const size_t &jCol)

  Divide a column by a scalar.
- void rowAdd (const vector< double > &scalars)

  Add a vector to rows.
- void rowAdd (const double &scalar, const size_t &iRow)

  Add a scalar to a row.
- void colAdd (const vector< double > &scalars)

  Add a vector to columns.
- void colAdd (const double &scalar, const size_t &jCol)

  Add a scalar to a column.
- void rowSub (const vector< double > &scalars)

  Subtract a vector from rows.
- void rowSub (const double &scalar, const size_t &iRow)

  Subtract a scalar from a row.
- void colSub (const vector< double > &scalars)

  Subtract a vector from columns.
- void colSub (const double &scalar, const size_t &jCol)

  Subtract a scalar from a column.
- void appendCol (const Matrix &cols)

  Append columns of a matrix.
- void appendRow (const Matrix &rows)

  Append rows of a matrix.
- void dropLeftCols (const size_t &newFirst)

Drop left columns.

- void dropRightCols (const size_t &newLast)

  Drop right columns.
- void dropTopRows (const size_t &newTop)

  Drop top rows.
- void dropBottomRows (const size_t &newBottom)

  Drop bottom rows.

Friends

- Matrix operator+ (const double &scal, const Matrix &m)

  Scalar-matrix addition.
- Matrix operator* (const double &scal, const Matrix &m)

  Scalar-matrix product.

7.5.1 Detailed Description

Test matrix class.

A test matrix class. The data type is *double*. The matrix is column-major to comply with LAPACK and BLAS routines. Columns and rows are base-0. Range checking is done unless the flag -DLMRG_CHECK_OFF is set at compile time.

TODO: transpose TODO: make thread-safe with C++-11 facilities

7.5.2 Constructor & Destructor Documentation

7.5.2.1 Matrix() [1/7]

```
Matrix::Matrix (
            const size_t & nrow,
            const size_t & ncol )
```

Allocating constructor.

Allocates memory but does not initialize

Parameters

in	*nrow*	number of rows
in	*ncol*	number of columns

7.5.2.2 Matrix() [2/7]

```
Matrix::Matrix (
            const double & val,
            const size_t & nrow,
            const size_t & ncol )
```

Initializing constructor.

Allocates memory and initializes the elements to the given value.

Parameters

in	*val*	initializing value
in	*nrow*	number of rows
in	*ncol*	number of columns

7.5.2.3 Matrix() [3/7]

```
Matrix::Matrix (
            const double inArr[],
            const size_t & nrow,
            const size_t & ncol )
```

Constructor from C array.

Allocates memory and initializes by copying values from the given array. The user is resposible for making sure the input array is long enough.

Parameters

in	*inArr*	initializing array
in	*nrow*	number of rows
in	*ncol*	number of columns

7.5.2.4 Matrix() [4/7]

```
Matrix::Matrix (
            const vector< double > & inVec,
            const size_t & nrow,
            const size_t & ncol )
```

Constructor from C++ vector.

Allocates memory and initializes by copying values from the given vector. Vector can be bigger (but not smaller) than necessary. If so, the beginning $N_{row} \times N_{col}$ elements are used.

Parameters

in	*inVec*	initializing array
in	*nrow*	number of rows
in	*ncol*	number of columns

7.5.2.5 Matrix() [5/7]

```
Matrix::Matrix (
            const string & fileName,
            const char & delim )
```

Constructor from file.

Gets data from a space-delimited file.

Parameters

in	*fileName*	input file name
in	*delim*	column delimiter

7.5.2.6 Matrix() [6/7]

```
Matrix::Matrix (
            const Matrix & inMat )
```

Copy constructor.

Parameters

in	*inMat*	object to be copied

7.5.2.7 Matrix() [7/7]

```
Matrix::Matrix (
            Matrix && inMat )
```

Move constructor.

Parameters

in	*inMat*	object to be moved

7.5.3 Member Function Documentation

7.5.3.1 appendCol()

```
void Matrix::appendCol (
            const Matrix & cols )
```

Append columns of a matrix.

Columns of a provided matrix are appended after the last column of the current object. The object is expanded accordingly. Number of rows does not change.

Parameters

in	*cols*	a *Matrix* object with columns to append

7.5.3.2 appendRow()

```
void Matrix::appendRow (
            const Matrix & rows )
```

Append rows of a matrix.

Rows of a provided matrix are appended after the last row of the current object. The object is expanded accordingly. Number of columns does not change.

Parameters

in	*rows*	a *Matrix* object with rows to append

7.5.3.3 chol() [1/2]

```
void Matrix::chol ( )
```

In-place Cholesky decomposition.

Performs the Cholesky decomposition and stores the resulting matrix in the lower triangle of the same object.

7.5.3.4 chol() [2/2]

```
void Matrix::chol (
            Matrix & out ) const
```

Copy Cholesky decomposition.

Performs the Cholesky decomposition and stores the result in the lower triangle of the provided Matrix object. The original object is left untouched.

Parameters

out	*out*	object where the result is to be stored

7.5.3.5 cholInv() [1/2]

```
void Matrix::cholInv ( )
```

In-place Cholesky inverse.

Computes the inverse of a Cholesky decomposition and stores the resulting matrix in the same object, resulting in a symmetric matrix. The object is assumed to be a Cholesky decomposition already.

7.5.3.6 cholInv() [2/2]

```
void Matrix::cholInv (
            Matrix & out ) const
```

Copy Cholesky inverse.

Computes the inverse of a Cholesky decomposition and stores the result in the provided Matrix object, resulting in a symmetric matrix. The original object is left untouched. The object is assumed to be a Cholesky decomposition already.

Parameters

out	*out*	object where the result is to be stored

7.5.3.7 colAdd() [1/2]

```
void Matrix::colAdd (
            const double & scalar,
            const size_t & jCol )
```

Add a scalar to a column.

Entry-wise addition of a scalar to the given column. The current object is modified.

Parameters

in	*scalar*	scalar to use for addition
in	*jCol*	column index

7.5.3.8 colAdd() [2/2]

```
void Matrix::colAdd (
            const vector< double > & scalars )
```

Add a vector to columns.

Entry-wise addition of a vector to each column. The current object is modified.

Parameters

in	*scalars*	vector of scalars to use for addition

7.5.3.9 colDivide() [1/2]

```
void Matrix::colDivide (
            const double & scalar,
            const size_t & jCol )
```

Divide a column by a scalar.

Entry-wise division of a given column by the provided scalar. The current object is modified.

Parameters

in	*scalar*	scalar to use for division
in	*jCol*	column index

7.5.3.10 colDivide() [2/2]

```
void Matrix::colDivide (
            const vector< double > & scalars )
```

Divide columns by a vector.

Entry-wise division of each column by the provided vector. The current object is modified.

Parameters

in	*scalars*	vector of scalars to use for division

7.5.3.11 colMeans()

```
void Matrix::colMeans (
            vector< double > & means ) const
```

Column means.

Calculates column means and stores them in the provided vector. If vector length is smaller than necessary, the vector is expanded. Otherwise, the first N_{col} elements are used.

Parameters

out	*means*	vector of means

7.5.3.12 colMultiply() [1/2]

```
void Matrix::colMultiply (
            const double & scalar,
            const size_t & jCol )
```

Multiply a column by a scalar.

Entry-wise multiplication of a given column by the provided scalar. The current object is modified.

Parameters

in	*scalar*	scalar to use for multiplication
in	*jCol*	column index

7.5.3.13 colMultiply() [2/2]

```
void Matrix::colMultiply (
            const vector< double > & scalars )
```

Multiply columns by a vector.

Entry-wise multiplication of each column by the provided vector. The current object is modified.

Parameters

in	*scalars*	vector of scalars to use for multiplication

7.5.3.14 colShuffle()

```
Matrix Matrix::colShuffle ( ) const
```

Shuffle columns.

Suffle the columns of the current object and return a matrix with of the same size but column order randomly permuted.

Returns

permuted Matrix object

7.5.3.15 colSub() [1/2]

```
void Matrix::colSub (
            const double & scalar,
            const size_t & jCol )
```

Subtract a scalar from a column.

Entry-wise subtraction of a scalar from the given column. The current object is modified.

Parameters

in	*scalar*	scalar to use for subtraction
in	*jCol*	column index

7.5.3.16 colSub() [2/2]

```
void Matrix::colSub (
            const vector< double > & scalars )
```

Subtract a vector from columns.

Entry-wise subtraction of a vector from each column. The current object is modified.

Parameters

in	*scalars*	vector of scalars to use for subtraction

7.5.3.17 colSums()

```
void Matrix::colSums (
            vector< double > & sums ) const
```

Column sums.

Calculates column sums and stores them in the provided vector. If vector length is smaller than necessary, the vector is expanded. Otherwise, the first N_{col} elements are used.

Parameters

out	*sums*	vector of sums

7.5.3.18 dropBottomRows()

```
void Matrix::dropBottomRows (
            const size_t & newBottom )
```

Drop bottom rows.

Delete rows lower than the one marked by the provided index.

Parameters

in	*newBottom*	index of the new bottom row

7.5.3.19 dropLeftCols()

```
void Matrix::dropLeftCols (
            const size_t & newFirst )
```

Drop left columns.

Delete columns to the left of the one marked by the provided index.

Parameters

in	*newFirst*	index of the new first column

7.5.3.20 dropRightCols()

```
void Matrix::dropRightCols (
            const size_t & newLast )
```

Drop right columns.

Delete columns to the right of the one marked by the provided index.

Parameters

in	*newLast*	index of the new last column

7.5.3.21 dropTopRows()

```
void Matrix::dropTopRows (
            const size_t & newTop )
```

Drop top rows.

Delete rows higher than the one marked by the provided index.

Parameters

in	*newTop*	index of the new top row

7.5.3.22 eigen() [1/2]

```
void Matrix::eigen (
            const char & tri,
            const size_t & n,
            Matrix & U,
            vector< double > & lam )
```

Some eigenvalues and vectors of a symmetric matrix.

Computes top *n* eigenvalues and vectors of a symmetric matrix. Interface to the *DSYEVR* LAPACK routine. This routine is recommended as the fastest (especially for smaller matrices) in LAPACK benchmarks. It is assumed that the current object is symmetric. It is only checked for being square. The data in the relevant triangle are destroyed. If the dimensions of the output matrix and vector are smaller than necessary, they are resized. If they are larger than necessary, only the first N^2 and N elements are used, respectively. For the matrix this means that the first N columns are used if the number of rows is the same as that in the current object. Otherwise, the columns are wrapped around.

Parameters

in	*tri*	triangle ID ('u' for upper or 'l' for lower)
in	*n*	number of largest eigenvalues to compute
out	*U*	matrix of eigenvectors
out	*lam*	vector of eigenvalues in ascending order

7.5.3.23 eigen() [2/2]

```
void Matrix::eigen (
            const char & tri,
            Matrix & U,
            vector< double > & lam )
```

All eigenvalues and vectors of a symmetric matrix.

Interface to the *DSYEVR* LAPACK routine. This routine is recommended as the fastest (especially for smaller matrices) in LAPACK benchmarks. It is assumed that the current object is symmetric. It is only checked for being square. The data in the relevant triangle are destroyed. If the dimensions of the output matrix and vector are smaller than necessary, they are resized. If they are larger than necessary, only the first N^2 and N elements are used, respectively. For the matrix this means that the first N columns are used if the number of rows is the same as that in the current object. Otherwise, the columns are wrapped around.

Parameters

in	*tri*	triangle ID ('u' for upper or 'l' for lower)
out	*U*	matrix of eigenvectors
out	*lam*	vector of eigenvalues in ascending order

7.5.3.24 eigenSafe() [1/2]

```
void Matrix::eigenSafe (
            const char & tri,
            const size_t & n,
            Matrix & U,
            vector< double > & lam )
```

Some eigenvalues and vectors of a symmetric matrix ("safe")

Computes the top *n* eigenvectors and values of a symmetric matrix. Interface to the *DSYEVR* LAPACK routine. This routine is recommended as the fastest (especially for smaller matrices) in LAPACK benchmarks. It is assumed that the current object is symmetric. It is only checked for being square. The data are preserved, leading to some loss of efficiency compared to eigen(). If the dimensions of the output matrix and vector are smaller than necessary, they are resized. If they are larger than necessary, only the first N^2 and N elements are used, respectively. For the matrix this means that the first N columns are used if the number of rows is the same as that in the current object. Otherwise, the columns are wrapped around.

Parameters

in	*tri*	triangle ID ('u' for upper or 'l' for lower)
in	*n*	number of largest eigenvalues to compute
out	*U*	matrix of eigenvectors
out	*lam*	vector of eigenvalues in ascending order

7.5.3.25 eigenSafe() [2/2]

```
void Matrix::eigenSafe (
            const char & tri,
            Matrix & U,
            vector< double > & lam )
```

All eigenvalues and vectors of a symmetric matrix ("safe")

Interface to the *DSYEVR* LAPACK routine. This routine is recommended as the fastest (especially for smaller matrices) in LAPACK benchmarks. It is assumed that the current object is symmetric. It is only checked for being square. The data are preserved, leading to some loss of efficiency compared to eigen(). If the dimensions of the output matrix and vector are smaller than necessary, they are resized. If they are larger than necessary, only the first N^2 and N elements are used, respectively. For the matrix this means that the first N columns are used if the number of rows is the same as that in the current object. Otherwise, the columns are wrapped around.

Parameters

in	*tri*	triangle ID ('u' for upper or 'l' for lower)
out	*U*	matrix of eigenvectors
out	*lam*	vector of eigenvalues in ascending order

7.5.3.26 gemc()

```
void Matrix::gemc (
            const bool & trans,
            const double & alpha,
            const Matrix & X,
            const size_t & xCol,
            const double & beta,
            vector< double > & y ) const
```

Multiply a general matrix by a column of another matrix.

Multiply the *Matrix* object by a specified column of another matrix. An interface for the BLAS *DGEMV* routine. Updates the input vector y

$$y \leftarrow \alpha A X_{\cdot j} + \beta y$$

or

$$y \leftarrow \alpha A^T X_{\cdot j} + \beta y$$

If the output vector is too short it is re-sized, adding zero elements as needed. If it is too long, only the first Nrow(A) elements are modified.

Parameters

in	*trans*	whether A (focal object) should be transposed
in	*alpha*	the α constant
in	*X*	matrix X whose column will be used
in	*xCol*	column of X to be used (0 base)
in	*beta*	the β constant
in,out	*y*	result vector

7.5.3.27 gemm()

```
void Matrix::gemm (
            const bool & transA,
            const double & alpha,
            const Matrix & A,
            const bool & transB,
            const double & beta,
            Matrix & C ) const
```

General matrix multiplication.

Interface for the BLAS *DGEMM* routine. Updates the input/output matrix C

$C \leftarrow \alpha op(A) op(B) + \beta C$

where $op(A)$ is A^T or A if *transA* is true or false, respectively, and similarly for $op(B)$. The object from which the method is called is B. If *C* does not have the right dimensions, it is re-sized and all elements are set to zero before the operation.

Parameters

in	*transA*	whether A should be transposed
in	*alpha*	the α constant
in	*A*	matrix A
in	*transB*	whether B should be transposed
in	*beta*	the β constant
in,out	*C*	the result C matrix

7.5.3.28 getElem()

```
double Matrix::getElem (
            const size_t & iRow,
            const size_t & jCol ) const
```

Access to an element.

Parameters

in	*iRow*	row number
in	*jCol*	column number

Returns

double element value

7.5.3.29 getNcols()

```
size_t BayesicSpace::Matrix::getNcols ( ) const  [inline]
```

Access to number of columns.

Returns

size_t number of columns

7.5.3.30 getNrows()

```
size_t BayesicSpace::Matrix::getNrows ( ) const  [inline]
```

Access to number of rows.

Returns

size_t number of rows

7.5.3.31 operator∗() [1/2]

```
Matrix Matrix::operator* (
            const double & scal ) const
```

Matrix-scalar product.

Parameters

in	*scal*	scalar

Returns

Matrix result

7.5.3.32 operator∗() [2/2]

```
Matrix Matrix::operator* (
            const Matrix & m ) const
```

Hadamard matrix product.

Parameters

in	*m*	right matrix

Returns

Matrix result

7.5.3.33 operator*=()

```
Matrix & Matrix::operator*= (
            const double & scal )
```

Matrix-scalar compound product.

Parameters

in	*scal*	scalar

Returns

Matrix result

7.5.3.34 operator+() [1/2]

```
Matrix Matrix::operator+ (
            const double & scal ) const
```

Matrix-scalar addition.

Parameters

in	*scal*	scalar

Returns

Matrix result

7.5.3.35 operator+() [2/2]

```
Matrix Matrix::operator+ (
            const Matrix & m ) const
```

Entrywise matrix addition.

Parameters

in	*m*	right matrix

Returns

Matrix result

7.5.3.36 operator+=()

```
Matrix & Matrix::operator+= (
            const double & scal )
```

Matrix-scalar compound addition.

Parameters

in	*scal*	scalar

Returns

Matrix result

7.5.3.37 operator-() [1/2]

```
Matrix Matrix::operator- (
            const double & scal ) const
```

Matrix-scalar subtraction.

Parameters

in	*scal*	scalar

Returns

Matrix result

7.5.3.38 operator-() [2/2]

```
Matrix Matrix::operator- (
            const Matrix & m ) const
```

Entrywise matrix subtraction.

Parameters

in	*m*	right matrix

Returns

Matrix result

7.5.3.39 operator-=()

```
Matrix & Matrix::operator-= (
            const double & scal )
```

Matrix-scalar compound subtraction.

Parameters

in	*scal*	scalar

Returns

Matrix result

7.5.3.40 operator/() [1/2]

```
Matrix Matrix::operator/ (
            const double & scal ) const
```

Matrix-scalar division.

Parameters

in	*scal*	scalar

Returns

Matrix result

7.5.3.41 operator/() [2/2]

```
Matrix Matrix::operator/ (
            const Matrix & m ) const
```

Entrywise matrix division.

Parameters

in	*m*	right matrix

Returns

Matrix result

7.5.3.42 operator/=()

```
Matrix & Matrix::operator/= (
            const double & scal )
```

Matrix-scalar compound division.

Parameters

in	*scal*	scalar

Returns

Matrix result

7.5.3.43 operator=() [1/2]

```
Matrix & Matrix::operator= (
            const Matrix & inMat )
```

Copy assignment operator.

Parameters

in	*inMat*	object to be copied

Returns

Matrix target object

7.5.3.44 operator=() [2/2]

```
Matrix & Matrix::operator= (
            Matrix && inMat )
```

Move assignment operator.

Parameters

in	*inMat*	object to be moved

Returns

Matrix target object

7.5.3.45 postmultZ() [1/2]

```
void Matrix::postmultZ (
            const Matrix & Z )
```

In-place multiply by a design matrix from the right.

Performs multiplication MZ where Z is a design matrix (Z is $m \times n$ and M is $m \times p$; in practice $m \geq n$) that has one or more elements set to 1, relating the columns to rows. The function avoids actual multiplication and simply shrinks the M matrix (in-place) to the number of columns the same as the number of columns in Z. Row order in the resulting matrix is the same as in the original object.

Parameters

in	*Z*	Z matrix (not transposed)

7.5.3.46 postmultZ() [2/2]

```
void Matrix::postmultZ (
            const Matrix & Z,
            Matrix & out ) const
```

Multiply by a design matrix from the right and copy result.

Performs multiplication MZ where Z is a design matrix (Z is $m \times n$ and M is $m \times p$; in practice $m \geq n$) that has one or more elements set to 1, relating the columns to rows. The function avoids actual multiplication and simply shrinks the M matrix (in-place) to the number of columns the same as the number of columns in Z. Row order in the resulting matrix is the same as in the original object. If the output *Matrix* object does not have the correct dimensions, it is resized.

Parameters

in	*Z*	Z matrix (not transposed)
out	*out*	output matrix

7.5.3.47 postmultZt() [1/2]

```
void Matrix::postmultZt (
            const Matrix & Z )
```

In-place multiply by the transpose of a design matrix from the right.

Performs multiplication MZ^T where Z is a design matrix (Z is $n \times m$ and M is $m \times p$; in practice $n \geq m$) that has one or more elements set to 1, relating the columns to rows. The function avoids actual multiplication and simply expands the M matrix (in-place) to the number of columns the same as the number of rows in Z. Row order in the resulting matrix is the same as in the original object.

Parameters

in	*Z*	Z matrix (not transposed)

7.5.3.48 postmultZt() [2/2]

```
void Matrix::postmultZt (
            const Matrix & Z,
            Matrix & out ) const
```

Multiply by the transpose of a design matrix from the right and copy result.

Performs multiplication MZ^T where Z is a design matrix (Z is $n \times m$ and M is $m \times p$; in practice $n \geq m$) that has one or more elements set to 1, relating the columns to rows. The function avoids actual multiplication and simply expands the M matrix (and copies to the provided Matrix object) to the number of columns the same as the number of rows in Z. Row order in the resulting matrix is the same as in the original object. If the output *Matrix* object does not have the correct dimensions, it is resized.

Parameters

in	*Z*	Z matrix (not transposed)
out	*out*	output matrix

7.5.3.49 premultZ() [1/2]

```
void Matrix::premultZ (
            const Matrix & Z )
```

In-place multiply by a design matrix from the left.

Performs multiplication ZM where Z is a design matrix (Z is $n \times m$ and M is $m \times p$; in practice $n \geq m$) that has one or more elements set to 1, relating the columns to rows. The function avoids actual multiplication and simply expands the M matrix (in-place) to the same number of rows as Z. Column order in the resulting matrix is the same as in the original object.

Parameters

in	*Z*	Z matrix

7.5.3.50 premultZ() [2/2]

```
void Matrix::premultZ (
            const Matrix & Z,
            Matrix & out ) const
```

Multiply by a design matrix from the left and copy result.

Performs multiplication ZM where Z is a design matrix (Z is $n \times m$ and M is $m \times p$; in practice $n \geq m$) that has one or more elements set to 1, relating the columns to rows. The function avoids actual multiplication and simply expands the M matrix (and copies to the provided Matrix object) to the same number of rows as Z. Column order in the resulting matrix is the same as in the original object. If the output *Matrix* object does not have the correct dimensions, it is resized.

Parameters

in	*Z*	Z matrix
out	*out*	output matrix

7.5.3.51 premultZt() [1/2]

```
void Matrix::premultZt (
            const Matrix & Z )
```

In-place multiply by the transpose of a design matrix from the left.

Performs multiplication $Z^T M$ where Z is a design matrix (Z is $m \times n$ and M is $m \times p$; in practice $m \geq n$) that has one or more elements set to 1, relating the columns to rows. The effect is to sum the rows of Y within categories represented by columns of Z. The function avoids actual multiplication and simply shrinks the M matrix (in-place) to the same number of rows as there are columns in Z. Column order in the resulting matrix is the same as in the original object.

Parameters

in	*Z*	Z matrix (not transposed)

7.5.3.52 premultZt() [2/2]

```
void Matrix::premultZt (
            const Matrix & Z,
            Matrix & out ) const
```

Multiply by the transpose of a design matrix from the left and copy result.

Performs multiplication $Z^T M$ where Z is a design matrix (Z is $m \times n$ and M is $m \times p$; in practice $m \geq n$) that has one or more elements set to 1, relating the columns to rows. The effect is to sum the rows of Y within categories represented by columns of Z. The function avoids actual multiplication and simply shrinks the M matrix to the same number of rows as there are columns in Z. Column order in the resulting matrix is the same as in the original object. If the output *Matrix* object does not have the correct dimensions, it is resized.

Parameters

in	*Z*	Z matrix (not transposed)
out	*out*	output matrix

7.5.3.53 resize()

```
void Matrix::resize (
            const size_t & nrow,
            const size_t & ncol )
```

Resize matrix.

Resizes the matrix to the dimensions provided. All elements are set to zero.

Parameters

in	*nrow*	new number of rows
in	*ncol*	new number of columns

7.5.3.54 rowAdd() [1/2]

```
void Matrix::rowAdd (
            const double & scalar,
            const size_t & iRow )
```

Add a scalar to a row.

Entry-wise addition of a scalar to the given row. The current object is modified.

Parameters

in	*scalar*	scalar to use for addition
in	*iRow*	row index

7.5.3.55 rowAdd() [2/2]

```
void Matrix::rowAdd (
            const vector< double > & scalars )
```

Add a vector to rows.

Entry-wise addition of a vector to each row. The current object is modified.

Parameters

in	*scalars*	vector of scalars to use for addition

7.5.3.56 rowDivide() [1/2]

```
void Matrix::rowDivide (
            const double & scalar,
            const size_t & iRow )
```

Divide a row by a scalar.

Entry-wise division of a given row by the provided scalar. The current object is modified.

Parameters

in	*scalar*	scalar to use for division
in	*iRow*	row index

7.5.3.57 rowDivide() [2/2]

```
void Matrix::rowDivide (
            const vector< double > & scalars )
```

Divide rows by a vector.

Entry-wise division of each row by the provided vector. The current object is modified.

Parameters

in	*scalars*	vector of scalars to use for division

7.5.3.58 rowMeans()

```
void Matrix::rowMeans (
            vector< double > & means ) const
```

Row means.

Calculates row means and stores them in the provided vector. If vector length is smaller than necessary, the vector is expanded. Otherwise, the first N_{row} elements are used.

Parameters

out	*means*	vector of means

7.5.3.59 rowMultiply() [1/2]

```
void Matrix::rowMultiply (
```

```
            const double & scalar,
            const size_t & iRow )
```

Multiply a row by a scalar.

Entry-wise multiplication of a given row by the provided scalar. The current object is modified.

Parameters

in	*scalar*	scalar to use for multiplication
in	*iRow*	row index

7.5.3.60 rowMultiply() [2/2]

```
void Matrix::rowMultiply (
            const vector< double > & scalars )
```

Multiply rows by a vector.

Entry-wise multiplication of each row by the provided vector. The current object is modified.

Parameters

in	*scalars*	vector of scalars to use for multiplication

7.5.3.61 rowShuffle()

```
Matrix Matrix::rowShuffle ( ) const
```

Shuffle rows.

Suffle the rows of the current object and return a matrix with of the same size but row order randomly permuted.

Returns

permuted Matrix object

7.5.3.62 rowSub() [1/2]

```
void Matrix::rowSub (
            const double & scalar,
            const size_t & iRow )
```

Subtract a scalar from a row.

Entry-wise subtraction of a scalar from the given row. The current object is modified.

Parameters

in	*scalar*	scalar to use for subtraction
in	*iRow*	row index

7.5.3.63 rowSub() [2/2]

```
void Matrix::rowSub (
            const vector< double > & scalars )
```

Subtract a vector from rows.

Entry-wise subtraction of a vector from each row. The current object is modified.

Parameters

in	*scalars*	vector of scalars to use for subtraction

7.5.3.64 rowSums()

```
void Matrix::rowSums (
            vector< double > & sums ) const
```

Row sums.

Calculates row sums and stores them in the provided vector. If vector length is smaller than necessary, the vector is expanded. Otherwise, the first N_{row} elements are used.

Parameters

out	*sums*	vector of sums

7.5.3.65 save()

```
void Matrix::save (
            const string & outFileName ) const
```

Save matrix contents to a tab-delimited file.

Parameters

in	*outFileName*	file name

7.5.3.66 setCol()

```
void Matrix::setCol (
            const size_t jCol,
            const vector< double > data )
```

Copy data from a vector to a column.

Copies data from a vector to a specified column. If the vector is too long, the first Nrow_ elements are used.

Parameters

in	*jCol*	column index (0 base)
in	*data*	vector with data

7.5.3.67 setElem()

```
void Matrix::setElem (
            const size_t & iRow,
            const size_t & jCol,
            const double & input )
```

Set element to a value.

Parameters

in	*iRow*	row number
in	*jCol*	column number
in	*input*	input value

7.5.3.68 svd()

```
void Matrix::svd (
            Matrix & U,
            vector< double > & s )
```

Perform SVD.

Performs SVD and stores the U vectors in a Matrix object and singular values in a C++ vector. For now, only does the *DGESVD* from LAPACK with no V^T matrix. The data in the object are destroyed.

Parameters

out	*U*	U vector matrix
out	*s*	singular value vector

7.5.3.69 svdSafe()

```
void Matrix::svdSafe (
            Matrix & U,
            vector< double > & s ) const
```

Perform "safe" SVD.

Performs SVD and stores the U vectors in a Matrix object and singular values in a C++ vector. For now, only does the *DGESVD* from LAPACK with no V^T matrix. The data in the object are preserved, leading to some loss of efficiency compared to svd().

Parameters

out	*U*	U vector matrix
out	*s*	singular value vector

7.5.3.70 symc()

```
void Matrix::symc (
            const char & tri,
            const double & alpha,
            const Matrix & X,
            const size_t & xCol,
            const double & beta,
            vector< double > & y ) const
```

Multiply symmetric matrix by a column of another matrix

Multiply the *Matrix* object, which is symmetric, by a specified column of another matrix. An interface for the BLAS *DSYMV* routine. Updates the input vector y

$$y \leftarrow \alpha A X_{\cdot j} + \beta y$$

If the output vector is too short it is re-sized, adding zero elements as needed. If it is too long, only the first Nrow(A) elements are modified.

Parameters

in	*tri*	A (focal object) triangle ID ('u' for upper or 'l' for lower)
in	*alpha*	the α constant
in	*X*	matrix X whose column will be used
in	*xCol*	column of X to be used (0 base)
in	*beta*	the β constant
in,out	*y*	result vector

7.5.3.71 symm()

```
void Matrix::symm (
            const char & tri,
            const char & side,
            const double & alpha,
            const Matrix & symA,
            const double & beta,
            Matrix & C ) const
```

Multiply by symmetric matrix.

Multiply the *Matrix* object by a symmetric matrix. The interface for the BLAS *DSYMM* routine. Updates the input/output matrix C

$$C \leftarrow \alpha AB + \beta C$$

if *side* is 'l' (left) and

$$C \leftarrow \alpha BA + \beta C$$

if *side* is 'r' (right). The symmetric A matrix is provided as input, the object from which the method is called is the B matrix. If *C* does not have the right dimensions, it is re-sized and all elements are set to zero before the operation. Otherwize, only the specified triangle is changed.

Parameters

in	*tri*	A triangle ID ('u' for upper or 'l' for lower)
in	*side*	multiplication side
in	*alpha*	the α constant
in	*symA*	symmetric matrix A
in	*beta*	the β constant
in,out	*C*	the result C matrix

7.5.3.72 syrk()

```
void Matrix::syrk (
            const char & tri,
            const double & alpha,
            const double & beta,
            Matrix & C ) const
```

Inner self crossproduct.

Interface for the BLAS *DSYRK* routine. This function updates the given symmetric matrix C with the operation

$$C \leftarrow \alpha A^T A + \beta C$$

The *char* parameter governs which triangle of C is used to store the result ('u' is upper and 'l' is lower). If *C* does not have the right dimensions, it is re-sized and all elements are set to zero before the operation. Otherwize, only the specified triangle is changed.

Parameters

in	*tri*	C triangle ID
in	*alpha*	the α parameter
in	*beta*	the β parameter
in,out	*C*	the result C matrix

7.5.3.73 tsyrk()

```
void Matrix::tsyrk (
            const char & tri,
            const double & alpha,
            const double & beta,
            Matrix & C ) const
```

Outer self crossproduct.

Interface for the BLAS *DSYRK* routine. This function updates the given symmetric matrix C with the operation

$$C \leftarrow \alpha A A^T + \beta C$$

The *char* parameter governs which triangle of C is used to store the result ('u' is upper and 'l' is lower). If *C* does not have the right dimensions, it is re-sized and all elements are set to zero before the operation. Otherwize, only the specified triangle is changed.

Parameters

in	*tri*	C triangle ID
in	*alpha*	the α parameter
in	*beta*	the β parameter
in,out	*C*	the result C matrix

7.5.3.74 vectorize()

```
void Matrix::vectorize (
            vector< double > & out ) const
```

Vectorize the matrix.

Vectorize the matrix by column.

Parameters

out	*out*	vector of matrix elements

7.5.4 Friends And Related Function Documentation

7.5.4.1 operator*

```
Matrix operator* (
            const double & scal,
            const Matrix & m )  [friend]
```

Scalar-matrix product.

Parameters

in	*scal*	scalar
in	*m*	matrix

Returns

Matrix result

7.5.4.2 operator+

```
Matrix operator+ (
            const double & scal,
            const Matrix & m )  [friend]
```

Scalar-matrix addition.

Parameters

in	*scal*	scalar
in	*m*	matrix

Returns

Matrix result

The documentation for this class was generated from the following files:

- src/locMatrix.hpp
- src/locMatrix.cpp

7.6 BayesicSpace::MixedModel Class Reference

Mixed model.

`#include <likeMeth.hpp>`

Public Member Functions

- MixedModel ()

  Default constructor

- MixedModel (const vector< double > &yvec, const vector< double > &kvec, const size_t &d, const size_t &Ngen)

  Basic constructor.

- MixedModel (const vector< double > &yvec, const vector< double > &kvec, const vector< size_t > &repFac, const size_t &d, const size_t &Ngen, const size_t &N)

  Constructor with replication.

- MixedModel (const vector< double > &yvec, const vector< double > &kvec, const vector< double > &xvec, const size_t &d, const size_t &Ngen)

  Constructor including a fixed effect but no replication.

- MixedModel (const vector< double > &yvec, const vector< double > &kvec, const vector< size_t > &repFac, const vector< double > &xvec, const size_t &d, const size_t &Ngen, const size_t &N)

  Constructor including a fixed effect and replication.

- MixedModel (const vector< double > &yvec, const vector< double > &kvec, const size_t &d, const size_t &Ngen, const vector< int32_t > *snps, const int32_t &misTok, vector< double > *lPval)

  Constructor with SNPs.

- MixedModel (const vector< double > &yvec, const vector< double > &kvec, const vector< size_t > &repFac, const size_t &d, const size_t &Ngen, const size_t &N, const vector< int32_t > *snps, const int32_t &misTok, vector< double > *lPval)

  Constructor with SNPs and replication.

- MixedModel (const vector< double > &yvec, const vector< double > &kvec, const vector< double > &xvec, const size_t &d, const size_t &Ngen, const vector< int32_t > *snps, const int32_t &misTok, vector< double > *lPval)

Constructor with SNPs and a fixed effect but no replication.

- MixedModel (const vector< double > &yvec, const vector< double > &kvec, const vector< size_t > &repFac, const vector< double > &xvec, const size_t &d, const size_t &Ngen, const size_t &N, const vector< int32_t > *snps, const int32_t &misTok, vector< double > *lPval)

  Constructor with SNPs, fixed effect, and replication.
- ~MixedModel ()

  Destructor.
- MixedModel (const MixedModel &in)=delete

  Copy constructor (not implemented)
- MixedModel & operator= (const MixedModel &in)=delete

  Copy assignement (not implemented)
- MixedModel (MixedModel &&in)

  Move constructor.
- MixedModel & operator= (MixedModel &&in)=delete

  Move assignment operator (not impplemented)
- void ranef (Matrix &u) const

  Access the random effects.
- void fixef (Matrix &beta) const

  Access the fixed effects.
- void hSq (vector< double > &out) const

  Marker heritability.
- void gwa (uint32_t nThr)

  Genome-wide association.
- void gwa (const uint32_t &nPer, uint32_t nThr, vector< double > &fdr)

  Genome-wide association with FDR.

Friends

- class **SNPblock**

7.6.1 Detailed Description

Mixed model.

Constructors solve a mixed model given inputs. Public functions do GWA.

7.6.2 Constructor & Destructor Documentation

7.6.2.1 MixedModel() [1/9]

```
MixedModel::MixedModel (
            const vector< double > & yvec,
            const vector< double > & kvec,
            const size_t & d,
            const size_t & Ngen )
```

Basic constructor.

Parameters

in	*yvec*	vectorized (by column) response matrix
in	*kvec*	vectorized relationship matrix
in	*d*	number of traits
in	*Ngen*	number of genotypes

7.6.2.2 MixedModel() [2/9]

```
MixedModel::MixedModel (
            const vector< double > & yvec,
            const vector< double > & kvec,
            const vector< size_t > & repFac,
            const size_t & d,
            const size_t & Ngen,
            const size_t & N )
```

Constructor with replication.

Parameters

in	*yvec*	vectorized (by column) response matrix
in	*kvec*	vectorized relationship matrix
in	*repFac*	factor relating genotypes to replicates
in	*d*	number of traits
in	*Ngen*	number of genotypes
in	*N*	number of data points

7.6.2.3 MixedModel() [3/9]

```
MixedModel::MixedModel (
            const vector< double > & yvec,
            const vector< double > & kvec,
            const vector< double > & xvec,
            const size_t & d,
            const size_t & Ngen )
```

Constructor including a fixed effect but no replication.

Parameters

in	*yvec*	vectorized (by column) response matrix

Parameters

in	*kvec*	vectorized relationship matrix
in	*xvec*	vectorized (by column) matrix of fixed effects
in	*d*	number of traits
in	*Ngen*	number of genotypes

7.6.2.4 MixedModel() [4/9]

```
MixedModel::MixedModel (
            const vector< double > & yvec,
            const vector< double > & kvec,
            const vector< size_t > & repFac,
            const vector< double > & xvec,
            const size_t & d,
            const size_t & Ngen,
            const size_t & N )
```

Constructor including a fixed effect and replication.

Parameters

in	*yvec*	vectorized (by column) response matrix
in	*kvec*	vectorized relationship matrix
in	*repFac*	factor relating genotypes to replicates
in	*xvec*	vectorized (by column) matrix of fixed effects
in	*d*	number of traits
in	*Ngen*	number of genotypes
in	*N*	number of data points

7.6.2.5 MixedModel() [5/9]

```
BayesicSpace::MixedModel::MixedModel (
            const vector< double > & yvec,
            const vector< double > & kvec,
            const size_t & d,
            const size_t & Ngen,
            const vector< int32_t > * snps,
            const int32_t & misTok,
            vector< double > * lPval )  [inline]
```

Constructor with SNPs.

Parameters

in	*yvec*	vectorized (by column) response matrix
in	*kvec*	vectorized relationship matrix
in	*d*	number of traits
in	*Ngen*	number of genotypes
in	*snps*	vectorized (by column) SNP matrix
in	*misTok*	token representing missing genotype data
out	*lPval*	address of the voectorized $-\log_{10} p$ matrix

7.6.2.6 MixedModel() [6/9]

```
BayesicSpace::MixedModel::MixedModel (
            const vector< double > & yvec,
            const vector< double > & kvec,
            const vector< size_t > & repFac,
            const size_t & d,
            const size_t & Ngen,
            const size_t & N,
            const vector< int32_t > * snps,
            const int32_t & misTok,
            vector< double > * lPval )  [inline]
```

Constructor with SNPs and replication.

Parameters

in	*yvec*	vectorized (by column) response matrix
in	*kvec*	vectorized relationship matrix
in	*repFac*	factor relating genotypes to replicates
in	*d*	number of traits
in	*Ngen*	number of genotypes
in	*N*	number of data points
in	*snps*	vectorized (by column) SNP matrix
in	*misTok*	token representing missing genotype data
out	*lPval*	address of the voectorized $-\log_{10} p$ matrix

7.6.2.7 MixedModel() [7/9]

```
BayesicSpace::MixedModel::MixedModel (
            const vector< double > & yvec,
```

```
            const vector< double > & kvec,
            const vector< double > & xvec,
            const size_t & d,
            const size_t & Ngen,
            const vector< int32_t > * snps,
            const int32_t & misTok,
            vector< double > * lPval )  [inline]
```

Constructor with SNPs and a fixed effect but no replication.

Parameters

in	*yvec*	vectorized (by column) response matrix
in	*kvec*	vectorized relationship matrix
in	*xvec*	vectorized (by column) matrix of fixed effects
in	*d*	number of traits
in	*Ngen*	number of genotypes
in	*snps*	vectorized (by column) SNP matrix
in	*misTok*	token representing missing genotype data
out	*lPval*	address of the voectorized $-\log_{10} p$ matrix

7.6.2.8 MixedModel() [8/9]

```
BayesicSpace::MixedModel::MixedModel (
            const vector< double > & yvec,
            const vector< double > & kvec,
            const vector< size_t > & repFac,
            const vector< double > & xvec,
            const size_t & d,
            const size_t & Ngen,
            const size_t & N,
            const vector< int32_t > * snps,
            const int32_t & misTok,
            vector< double > * lPval )  [inline]
```

Constructor with SNPs, fixed effect, and replication.

Parameters

in	*yvec*	vectorized (by column) response matrix
in	*kvec*	vectorized relationship matrix
in	*repFac*	factor relating genotypes to replicates
in	*xvec*	vectorized (by column) matrix of fixed effects
in	*d*	number of traits
in	*Ngen*	number of genotypes
in	*N*	number of data points
in	*snps*	vectorized (by column) SNP matrix
in	*misTok*	token representing missing genotype data
out	*lPval*	address of the voectorized $-\log_{10} p$ matrix

7.6.2.9 MixedModel() [9/9]

```
BayesicSpace::MixedModel::MixedModel (
            MixedModel && in )  [inline]
```

Move constructor.

Parameters

in	*in*	object to be moved

7.6.3 Member Function Documentation

7.6.3.1 fixef()

```
void BayesicSpace::MixedModel::fixef (
            Matrix & beta ) const  [inline]
```

Access the fixed effects.

Returns empty matrix if there are no fixed effects.

Parameters

out	*beta*	fixed effect matrix

7.6.3.2 gwa() [1/2]

```
void MixedModel::gwa (
            const uint32_t & nPer,
            uint32_t nThr,
            vector< double > & fdr )
```

Genome-wide association with FDR.

The same as gwa(), but does permutations to calculate emprical FDR for each SNP.

Parameters

in	*nPer*	number of permutations
in	*nThr*	number of threads; set automatically if 0
out	*fdr*	vectorized (by column) matrix of FDR values

7.6.3.3 gwa() [2/2]

```
void MixedModel::gwa (
            uint32_t nThr )
```

Genome-wide association.

Estimates regression $-\log_{10} p$ for SNPs with missing genotype data and multiple traits in a table. Genotypes should be coded as (0,1,2) for homozygous, hetereozygous and homozygous alternative. It should run faster of the major allele homozygotes are coded as 0. Each trait is treated separately but it helps to include multiple traits because some calculations are common and can be performed only once. The $-\log_{10} p$ matrix has each trait in a column.

Parameters

in	*nThr*	number of threads; set automatically if 0

7.6.3.4 hSq()

```
void MixedModel::hSq (
            vector< double > & out ) const
```

Marker heritability.

Parameters

out	*marker*	heritability h^2_{M}

7.6.3.5 ranef()

```
void BayesicSpace::MixedModel::ranef (
            Matrix & u ) const  [inline]
```

Access the random effects.

Parameters

out	*u*	random effect matrix

The documentation for this class was generated from the following files:

- src/likeMeth.hpp
- src/likeMeth.cpp

7.7 BayesicSpace::RanDraw Class Reference

Random number generating class.

`#include <random.hpp>`

Public Member Functions

- RanDraw ()
  Default constructor.
- ~RanDraw ()
  Destructor.
- RanDraw (const RanDraw &old)=default
  Copy constructor.
- RanDraw (RanDraw &&old)=default
  Move constructor.
- RanDraw & operator= (const RanDraw &old)=default
  Copy assignment.
- RanDraw & operator= (RanDraw &&old)=default
  Move assignment.
- string type () const
  Query RNG kind.
- uint64_t ranInt () const
  Generate random integer.
- uint64_t sampleInt (const uint64_t &max) const
  Sample and integer from the $[0, n)$ interval.
- uint64_t sampleInt (const uint64_t &min, const uint64_t &max) const
  Sample and integer from the $[m, n)$ interval.
- vector< uint64_t > shuffleUint (const uint64_t &N)
  Draw non-negative intergers in random order.
- double runif () const
  Generate a uniform deviate.
- double runifnz () const
  Generate a non-zero uniform deviate.
- double runifno () const

Generate a non-one uniform deviate.

- double runifop () const

  Generate an open-interval uniform deviate.
- double rnorm () const

  A standard Gaussian deviate.
- double rnorm (const double &sigma) const

  A zero-mean Gaussian deviate.
- double rnorm (const double &mu, const double &sigma) const

  A Gaussian deviate.
- double rgamma (const double &alpha) const

  A standard Gamma deviate.
- double rgamma (const double &alpha, const double &beta) const

  A general Gamma deviate.
- double rchisq (const double &nu) const

  A chi-square deviate.
- uint64_t vitterA (const double &n, const double &N) const

  Sample from Vitter's distribution, method A.
- uint64_t vitter (const double &n, const double &N) const

  Sample from Vitter's distribution, method D.

7.7.1 Detailed Description

Random number generating class.

Generates (pseudo-)random deviates from a number of distributions. If hardware random numbers are supported, uses them. Otherwise, falls back to 64-bit MT19937 ("Mersenne Twister").

7.7.2 Constructor & Destructor Documentation

7.7.2.1 RanDraw() [1/3]

```
RanDraw::RanDraw ( )
```

Default constructor.

Checks if the processor provides hardware random number support. Seeds the Mersenne Twister if not. Throws "CPU_unsupported" string object if the CPU is not AMD or Intel.

7.7.2.2 RanDraw() [2/3]

```
BayesicSpace::RanDraw::RanDraw (
            const RanDraw & old )  [default]
```

Copy constructor.

Parameters

in	*old*	pbject to be copied

7.7.2.3 RanDraw() [3/3]

```
BayesicSpace::RanDraw::RanDraw (
            RanDraw && old )  [default]
```

Move constructor.

Parameters

in	*old*	pbject to be moved

7.7.3 Member Function Documentation

7.7.3.1 operator=() [1/2]

```
RanDraw& BayesicSpace::RanDraw::operator= (
            const RanDraw & old )  [default]
```

Copy assignment.

Parameters

in	*old*	pbject to be copied

7.7.3.2 operator=() [2/2]

```
RanDraw& BayesicSpace::RanDraw::operator= (
            RanDraw && old )  [default]
```

Move assignment.

Parameters

in	*old*	pbject to be moved

7.7.3.3 ranInt()

```
uint64_t BayesicSpace::RanDraw::ranInt ( ) const  [inline]
```

Generate random integer.

Returns

An unsigned random 64-bit integer

7.7.3.4 rchisq()

```
double BayesicSpace::RanDraw::rchisq (
            const double & nu ) const  [inline]
```

A chi-square deviate.

Generates a χ^2 random variable with degrees of freedom $\nu > 0.0$.

Parameters

in	*nu*	degrees of freedom

Returns

a sample from the χ^2 distribution

7.7.3.5 rgamma() [1/2]

```
double RanDraw::rgamma (
            const double & alpha ) const
```

A standard Gamma deviate.

Generates a Gamma random variable with shape $\alpha > 0$ and standard scale $\beta = 1.0$. Implements the Marsaglia and Tsang (2000) method.

Parameters

in	*alpha*	shape parameter α

Returns

a sample from the standard Gamma distribution

7.7.3.6 rgamma() [2/2]

```
double BayesicSpace::RanDraw::rgamma (
            const double & alpha,
            const double & beta ) const  [inline]
```

A general Gamma deviate.

Generates a Gamma random variable with shape $\alpha > 0$ and scale $\beta > 0$.

Parameters

in	*alpha*	shape parameter α
in	*beta*	scale parameter β

Returns

a sample from the general Gamma distribution

7.7.3.7 rnorm() [1/3]

```
double RanDraw::rnorm ( ) const
```

A standard Gaussian deviate.

Generates a Gaussian random value with mean $\mu = 0.0$ and standard deviation $\sigma = 1.0$. Implemented using a version of the Marsaglia and Tsang (2000) ziggurat algorithm, modified according to suggestions in the GSL implementation of the function.

Returns

a sample from the standard Gaussian distribution

7.7.3.8 rnorm() [2/3]

```
double BayesicSpace::RanDraw::rnorm (
            const double & mu,
            const double & sigma ) const  [inline]
```

A Gaussian deviate.

Generates a Gaussian random value with mean μ and standard deviation σ. Implemented using a version of the Marsaglia and Tsang (2000) ziggurat algorithm, modified according to suggestions in the GSL implementation of the function.

Parameters

in	*mu*	standard deviation
in	*sigma*	standard deviation

Returns

a sample from the Gaussian distribution

7.7.3.9 rnorm() [3/3]

```
double BayesicSpace::RanDraw::rnorm (
            const double & sigma ) const  [inline]
```

A zero-mean Gaussian deviate.

Generates a Gaussian random value with mean $\mu = 0.0$ and standard deviation σ. Implemented using a version of the Marsaglia and Tsang (2000) ziggurat algorithm, modified according to suggestions in the GSL implementation of the function.

Parameters

in	*sigma*	standard deviation

Returns

a sample from the zero-mean Gaussian distribution

7.7.3.10 runif()

```
double BayesicSpace::RanDraw::runif ( ) const  [inline]
```

Generate a uniform deviate.

Returns

A double-precision value from the $U[0,1]$ distribution

7.7.3.11 runifno()

```
double RanDraw::runifno ( ) const
```

Generate a non-one uniform deviate.

Returns

A double-precision value from the $U[0,1)$ distribution

7.7.3.12 runifnz()

```
double RanDraw::runifnz ( ) const
```

Generate a non-zero uniform deviate.

Returns

A double-precision value from the $U(0,1]$ distribution

7.7.3.13 runifop()

```
double RanDraw::runifop ( ) const
```

Generate an open-interval uniform deviate.

Returns

A double-precision value from the $U(0,1)$ distribution

7.7.3.14 sampleInt() [1/2]

```
uint64_t BayesicSpace::RanDraw::sampleInt (
            const uint64_t & max ) const  [inline]
```

Sample and integer from the $[0,n)$ interval.

Parameters

in	*max*	the maximal value n (does not appear in the sample)

Returns

sampled value

7.7.3.15 sampleInt() [2/2]

```
uint64_t RanDraw::sampleInt (
            const uint64_t & min,
            const uint64_t & max ) const
```

Sample and integer from the $[m, n)$ interval.

Throws `string` "Lower bound not smaller than upper bound" if $m \geq n$.

Parameters

in	*min*	the minimal value m (can appear in the sample)
in	*max*	the maximal value n (does not appear in the sample)

Returns

sampled value

7.7.3.16 shuffleUint()

```
vector< uint64_t > RanDraw::shuffleUint (
            const uint64_t & N )
```

Draw non-negative intergers in random order.

Uses Fisher-Yates-Durstenfeld algorithm to produce a random shuffle of integers in $[0, N)$.

Parameters

in	*Nmax*	the upper bound of the integer sequence

Returns

vector of N shuffled integers

7.7.3.17 type()

```
string BayesicSpace::RanDraw::type ( ) const  [inline]
```

Query RNG kind.

Find out the kind of (P)RNG in use.

Returns

String reflecting the RNG type

7.7.3.18 vitter()

```
uint64_t RanDraw::vitter (
            const double & n,
            const double & N ) const
```

Sample from Vitter's distribution, method D.

Given the number of remaining records in a file N and the number of records n remaining to be selected, sample the number of records to skip over. This function implements Vitter's **[vitter84a] [vitter87a]** method D. It is useful for online one-pass sampling of records from a file. While the inputs are integer, we pass them in as *double* because that is more efficient for calculations.

Parameters

in	*n*	number of records remaining to be picked
in	*N*	number of remaining records in the file

Returns

the number of records to skip

7.7.3.19 vitterA()

```
uint64_t RanDraw::vitterA (
            const double & n,
            const double & N ) const
```

Sample from Vitter's distribution, method A.

Given the number of remaining records in a file N and the number of records n remaining to be selected, sample the number of records to skip over. This function implements Vitter's **[vitter84a] [vitter87a]** method A. It is useful for online one-pass sampling of records from a file. While the inputs are integer, we pass them in as *double* because that is more efficient for calculations.

Parameters

in	*n*	number of records remaining to be picked
in	*N*	number of remaining records in the file

Returns

the number of records to skip

The documentation for this class was generated from the following files:

- src/random.hpp
- src/random.cpp

7.8 BayesicSpace::SNPblock Class Reference

A SNP block functor class.

`#include <likeMeth.hpp>`

Public Member Functions

- SNPblock ()
  Default constructor.
- SNPblock (MixedModel &parent, const size_t &bStart, const size_t &bSize)
  Constructor.
- SNPblock (MixedModel &parent, const size_t &bStart, const size_t &bSize, vector< double > &plPval, const size_t &perOff, const size_t &snpOff)
  Constructor for permutations.
- ~SNPblock ()
  Destructor.
- SNPblock (const SNPblock &)=delete

Copy constructor (not implemented)

- SNPblock & operator= (const SNPblock &)=delete

  Copy assignment operator (not implemented)

- SNPblock (SNPblock &&in)

  Move constructor.

- void operator() ()

  Function operator.

7.8.1 Detailed Description

A SNP block functor class.

A class to facilitate GWA multithreading in the `MixedModel` class. Points to a block of SNPs and runs `oneSNP_()` on each locus.

7.8.2 Constructor & Destructor Documentation

7.8.2.1 SNPblock() [1/2]

```
BayesicSpace::SNPblock::SNPblock (
            MixedModel & parent,
            const size_t & bStart,
            const size_t & bSize )  [inline]
```

Constructor.

Sets up the pointer to the `MixedModel` object calling the current instance of this functor.

Parameters

in,out	*parent*	address of the `MixedModel` object calling this functor
in	*bStart*	block start position
in	*bSize*	block size (number of SNPs)

7.8.2.2 SNPblock() [2/2]

```
BayesicSpace::SNPblock::SNPblock (
            MixedModel & parent,
            const size_t & bStart,
```

```
const size_t & bSize,
vector< double > & plPval,
const size_t & perOff,
const size_t & snpOff )  [inline]
```

Constructor for permutations.

Sets up the pointer to the `MixedModel` object calling the current instance of this functor, adding the info to work on permuted data.

Parameters

`in,out`	*parent*	address of the `MixedModel` object calling this functor
`in`	*bStart*	block start position
`in`	*bSize*	block size (number of SNPs)
`out`	*plPval*	address of the permuted $-\log_{10} p$ vector
`in`	*perOff*	permutation offset (`Nsnp*nPer`)
`in`	*snpOff*	SNP offset (`Nsnp*perIdx`)

7.8.3 Member Function Documentation

7.8.3.1 operator()()

```
void SNPblock::operator() ( )
```

Function operator.

Performs GWA on each SNP in the block.

The documentation for this class was generated from the following files:

- src/likeMeth.hpp
- src/likeMeth.cpp

Chapter 8

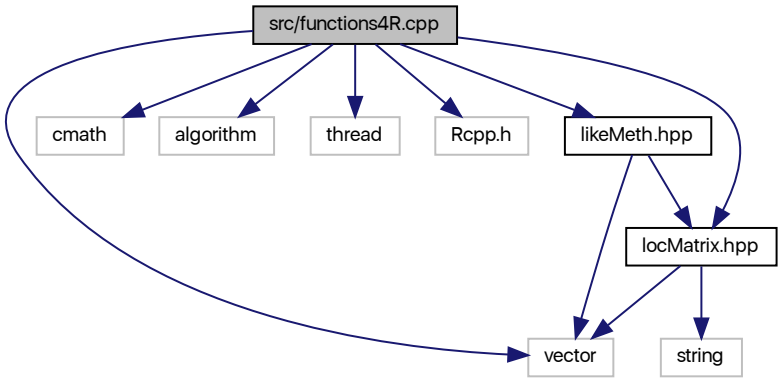
File Documentation

8.1 src/functions4R.cpp File Reference

GWA on replicated data with a mixed model.

```
#include <vector>
#include <cmath>
#include <algorithm>
#include <thread>
#include <Rcpp.h>
#include "locMatrix.hpp"
#include "likeMeth.hpp"
```

Include dependency graph for functions4R.cpp:



Functions

- Rcpp::List **reFit** (const std::vector< double > &yVec, const std::vector< double > &kVec, const int32_t &d, const int32_t &Ngen)
- Rcpp::List **reFitR** (const std::vector< double > &yVec, const std::vector< double > &kVec, const std::vector< int32_t > &repFac, const int32_t &d, const int32_t &Ngen)
- Rcpp::List **reFitF** (const std::vector< double > &yVec, const std::vector< double > &kVec, const std::vector< double > &xVec, const int32_t &d, const int32_t &Ngen)
- Rcpp::List **reFitRF** (const std::vector< double > &yVec, const std::vector< double > &kVec, const std::vector< int32_t > &repFac, const std::vector< double > &xVec, const int32_t &d, const int32_t &Ngen)
- Rcpp::List **gwa** (const std::vector< double > &yVec, const std::vector< double > &kVec, const std::vector< int32_t > &snps, const int32_t &d, const int32_t &Ngen, const int32_t &nThr)
- Rcpp::List **gwaF** (const std::vector< double > &yVec, const std::vector< double > &kVec, const std::vector< double > &xVec, const std::vector< int32_t > &snps, const int32_t &d, const int32_t &Ngen, const int32_t &nThr)
- Rcpp::List **gwaR** (const std::vector< double > &yVec, const std::vector< double > &kVec, const std::vector< int32_t > &repFac, const std::vector< int32_t > &snps, const int32_t &d, const int32_t &Ngen, const int32_t &nThr)
- Rcpp::List **gwaRF** (const std::vector< double > &yVec, const std::vector< double > &kVec, const std::vector< int32_t > &repFac, const std::vector< double > xVec, const std::vector< int32_t > &snps, const int32_t &d, const int32_t &Ngen, const int32_t &nThr)
- Rcpp::List **gwaFDR** (const std::vector< double > &yVec, const std::vector< double > &kVec, const std::vector< int32_t > &snps, const int32_t &d, const int32_t &Ngen, const int32_t &nPer, const int32_t &nThr)
- Rcpp::List **gwaFDRR** (const std::vector< double > &yVec, const std::vector< double > &kVec, const std↩ ::vector< int32_t > &repFac, const std::vector< int32_t > &snps, const int32_t &d, const int32_t &Ngen, const int32_t &nPer, const int32_t &nThr)
- Rcpp::List **gwaFDRF** (const std::vector< double > &yVec, const std::vector< double > &kVec, const std↩ ::vector< double > &xVec, const std::vector< int32_t > &snps, const int32_t &d, const int32_t &Ngen, const int32_t &nPer, const int32_t &nThr)
- Rcpp::List **gwaFDRRF** (const std::vector< double > &yVec, const std::vector< double > &kVec, const std↩ ::vector< int32_t > &repFac, const std::vector< double > &xVec, const std::vector< int32_t > &snps, const int32_t &d, const int32_t &Ngen, const int32_t &nPer, const int32_t &nThr)

8.1.1 Detailed Description

GWA on replicated data with a mixed model.

Author

Anthony J. Greenberg

Copyright

Version

0.1

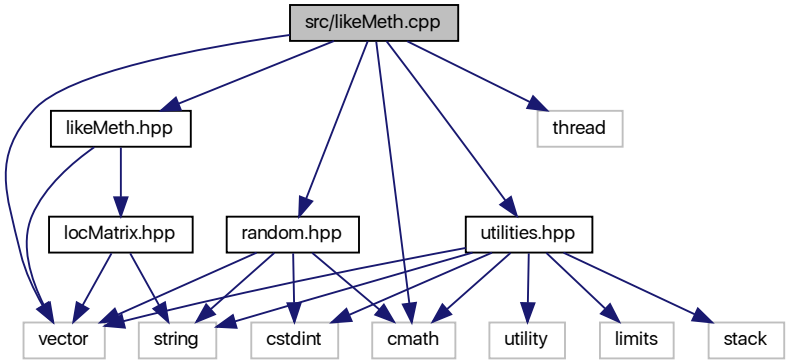
R interface functions that perform GWA on replicated data. Fixed-effect covariates and missing SNP data are allowed. SNPs have to be coded as (0,1,2) with missing data marked as -9. The implementation depends on C++-11. S↩ NP regression is multi-threaded. Vectorized matrices can be passed directly from R, no trasition to row-major storage needed. Multiple traits from the same experiment are treated at once, but the statistics are calculated independently (covariances effectively set to zero).

8.2 src/likeMeth.cpp File Reference

Likelihood methods for quantitative genetics.

```
#include <vector>
#include <cmath>
#include <thread>
#include "likeMeth.hpp"
#include "utilities.hpp"
#include "random.hpp"
```

Include dependency graph for likeMeth.cpp:



8.2.1 Detailed Description

Likelihood methods for quantitative genetics.

Author

Anthony J. Greenberg

Copyright

Version

0.1

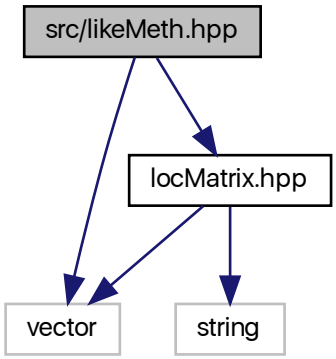
This is the file containing function implementations.

8.3 src/likeMeth.hpp File Reference

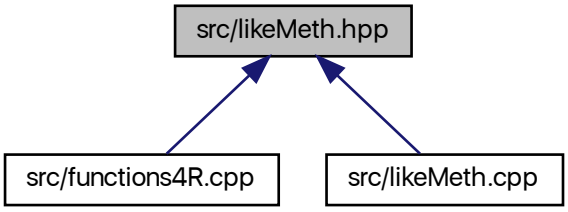
Likelihood methods for quantitative genetics.

```
#include <vector>
#include "locMatrix.hpp"
```

Include dependency graph for likeMeth.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class BayesicSpace::EmmREML

  EMMA REML functor class.
- class BayesicSpace::MixedModel

  Mixed model.
- class BayesicSpace::SNPblock

  A SNP block functor class.

8.3.1 Detailed Description

Likelihood methods for quantitative genetics.

Author

Anthony J. Greenberg

Copyright

Version

0.1

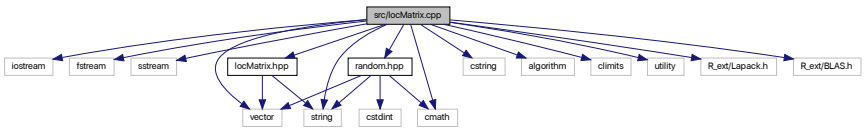
This is the project header file containing class definitions and interface documentation.

8.4 src/locMatrix.cpp File Reference

C++ matrix class for development.

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <vector>
#include <string>
#include <cstring>
#include <algorithm>
#include <cmath>
#include <climits>
#include <utility>
#include <R_ext/Lapack.h>
#include <R_ext/BLAS.h>
#include "locMatrix.hpp"
#include "random.hpp"
```

Include dependency graph for locMatrix.cpp:



8.4.1 Detailed Description

C++ matrix class for development.

Author

Anthony J. Greenberg

Copyright

Version

0.1

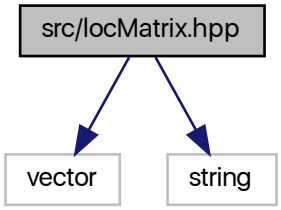
This is the class implementation file for the experimental Matrix class. This version is for including in R packages, so it uses the R BLAS and LAPACK interfaces.

8.5 src/locMatrix.hpp File Reference

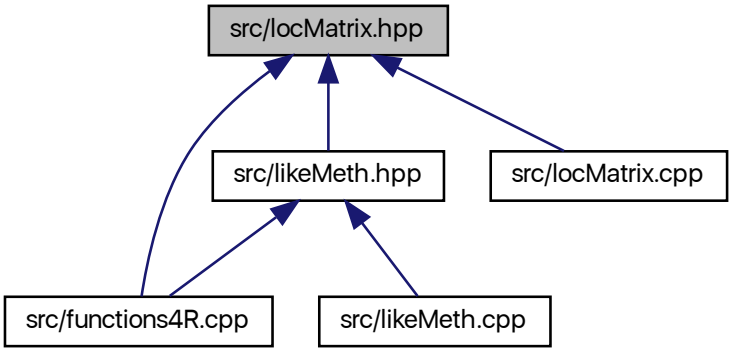
C++ matrix class for development.

```
#include <vector>
#include <string>
```

Include dependency graph for locMatrix.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class BayesicSpace::Matrix

  Test matrix class.

Functions

- Matrix BayesicSpace::operator∗ (const double &scal, const Matrix &m)

  Scalar-matrix product.
- Matrix BayesicSpace::operator+ (const double &scal, const Matrix &m)

  Scalar-matrix addition.

8.5.1 Detailed Description

C++ matrix class for development.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2016 Anthony J. Greenberg

Version

0.1

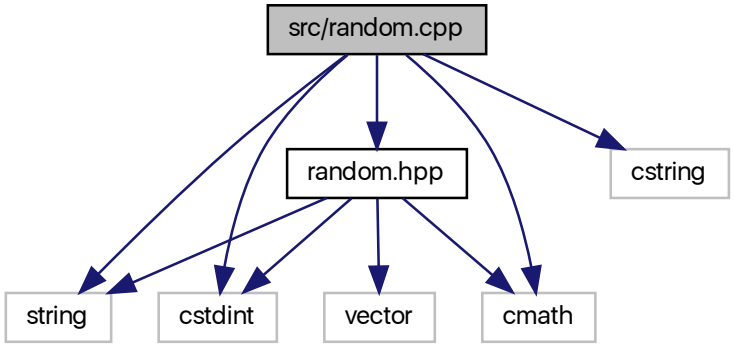
This is the project header file containing class definitions and interface documentation.

8.6 src/random.cpp File Reference

Random number generation.

```
#include <string>
#include <cstring>
#include <cstdint>
#include <cmath>
#include "random.hpp"
```

Include dependency graph for random.cpp:



8.6.1 Detailed Description

Random number generation.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2017 Anthony J. Greenberg

Version

1.0

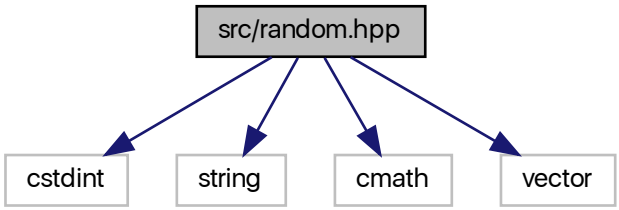
Class implementation for facilities that generate random draws from various distributions.

8.7 src/random.hpp File Reference

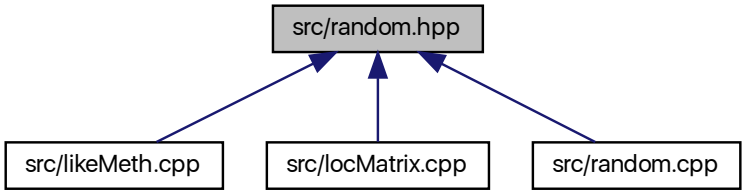
Random number generation.

```
#include <cstdint>
#include <string>
#include <cmath>
#include <vector>
```

Include dependency graph for random.hpp:



This graph shows which files directly or indirectly include this file:



Classes

- class BayesicSpace::Generate

  Abstract base random number class.
- class BayesicSpace::GenerateHR

  Hardware random number generating class.
- class BayesicSpace::GenerateMT

  Pseudo-random number generator.
- class BayesicSpace::RanDraw

  Random number generating class.

8.7.1 Detailed Description

Random number generation.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2017 Anthony J. Greenberg

Version

1.0

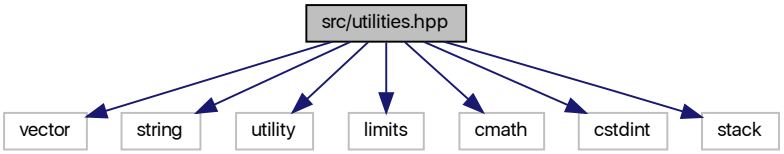
Class definition and interface documentation for facilities that generate random draws from various distributions.

8.8 src/utilities.hpp File Reference

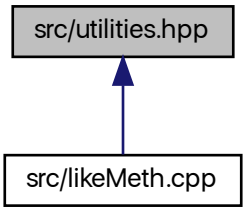
Miscellaneous functions and algorithms.

```
#include <vector>
#include <string>
#include <utility>
#include <limits>
#include <cmath>
#include <cstdint>
#include <stack>
```

Include dependency graph for utilities.hpp:



This graph shows which files directly or indirectly include this file:



Functions

- void BayesicSpace::swapXOR (uint64_t &i, uint64_t &j)

  Swap two `uint64_t` *values.*
- void BayesicSpace::swapXOR (int64_t &i, int64_t &j)

  Swap two `in64_t` *values.*
- void BayesicSpace::swapXOR (uint32_t &i, uint32_t &j)

  Swap two `uint32_t` *values.*
- void BayesicSpace::swapXOR (int32_t &i, int32_t &j)

  Swap two `int32_t` *values.*
- double BayesicSpace::mean (const double ∗arr, const size_t &len)

  Mean of a C array.
- double BayesicSpace::mean (const double ∗arr, const size_t &len, const size_t &stride)

  Mean of a C array with stride.
- double BayesicSpace::mean (const vector< double > &vec)

  Mean of a C++ vector.
- double BayesicSpace::mean (const vector< double > &vec, const size_t &stride)

  Mean of a C++ vector with stride.
- double BayesicSpace::pow2 (const double &x)

  Square of a double.
- void BayesicSpace::shft3 (double &a, double &b, double &c, const double &d)

  Shift three values left.
- template<class T >
  void BayesicSpace::bracketMax (T &func, const double &startA, const double &startB, double &candA, double &candB, double &candC)

  Bracket a maximum.
- template<class T >
  void BayesicSpace::maximizer (T &func, const double &startX, double &xMax, double &fMax)

  Find the value that maximizes a function.
- double BayesicSpace::lnGamma (const double &x)

  Logarithm of the Gamma function.
- double BayesicSpace::betacf (const double &x, const double &a, const double &b)

Continued fraction of the Beta function.

- double BayesicSpace::betaiapprox (const double &x, const double &a, const double &b)

  Regularized incomplete Beta function.
- double BayesicSpace::betai (const double &x, const double &a, const double &b)

  Regularized incomplete Beta function.
- void BayesicSpace::shellSort (const vector< double > &target, const size_t &beg, const size_t &end, vector< size_t > &outIdx)

  Shell sort.
- void BayesicSpace::quickSort (const vector< double > &target, const size_t &beg, const size_t &end, vector< size_t > &outIdx)

Variables

- const double BayesicSpace::BS_PI = 3.14159265358979323846264338328

  The definition of π.
- const double BayesicSpace::BS_EPS = numeric_limits<double>::epsilon()

  Machine ϵ.
- const double BayesicSpace::BS_FPMIN = numeric_limits<double>::min()/BS_EPS

  Tiny value to guard agains underflow.

8.8.1 Detailed Description

Miscellaneous functions and algorithms.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2016 Anthony J. Greenberg

Version

0.1

This is the project header file containing function definitions and constants.

8.8.2 Function Documentation

8.8.2.1 betacf()

```
double BayesicSpace::betacf (
            const double & x,
            const double & a,
            const double & b )
```

Continued fraction of the Beta function.

Computes the continued fraction of the Beta function following the Lenz method (see Numerical Recipes in C++). To be used in the *betai* function.

Parameters

in	*x*	value
in	*a*	shape parameter a
in	*b*	shape parameter b

Returns

continued fraction value

8.8.2.2 betai()

```
double BayesicSpace::betai (
            const double & x,
            const double & a,
            const double & b )
```

Regularized incomplete Beta function.

Computes the regularized incomplete Beta function following the method in Numerical Recipes in C++.

Parameters

in	*x*	value
in	*a*	shape parameter a
in	*b*	shape parameter b

Returns

$I_x(a, b)$

8.8.2.3 betaiapprox()

```
double BayesicSpace::betaiapprox (
            const double & x,
            const double & a,
            const double & b )
```

Regularized incomplete Beta function.

Computes a quadrature approximatino of the regularized incomplete Beta function following the method in Numerical Recipes in C++. To be used in the *betai* function.

Parameters

in	*x*	value
in	*a*	shape parameter a
in	*b*	shape parameter b

Returns

approximate $I_x(a, b)$ s

8.8.2.4 bracketMax()

```
template<class T >
void BayesicSpace::bracketMax (
            T & func,
            const double & startA,
            const double & startB,
            double & candA,
            double & candB,
            double & candC )
```

Bracket a maximum.

Brackets a maximum of a function given two initial guesses. Based on the Numerical Recipes in C++ function. Using max rather than min because max is more important in statistical applications. The resulting bracketing values are candA $<$ candB $<$ candC.

Parameters

in	*func*	function to maximize. Has to take a double and return a double
in	*startA*	starting value A
in	*startB*	starting value B
out	*candA*	first bracketing value
out	*candB*	second bracketing value
out	*candC*	third breacketing value

8.8.2.5 lnGamma()

```
double BayesicSpace::lnGamma (
            const double & x )
```

Logarithm of the Gamma function.

The log of the $\Gamma(x)$ function. Implementing the Lanczos algorythm following Numerical Recipes in C++.

Parameters

in	*x*	value

Returns

$\log \Gamma(x)$

8.8.2.6 maximizer()

```
template<class T >
void BayesicSpace::maximizer (
            T & func,
            const double & startX,
            double & xMax,
            double & fMax )
```

Find the value that maximizes a function.

Uses the Brent method to find the value of x that maximizes a function. Modification of the implementation found in Numerical Recipes in C++. Maximizing rather than minimizing because that is the most common application in statistics. Tolerance is set at $1.001 \times \sqrt{\epsilon}$, where ϵ is machine floating-point precision for *double*. This is just above the theoretical limit of precision.

Parameters

in	*func*	functor that represents the function to be maximized
in	*startX*	starting value
out	*xMax*	value of x at maximum
out	*fMax*	function value at maximum

8.8.2.7 mean() [1/4]

```
double BayesicSpace::mean (
            const double * arr,
            const size_t & len )
```

Mean of a C array.

Calculates a mean of an array. Uses the recursive algorithm for numerical stability.

Parameters

in	*arr*	array to average
in	*len*	array length

8.8.2.8 mean() [2/4]

```
double BayesicSpace::mean (
            const double * arr,
            const size_t & len,
            const size_t & stride )
```

Mean of a C array with stride.

Calculates a mean of an array with stride (i.e., using every *stride* element). Uses the recursive algorithm for numerical stability.

Parameters

in	*arr*	array to average
in	*len*	array length
in	*stride*	stride

8.8.2.9 mean() [3/4]

```
double BayesicSpace::mean (
            const vector< double > & vec )
```

Mean of a C++ vector.

Calculates a mean of a vector. Uses the recursive algorithm for numerical stability.

Parameters

in	*vec*	vector to average

8.8.2.10 mean() [4/4]

```
double BayesicSpace::mean (
```

```
            const vector< double > & vec,
            const size_t & stride )
```

Mean of a C++ vector with stride.

Calculates a mean of a vector with stride (i.e., using every *stride* element). Uses the recursive algorithm for numerical stability.

Parameters

in	*vec*	vector to average
in	*stride*	stride

8.8.2.11 pow2()

```
double BayesicSpace::pow2 (
            const double & x )  [inline]
```

Square of a double.

Parameters

in	*x*	value to square

Returns

double square of the input

8.8.2.12 quickSort()

```
void BayesicSpace::quickSort (
            const vector< double > & target,
            const size_t & beg,
            const size_t & end,
            vector< size_t > & outIdx )
```

Quicksort

This function implements the Quicksort algorithm, taking the Numerical Recipes implementation as a base. It rearranges the indexes in the output vector rather than move around the elements of the target vector. The output index must be the same size as the target (this is checked and exception thown if the condidition is not met). The output index is initialized with the correct index values.

Parameters

in	*target*	vector to be sorted
in	*beg*	index of the first element
in	*end*	index of one past the last element to be included
out	*outIdx*	vector of indexes

8.8.2.13 shellSort()

```
void BayesicSpace::shellSort (
            const vector< double > & target,
            const size_t & beg,
            const size_t & end,
            vector< size_t > & outIdx )
```

Shell sort.

Sorts the provided vector in ascending order using Shell's method. Rather than move the elements themselves, save their indexes to the output vector. The first element of the index vector points to the smallest element of the input vector etc. The implementation is modified from code in Numerical Recipes in C++. NOTE: This algorithm is too slow for vectors of > 50 elements. I am using it to finish off the quickSort, this is why I am giving it a range within a larger vector.

Parameters

in	*target*	vector to be sorted
in	*beg*	index of the first element
in	*end*	index of one past the last element to be included
out	*outIdx*	vector of indexes

8.8.2.14 shft3()

```
void BayesicSpace::shft3 (
            double & a,
            double & b,
            double & c,
            const double & d )  [inline]
```

Shift three values left.

Shifts a new value, moving the old to the left. The first value is discarded.

Parameters

in,out	*a*	first value (becomes *b*)
in,out	*b*	second value (becomes *c*)
in,out	*c*	third value (becomes *d*)
in	*d*	unchanged fourth value (shifted to *c*)

8.8.2.15 swapXOR() [1/4]

```
void BayesicSpace::swapXOR (
            int32_t & i,
            int32_t & j )
```

Swap two `int32_t` values.

Uses the three XORs trick to swap two integers. Safe if the variables happen to refer to the same address.

Parameters

in,out	*i*	first integer
in,out	*j*	second integer

8.8.2.16 swapXOR() [2/4]

```
void BayesicSpace::swapXOR (
            int64_t & i,
            int64_t & j )
```

Swap two `in64_t` values.

Uses the three XORs trick to swap two integers. Safe if the variables happen to refer to the same address.

Parameters

in,out	*i*	first integer
in,out	*j*	second integer

8.8.2.17 swapXOR() [3/4]

```
void BayesicSpace::swapXOR (
            uint32_t & i,
            uint32_t & j )
```

Swap two uint32_t values.

Uses the three XORs trick to swap two integers. Safe if the variables happen to refer to the same address.

Parameters

in,out	*i*	first integer
in,out	*j*	second integer

8.8.2.18 swapXOR() [4/4]

```
void BayesicSpace::swapXOR (
            uint64_t & i,
            uint64_t & j )
```

Swap two uint64_t values.

Uses the three XORs trick to swap two integers. Safe if the variables happen to refer to the same address.

Parameters

in,out	*i*	first integer
in,out	*j*	second integer

Index