Gaussian mixture models

Generated by Doxygen 1.8.20

Chapter 1

Package description

This is an R package to assign individuals to groups based on phenotypic data from multiple traits using Bayesian inference. While it is geared towards biological data, any multivariate data set can be analyzed this way. The model is a Gaussian mixture, fit using variational Bayes. A previous version hosted here used Hamiltonian Monte Carlo. This is still in development and will be added later. Goodness of fit is assessed using the deviance information criterion (DIC). The implementation is based on McGrory and Titterington (2007)

The package is still in development. Basic model fitting with no replication and no covariates is functional. The following functionality is in development:

- Full Bayesian treatment with MCMC
- Functions to summarize results and provide convergence diagnostics
- Support for covariates
- Arbitrary replication levels

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

To fit the model, run the `quickFitModel` function (`?quickFitModel` for help). There is a plot method for the object returned by this function, which shows group assignment probabilities for each individual, similar to the plots that are used with `STRUCTURE`.

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

Chapter 5

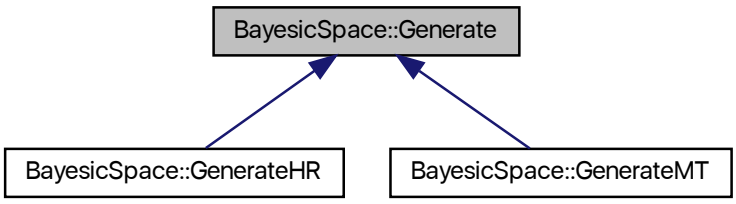
Class Documentation

5.1 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.

5.1.1 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.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Generate() [1/2]

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

Protected copy constructor.

Parameters

in	*old*	object to copy

5.1.2.2 Generate() [2/2]

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

Protected move constructor.

Parameters

in	*old*	object to move

5.1.3 Member Function Documentation

5.1.3.1 operator=() [1/2]

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

Protected copy assignment operator.

Parameters

in	*old*	object to copy

5.1.3.2 operator=() [2/2]

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

Protected move assignment.

Parameters

in	*old*	object to move

5.1.3.3 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:

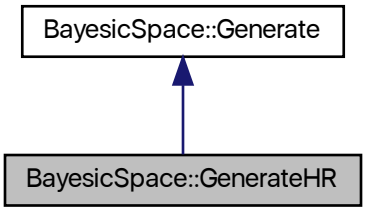
- src/random.hpp

5.2 BayesicSpace::GenerateHR Class Reference

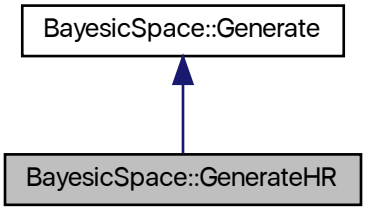
Hardware random number generating class.

`#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 override

  Generate a random 64-bit unsigned integer.

Additional Inherited Members

5.2.1 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`.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 GenerateHR() [1/2]

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

Copy constructor.

Parameters

in	*old*	object to copy

5.2.2.2 GenerateHR() [2/2]

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

Move constructor.

Parameters

in	*old*	object to move

5.2.3 Member Function Documentation

5.2.3.1 operator=() [1/2]

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

Copy assignment operator.

Parameters

in	*old*	object to copy

5.2.3.2 operator=() [2/2]

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

Move assignment.

Parameters

in	*old*	object to move

5.2.3.3 ranInt()

```
uint64_t GenerateHR::ranInt ( ) const  [override], [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:

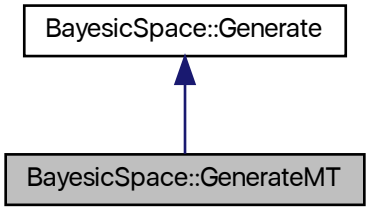
- src/random.hpp
- src/random.cpp

5.3 BayesicSpace::GenerateMT Class Reference

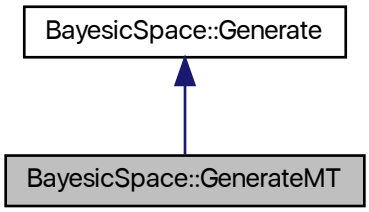
Pseudo-random number generator.

```
#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 override

  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

5.3.1 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.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 GenerateMT() [1/3]

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

Default constructor.

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

5.3.2.2 GenerateMT() [2/3]

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

Copy constructor.

Parameters

in	*old*	object to copy

5.3.2.3 GenerateMT() [3/3]

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

Move constructor.

Parameters

in	*old*	object to move

5.3.3 Member Function Documentation

5.3.3.1 operator=() [1/2]

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

Copy assignment operator.

Parameters

in	*old*	object to copy

5.3.3.2 operator=() [2/2]

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

Move assignment.

Parameters

in	*old*	object to move

5.3.3.3 ranInt()

```
uint64_t GenerateMT::ranInt ( ) const  [override], [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:

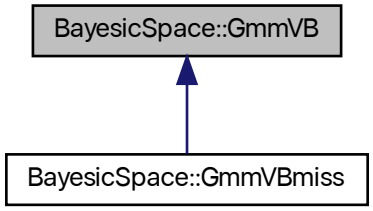
- src/random.hpp
- src/random.cpp

5.4 BayesicSpace::GmmVB Class Reference

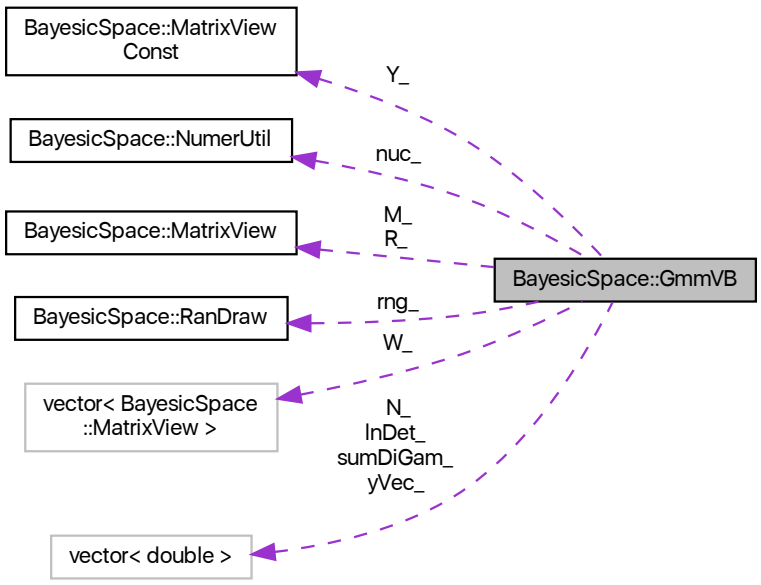
Variational Bayes class.

```
#include <gmmvb.hpp>
```

Inheritance diagram for BayesicSpace::GmmVB:



Collaboration diagram for BayesicSpace::GmmVB:



Public Member Functions

- GmmVB ()

  Default constructor.
- GmmVB (const vector< double > *yVec, const double &lambda0, const double &sigmaSq0, const double alpha0, const size_t &nPop, const size_t &d, vector< double > *vPopMn, vector< double > *vSm, vector< double > *resp, vector< double > *Nm)

  Constructor.
- ~GmmVB ()

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

  Copy constructor (deleted)
- GmmVB & operator= (const GmmVB &in)=delete

  Copy assignment (deleted)
- GmmVB (GmmVB &&in)

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

  Move assignment operator (deleted)
- virtual void fitModel (vector< double > &logPost, double &dic)

  Fit model.

Protected Member Functions

- virtual void eStep_ ()

  The E-step.
- virtual void mStep_ ()

  The M-step.
- virtual double logPost_ ()

  Log-posterior function.
- double rowDistance_ (const MatrixViewConst &m1, const size_t &row1, const MatrixView &m2, const size_t &row2)

  Euclidean distance between matrix rows.
- virtual void kMeans_ (const MatrixViewConst &X, const size_t &Kclust, const uint32_t &maxIt, Index &x2m, MatrixView &M)

  K-means clustering.

Protected Attributes

- const vector< double > * yVec_

  Pointer to vectorized data matrix.
- MatrixViewConst Y_

  Matrix view of the data.
- MatrixView M_

  Population means matrix view.
- vector< MatrixView > W_

  Vector of weighted covariance matrix views.
- vector< double > lnDet_

  W_ log-determinants
- vector< double > sumDiGam_

  Sum of digammas.
- MatrixView R_

  Matrix view of responsibilities.
- vector< double > * N_

  Pointer to vector of effective population sizes.
- const double lambda0_

  Prior covariance scale factor.
- const double nu0_

  Prior precision degrees of freedom.
- const double sigmaSq0_

  Prior variance.
- const double alpha0_

  Prior population size.
- const double d_

  Double version of the trait number.
- const double nu0p2_

  nu_0 + 2
- const double nu0p1_

  nu_0 + 1

- const double dln2_

  $d \ln 2$
- const uint16_t maxIt_

  Maximum number of iterations.
- const double stoppingDiff_

  Stopping criterion.
- NumerUtil nuc_

  Numerical utilities.
- RanDraw rng_

  Random numbers.

Static Protected Attributes

- static const double lnMaxDbl_ = log(numeric_limits<double>::max())

  Natural log of `DBL_MAX`

5.4.1 Detailed Description

Variational Bayes class.

Implements Gaussian mixture model fit using variational Bayes

5.4.2 Constructor & Destructor Documentation

5.4.2.1 GmmVB() [1/2]

```
GmmVB::GmmVB (
            const vector< double > * yVec,
            const double & lambda0,
            const double & sigmaSq0,
            const double alpha0,
            const size_t & nPop,
            const size_t & d,
            vector< double > * vPopMn,
            vector< double > * vSm,
            vector< double > * resp,
            vector< double > * Nm )
```

Constructor.

The vectorized matrices must be in the column major format (as in R and FORTRAN). For larger population numbers, make sure $\nu_0 > d - 2$.

Parameters

in	*yVec*	pointer to vectorized data matrix
in	*lambda0*	prior precision scale factor
in	*sigmaSq0*	prior variance
in	*alpha0*	prior population size
in	*nPop*	number of populations
in	*d*	number of traits
in,out	*vPopMn*	pointer to vectorized matrix of population means
in,out	*vSm*	pointer to vectorized collection of population covariances
in,out	*resp*	pointer to vectorized matrix responsibilities
in,out	*Nm*	pointer to vector of effective population sizes

5.4.2.2 GmmVB() [2/2]

```
GmmVB::GmmVB (
            GmmVB && in )
```

Move constructor.

Parameters

in	*in*	object to move

5.4.3 Member Function Documentation

5.4.3.1 fitModel()

```
void GmmVB::fitModel (
            vector< double > & logPost,
            double & dic )  [virtual]
```

Fit model.

Fits the model, returning the log-posterior for each step and the end-result deviance information criterion (DIC).

Parameters

out	*logPost*	vector of lower bounds
out	*dic*	the DIC value

Reimplemented in BayesicSpace::GmmVBmiss.

5.4.3.2 kMeans_()

```
void GmmVB::kMeans_ (
            const MatrixViewConst & X,
            const size_t & Kclust,
            const uint32_t & maxIt,
            Index & x2m,
            MatrixView & M )  [protected], [virtual]
```

K-means clustering.

Performs k-means clustering on a matrix of values. Each row of the input matrix is an item with observed values in columns.

Parameters

in	*X*	matrix of observations to be clustered
in	*Kclust*	number of clusters
in	*maxIt*	maximum number of iterations
out	*x2m*	Index relating clusters to values
out	*M*	matrix of cluster means (clusters in rows)

Reimplemented in BayesicSpace::GmmVBmiss.

5.4.3.3 logPost_()

```
double GmmVB::logPost_ ( )  [protected], [virtual]
```

Log-posterior function.

Calculates the log-posterior to monitor convergence.

Returns

the log-posterior value

Reimplemented in BayesicSpace::GmmVBmiss.

5.4.3.4 operator=()

```
GmmVB& BayesicSpace::GmmVB::operator= (
            GmmVB && in )  [delete]
```

Move assignment operator (deleted)

Parameters

in	*in*	object to be moved

Returns

target object

5.4.3.5 rowDistance_()

```
double GmmVB::rowDistance_ (
            const MatrixViewConst & m1,
            const size_t & row1,
            const MatrixView & m2,
            const size_t & row2 )  [protected]
```

Euclidean distance between matrix rows.

Parameters

in	*m1*	first matrix
in	*row1*	index of the first matrix row
in	*m2*	second matrix
in	*row2*	index of the second matrix row

Returns

euclidean distance between the rows

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

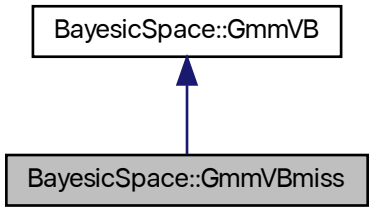
- src/gmmvb.hpp
- src/gmmvb.cpp

5.5 BayesicSpace::GmmVBmiss Class Reference

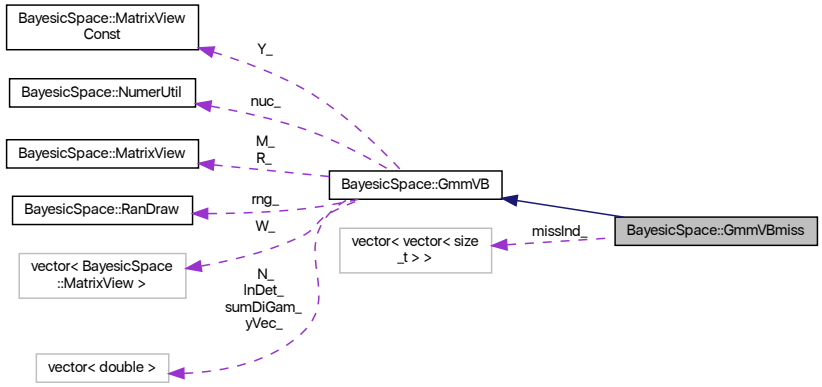
Variational Bayes with missing data.

```
#include <gmmvb.hpp>
```

Inheritance diagram for BayesicSpace::GmmVBmiss:



Collaboration diagram for BayesicSpace::GmmVBmiss:



Public Member Functions

- GmmVBmiss ()

  Default constructor.
- GmmVBmiss (vector< double > *yVec, const double &lambda0, const double &sigmaSq0, const double alpha0, const size_t &nPop, const size_t &d, vector< double > *vPopMn, vector< double > *vSm, vector< double > *resp, vector< double > *Nm)

  Constructor.
- ~GmmVBmiss ()

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

  Copy constructor (deleted)
- GmmVBmiss & operator= (const GmmVBmiss &in)=delete

  Copy assignment (deleted)

- GmmVBmiss (GmmVBmiss &&in)

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

  Move assignment operator (deleted)
- void fitModel (vector< double > &logPost, double &dic) override

  Fit model.

Protected Member Functions

- void eStep_ () override

  The E-step.
- void mStep_ () override

  The M-step.
- double logPost_ () override

  Log-posterior function.
- double rowDistance_ (const MatrixViewConst &m1, const size_t &row1, const MatrixView &m2, const size_t &row2, const vector< size_t > &presInd)

  Euclidean distance between matrix rows.
- void kMeans_ (const MatrixViewConst &X, const size_t &Kclust, const uint32_t &maxIt, Index &x2m, MatrixView &M) override

  K-means clustering.

Protected Attributes

- vector< vector< size_t > > missInd_

  Missing data indixes.

Additional Inherited Members

5.5.1 Detailed Description

Variational Bayes with missing data.

Implements Gaussian mixture model fit using variational Bayes. Missing data are allowed.

5.5.2 Constructor & Destructor Documentation

5.5.2.1 GmmVBmiss() [1/2]

```
GmmVBmiss::GmmVBmiss (
            vector< double > * yVec,
            const double & lambda0,
            const double & sigmaSq0,
            const double alpha0,
            const size_t & nPop,
            const size_t & d,
            vector< double > * vPopMn,
            vector< double > * vSm,
            vector< double > * resp,
            vector< double > * Nm )
```

Constructor.

The vectorized matrices must be in the column major format (as in R and FORTRAN). Missing values must be marked with `nan("")`. For larger population numbers, make sure $\nu_0 > d - 2$.

Parameters

in	*yVec*	pointer to vectorized data matrix
in	*lambda0*	prior precision scale factor
in	*sigmaSq0*	prior variance
in	*alpha0*	prior population size
in	*nPop*	number of populations
in	*d*	number of traits
in,out	*vPopMn*	pointer to vectorized matrix of population means
in,out	*vSm*	pointer to vectorized collection of population covariances
in,out	*resp*	pointer to vectorized matrix responsibilities
in,out	*Nm*	pointer to vector of effective population sizes

5.5.2.2 GmmVBmiss() [2/2]

```
BayesicSpace::GmmVBmiss::GmmVBmiss (
            GmmVBmiss && in )
```

Move constructor.

Parameters

in	*in*	object to move

5.5.3 Member Function Documentation

5.5.3.1 fitModel()

```
void GmmVBmiss::fitModel (
            vector< double > & logPost,
            double & dic )  [override], [virtual]
```

Fit model.

Fits the model, returning the log-posterior for each step and the end-result deviance information criterion (DIC).

Parameters

out	*logPost*	vector of lower bounds
out	*dic*	the DIC value

Reimplemented from BayesicSpace::GmmVB.

5.5.3.2 kMeans_()

```
void GmmVBmiss::kMeans_ (
            const MatrixViewConst & X,
            const size_t & Kclust,
            const uint32_t & maxIt,
            Index & x2m,
            MatrixView & M )  [override], [protected], [virtual]
```

K-means clustering.

Performs k-means clustering on a matrix of values. Each row of the input matrix is an item with observed values in columns.

Parameters

in	*X*	matrix of observations to be clustered
in	*Kclust*	number of clusters
in	*maxIt*	maximum number of iterations
out	*x2m*	Index relating clusters to values
out	*M*	matrix of cluster means (clusters in rows)

Reimplemented from BayesicSpace::GmmVB.

5.5.3.3 logPost_()

```
double GmmVBmiss::logPost_ ( )  [override], [protected], [virtual]
```

Log-posterior function.

Calculates the log-posterior to monitor convergence.

Returns

the log-posterior value

Reimplemented from BayesicSpace::GmmVB.

5.5.3.4 operator=()

```
GmmVBmiss& BayesicSpace::GmmVBmiss::operator= (
            GmmVBmiss && in )  [delete]
```

Move assignment operator (deleted)

Parameters

in	*in*	object to be moved

Returns

target object

5.5.3.5 rowDistance_()

```
double GmmVBmiss::rowDistance_ (
            const MatrixViewConst & m1,
            const size_t & row1,
            const MatrixView & m2,
            const size_t & row2,
            const vector< size_t > & presInd )  [protected]
```

Euclidean distance between matrix rows.

Parameters

in	*m1*	first matrix
in	*row1*	index of the first matrix row
in	*m2*	second matrix
in	*row2*	index of the second matrix row
in	*presInd*	index of present (not missing) values

Returns

euclidean distance between the rows

5.5.4 Member Data Documentation

5.5.4.1 missInd_

```
vector< vector<size_t> > BayesicSpace::GmmVBmiss::missInd_  [protected]
```

Missing data indixes.

The outer vector corresponds to data matrix columns. The inner vectors contain row indexes of missing data positions.

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

- src/gmmvb.hpp
- src/gmmvb.cpp

5.6 BayesicSpace::Index Class Reference

Group index.

```
#include <index.hpp>
```

Public Member Functions

- Index ()

  Default constructor.
- Index (const size_t &Ngroups)

  Group constructor.
- Index (const size_t *arr, const size_t &N)

  Array constructor.
- Index (const vector< size_t > &vec)

  Vector constructor.
- Index (const string &inFileName)

  File read constructor.
- Index (const Index &in)

  Copy constructor.
- Index & operator= (const Index &in)

  Copy assignment operator.
- Index (Index &&in)

  Move constructor.
- Index & operator= (Index &&in)

  Move assignment operator.
- ~Index ()

  Destructor.
- const vector< size_t > & operator[ ] (const size_t &i) const

  Vector subscript operator.
- size_t groupSize (const size_t &i) const

  Group size.
- size_t size () const

  Total sample size.
- size_t groupNumber () const

  Number of groups.
- size_t neGroupNumber () const

  Number of non-empty groups.
- size_t groupID (const size_t &ind) const

  Group ID.
- void update (const vector< size_t > &newVec)

  Update the index.

5.6.1 Detailed Description

Group index.

For each group, contains indexes of the lines that belong to it. Can also identify the group a given element belongs to. Group numbers need not be consecutive. Although group IDs are assumed to be base-0, everything should work even if they are not.

5.6.2 Constructor & Destructor Documentation

5.6.2.1 Index() [1/6]

```
Index::Index (
            const size_t & Ngroups )
```

Group constructor.

Sets up empty groups.

Parameters

in	*Ngroups*	number of groups to set up

5.6.2.2 Index() [2/6]

```
Index::Index (
            const size_t * arr,
            const size_t & N )
```

Array constructor.

The input array has an element for each line, and the value of that element is the base-0 group ID (i.e., if line *n* is in the first group, then `arr[n] == 0`).

Parameters

in	*arr*	array of group IDs
in	*N*	array length

5.6.2.3 Index() [3/6]

```
Index::Index (
            const vector< size_t > & vec )
```

Vector constructor.

The input vector has an element for each line, and the value of that element is the base-0 group ID (i.e., if line *n* is in the first group, then `vec[n] == 0`).

Parameters

in	*vec*	array of group IDs

5.6.2.4 Index() [4/6]

```
Index::Index (
            const string & inFileName )
```

File read constructor.

The input file has an entry for each line (separated by white space), and the value of that entry is the base-0 group ID. If the file cannot be opened, throws "Cannot open file file_name". If a negative group value is detected, thorws "Negative group ID".

Parameters

in	*inFileName*	input file name

5.6.2.5 Index() [5/6]

```
BayesicSpace::Index::Index (
            const Index & in )  [inline]
```

Copy constructor.

Parameters

in	*in*	Index to be copied

Returns

Index object

5.6.2.6 Index() [6/6]

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

Move constructor.

Parameters

in	*in*	Index object to be moved

Returns

Index object

5.6.3 Member Function Documentation

5.6.3.1 groupID()

```
size_t BayesicSpace::Index::groupID (
            const size_t & ind ) const  [inline]
```

Group ID.

Returns the group ID for a given individual.

Parameters

in	*ind*	index of an individual

Returns

group ID

5.6.3.2 groupNumber()

```
size_t BayesicSpace::Index::groupNumber ( ) const  [inline]
```

Number of groups.

Returns

number of groups

5.6.3.3 groupSize()

```
size_t BayesicSpace::Index::groupSize (
            const size_t & i ) const  [inline]
```

Group size.

Parameters

in	*i*	group index

Returns

size of the _i_th group

5.6.3.4 neGroupNumber()

```
size_t Index::neGroupNumber ( ) const
```

Number of non-empty groups.

Returns

number of non-empty groups

5.6.3.5 operator=() [1/2]

```
Index & Index::operator= (
            const Index & in )
```

Copy assignment operator.

Parameters

in	*in*	object to be copied

Returns

an Index object

5.6.3.6 operator=() [2/2]

```
Index & Index::operator= (
            Index && in )
```

Move assignment operator.

Parameters

in	*in*	object to be moved

Returns

an Index object

5.6.3.7 operator[]()

```
const vector<size_t>& BayesicSpace::Index::operator[] (
            const size_t & i ) const  [inline]
```

Vector subscript operator.

Returns the index of group *i*.

Parameters

in	*i*	group index

Returns

index of line IDs

5.6.3.8 size()

```
size_t BayesicSpace::Index::size ( ) const  [inline]
```

Total sample size.

Returns

total sample size

5.6.3.9 update()

```
void Index::update (
            const vector< size_t > & newVec )
```

Update the index.

Updates the groups with a new index. If a group is not present in the new vector, it is left empty but still exists.

Parameters

in	*newVec*	new vector of group IDs

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

- src/index.hpp
- src/index.cpp

5.7 BayesicSpace::MatrixView Class Reference

Matrix view of a `vector`

`#include <matrixView.hpp>`

Public Member Functions

- MatrixView ()
  Default constructor.
- MatrixView (vector< double > *inVec, const size_t &idx, const size_t &nrow, const size_t &ncol)
  Constructor pointing to a C++ `vector`
- ~MatrixView ()
  Destructor.
- MatrixView (const MatrixView &inMat)=delete
  Copy constructor (deleted)
- MatrixView & operator= (const MatrixView &inMat)=delete
  Copy assignment operator (deleted)
- MatrixView (MatrixView &&inMat)
  Move constructor.
- MatrixView & operator= (MatrixView &&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 addToElem (const size_t &iRow, const size_t &jCol, const double &input)
  Add a scalar to an element.
- void subtractFromElem (const size_t &iRow, const size_t &jCol, const double &input)

Subtract a scalar from an element.

- void multiplyElem (const size_t &iRow, const size_t &jCol, const double &input)

  Multiply an element by a scalar.
- void divideElem (const size_t &iRow, const size_t &jCol, const double &input)

  Divide an element by a scalar.
- void permuteCols (const vector< size_t > &idx)

  Permute columns.
- void permuteRows (const vector< size_t > &idx)

  Permute rows.
- void chol ()

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

  Copy Cholesky decomposition.
- void cholInv ()

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

  Copy Cholesky inverse.
- void pseudoInv ()

  In-place pseudoinverse.
- void pseudoInv (double &lDet)

  In-place pseudoinverse with log-determinant.
- void pseudoInv (MatrixView &out) const

  Copy pseudoinverse.
- void pseudoInv (MatrixView &out, double &lDet) const

  Copy pseudoinverse with log-determinant.
- void svd (MatrixView &U, vector< double > &s)

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

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

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

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

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

  Some eigenvalues and vectors of a symmetric matrix ("safe")
- void syrk (const char &tri, const double &alpha, const double &beta, MatrixView &C) const

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

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

  Multiply by symmetric matrix.
- void symm (const char &tri, const char &side, const double &alpha, const MatrixViewConst &symA, const double &beta, MatrixView &C) const

  Multiply by symmetric `MatrixViewConst`

- void symc (const char &tri, const double &alpha, const MatrixView &X, const size_t &xCol, const double &beta, vector< double > &y) const
- void symc (const char &tri, const double &alpha, const MatrixViewConst &X, const size_t &xCol, const double &beta, vector< double > &y) const
- void trm (const char &tri, const char &side, const bool &transA, const bool &uDiag, const double &alpha, const MatrixView &trA)

  Multiply by triangular matrix.
- void trm (const char &tri, const char &side, const bool &transA, const bool &uDiag, const double &alpha, const MatrixViewConst &trA)

  Multiply by triangular MatrixViewConst
- void gemm (const bool &transA, const double &alpha, const MatrixView &A, const bool &transB, const double &beta, MatrixView &C) const

  General matrix multiplication.
- void gemm (const bool &transA, const double &alpha, const MatrixViewConst &A, const bool &transB, const double &beta, MatrixView &C) const

  General matrix multiplication with `MatrixViewConst`
- void gemc (const bool &trans, const double &alpha, const MatrixView &X, const size_t &xCol, const double &beta, vector< double > &y) const

  Multiply a general matrix by a column of another matrix.
- MatrixView & operator+= (const double &scal)

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

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

  MatrixView-scalar compound subtraction.
- MatrixView & operator/= (const double &scal)

  MatrixView-scalar compound division.
- void rowExpand (const Index &ind, MatrixView &out) const

  Expand rows according to the provided index.
- void rowSums (vector< double > &sums) const

  Row sums.
- void rowSums (const vector< vector< size_t > > &missInd, vector< double > &sums) const

  Row sums with missing data.
- void rowSums (const Index &ind, MatrixView &out) const

  Sum rows in groups.
- void rowSums (const Index &ind, const vector< vector< size_t > > &missInd, MatrixView &out) const

  Sum rows in groups with missing values.
- void rowMeans (vector< double > &means) const

  Row means.
- void rowMeans (const vector< vector< size_t > > &missInd, vector< double > &means) const

  Row means with missing data.
- void rowMeans (const Index &ind, MatrixView &out) const

  Row means in groups.
- void rowMeans (const Index &ind, const vector< vector< size_t > > &missInd, MatrixView &out) const

  Row means in groups with missing data.
- void colSums (vector< double > &sums) const

  Column sums.
- void colSums (const vector< vector< size_t > > &missInd, vector< double > &sums) const

Column sums with missing data.

- void colSums (const Index &ind, MatrixView &out) const

  Sum columns in groups.

- void colSums (const Index &ind, const vector< vector< size_t > > &missInd, MatrixView &out) const

  Sum columns in groups with missing data.

- void colExpand (const Index &ind, MatrixView &out) const

  Expand columns accoring to the provided index.

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

  Column means.

- void colMeans (const vector< vector< size_t > > &missInd, vector< double > &means) const

  Column means with missing data.

- void colMeans (const Index &ind, MatrixView &out) const

  Column means in groups.

- void colMeans (const Index &ind, const vector< vector< size_t > > &missInd, MatrixView &out) const

  Column means in groups with missing data.

- 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.

Friends

- class **MatrixViewConst**

5.7.1 Detailed Description

Matrix view of a `vector`

This matrix class creates points to a portion of a vector, presenting it as a matrix. Matrix operations can then be perfomed, possibly modifying the data pointed to. The idea is similar to GSL's `matrix_view`. 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. Methods that support missing data assume that missing values are coded with `nan("")`.

5.7.2 Constructor & Destructor Documentation

5.7.2.1 MatrixView() [1/2]

```
MatrixView::MatrixView (
            vector< double > * inVec,
            const size_t & idx,
            const size_t & nrow,
            const size_t & ncol )
```

Constructor pointing to a C++ `vector`

Points to a portion of the vector starting from the provided index.

Parameters

in	*inVec*	target vector
in	*idx*	start index
in	*nrow*	number of rows
in	*ncol*	number of columns

5.7.2.2 MatrixView() [2/2]

```
MatrixView::MatrixView (
            MatrixView && inMat )
```

Move constructor.

Parameters

in	*inMat*	object to be moved

5.7.3 Member Function Documentation

5.7.3.1 addToElem()

```
void MatrixView::addToElem (
            const size_t & iRow,
            const size_t & jCol,
            const double & input )
```

Add a scalar to an element.

Parameters

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

5.7.3.2 chol() [1/2]

```
void MatrixView::chol ( )
```

In-place Cholesky decomposition.

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

5.7.3.3 chol() [2/2]

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

Copy Cholesky decomposition.

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

Parameters

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

5.7.3.4 cholInv() [1/2]

```
void MatrixView::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.

5.7.3.5 cholInv() [2/2]

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

Copy Cholesky inverse.

Computes the inverse of a Cholesky decomposition and stores the result in the provided MatrixView 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

5.7.3.6 colAdd() [1/2]

```
void MatrixView::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

5.7.3.7 colAdd() [2/2]

```
void MatrixView::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

5.7.3.8 colDivide() [1/2]

```
void MatrixView::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

5.7.3.9 colDivide() [2/2]

```
void MatrixView::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

5.7.3.10 colExpand()

```
void MatrixView::colExpand (
            const Index & ind,
            MatrixView & out ) const
```

Expand columns accoring to the provided index.

Columns are expanded to make more rows. The output matrix must be of correct size.

Parameters

in	*ind*	Index with groups corresponding to columns
out	*out*	output MatrixView

5.7.3.11 colMeans() [1/4]

```
void MatrixView::colMeans (
            const Index & ind,
            const vector< vector< size_t > > & missInd,
            MatrixView & out ) const
```

Column means in groups with missing data.

Means among column elements are calculated within each group of the Index. Missing data are ignored. The output matrix must have the correct dimensions. The positions where the data are missing are identified by a vector of index vectors. The size of the outer vector equals the number of columns. The inner vector has row IDs of the missing data. Some of these inner vectors may be empty.

Parameters

in	*ind*	Index with elements corresponding to columns
in	*missInd*	vector of vectors of missing data indexes
out	*out*	output MatrixView

5.7.3.12 colMeans() [2/4]

```
void MatrixView::colMeans (
            const Index & ind,
            MatrixView & out ) const
```

Column means in groups.

Means among column elements are calculated within each group of the Index. The output matrix must have the correct dimensions.

Parameters

in	*ind*	Index with elements corresponding to columns
out	*out*	output MatrixView

5.7.3.13 colMeans() [3/4]

```
void MatrixView::colMeans (
            const vector< vector< size_t > > & missInd,
            vector< double > & means ) const
```

Column means with missing data.

Calculates means among rows in each column and stores them in the provided vector. Missing data are ignored. If vector length is smaller than necessary, the vector is expanded. Otherwise, the first N_{row} elements are used. The positions where the data are missing are identified by a vector of index vectors. The size of the outer vector equals the number of columns. The inner vector has row IDs of the missing data. Some of these inner vectors may be empty.

Parameters

in	*missInd*	vector of vectors of missing data indexes
out	*means*	vector of means

5.7.3.14 colMeans() [4/4]

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

Column means.

Calculates means among rows in each column 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

5.7.3.15 colMultiply() [1/2]

```
void MatrixView::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

5.7.3.16 colMultiply() [2/2]

```
void MatrixView::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

5.7.3.17 colSub() [1/2]

```
void MatrixView::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

5.7.3.18 colSub() [2/2]

```
void MatrixView::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

5.7.3.19 colSums() [1/4]

```
void MatrixView::colSums (
            const Index & ind,
            const vector< vector< size_t > > & missInd,
            MatrixView & out ) const
```

Sum columns in groups with missing data.

The column elements are summed within each group of the Index. Missing data are ignored. The output matrix must have the correct dimensions. The positions where the data are missing are identified by a vector of index vectors. The size of the outer vector equals the number of columns. The inner vector has row IDs of the missing data. Some of these inner vectors may be empty.

Parameters

in	*ind*	Index with elements corresponding to columns
in	*missInd*	vector of vectors of missing data indexes
out	*out*	output MatrixView

5.7.3.20 colSums() [2/4]

```
void MatrixView::colSums (
            const Index & ind,
            MatrixView & out ) const
```

Sum columns in groups.

The column elements are summed within each group of the Index. The output matrix must have the correct dimensions.

Parameters

in	*ind*	Index with elements corresponding to columns
out	*out*	output MatrixView

5.7.3.21 colSums() [3/4]

```
void MatrixView::colSums (
            const vector< vector< size_t > > & missInd,
            vector< double > & sums ) const
```

Column sums with missing data.

Calculates sums of column elements and stores them in the provided vector. Missing data are ignored. If vector length is smaller than necessary, the vector is expanded. Otherwise, the first N_{col} elements are used. The positions where the data are missing are identified by a vector of index vectors. The size of the outer vector equals the number of columns. The inner vector has row IDs of the missing data. Some of these inner vectors may be empty.

Parameters

in	*missInd*	vector of vectors of missing data indexes
out	*sums*	vector of sums

5.7.3.22 colSums() [4/4]

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

Column sums.

Calculates sums of column elements 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

5.7.3.23 divideElem()

```
void MatrixView::divideElem (
            const size_t & iRow,
            const size_t & jCol,
            const double & input )
```

Divide an element by a scalar.

Parameters

in	*iRow*	row number
in	*jCol*	column number
in	*input*	value to divide by

5.7.3.24 eigen() [1/2]

```
void MatrixView::eigen (
            const char & tri,
            const size_t & n,
            MatrixView & 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.

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

5.7.3.25 eigen() [2/2]

```
void MatrixView::eigen (
            const char & tri,
            MatrixView & 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.

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

5.7.3.26 eigenSafe() [1/2]

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

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().

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

5.7.3.27 eigenSafe() [2/2]

```
void MatrixView::eigenSafe (
```

```
            const char & tri,
            MatrixView & U,
            vector< double > & lam ) const
```

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().

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

5.7.3.28 gemc()

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

Multiply a general matrix by a column of another matrix.

Multiply the *MatrixView* 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 resized, 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

5.7.3.29 gemm() [1/2]

```
void MatrixView::gemm (
            const bool & transA,
            const double & alpha,
            const MatrixView & A,
            const bool & transB,
            const double & beta,
            MatrixView & 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 method is called from B.

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

5.7.3.30 gemm() [2/2]

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

General matrix multiplication with MatrixViewConst

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 method is called from B.

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

5.7.3.31 getElem()

```
double MatrixView::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

5.7.3.32 getNcols()

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

Access to number of columns.

Returns

size_t number of columns

5.7.3.33 getNrows()

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

Access to number of rows.

Returns

size_t number of rows

5.7.3.34 multiplyElem()

```
void MatrixView::multiplyElem (
            const size_t & iRow,
            const size_t & jCol,
            const double & input )
```

Multiply an element by a scalar.

Parameters

in	*iRow*	row number
in	*jCol*	column number
in	*input*	value to multiply by

5.7.3.35 operator*=()

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

MatrixView-scalar compound product.

Parameters

in	*scal*	scalar

Returns

MatrixView result

5.7.3.36 operator+=()

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

MatrixView-scalar compound addition.

Parameters

in	*scal*	scalar

Returns

MatrixView result

5.7.3.37 operator-=()

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

MatrixView-scalar compound subtraction.

Parameters

in	*scal*	scalar

Returns

MatrixView result

5.7.3.38 operator/=()

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

MatrixView-scalar compound division.

Parameters

in	*scal*	scalar

Returns

MatrixView result

5.7.3.39 operator=()

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

Move assignment operator.

Parameters

in	*inMat*	object to be moved

Returns

MatrixView target object

5.7.3.40 permuteCols()

```
void MatrixView::permuteCols (
            const vector< size_t > & idx )
```

Permute columns.

Re-order columns of the matrix according to the provided index vector.

Parameters

in	*idx*	index vector

5.7.3.41 permuteRows()

```
void MatrixView::permuteRows (
            const vector< size_t > & idx )
```

Permute rows.

Re-order rows of the matrix according to the provided index vector.

Parameters

in	*idx*	index vector

5.7.3.42 pseudoInv() [1/4]

```
void MatrixView::pseudoInv ( )
```

In-place pseudoinverse.

Computes a pseudoinverse of a symmetric square matrix using eigendecomposition (using the LAPACK *DSYEVR* routine). The matrix is replaced with its inverse. Only the lower triangle of the input matrix is addressed.

5.7.3.43 pseudoInv() [2/4]

```
void MatrixView::pseudoInv (
            double & lDet )
```

In-place pseudoinverse with log-determinant.

Computes a pseudoinverse and its log-pseudodeterminant of a symmetric square matrix using eigendecomposition (using the LAPACK *DSYEVR* routine). The matrix is replaced with its inverse. Only the lower triangle of the input matrix is addressed.

Parameters

out	*lDet*	log-pseudodeterminant of the inverted matrix

5.7.3.44 pseudoInv() [3/4]

```
void MatrixView::pseudoInv (
            MatrixView & out ) const
```

Copy pseudoinverse.

Computes a pseudoinverse of a symmetric square matrix using eigendecomposition (using the LAPACK *DSYEVR* routine). The calling matrix is left intact and the result is copied to the output. Only the lower triangle of the input matrix is addressed.

Parameters

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

5.7.3.45 pseudoInv() [4/4]

```
void MatrixView::pseudoInv (
            MatrixView & out,
            double & lDet ) const
```

Copy pseudoinverse with log-determinant.

Computes a pseudoinverse and its log-pseudodeterminant of a symmetric square matrix using eigendecomposition (using the LAPACK *DSYEVR* routine). The calling matrix is left intact and the result is copied to the output. Only the lower triangle of the input matrix is addressed.

Parameters

out	*out*	object where the result is to be stored
out	*lDet*	log-pseudodeterminant of the inverted matrix

5.7.3.46 rowAdd() [1/2]

```
void MatrixView::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

5.7.3.47 rowAdd() [2/2]

```
void MatrixView::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

5.7.3.48 rowDivide() [1/2]

```
void MatrixView::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

5.7.3.49 rowDivide() [2/2]

```
void MatrixView::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

5.7.3.50 rowExpand()

```
void MatrixView::rowExpand (
            const Index & ind,
            MatrixView & out ) const
```

Expand rows according to the provided index.

Each row is expanded, creating more columns. The output matrix must be of the correct size.

Parameters

in	*ind*	Index with groups corresponding to existing rows
out	*out*	output MatrixView

5.7.3.51 rowMeans() [1/4]

```
void MatrixView::rowMeans (
            const Index & ind,
            const vector< vector< size_t > > & missInd,
            MatrixView & out ) const
```

Row means in groups with missing data.

Means among row elements are calculated within each group of the Index. Missing data are ignored. The output matrix must have the correct dimensions (N_{col} the new matrix equal to the number of groups). The positions where the data are missing are identified by a vector of index vectors. The size of the outer vector equals the number of columns. The inner vector has row IDs of the missing data. Some of these inner vectors may be empty.

Parameters

in	*ind*	Index with elements corresponding to rows
in	*missInd*	vector of vectors of missing data indexes
out	*out*	output MatrixView

5.7.3.52 rowMeans() [2/4]

```
void MatrixView::rowMeans (
            const Index & ind,
            MatrixView & out ) const
```

Row means in groups.

Means among row elements are calculated within each group of the Index. The output matrix must have the correct dimensions (N_{col} the new matrix equal to the number of groups).

Parameters

in	*ind*	Index with elements corresponding to rows
out	*out*	output MatrixView

5.7.3.53 rowMeans() [3/4]

```
void MatrixView::rowMeans (
            const vector< vector< size_t > > & missInd,
            vector< double > & means ) const
```

Row means with missing data.

Calculates means among row elements and stores them in the provided vector. Missing data a ignored. If vector length is smaller than necessary, the vector is expanded. Otherwise, the first N_{row} elements are used. The positions where the data are missing are identified by a vector of index vectors. The size of the outer vector equals the number of columns. The inner vector has row IDs of the missing data. Some of these inner vectors may be empty.

Parameters

in	*missInd*	vector of vectors of missing data indexes
out	*means*	vector of means

5.7.3.54 rowMeans() [4/4]

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

Row means.

Calculates means among row elements 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

5.7.3.55 rowMultiply() [1/2]

```
void MatrixView::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

5.7.3.56 rowMultiply() [2/2]

```
void MatrixView::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

5.7.3.57 rowSub() [1/2]

```
void MatrixView::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

5.7.3.58 rowSub() [2/2]

```
void MatrixView::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

5.7.3.59 rowSums() [1/4]

```
void MatrixView::rowSums (
            const Index & ind,
            const vector< vector< size_t > > & missInd,
            MatrixView & out ) const
```

Sum rows in groups with missing values.

The row elements are summed within each group of the Index. Missing values are ignored. The output matrix must have the correct dimensions (N_{col} the new matrix equal to the number of groups). The positions where the data are missing are identified by a vector of index vectors. The size of the outer vector equals the number of columns. The inner vector has row IDs of the missing data. Some of these inner vectors may be empty.

Parameters

in	*ind*	Index with elements corresponding to rows
in	*missInd*	vector of vectors of missing data indexes
out	*out*	output MatrixView

5.7.3.60 rowSums() [2/4]

```
void MatrixView::rowSums (
            const Index & ind,
            MatrixView & out ) const
```

Sum rows in groups.

The row elements are summed within each group of the Index. The output matrix must have the correct dimensions (N_{col} the new matrix equal to the number of groups).

Parameters

in	*ind*	Index with elements corresponding to rows
out	*out*	output MatrixView

5.7.3.61 rowSums() [3/4]

```
void MatrixView::rowSums (
            const vector< vector< size_t > > & missInd,
            vector< double > & sums ) const
```

Row sums with missing data.

Sums row elements and stores them in the provided vector. Missing values are ignored. If vector length is smaller than necessary, the vector is expanded. Otherwise, the first N_{row} elements are used. The positions where the data are missing are identified by a vector of index vectors. The size of the outer vector equals the number of columns. The inner vector has row IDs of the missing data. Some of these inner vectors may be empty.

Parameters

in	*missInd*	vector of vectors of missing data indexes
out	*sums*	vector of sums

5.7.3.62 rowSums() [4/4]

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

Row sums.

Sums row elements 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

5.7.3.63 setCol()

```
void MatrixView::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

5.7.3.64 setElem()

```
void MatrixView::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

5.7.3.65 subtractFromElem()

```
void MatrixView::subtractFromElem (
            const size_t & iRow,
            const size_t & jCol,
            const double & input )
```

Subtract a scalar from an element.

Parameters

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

5.7.3.66 svd()

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

Perform SVD.

Performs SVD and stores the U vectors in a MatrixView 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

5.7.3.67 svdSafe()

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

Perform "safe" SVD.

Performs SVD and stores the U vectors in a MatrixView 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

5.7.3.68 symc() [1/2]

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

Multiply symmetric matrix by a column of another matrix

Multiply the *MatrixView* object, which is symmetric, by a specified column of a *MatrixView*. 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 resized, 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

5.7.3.69 symc() [2/2]

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

Multiply symmetric matrix by a column of another matrix

Multiply the *MatrixView* object, which is symmetric, by a specified column of a *MatrixViewConst*. 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 resized, 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

5.7.3.70 symm() [1/2]

```
void MatrixView::symm (
            const char & tri,
```

```
            const char & side,
            const double & alpha,
            const MatrixView & symA,
            const double & beta,
            MatrixView & C ) const
```

Multiply by symmetric matrix.

Multiply the *MatrixView* 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 method is called from the B matrix.

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

5.7.3.71 symm() [2/2]

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

Multiply by symmetric MatrixViewConst

Multiply the *MatrixView* 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 method is called from the B matrix.

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

5.7.3.72 syrk()

```
void MatrixView::syrk (
            const char & tri,
            const double & alpha,
            const double & beta,
            MatrixView & 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). Only the specified triangle of *C* 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

5.7.3.73 trm() [1/2]

```
void MatrixView::trm (
            const char & tri,
            const char & side,
            const bool & transA,
            const bool & uDiag,
            const double & alpha,
            const MatrixView & trA )
```

Multiply by triangular matrix.

Multiply the *MatrixView* object by a triangular matrix A. The interface for the BLAS *DTRMM* routine. Updates current object B

$$B \leftarrow \alpha op(A)B$$

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

$$B \leftarrow \alpha Bop(A)$$

if *side* is 'r' (right). $op(A)$ is A^T or A if *transA* is true or false, respectively. The triangular A matrix is provided as input, the method is called from the B matrix. The current object is replaced by the transformed resulting matrix.

Parameters

in	*tri*	A triangle ID ('u' for upper or 'l' for lower)
in	*side*	multiplication side
in	*transA*	whether matrix A should be transposed
in	*uDiag*	whether A unit-diagonal or not
in	*alpha*	the α constant
in	*trA*	triangular matrix A

5.7.3.74 trm() [2/2]

```
void MatrixView::trm (
            const char & tri,
            const char & side,
            const bool & transA,
            const bool & uDiag,
            const double & alpha,
            const MatrixViewConst & trA )
```

Multiply by triangular *MatrixViewConst*

Multiply the *MatrixView* object by a triangular matrix A. The interface for the BLAS *DTRMM* routine. Updates current object B

$$B \leftarrow \alpha op(A)B$$

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

$$B \leftarrow \alpha Bop(A)$$

if *side* is 'r' (right). $op(A)$ is A^T or A if *transA* is true or false, respectively. The triangular A matrix is provided as input, the method is called from the B matrix. The current object is replaced by the transformed resulting matrix.

Parameters

in	*tri*	A triangle ID ('u' for upper or 'l' for lower)
in	*side*	multiplication side
in	*transA*	whether matrix A should be transposed
in	*uDiag*	whether A unit-diagonal or not
in	*alpha*	the α constant
in	*trA*	triangular matrix A

5.7.3.75 tsyrk()

```
void MatrixView::tsyrk (
            const char & tri,
            const double & alpha,
            const double & beta,
            MatrixView & 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). Only the specified triangle of *C* 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

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

- src/matrixView.hpp
- src/matrixView.cpp

5.8 BayesicSpace::MatrixViewConst Class Reference

A `const` version of `MatrixView`

`#include <matrixView.hpp>`

Public Member Functions

- MatrixViewConst ()

  Default constructor.
- MatrixViewConst (const vector< double > *inVec, const size_t &idx, const size_t &nrow, const size_t &ncol)

  Constructor pointing to a C++ `vector`
- ~MatrixViewConst ()

  Destructor.
- MatrixViewConst (const MatrixViewConst &inMat)=delete

  Copy constructor (deleted)
- MatrixViewConst & operator= (const MatrixViewConst &inMat)=delete

  Copy assignment operator (deleted)
- MatrixViewConst (MatrixViewConst &&inMat)

  Move constructor.
- MatrixViewConst (MatrixView &&inMat)

  Move constructor from `MatrixView`
- MatrixViewConst & operator= (MatrixViewConst &&inMat)

  Move assignment operator.
- MatrixViewConst & operator= (MatrixView &&inMat)

  Move assignment operator from `MatrixView`
- 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 chol (MatrixView &out) const

  Copy Cholesky decomposition.
- void cholInv (MatrixView &out) const

  Copy Cholesky inverse.
- void pseudoInv (MatrixView &out) const

  Copy pseudoinverse.
- void pseudoInv (MatrixView &out, double &lDet) const

  Copy pseudoinverse with log-determinant.
- void svdSafe (MatrixView &U, vector< double > &s) const

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

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

  Some eigenvalues and vectors of a symmetric matrix ("safe")
- void syrk (const char &tri, const double &alpha, const double &beta, MatrixView &C) const

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

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

  Multiply by symmetric matrix.

- void symm (const char &tri, const char &side, const double &alpha, const MatrixViewConst &symA, const double &beta, MatrixView &C) const

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

  General matrix multiplication.
- void gemm (const bool &transA, const double &alpha, const MatrixViewConst &A, const bool &transB, const double &beta, MatrixView &C) const

  General matrix multiplication with `MatrixViewConst`
- void gemc (const bool &trans, const double &alpha, const MatrixView &X, const size_t &xCol, const double &beta, vector< double > &y) const

  Multiply a general matrix by a column of another matrix.
- void rowExpand (const Index &ind, MatrixView &out) const

  Expand rows according to the provided index.
- void rowSums (const Index &ind, MatrixView &out) const

  Sum rows in groups.
- void rowSums (const Index &ind, const vector< vector< size_t > > &missInd, MatrixView &out) const

  Sum rows in groups with missing data.
- void rowSums (vector< double > &sums) const

  Row sums.
- void rowSums (const vector< vector< size_t > > &missInd, vector< double > &sums) const

  Row sums with missing data.
- void rowMeans (vector< double > &means) const

  Row means.
- void rowMeans (const vector< vector< size_t > > &missInd, vector< double > &means) const

  Row means with missing data.
- void rowMeans (const Index &ind, MatrixView &out) const

  Row means in groups.
- void rowMeans (const Index &ind, const vector< vector< size_t > > &missInd, MatrixView &out) const

  Row means in groups with missing data.
- void colSums (vector< double > &sums) const

  Column sums.
- void colSums (const vector< vector< size_t > > &missInd, vector< double > &sums) const

  Column sums with missing data.
- void colSums (const Index &ind, MatrixView &out) const

  Sum columns in groups.
- void colSums (const Index &ind, const vector< vector< size_t > > &missInd, MatrixView &out) const

  Sum columns in groups with missing data.
- void colExpand (const Index &ind, MatrixView &out) const

  Expand columns accoring to the provided index.
- void colMeans (vector< double > &means) const

  Column means.
- void colMeans (const vector< vector< size_t > > &missInd, vector< double > &means) const

  Column means with missing data.
- void colMeans (const Index &ind, MatrixView &out) const

  Column means in groups.
- void colMeans (const Index &ind, const vector< vector< size_t > > &missInd, MatrixView &out) const

  Column means in groups with missing data.

Friends

- class **MatrixView**

5.8.1 Detailed Description

A `const` version of `MatrixView`

This matrix class creates points to a portion of a vector, presenting it as a matrix. Matrix operations can then be perfomed, but are aguaranteed not to modify the vector pointed to. The idea is similar to GSL's `matrix_view`. 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.

5.8.2 Constructor & Destructor Documentation

5.8.2.1 MatrixViewConst() [1/3]

```
BayesicSpace::MatrixViewConst::MatrixViewConst (
            const vector< double > * inVec,
            const size_t & idx,
            const size_t & nrow,
            const size_t & ncol )  [inline]
```

Constructor pointing to a C++ `vector`

Points to a portion of the vector starting from the provided index.

Parameters

in	*inVec*	target vector
in	*idx*	start index
in	*nrow*	number of rows
in	*ncol*	number of columns

5.8.2.2 MatrixViewConst() [2/3]

```
MatrixViewConst::MatrixViewConst (
            MatrixViewConst && inMat )
```

Move constructor.

Parameters

in	*inMat*	object to be moved

5.8.2.3 MatrixViewConst() [3/3]

```
MatrixViewConst::MatrixViewConst (
            MatrixView && inMat )
```

Move constructor from MatrixView

Parameters

in	*inMat*	object to be moved

5.8.3 Member Function Documentation

5.8.3.1 chol()

```
void MatrixViewConst::chol (
            MatrixView & out ) const
```

Copy Cholesky decomposition.

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

Parameters

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

5.8.3.2 cholInv()

```
void MatrixViewConst::cholInv (
            MatrixView & out ) const
```

Copy Cholesky inverse.

Computes the inverse of a Cholesky decomposition and stores the result in the provided MatrixView 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

5.8.3.3 colExpand()

```
void MatrixViewConst::colExpand (
            const Index & ind,
            MatrixView & out ) const
```

Expand columns accoring to the provided index.

Columns are expanded to make more rows. The output matrix must be of correct size.

Parameters

in	*ind*	Index with groups corresponding to columns
out	*out*	output MatrixView

5.8.3.4 colMeans() [1/4]

```
void MatrixViewConst::colMeans (
            const Index & ind,
            const vector< vector< size_t > > & missInd,
            MatrixView & out ) const
```

Column means in groups with missing data.

Means among column elements are calculated within each group of the Index. Missing data are ignored. The output matrix must have the correct dimensions. The positions where the data are missing are identified by a vector of index vectors. The size of the outer vector equals the number of columns. The inner vector has row IDs of the missing data. Some of these inner vectors may be empty.

Parameters

in	*ind*	Index with elements corresponding to columns
in	*missInd*	vector of vectors of missing data indexes
out	*out*	output MatrixView

5.8.3.5 colMeans() [2/4]

```
void MatrixViewConst::colMeans (
            const Index & ind,
            MatrixView & out ) const
```

Column means in groups.

Means among column elements are calculated within each group of the Index. The output matrix must have the correct dimensions.

Parameters

in	*ind*	Index with elements corresponding to columns
out	*out*	output MatrixView

5.8.3.6 colMeans() [3/4]

```
void MatrixViewConst::colMeans (
            const vector< vector< size_t > > & missInd,
            vector< double > & means ) const
```

Column means with missing data.

Calculates means among rows in each column and stores them in the provided vector. Missing data are ignored. If vector length is smaller than necessary, the vector is expanded. Otherwise, the first N_{col} elements are used. The positions where the data are missing are identified by a vector of index vectors. The size of the outer vector equals the number of columns. The inner vector has row IDs of the missing data. Some of these inner vectors may be empty.

Parameters

in	*missInd*	vector of vectors of missing data indexes
out	*means*	vector of means

5.8.3.7 colMeans() [4/4]

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

Column means.

Calculates means among rows in each column 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

5.8.3.8 colSums() [1/4]

```
void MatrixViewConst::colSums (
            const Index & ind,
            const vector< vector< size_t > > & missInd,
            MatrixView & out ) const
```

Sum columns in groups with missing data.

The column elements are summed within each group of the Index. Missing data are ignored. The output matrix must have the correct dimensions. The positions where the data are missing are identified by a vector of index vectors. The size of the outer vector equals the number of columns. The inner vector has row IDs of the missing data. Some of these inner vectors may be empty.

Parameters

in	*ind*	Index with elements corresponding to columns
in	*missInd*	vector of vectors of missing data indexes
out	*out*	output MatrixView

5.8.3.9 colSums() [2/4]

```
void MatrixViewConst::colSums (
            const Index & ind,
            MatrixView & out ) const
```

Sum columns in groups.

The column elements are summed within each group of the Index. The output matrix must have the correct dimensions.

Parameters

in	*ind*	Index with elements corresponding to columns
out	*out*	output MatrixView

5.8.3.10 colSums() [3/4]

```
void MatrixViewConst::colSums (
            const vector< vector< size_t > > & missInd,
            vector< double > & sums ) const
```

Column sums with missing data.

Calculates sums of column elements and stores them in the provided vector. Missing data are ignored. If vector length is smaller than necessary, the vector is expanded. Otherwise, the first N_{col} elements are used. The positions where the data are missing are identified by a vector of index vectors. The size of the outer vector equals the number of columns. The inner vector has row IDs of the missing data. Some of these inner vectors may be empty.

Parameters

in	*missInd*	vector of vectors of missing data indexes
out	*sums*	vector of sums

5.8.3.11 colSums() [4/4]

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

Column sums.

Calculates sums of column elements 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

5.8.3.12 eigenSafe() [1/2]

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

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().

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

5.8.3.13 eigenSafe() [2/2]

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

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().

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

5.8.3.14 gemc()

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

Multiply a general matrix by a column of another matrix.

Multiply the MatrixViewConst 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 resized, 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

5.8.3.15 gemm() [1/2]

```
void MatrixViewConst::gemm (
            const bool & transA,
            const double & alpha,
            const MatrixView & A,
            const bool & transB,
            const double & beta,
            MatrixView & 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 method is called from B.

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

5.8.3.16 gemm() [2/2]

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

General matrix multiplication with MatrixViewConst

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 method is called from B.

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

5.8.3.17 getElem()

```
double MatrixViewConst::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

5.8.3.18 getNcols()

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

Access to number of columns.

Returns

size_t number of columns

5.8.3.19 getNrows()

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

Access to number of rows.

Returns

size_t number of rows

5.8.3.20 operator=() [1/2]

```
MatrixViewConst & MatrixViewConst::operator= (
            MatrixView && inMat )
```

Move assignment operator from MatrixView

Parameters

in	*inMat*	object to be moved

Returns

MatrixViewConst target object

5.8.3.21 operator=() [2/2]

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

Move assignment operator.

Parameters

in	*inMat*	object to be moved

Returns

MatrixViewConst target object

5.8.3.22 pseudoInv() [1/2]

```
void MatrixViewConst::pseudoInv (
            MatrixView & out ) const
```

Copy pseudoinverse.

Computes a pseudoinverse of a symmetric square matrix using eigendecomposition (using the LAPACK *DSYEVR* routine). The calling matrix is left intact and the result is copied to the output. Only the lower triangle of the input matrix is addressed.

Parameters

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

5.8.3.23 pseudoInv() [2/2]

```
void MatrixViewConst::pseudoInv (
            MatrixView & out,
            double & lDet ) const
```

Copy pseudoinverse with log-determinant.

Computes a pseudoinverse and its log-pseudodeterminant of a symmetric square matrix using eigendecomposition (using the LAPACK *DSYEVR* routine). The calling matrix is left intact and the result is copied to the output. Only the lower triangle of the input matrix is addressed.

Parameters

out	*out*	object where the result is to be stored
out	*lDet*	log-pseudodeterminant of the inverted matrix

5.8.3.24 rowExpand()

```
void MatrixViewConst::rowExpand (
            const Index & ind,
            MatrixView & out ) const
```

Expand rows according to the provided index.

Each row is expanded, creating more columns. The output matrix must be of the correct size.

Parameters

in	*ind*	Index with groups corresponding to existing rows
out	*out*	output MatrixView

5.8.3.25 rowMeans() [1/4]

```
void MatrixViewConst::rowMeans (
            const Index & ind,
            const vector< vector< size_t > > & missInd,
            MatrixView & out ) const
```

Row means in groups with missing data.

Means among row elements are calculated within each group of the Index. Missing data are ignored. The output matrix must have the correct dimensions. The positions where the data are missing are identified by a vector of index vectors. The size of the outer vector equals the number of columns. The inner vector has row IDs of the missing data. Some of these inner vectors may be empty.

Parameters

in	*ind*	Index with elements corresponding to rows
in	*missInd*	vector of vectors of missing data indexes
out	*out*	output MatrixView

5.8.3.26 rowMeans() [2/4]

```
void MatrixViewConst::rowMeans (
            const Index & ind,
            MatrixView & out ) const
```

Row means in groups.

Means among row elements are calculated within each group of the Index. The output matrix must have the correct dimensions.

Parameters

in	*ind*	Index with elements corresponding to rows
out	*out*	output MatrixView

5.8.3.27 rowMeans() [3/4]

```
void MatrixViewConst::rowMeans (
            const vector< vector< size_t > > & missInd,
            vector< double > & means ) const
```

Row means with missing data.

Calculates means among row elements and stores them in the provided vector. Missing data are ignored. If vector length is smaller than necessary, the vector is expanded. Otherwise, the first N_{row} elements are used. The positions where the data are missing are identified by a vector of index vectors. The size of the outer vector equals the number of columns. The inner vector has row IDs of the missing data. Some of these inner vectors may be empty.

Parameters

in	*missInd*	vector of vectors of missing data indexes
out	*means*	vector of means

5.8.3.28 rowMeans() [4/4]

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

Row means.

Calculates means among row elements 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

5.8.3.29 rowSums() [1/4]

```
void MatrixViewConst::rowSums (
            const Index & ind,
            const vector< vector< size_t > > & missInd,
            MatrixView & out ) const
```

Sum rows in groups with missing data.

The row elements are summed within each group of the Index. Missing data are ignored. The output matrix must have the correct dimensions (N_{col} the new matrix equal to the number of groups). The positions where the data are missing are identified by a vector of index vectors. The size of the outer vector equals the number of columns. The inner vector has row IDs of the missing data. Some of these inner vectors may be empty.

Parameters

in	*ind*	Index with elements corresponding to rows
in	*missInd*	vector of vectors of missing data indexes
out	*out*	output MatrixView

5.8.3.30 rowSums() [2/4]

```
void MatrixViewConst::rowSums (
            const Index & ind,
            MatrixView & out ) const
```

Sum rows in groups.

The row elements are summed within each group of the Index. The output matrix must have the correct dimensions (N_{col} the new matrix equal to the number of groups).

Parameters

in	*ind*	Index with elements corresponding to rows
out	*out*	output MatrixView

5.8.3.31 rowSums() [3/4]

```
void MatrixViewConst::rowSums (
            const vector< vector< size_t > > & missInd,
            vector< double > & sums ) const
```

Row sums with missing data.

Sums row elements and stores them in the provided vector. Missing data are ignored. If vector length is smaller than necessary, the vector is expanded. Otherwise, the first N_{row} elements are used. The positions where the data are missing are identified by a vector of index vectors. The size of the outer vector equals the number of columns. The inner vector has row IDs of the missing data. Some of these inner vectors may be empty.

Parameters

in	*missInd*	vector of vectors of missing data indexes
out	*sums*	vector of sums

5.8.3.32 rowSums() [4/4]

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

Row sums.

Sums row elements 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

5.8.3.33 svdSafe()

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

Perform "safe" SVD.

Performs SVD and stores the U vectors in a MatrixView 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

5.8.3.34 symc()

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

Multiply symmetric matrix by a column of another matrix

Multiply the MatrixViewConst 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 resized, 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

5.8.3.35 symm() [1/2]

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

Multiply by symmetric matrix.

Multiply the MatrixViewConst 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 method is called from the B matrix.

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

5.8.3.36 symm() [2/2]

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

Multiply by symmetric MatrixViewConst

Multiply the MatrixViewConst 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 method is called from the B matrix.

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

5.8.3.37 syrk()

```
void MatrixViewConst::syrk (
            const char & tri,
            const double & alpha,
            const double & beta,
            MatrixView & 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). 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

5.8.3.38 tsyrk()

```
void MatrixViewConst::tsyrk (
            const char & tri,
            const double & alpha,
            const double & beta,
            MatrixView & 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). 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

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

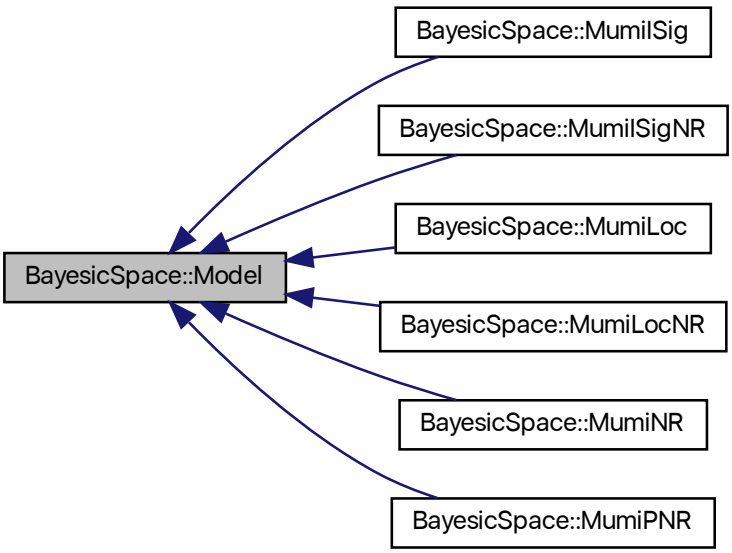
- src/matrixView.hpp
- src/matrixView.cpp

5.9 BayesicSpace::Model Class Reference

Model class.

`#include <model.hpp>`

Inheritance diagram for BayesicSpace::Model:



Public Member Functions

- virtual ~Model ()

  Destructor.
- virtual double logPost (const vector< double > &theta) const =0

  Virtual log-posterior function.
- virtual void gradient (const vector< double > &theta, vector< double > &grad) const =0

  Virtual gradient of the log-posterior.

Protected Member Functions

- Model ()

  Default constructor.

5.9.1 Detailed Description

Model class.

Abstract class that points to an implementation of a particular model. Must surface a call to a log-posterior function and its gradient. No other public methods are required.

5.9.2 Member Function Documentation

5.9.2.1 gradient()

```
virtual void BayesicSpace::Model::gradient (
            const vector< double > & theta,
            vector< double > & grad ) const  [pure virtual]
```

Virtual gradient of the log-posterior.

Calculates the patial derivative of the log-posterior for each element in the provided parameter vector.

Parameters

in	*theta*	parameter vector
out	*grad*	partial derivative (gradient) vector

Implemented in BayesicSpace::MumiPNR, BayesicSpace::MumiISig, BayesicSpace::MumiISigNR, BayesicSpace::MumiLoc, BayesicSpace::MumiLocNR, and BayesicSpace::MumiNR.

5.9.2.2 logPost()

```
virtual double BayesicSpace::Model::logPost (
            const vector< double > & theta ) const  [pure virtual]
```

Virtual log-posterior function.

Returns the value of the log-posterior.

Parameters

in	*theta*	parameter vector

Returns

Value of the log-posterior

Implemented in BayesicSpace::MumiPNR, BayesicSpace::MumiISig, BayesicSpace::MumiISigNR, BayesicSpace::MumiLoc, BayesicSpace::MumiLocNR, and BayesicSpace::MumiNR.

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

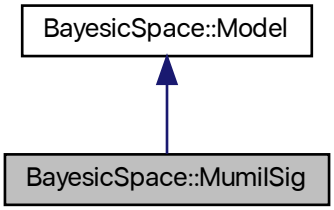
- src/model.hpp

5.10 BayesicSpace::MumiISig Class Reference

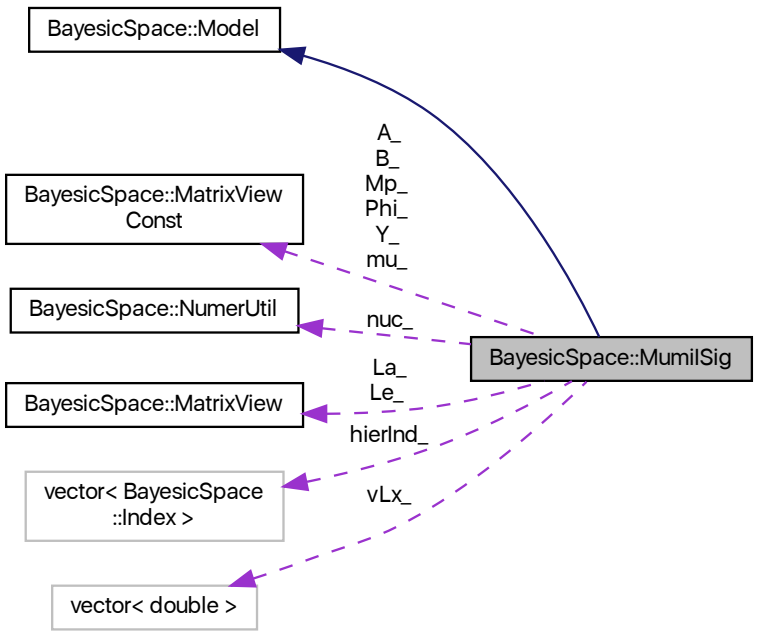
Model for inverse covariances.

```
#include <mumimo.hpp>
```

Inheritance diagram for BayesicSpace::MumiISig:



Collaboration diagram for BayesicSpace::MumiISig:



Public Member Functions

- MumiISig ()

  Default constructor.
- MumiISig (const vector< double > *yVec, const vector< double > *vTheta, const vector< Index > *hierInd, const double &nu0, const double &invAsq, const size_t &nPops)

  Constructor.
- ~MumiISig ()

  Destructor.
- MumiISig (const MumiLoc &in)=delete

  Copy constructor (deleted)
- MumiISig & operator= (const MumiISig &in)=delete

  Copy assignment (deleted)
- MumiISig (MumiISig &&in)

  Move constructor.
- MumiISig & operator= (MumiISig &&in)

  Move assignment operator.
- double logPost (const vector< double > &viSig) const override

  Log-posterior function.
- void gradient (const vector< double > &viSig, vector< double > &grad) const override

  Gradient of the log-posterior.

Protected Member Functions

- void expandISvec_ (const vector< double > &viSig) const

  Expand the vector of factorized precision matrices.

Protected Attributes

- const vector< Index > * hierInd_

  Pointer to vector of indexes connecting hierarchy levels.
- double nu0_

  Prior degrees of freedom.
- double invAsq_

  Prior inverse-variance.
- MatrixViewConst Y_

  Data view.
- MatrixViewConst A_

  Line mean view.
- MatrixViewConst B_

  Covariate effect view.
- MatrixViewConst Mp_

  Population mean view.
- MatrixViewConst mu_

  Overall mean view.
- MatrixViewConst Phi_

  Population assignment logit-probability view.
- MatrixView Le_

  Error factorized precision matrix view.
- MatrixView La_

  Line factorized precision matrix view.
- vector< double > vLx_

  Expanded $\boldsymbol{L}$ matrices.
- size_t fTeInd_

  Index of the first $\boldsymbol{T}_E$ element.
- size_t fLaInd_

  Index of the first $\boldsymbol{L}_A$ element.
- size_t fTaInd_

  Index of the first $\boldsymbol{T}_A$ element.
- size_t fTpInd_

  Index of the first $\boldsymbol{T}_P$ element.
- double nxnd_

  nu0(nu0 + 2d)*
- double Nnd_

  N + nu0 + 2d.
- double NAnd_

  N_A + nu0 + 2d.
- double NPnd_

  N_P + nu0 + 2d.
- NumerUtil nuc_

5.10.1 Detailed Description

Model for inverse covariances.

Implements log-posterior and gradient for inverse covariances. The inverse-covariances are factorized and stored compactly in the vectors provided to the methods of this class. The error matrix is stored first, then the line precision matrix. The unit lower-triangular $\boldsymbol{L}_X$ is stored first (by column and excluding the diagonal), then the diagonal log-precision matrix $\boldsymbol{T}_X$ (see the model description for notation).

5.10.2 Constructor & Destructor Documentation

5.10.2.1 MumiISig() [1/2]

```
MumiISig::MumiISig (
            const vector< double > * yVec,
            const vector< double > * vTheta,
            const vector< Index > * hierInd,
            const double & nu0,
            const double & invAsq,
            const size_t & nPops )
```

Constructor.

Parameters

in	*yVec*	pointer to data
in	*vTheta*	pointer to vector of location parameters
in	*xVec*	pointer to vectorized covariate matrix (with intercept)
in	*hierInd*	pointer to a vector with hierarchical indexes
in	*nu0*	prior degrees of freedom ν_0
in	*invAsq*	prior precision a^{-2}
in	*nPops*	number of populations

5.10.2.2 MumiISig() [2/2]

```
MumiISig::MumiISig (
            MumiISig && in )
```

Move constructor.

Parameters

in	*in*	object to move

5.10.3 Member Function Documentation

5.10.3.1 expandISvec_()

```
void MumiISig::expandISvec_ (
            const vector< double > & viSig ) const  [protected]
```

Expand the vector of factorized precision matrices.

Expands the triangular $\boldsymbol{L}_X$ matrices contained in the provided vector into the internal `L_` vector. The input vector stores only the non-zero elements of these matrices.

Parameters

in	*viSig*	compressed vector of factorized precision matrices

5.10.3.2 gradient()

```
void MumiISig::gradient (
            const vector< double > & viSig,
            vector< double > & grad ) const  [override], [virtual]
```

Gradient of the log-posterior.

Calculates the partial derivative of the log-posterior for each element in the provided parameter vector.

Parameters

in	*viSig*	parameter vector
out	*grad*	partial derivative (gradient) vector

Implements BayesicSpace::Model.

5.10.3.3 logPost()

```
double MumiISig::logPost (
            const vector< double > & viSig ) const  [override], [virtual]
```

Log-posterior function.

Returns the value of the log-posterior given the data provided at construction and the passed-in parameter vector.

Parameters

in	*viSig*	parameter vector

Returns

Value of the log-posterior

Implements BayesicSpace::Model.

5.10.3.4 operator=()

```
MumiISig & MumiISig::operator= (
            MumiISig && in )
```

Move assignment operator.

Parameters

in	*in*	object to be moved

Returns

target object

5.10.4 Member Data Documentation

5.10.4.1 invAsq_

```
double BayesicSpace::MumiISig::invAsq_  [protected]
```

Prior inverse-variance.

Inverse variance of the prior. Should be set to a large value for a vague prior.

5.10.4.2 La_

```
MatrixView BayesicSpace::MumiISig::La_ [mutable], [protected]
```

Line factorized precision matrix view.

Points to $\mathrm{vLx\_}$.

5.10.4.3 Le_

```
MatrixView BayesicSpace::MumiISig::Le_ [mutable], [protected]
```

Error factorized precision matrix view.

Points to $\mathrm{vLx\_}$.

5.10.4.4 nu0_

```
double BayesicSpace::MumiISig::nu0_ [protected]
```

Prior degrees of freedom.

Degrees of freedom of the half- t prior distribution on the covariance matrix. If $\nu_0 = 2$, the prior is half-Cauchy. Should not be large for a vague prior.

5.10.4.5 nuc_

```
NumerUtil BayesicSpace::MumiISig::nuc_ [protected]
```

Numerical utility collection

5.10.4.6 vLx_

```
vector<double> BayesicSpace::MumiISig::vLx_ [mutable], [protected]
```

Expanded $\boldsymbol{L}$ matrices.

Vectorized error and line unity triangular matrices ($\boldsymbol{L}_X$ in the model description).

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

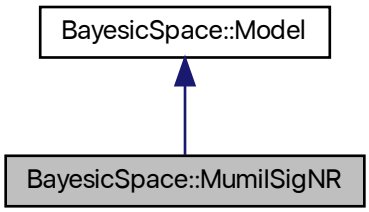
- src/mumimo.hpp
- src/mumimo.cpp

5.11 BayesicSpace::MumilSigNR Class Reference

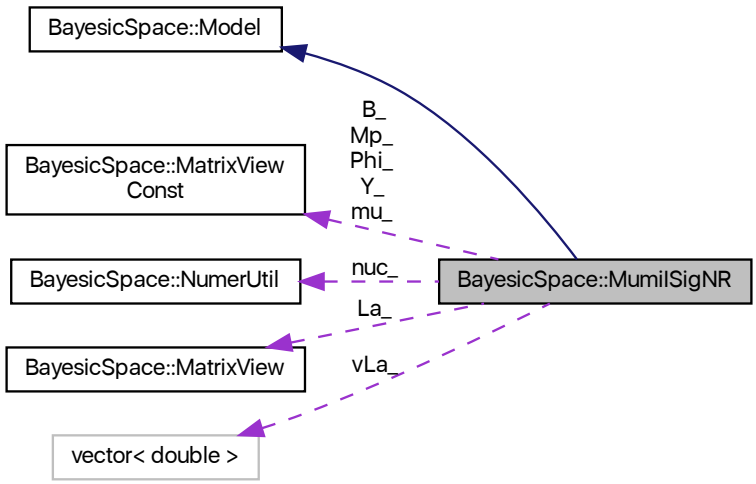
Model for inverse covariances with no replication.

`#include <mumimo.hpp>`

Inheritance diagram for BayesicSpace::MumilSigNR:



Collaboration diagram for BayesicSpace::MumilSigNR:



Public Member Functions

- MumilSigNR ()

Default constructor.

- MumiISigNR (const vector< double > *yVec, const size_t &d, const vector< double > *vTheta, const double &nu0, const double &invAsq, const size_t &nPops)

  Constructor.
- ~MumiISigNR ()

  Destructor.
- MumiISigNR (const MumiLocNR &in)=delete

  Copy constructor (deleted)
- MumiISigNR & operator= (const MumiISigNR &in)=delete

  Copy assignment (deleted)
- MumiISigNR (MumiISigNR &&in)

  Move constructor.
- MumiISigNR & operator= (MumiISigNR &&in)

  Move assignment operator.
- double logPost (const vector< double > &viSig) const override

  Log-posterior function.
- void gradient (const vector< double > &viSig, vector< double > &grad) const override

  Gradient of the log-posterior.

Protected Member Functions

- void expandISvec_ (const vector< double > &viSig) const

  Expand the vector of factorized precision matrices.

Protected Attributes

- double nu0_

  Prior degrees of freedom.
- double invAsq_

  Prior inverse-variance.
- MatrixViewConst Y_

  Data view.
- MatrixViewConst B_

  Covariate effect view.
- MatrixViewConst Mp_

  Population mean view.
- MatrixViewConst mu_

  Overall mean view.
- MatrixViewConst Phi_

  Population assignment logit-probability view.
- MatrixView La_

  Line factorized precision matrix view.
- vector< double > vLa_

  Expanded $\boldsymbol{L}_A$ matrix.
- size_t fTaInd_

  Index of the first $\boldsymbol{T}_A$ element.

- size_t fTpInd_

  Index of the first $\boldsymbol{T}_P$ element.
- double nxnd_

  nu0(nu0 + 2d)*
- double NAnd_

  N_A + nu0 + 2d.
- double NPnd_

  N_P + nu0 + 2d.
- NumerUtil nuc_

5.11.1 Detailed Description

Model for inverse covariances with no replication.

Implements log-posterior and gradient for inverse covariances for the multiplicative mixture model with no replication. The inverse-covariances are factorized and stored compactly in the vectors provided to the methods of this class. The unit lower-triangular $\boldsymbol{L}_A$ is stored first (by column and excluding the diagonal), then the diagonal log-precision matrix $\boldsymbol{T}_A$, then $\boldsymbol{T}_M$ (see the model description for notation).

5.11.2 Constructor & Destructor Documentation

5.11.2.1 MumilSigNR() [1/2]

```
MumiISigNR::MumiISigNR (
            const vector< double > * yVec,
            const size_t & d,
            const vector< double > * vTheta,
            const double & nu0,
            const double & invAsq,
            const size_t & nPops )
```

Constructor.

Parameters

in	*yVec*	pointer to data
in	*d*	number of traits
in	*vTheta*	pointer to vector of location parameters
in	*xVec*	pointer to vectorized covariate matrix (with intercept)
in	*nu0*	prior degrees of freedom ν_0
in	*invAsq*	prior precision a^{-2}
in	*nPops*	number of populations

5.11.2.2 MumiISigNR() [2/2]

```
MumiISigNR::MumiISigNR (
            MumiISigNR && in )
```

Move constructor.

Parameters

in	*in*	object to move

5.11.3 Member Function Documentation

5.11.3.1 expandISvec_()

```
void MumiISigNR::expandISvec_ (
            const vector< double > & viSig ) const  [protected]
```

Expand the vector of factorized precision matrices.

Expands the triangular $\boldsymbol{L}_A$ matrix contained in the provided vector into the internal L_ vector. The input vector stores only the non-zero elements of these matrices.

Parameters

in	*viSig*	compressed vector of factorized precision matrices

5.11.3.2 gradient()

```
void MumiISigNR::gradient (
            const vector< double > & viSig,
            vector< double > & grad ) const  [override], [virtual]
```

Gradient of the log-posterior.

Calculates the partial derivative of the log-posterior for each element in the provided parameter vector.

Parameters

in	*viSig*	parameter vector
out	*grad*	partial derivative (gradient) vector

Implements BayesicSpace::Model.

5.11.3.3 logPost()

```
double MumiISigNR::logPost (
            const vector< double > & viSig ) const  [override], [virtual]
```

Log-posterior function.

Returns the value of the log-posterior given the data provided at construction and the passed-in parameter vector.

Parameters

in	*viSig*	parameter vector

Returns

Value of the log-posterior

Implements BayesicSpace::Model.

5.11.3.4 operator=()

```
MumiISigNR & MumiISigNR::operator= (
            MumiISigNR && in )
```

Move assignment operator.

Parameters

in	*in*	object to be moved

Returns

target object

5.11.4 Member Data Documentation

5.11.4.1 invAsq_

```
double BayesicSpace::MumiISigNR::invAsq_  [protected]
```

Prior inverse-variance.

Inverse variance of the prior. Should be set to a large value for a vague prior.

5.11.4.2 La_

```
MatrixView BayesicSpace::MumiISigNR::La_  [mutable], [protected]
```

Line factorized precision matrix view.

Points to vLa_.

5.11.4.3 nu0_

```
double BayesicSpace::MumiISigNR::nu0_  [protected]
```

Prior degrees of freedom.

Degrees of freedom of the half- t prior distribution on the covariance matrix. If $\nu_0 = 2$, the prior is half-Cauchy. Should not be large for a vague prior.

5.11.4.4 nuc_

```
NumerUtil BayesicSpace::MumiISigNR::nuc_  [protected]
```

Numerical utility collection

5.11.4.5 vLa_

```
vector<double> BayesicSpace::MumiISigNR::vLa_  [mutable], [protected]
```

Expanded $\boldsymbol{L}_A$ matrix.

Vectorized line unity triangular matrix ($\boldsymbol{L}_A$ in the model description).

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

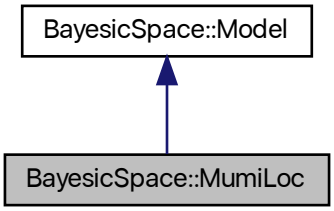
- src/mumimo.hpp
- src/mumimo.cpp

5.12 BayesicSpace::MumiLoc Class Reference

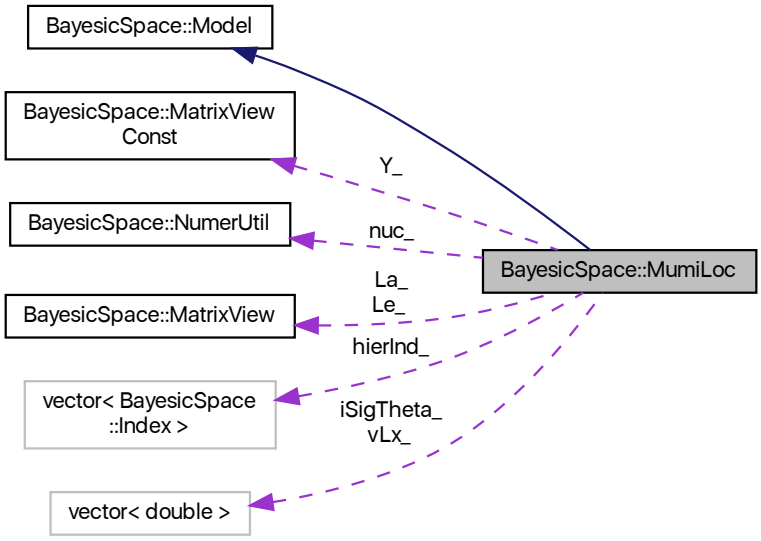
Mixture model for location parameters.

`#include <mumimo.hpp>`

Inheritance diagram for BayesicSpace::MumiLoc:



Collaboration diagram for BayesicSpace::MumiLoc:



Public Member Functions

- MumiLoc ()

  Default constructor.
- MumiLoc (const vector< double > *yVec, const vector< double > *iSigVec, const vector< Index > *hierInd, const double &tau, const size_t &nPops, const double &tauPrPhi, const double &alphaPr)

  Constructor.
- ~MumiLoc ()

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

  Copy constructor (deleted)
- MumiLoc & operator= (const MumiLoc &in)=delete

  Copy assignment (deleted)
- MumiLoc (MumiLoc &&in)

  Move constructor.
- MumiLoc & operator= (MumiLoc &&in)

  Move assignment operator.
- double logPost (const vector< double > &theta) const override

  Log-posterior function.
- void gradient (const vector< double > &theta, vector< double > &grad) const override

  Gradient of the log-posterior.

Protected Member Functions

- void expandISvec_ () const

  Expand the vector of factorized precision matrices.

Protected Attributes

- MatrixViewConst Y_

  Matrix view of data.
- const vector< Index > * hierInd_

  Pointer to vector of indexes connecting hierarchy levels.
- double tau0_

  Fixed prior precision for unmodeled effects.
- const vector< double > * iSigTheta_

  Pointer to a precision parameter vector.
- MatrixView Le_

  Error factorized precision matrix view.
- MatrixView La_

  Line factorized precision matrix view.
- vector< double > vLx_

  Expanded $\boldsymbol{L}$ matrices.
- size_t fTeInd_

  Index of the first $\boldsymbol{T}_E$ element.
- size_t fLaInd_

Index of the first $\boldsymbol{L}_A$ element.

- size_t fTaInd_

  Index of the first $\boldsymbol{T}_A$ element.

- size_t fTpInd_

  Index of the first $\boldsymbol{T}_P$ element.

- size_t PhiBegInd_

  Index of the first probability element.

- size_t Npop_

  Number of populations.

- double tauPrPhi_

  The τ_ϕ prior precision.

- double alphaPr_

  Prior population assignment probability.

- NumerUtil nuc_

Static Protected Attributes

- static const double pSumCutOff_ = 0.003

  Cut-off for p_{jm} approximation.

5.12.1 Detailed Description

Mixture model for location parameters.

Implements log-posterior and gradient for the location parameters of the multiplicative mixture model with replicated observations.

5.12.2 Constructor & Destructor Documentation

5.12.2.1 MumiLoc() [1/2]

```
MumiLoc::MumiLoc (
            const vector< double > * yVec,
            const vector< double > * iSigVec,
            const vector< Index > * hierInd,
            const double & tau,
            const size_t & nPops,
            const double & tauPrPhi,
            const double & alphaPr )
```

Constructor.

Parameters

in	*yVec*	pointer vectorized data matrix
in	*iSigVec*	pointer to vectorized inverse-covariance matrix collection
in	*xVec*	pointer to vectorized covariate predictor matrix
in	*hierInd*	pointer to vector of hierarchical indexes
in	*tau*	fixed prior for the unmodeled ("fixed") effects and overall mean (intercept)
in	*nPops*	number of populations
in	*tauPrPhi*	τ_ϕ population assignment probability prior precision
in	*alphaPr*	prior on α on population assignment probabilities

5.12.2.2 MumiLoc() [2/2]

```
MumiLoc::MumiLoc (
            MumiLoc && in )
```

Move constructor.

Parameters

in	*in*	object to move

5.12.3 Member Function Documentation

5.12.3.1 expandISvec_()

```
void MumiLoc::expandISvec_ ( ) const  [protected]
```

Expand the vector of factorized precision matrices.

Expands the triangular $\boldsymbol{L}_X$ matrices contained in the precision matrix vector into the internal L_ vector. The input vector stores only the non-zero elements of these matrices.

5.12.3.2 gradient()

```
void MumiLoc::gradient (
            const vector< double > & theta,
            vector< double > & grad ) const  [override], [virtual]
```

Gradient of the log-posterior.

Calculates the partial derivative of the log-posterior for each element in the provided parameter vector.

Parameters

in	*theta*	parameter vector
out	*grad*	partial derivative (gradient) vector

Implements BayesicSpace::Model.

5.12.3.3 logPost()

```
double MumiLoc::logPost (
            const vector< double > & theta ) const  [override], [virtual]
```

Log-posterior function.

Returns the value of the log-posterior given the data provided at construction and the passed-in parameter vector. The parameter vector has the covariates, line means, and population means in that order.

Parameters

in	*theta*	parameter vector

Returns

Value of the log-posterior

Implements BayesicSpace::Model.

5.12.3.4 operator=()

```
MumiLoc & MumiLoc::operator= (
            MumiLoc && in )
```

Move assignment operator.

Parameters

in	*in*	object to be moved

Returns

target object

5.12.4 Member Data Documentation

5.12.4.1 La_

`MatrixView BayesicSpace::MumiLoc::La_ [mutable], [protected]`

Line factorized precision matrix view.

Points to `vLx_`.

5.12.4.2 Le_

`MatrixView BayesicSpace::MumiLoc::Le_ [mutable], [protected]`

Error factorized precision matrix view.

Points to `vLx_`.

5.12.4.3 nuc_

`NumerUtil BayesicSpace::MumiLoc::nuc_ [protected]`

Numerical utility collection

5.12.4.4 vLx_

`vector<double> BayesicSpace::MumiLoc::vLx_ [mutable], [protected]`

Expanded $\boldsymbol{L}$ matrices.

Vectorized error and line unity triangular matrices ($\boldsymbol{L}_X$ in the model description).

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

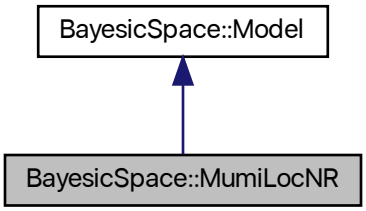
- src/mumimo.hpp
- src/mumimo.cpp

5.13 BayesicSpace::MumiLocNR Class Reference

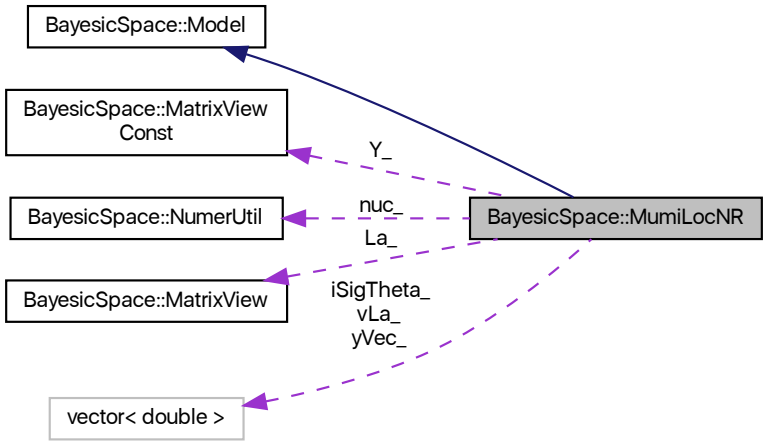
Mixture model for location parameters, no replication.

`#include <mumimo.hpp>`

Inheritance diagram for BayesicSpace::MumiLocNR:



Collaboration diagram for BayesicSpace::MumiLocNR:



Public Member Functions

- MumiLocNR ()

  Default constructor.

- MumiLocNR (const vector< double > *yVec, const size_t &d, const vector< double > *iSigVec, const double &tau, const size_t &nPops, const double &alphaPr)

  Constructor.
- ~MumiLocNR ()

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

  Copy constructor (deleted)
- MumiLocNR & operator= (const MumiLocNR &in)=delete

  Copy assignment (deleted)
- MumiLocNR (MumiLocNR &&in)

  Move constructor.
- MumiLocNR & operator= (MumiLocNR &&in)

  Move assignment operator.
- double logPost (const vector< double > &theta) const override

  Log-posterior function.
- void gradient (const vector< double > &theta, vector< double > &grad) const override

  Gradient of the log-posterior.

Protected Member Functions

- void expandISvec_ () const

  Expand the vector of factorized precision matrices.

Protected Attributes

- const vector< double > * yVec_

  Pointer to the data vector.
- MatrixViewConst Y_

  Matrix view of data.
- double tau0_

  Fixed prior precision for unmodeled effects.
- const vector< double > * iSigTheta_

  Pointer to a precision parameter vector.
- MatrixView La_

  Line factorized precision matrix view.
- vector< double > vLa_

  Expanded $\boldsymbol{L}_A$ matrix.
- size_t fTaInd_

  Index of the first $\boldsymbol{T}_A$ element.
- size_t fTpInd_

  Index of the first $\boldsymbol{T}_P$ element.
- size_t PhiBegInd_

  Index of the first probability element.
- size_t Npop_

  Number of populations.
- double alphaPr_

  Prior population assignment probability $\alpha_0 - 1$.
- NumerUtil nuc_

5.13.1 Detailed Description

Mixture model for location parameters, no replication.

Implements log-posterior and gradient for the location parameters of the multiplicative mixture model with no replication.

5.13.2 Constructor & Destructor Documentation

5.13.2.1 MumiLocNR() [1/2]

```
MumiLocNR::MumiLocNR (
            const vector< double > * yVec,
            const size_t & d,
            const vector< double > * iSigVec,
            const double & tau,
            const size_t & nPops,
            const double & alphaPr )
```

Constructor.

Parameters

in	*yVec*	pointer vectorized data matrix
in	*d*	number of traits
in	*iSigVec*	pointer to vectorized inverse-covariance matrix collection
in	*xVec*	pointer to vectorized covariate predictor matrix
in	*tau*	fixed prior for the unmodeled ("fixed") effects and overall mean (intercept)
in	*nPops*	number of populations
in	*alphaPr*	prior on α on population assignment probabilities

5.13.2.2 MumiLocNR() [2/2]

```
MumiLocNR::MumiLocNR (
            MumiLocNR && in )
```

Move constructor.

Parameters

in	*in*	object to move

5.13.3 Member Function Documentation

5.13.3.1 expandISvec_()

```
void MumiLocNR::expandISvec_ ( ) const  [protected]
```

Expand the vector of factorized precision matrices.

Expands the triangular $\boldsymbol{L}_X$ matrices contained in the precision matrix vector into the internal L_ vector. The input vector stores only the non-zero elements of these matrices.

5.13.3.2 gradient()

```
void MumiLocNR::gradient (
            const vector< double > & theta,
            vector< double > & grad ) const  [override], [virtual]
```

Gradient of the log-posterior.

Calculates the partial derivative of the log-posterior for each element in the provided parameter vector.

Parameters

in	*theta*	parameter vector
out	*grad*	partial derivative (gradient) vector

Implements BayesicSpace::Model.

5.13.3.3 logPost()

```
double MumiLocNR::logPost (
            const vector< double > & theta ) const  [override], [virtual]
```

Log-posterior function.

Returns the value of the log-posterior given the data provided at construction and the passed-in parameter vector. The parameter vector has the population means and population assignment probabilities in that order.

Parameters

in	*theta*	parameter vector

Returns

Value of the log-posterior

Implements BayesicSpace::Model.

5.13.3.4 operator=()

```
MumiLocNR & MumiLocNR::operator= (
            MumiLocNR && in )
```

Move assignment operator.

Parameters

in	*in*	object to be moved

Returns

target object

5.13.4 Member Data Documentation

5.13.4.1 La_

```
MatrixView BayesicSpace::MumiLocNR::La_  [mutable], [protected]
```

Line factorized precision matrix view.

Points to vLa_.

5.13.4.2 nuc_

```
NumerUtil BayesicSpace::MumiLocNR::nuc_  [protected]
```

Numerical utility collection

5.13.4.3 vLa_

```
vector<double> BayesicSpace::MumiLocNR::vLa_  [mutable], [protected]
```

Expanded $\boldsymbol{L}_A$ matrix.

Vectorized line unity triangular matrix ($\boldsymbol{L}_A$ in the model description).

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

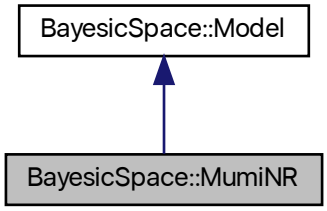
- src/mumimo.hpp
- src/mumimo.cpp

5.14 BayesicSpace::MumiNR Class Reference

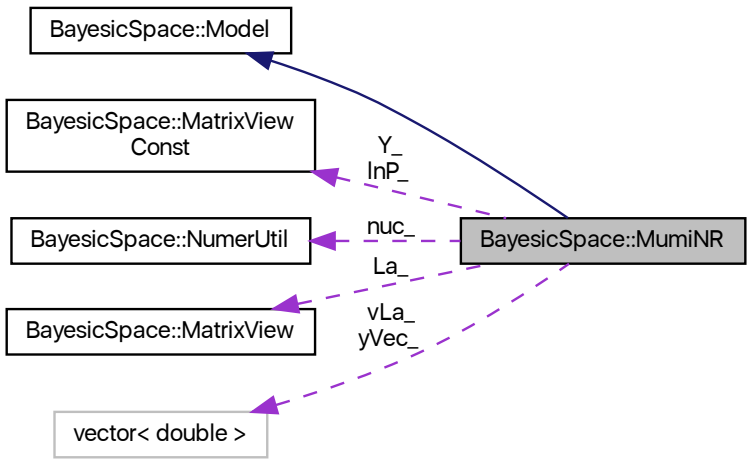
No-replication multiplicative mixture model parameter component.

```
#include <mumimo.hpp>
```

Inheritance diagram for BayesicSpace::MumiNR:



Collaboration diagram for BayesicSpace::MumiNR:



Public Member Functions

- MumiNR ()

  Default constructor.
- MumiNR (const vector< double > *yVec, const vector< double > *lnpVec, const size_t &d, const size_t &Npop, const double &tau0, const double &nu0, const double &invAsq)

  Constructor.
- ~MumiNR ()

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

  Copy constructor (deleted)
- MumiNR & operator= (const MumiNR &in)=delete

  Copy assignment (deleted)
- MumiNR (MumiNR &&in)

  Move constructor.
- MumiNR & operator= (MumiNR &&in)

  Move assignment operator.
- double logPost (const vector< double > &theta) const override

  Log-posterior function.
- void gradient (const vector< double > &theta, vector< double > &grad) const override

  Gradient of the log-posterior.

Protected Member Functions

- void expandISvec_ (const vector< double > &theta) const

  Expand the vector of factorized precision matrices.

Protected Attributes

- const vector< double > * yVec_

  Pointer to the data vector.
- MatrixViewConst Y_

  Matrix view of data.
- MatrixViewConst lnP_

  Population assignment log-probability matrix.
- MatrixView La_

  Line factorized precision matrix view.
- vector< double > vLa_

  Expanded $\boldsymbol{L}_A$ matrix.
- double tau0_

  Grand mean prior precision.
- double nu0_

  Prior degrees of freedom.
- double invAsq_

  Prior inverse-variance.
- size_t LaInd_

  First element of the among-individual inverse-covariances.
- size_t TaInd_

  First element of the among-individual precisions.
- size_t TpInd_

  First element of the among-population precisions.
- double NAnd_

  N_A + nu0 + 2d.
- double NPnd_

  N_P + nu0 + 2d.
- double nxnd_

  nu0(nu0 + 2d)*
- NumerUtil nuc_

Static Protected Attributes

- static const double lnMaxDbl_ = log(numeric_limits<double>::max())

  Natural log of `DBL_MAX`

5.14.1 Detailed Description

No-replication multiplicative mixture model parameter component.

Models all mixture model components given population assignment probabilities. Implements the log-posterior and gradient methods.

5.14.2 Constructor & Destructor Documentation

5.14.2.1 MumiNR() [1/2]

```
MumiNR::MumiNR (
            const vector< double > * yVec,
            const vector< double > * lnpVec,
            const size_t & d,
            const size_t & Npop,
            const double & tau0,
            const double & nu0,
            const double & invAsq )
```

Constructor.

Parameters

in	*yVec*	pointer to the data vectorized matrix
in	*lnpVec*	pointer to the population assignment log-probability vectorized matrix
in	*d*	number of traits
in	*Npop*	number of populations
in	*tau0*	grand mean prior precision
in	*nu0*	inverse covariance prior degrees of freedom
in	*invAsq*	inverse covariance prior precision

5.14.2.2 MumiNR() [2/2]

```
MumiNR::MumiNR (
            MumiNR && in )
```

Move constructor.

Parameters

in	*in*	object to move

5.14.3 Member Function Documentation

5.14.3.1 expandISvec_()

```
void MumiNR::expandISvec_ (
            const vector< double > & theta ) const  [protected]
```

Expand the vector of factorized precision matrices.

Expands the triangular $\boldsymbol{L}_A$ matrix contained in the precision matrix potion of the parameter vector into the internal L_ vector. The parameter vector stores only the non-zero elements of these matrix.

Parameters

in	*theta*	the parameter vector

5.14.3.2 gradient()

```
void MumiNR::gradient (
            const vector< double > & theta,
            vector< double > & grad ) const  [override], [virtual]
```

Gradient of the log-posterior.

Calculates the partial derivative of the log-posterior for each element in the provided parameter vector.

Parameters

in	*theta*	parameter vector
out	*grad*	partial derivative (gradient) vector

Implements BayesicSpace::Model.

5.14.3.3 logPost()

```
double MumiNR::logPost (
            const vector< double > & theta ) const  [override], [virtual]
```

Log-posterior function.

Returns the value of the log-posterior given the data provided at construction and the passed-in parameter vector. The parameter vector has population means, among-individual inverse covariances, among-individual precisions, and among-population precisions in that order.

Parameters

in	*theta*	parameter vector

Returns

Value of the log-posterior

Implements BayesicSpace::Model.

5.14.3.4 operator=()

```
MumiNR & MumiNR::operator= (
            MumiNR && in )
```

Move assignment operator.

Parameters

in	*in*	object to be moved

Returns

target object

5.14.4 Member Data Documentation

5.14.4.1 invAsq_

```
double BayesicSpace::MumiNR::invAsq_  [protected]
```

Prior inverse-variance.

Inverse variance of the prior. Should be set to a large value for a vague prior.

5.14.4.2 La_

```
MatrixView BayesicSpace::MumiNR::La_  [mutable], [protected]
```

Line factorized precision matrix view.

Points to vLa_.

5.14.4.3 nu0_

```
double BayesicSpace::MumiNR::nu0_  [protected]
```

Prior degrees of freedom.

Degrees of freedom of the half- t prior distribution on the covariance matrix. If $\nu_0 = 2$, the prior is half-Cauchy. Should not be large for a vague prior.

5.14.4.4 nuc_

```
NumerUtil BayesicSpace::MumiNR::nuc_  [protected]
```

Numerical utility collection

5.14.4.5 vLa_

```
vector<double> BayesicSpace::MumiNR::vLa_  [mutable], [protected]
```

Expanded $\boldsymbol{L}_A$ matrix.

Vectorized line unity triangular matrix ($\boldsymbol{L}_A$ in the model description).

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

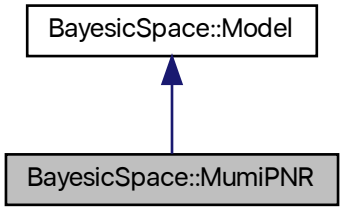
- src/mumimo.hpp
- src/mumimo.cpp

5.15 BayesicSpace::MumiPNR Class Reference

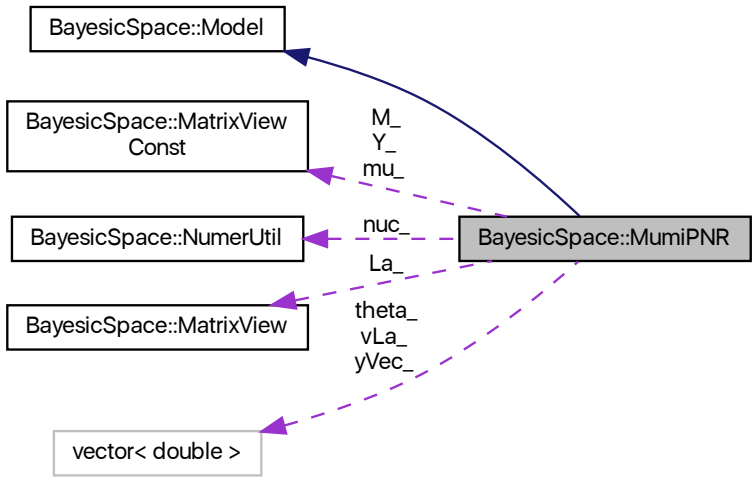
Population assignement probability component.

```
#include <mumimo.hpp>
```

Inheritance diagram for BayesicSpace::MumiPNR:



Collaboration diagram for BayesicSpace::MumiPNR:



Public Member Functions

- MumiPNR ()

  Default constructor.
- MumiPNR (const vector< double > *yVec, const vector< double > *theta, const size_t &d, const size_t &Npop, const double &alphaPr)

  Constructor.
- ~MumiPNR ()

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

  Copy constructor (deleted)
- MumiPNR & operator= (const MumiPNR &in)=delete

  Copy assignment (deleted)
- MumiPNR (MumiPNR &&in)

  Move constructor.
- MumiPNR & operator= (MumiPNR &&in)

  Move assignment operator.
- double logPost (const vector< double > &vPhi) const override

  Log-posterior function.
- void gradient (const vector< double > &vPhi, vector< double > &grad) const override

  Gradient of the log-posterior.

Protected Member Functions

- void expandISvec_ () const

  Expand the vector of factorized precision matrices.

Protected Attributes

- const vector< double > * yVec_

  Pointer to vectorized data.
- MatrixViewConst Y_

  Matrix view of data.
- double alphaPr_

  Prior population proportions.
- const vector< double > * theta_

  Pointer to model data.
- MatrixViewConst M_
- MatrixViewConst mu_
- MatrixView La_

  Line factorized precision matrix view.
- vector< double > vLa_

  Expanded $\boldsymbol{L}_A$ matrix.
- size_t LaInd_

  First element of the among-individual inverse-covariances.
- size_t TaInd_

  First element of the among-individual precisions.
- size_t TpInd_

  First element of the among-population precisions.
- NumerUtil nuc_

5.15.1 Detailed Description

Population assignement probability component.

Models population assignment probabilities given current values of parameters for the rest of the unreplicated hierarchical model. Implements log-posterior and gradient methods.

5.15.2 Constructor & Destructor Documentation

5.15.2.1 MumiPNR() [1/2]

```
MumiPNR::MumiPNR (
            const vector< double > * yVec,
            const vector< double > * theta,
            const size_t & d,
            const size_t & Npop,
            const double & alphaPr )
```

Constructor.

Parameters

in	*yVec*	pointer to the data vector
in	*theta*	pointer to hierarchical model parameters
in	*d*	number of traits
in	*Npop*	number of populations
in	*alphaPr*	prior population proportions

5.15.2.2 MumiPNR() [2/2]

```
MumiPNR::MumiPNR (
            MumiPNR && in )
```

Move constructor.

Parameters

in	*in*	object to move

5.15.3 Member Function Documentation

5.15.3.1 expandlSvec_()

```
void MumiPNR::expandISvec_ ( ) const  [protected]
```

Expand the vector of factorized precision matrices.

Expands the triangular $\boldsymbol{L}_A$ matrix contained in the precision matrix potion of the parameter vector into the internal L_ vector. The parameter vector stores only the non-zero elements of these matrix.

5.15.3.2 gradient()

```
void MumiPNR::gradient (
            const vector< double > & vPhi,
            vector< double > & grad ) const  [override], [virtual]
```

Gradient of the log-posterior.

Calculates the partial derivative of the log-posterior for each element in the provided parameter vector.

Parameters

in	*vPhi*	vectorized matrix of transformed population assignment probabilities
out	*grad*	partial derivative (gradient) vector

Implements BayesicSpace::Model.

5.15.3.3 logPost()

```
double MumiPNR::logPost (
            const vector< double > & vPhi ) const  [override], [virtual]
```

Log-posterior function.

Returns the value of the log-posterior given the data provided at construction and the passed-in parameter vector. The parameter vector has population means, among-individual inverse covariances, among-individual precisions, and among-population precisions in that order.

Parameters

in	*vPhi*	vectorized matrix of transformed population assignment probabilities

Returns

Value of the log-posterior

Implements BayesicSpace::Model.

5.15.3.4 operator=()

```
MumiPNR & MumiPNR::operator= (
            MumiPNR && in )
```

Move assignment operator.

Parameters

in	*in*	object to be moved

Returns

target object

5.15.4 Member Data Documentation

5.15.4.1 alphaPr_

`double BayesicSpace::MumiPNR::alphaPr_ [protected]`

Prior population proportions.

To save computation, actually stores $\alpha - 1$

5.15.4.2 La_

`MatrixView BayesicSpace::MumiPNR::La_ [mutable], [protected]`

Line factorized precision matrix view.

Points to `vLa_`.

5.15.4.3 M_

`MatrixViewConst BayesicSpace::MumiPNR::M_ [protected]`

Population mean matrix view

5.15.4.4 mu_

`MatrixViewConst BayesicSpace::MumiPNR::mu_ [protected]`

Overall mean matrix view

5.15.4.5 nuc_

`NumerUtil BayesicSpace::MumiPNR::nuc_ [protected]`

Numerical utility collection

5.15.4.6 vLa_

```
vector<double> BayesicSpace::MumiPNR::vLa_  [mutable], [protected]
```

Expanded $\boldsymbol{L}_A$ matrix.

Vectorized line unity triangular matrix ($\boldsymbol{L}_A$ in the model description).

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

- src/mumimo.hpp
- src/mumimo.cpp

5.16 BayesicSpace::NumerUtil Class Reference

Numerical utilities collection.

```
#include <utilities.hpp>
```

Public Member Functions

- void swapXOR (size_t &i, size_t &j) const
  Swap two `size_t` values.
- void insertionSort (const vector< size_t > &vec, vector< size_t > &ind) const
  Insertion sort.
- double logit (const double &p) const
  Logit function.
- double logistic (const double &x) const
  Logistic function.
- void sort (const vector< double > &target, vector< size_t > &outIdx) const
  Shell sort.
- double lnGamma (const double &x) const
  Logarithm of the Gamma function.
- double digamma (const double &x) const
  Digamma function.
- void phi2p (const MatrixViewConst &Phi, MatrixView &P) const
- void phi2lnp (const MatrixViewConst &Phi, MatrixView &lnP) const
- void w2p (const MatrixViewConst &W, MatrixView &P) const
- void phi2p (const MatrixView &Phi, MatrixView &P) const
- void phi2lnp (const MatrixView &Phi, MatrixView &lnP) const
- void w2p (const MatrixView &W, MatrixView &P) const
- double dotProd (const vector< double > &v) const
  Vector self-dot-product.
- double dotProd (const vector< double > &v1, const vector< double > &v2) const
  Dot-product of two vectors.
- void updateWeightedMean (const double &xn, const double &wn, double &mu, double &w) const
  Weighted mean update.
- double mean (const double arr[ ], const size_t &len)
  Mean of an array.

5.16.1 Detailed Description

Numerical utilities collection.

Implements numerical functions for use throughout the project.

5.16.2 Member Function Documentation

5.16.2.1 digamma()

```
double NumerUtil::digamma (
            const double & x ) const
```

Digamma function.

Defined only for $x > 0$, will return *NaN* otherwise. Adopted from the `dpsifn` function in R.

Parameters

in	*x*	function argument (must be positive)

Returns

value of the digamma function

5.16.2.2 dotProd() [1/2]

```
double NumerUtil::dotProd (
            const vector< double > & v ) const
```

Vector self-dot-product.

Parameters

in	*v*	vector

Returns

dot-product value

5.16.2.3 dotProd() [2/2]

```
double NumerUtil::dotProd (
            const vector< double > & v1,
            const vector< double > & v2 ) const
```

Dot-product of two vectors.

Parameters

in	*v1*	vector 1
in	*v2*	vector 2

Returns

dot-product value

5.16.2.4 insertionSort()

```
void NumerUtil::insertionSort (
            const vector< size_t > & vec,
            vector< size_t > & ind ) const
```

Insertion sort.

Performs an insertion sort on a vector, outputting the position of each element in a sorted vector. Sorting is done in order of increase.

Parameters

in	*vec*	vector to be sorted
out	*ind*	vector of sorted indexes, will replace contents of non-empty vector

5.16.2.5 lnGamma()

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

Logarithm of the Gamma function.

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

Parameters

in	*x*	value

Returns

$\log \Gamma(x)$

5.16.2.6 logistic()

```
double NumerUtil::logistic (
            const double & x ) const
```

Logistic function.

There is a guard against under- and overflow: the function returns 0.0 for $x \leq -35.0$ and 1.0 for $x \geq 35.0$.

Parameters

in	*x*	value to be projected to the (0, 1) interval

Returns

logistic transformation

5.16.2.7 logit()

```
double BayesicSpace::NumerUtil::logit (
            const double & p ) const  [inline]
```

Logit function.

Parameters

in	*p*	probability in the (0, 1) interval

Returns

logit transformation

5.16.2.8 mean()

```
double NumerUtil::mean (
            const double arr[],
            const size_t & len )
```

Mean of an array.

Uses the numerically stable recursive algorithm.

Parameters

in	*arr*	c-style array of values
in	*len*	array length

Returns

mean value

5.16.2.9 phi2lnp() [1/2]

```
void NumerUtil::phi2lnp (
            const MatrixView & Phi,
            MatrixView & lnP ) const
```

brief Unrestricted $\mathbf{\Phi}$ to log-probability matrix conversion

Does the hyper-spherical back-transformation of the free logit-space population assignment probability matrix to the true log-probability matrix (with all rows summing to 1). The $\mathbf{\Phi}$ matrix must have one fewer columns than $\ln(\boldsymbol{P})$.

Parameters

in	*Phi*	free-parameter matrix
out	*lnP*	population assignment probability matrix

5.16.2.10 phi2lnp() [2/2]

```
void NumerUtil::phi2lnp (
            const MatrixViewConst & Phi,
            MatrixView & lnP ) const
```

brief Unrestricted $\mathbf{\Phi}$ to log-probability matrix conversion

Does the hyper-spherical back-transformation of the free logit-space population assignment probability matrix to the true log-probability matrix (with all rows summing to 1). The $\mathbf{\Phi}$ matrix must have one fewer columns than $\ln(\boldsymbol{P})$.

Parameters

in	*Phi*	free-parameter matrix
out	*lnP*	population assignment probability matrix

5.16.2.11 phi2p() [1/2]

```
void NumerUtil::phi2p (
            const MatrixView & Phi,
            MatrixView & P ) const
```

brief Unrestricted Φ to probability matrix conversion

Does the hyper-spherical back-transformation of the free logit-space population assignment probability matrix to the true probability matrix (with all rows summing to 1).

Parameters

in	*Phi*	the free-parameter matrix
out	*P*	the population assignment probability matrix

5.16.2.12 phi2p() [2/2]

```
void NumerUtil::phi2p (
            const MatrixViewConst & Phi,
            MatrixView & P ) const
```

brief Unrestricted Φ to probability matrix conversion

Does the hyper-spherical back-transformation of the free logit-space population assignment probability matrix to the true probability matrix (with all rows summing to 1). The Φ matrix must have one fewer columns than P.

Parameters

in	*Phi*	free-parameter matrix
out	*P*	population assignment probability matrix

5.16.2.13 sort()

```
void NumerUtil::sort (
            const vector< double > & target,
            vector< size_t > & outIdx ) const
```

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 for population projection ordering, where the number of populations will typically not exceed 10.

Parameters

in	*target*	vector to be sorted
out	*outIdx*	vector of indexes

5.16.2.14 swapXOR()

```
void NumerUtil::swapXOR (
            size_t & i,
            size_t & j ) const
```

Swap two `size_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

5.16.2.15 updateWeightedMean()

```
void NumerUtil::updateWeightedMean (
            const double & xn,
            const double & wn,
            double & mu,
            double & w ) const
```

Weighted mean update.

Takes the current weighted mean and updates using the new data point and weight. The formula is

$$\bar{\mu}_n = \frac{\bar{\mu}_{n-1} \sum_{i=1}^{n-1} w_i + w_n x_n}{\sum_{i=1}^{n-1} w_i + w_n}$$

Parameters

in	*xn*	new point x_n
in	*wn*	weight w_n
out	*mu*	new mean
out	*w*	new weight

5.16.2.16 w2p() [1/2]

```
void NumerUtil::w2p (
            const MatrixView & W,
            MatrixView & P ) const
```

brief Weight matrix to probability matrix conversion

Does the hyper-spherical back-transformation of the free population assignment weight ($w = \text{logistic}(\phi)$) matrix to the true probability matrix (with all rows summing to 1).

Parameters

in	*W*	the free-parameter matrix
out	*P*	the population assignment probability matrix

5.16.2.17 w2p() [2/2]

```
void NumerUtil::w2p (
            const MatrixViewConst & W,
            MatrixView & P ) const
```

brief Weight matrix $\boldsymbol{W}$ to probability matrix conversion

Does the hyper-spherical back-transformation of the free population assignment weight ($w = \text{logistic}(\phi)$) matrix to the true probability matrix (with all rows summing to 1).

Parameters

in	*W*	free-parameter matrix
out	*P*	population assignment probability matrix

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

- src/utilities.hpp
- src/utilities.cpp

5.17 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.
- void rdirichlet (const vector< double > &alpha, vector< double > &p) const

  A Dirichlet 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.

5.17.1 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").

5.17.2 Constructor & Destructor Documentation

5.17.2.1 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.

5.17.2.2 RanDraw() [2/3]

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

Copy constructor.

Parameters

in	*old*	pbject to be copied

5.17.2.3 RanDraw() [3/3]

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

Move constructor.

Parameters

in	*old*	pbject to be moved

5.17.3 Member Function Documentation

5.17.3.1 operator=() [1/2]

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

Copy assignment.

Parameters

in	*old*	pbject to be copied

5.17.3.2 operator=() [2/2]

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

Move assignment.

Parameters

in	*old*	pbject to be moved

5.17.3.3 ranInt()

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

Generate random integer.

Returns

An unsigned random 64-bit integer

5.17.3.4 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

5.17.3.5 rdirichlet()

```
void RanDraw::rdirichlet (
            const vector< double > & alpha,
            vector< double > & p ) const
```

A Dirichlet deviate.

Generates a vector of probabilities, given a vector of concetration parameters $\alpha_K > 0$.

Parameters

in	*alpha*	vector of concentration parameters
out	*p*	vector of probabilities, must be the same length as α.

5.17.3.6 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

5.17.3.7 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

5.17.3.8 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

5.17.3.9 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

5.17.3.10 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

5.17.3.11 runif()

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

Generate a uniform deviate.

Returns

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

5.17.3.12 runifno()

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

Generate a non-one uniform deviate.

Returns

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

5.17.3.13 runifnz()

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

Generate a non-zero uniform deviate.

Returns

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

5.17.3.14 runifop()

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

Generate an open-interval uniform deviate.

Returns

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

5.17.3.15 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

5.17.3.16 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

5.17.3.17 shuffleUint()

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

Draw non-negative intergers in random order.

Uses the 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

5.17.3.18 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

5.17.3.19 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

5.17.3.20 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:

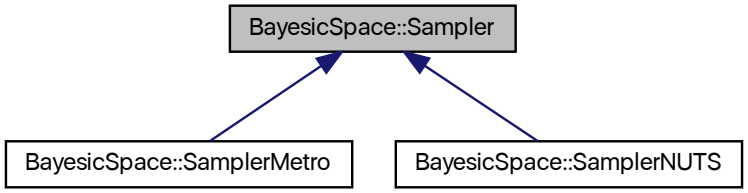
- src/random.hpp
- src/random.cpp

5.18 BayesicSpace::Sampler Class Reference

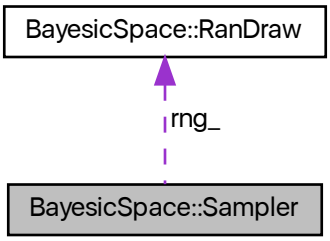
Sampler abstract base class.

`#include <sampler.hpp>`

Inheritance diagram for BayesicSpace::Sampler:



Collaboration diagram for BayesicSpace::Sampler:



Public Member Functions

- virtual ~Sampler ()
  Destructor.
- virtual int16_t adapt ()=0
  Adaptation (burn-in) phase update.
- virtual int16_t update ()=0
  Sampling phase update.

Protected Member Functions

- Sampler ()

  Default constructor.

Protected Attributes

- RanDraw rng_

  Random number generator.

5.18.1 Detailed Description

Sampler abstract base class.

Abstract base class for MCMC sampling methods.

5.18.2 Member Function Documentation

5.18.2.1 adapt()

```
virtual int16_t BayesicSpace::Sampler::adapt ( )  [pure virtual]
```

Adaptation (burn-in) phase update.

Returns

Implementation-dependent exit value

Implemented in BayesicSpace::SamplerMetro, and BayesicSpace::SamplerNUTS.

5.18.2.2 update()

```
virtual int16_t BayesicSpace::Sampler::update ( )  [pure virtual]
```

Sampling phase update.

Returns

Implementation-dependent exit value

Implemented in BayesicSpace::SamplerMetro, and BayesicSpace::SamplerNUTS.

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

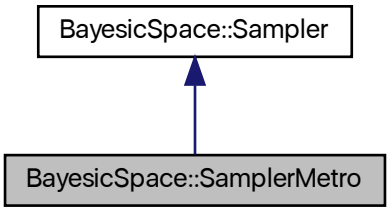
- src/sampler.hpp

5.19 BayesicSpace::SamplerMetro Class Reference

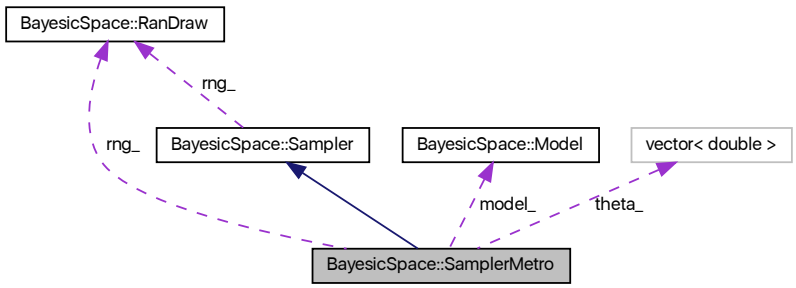
Metropolis sampler.

`#include <danuts.hpp>`

Inheritance diagram for BayesicSpace::SamplerMetro:



Collaboration diagram for BayesicSpace::SamplerMetro:



Public Member Functions

- SamplerMetro ()
  Default constructor.
- SamplerMetro (const Model *model, vector< double > *theta, const double &incr)
  Constructor.
- SamplerMetro (const SamplerMetro &in)=delete
  Copy constructor (deleted)
- SamplerMetro & operator= (const SamplerMetro &in)=delete
  Copy assignment operator (deleted)

- SamplerMetro (SamplerMetro &&in)

  Move constructor.
- SamplerMetro & operator= (SamplerMetro &&in)

  Move assignment operator.
- ~SamplerMetro ()

  Destructor.
- int16_t adapt () override

  Adaptation step.
- int16_t update () override

  Sampling step.

Protected Attributes

- const Model * model_

  Pointer to a model object.
- vector< double > * theta_

  Pointer to the parameter vector.
- double incr_

  Gaussian proposal standard deviation (step size)
- RanDraw rng_

  Random number generator.

Additional Inherited Members

5.19.1 Detailed Description

Metropolis sampler.

Simple Metropolis sampler with a Gaussian proposal.

5.19.2 Constructor & Destructor Documentation

5.19.2.1 SamplerMetro() [1/2]

```
BayesicSpace::SamplerMetro::SamplerMetro (
            const Model * model,
            vector< double > * theta,
            const double & incr )  [inline]
```

Constructor.

Parameters

in	*model*	pointer to a model object that has a logPost() function
in	*theta*	pointer to a parameter vector
in	*incr*	standard deviation of the Gaussian proposal

5.19.2.2 SamplerMetro() [2/2]

```
SamplerMetro::SamplerMetro (
            SamplerMetro && in )
```

Move constructor.

Parameters

in	*in*	object to be moved

5.19.3 Member Function Documentation

5.19.3.1 adapt()

```
int16_t SamplerMetro::adapt ( )  [override], [virtual]
```

Adaptation step.

Returns

accept/reject indicator (1 for accept, 0 for reject)

Implements BayesicSpace::Sampler.

5.19.3.2 operator=()

```
SamplerMetro & SamplerMetro::operator= (
            SamplerMetro && in )
```

Move assignment operator.

Parameters

in	*in*	object to be moved

Returns

Output object

5.19.3.3 update()

```
int16_t SamplerMetro::update ( )  [override], [virtual]
```

Sampling step.

Returns

accept/reject indicator (1 for accept, 0 for reject)

Implements BayesicSpace::Sampler.

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

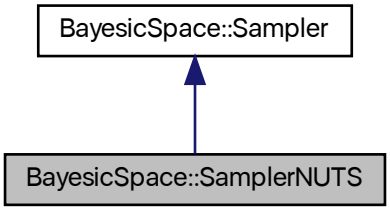
- src/danuts.hpp
- src/danuts.cpp

5.20 BayesicSpace::SamplerNUTS Class Reference

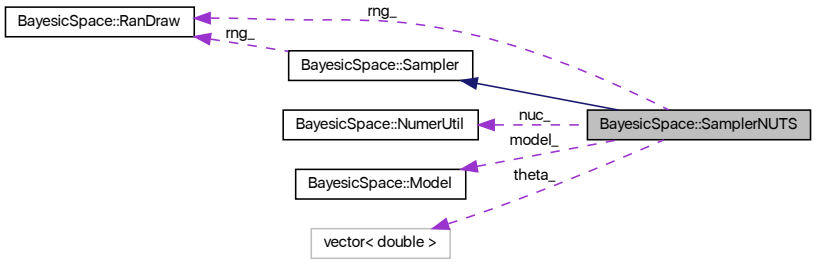
NUTS sampler class.

```
#include <danuts.hpp>
```

Inheritance diagram for BayesicSpace::SamplerNUTS:



Collaboration diagram for BayesicSpace::SamplerNUTS:



Public Member Functions

- SamplerNUTS ()

  Default constructor.
- SamplerNUTS (const Model *model, vector< double > *theta)

  Constructor.
- SamplerNUTS (const SamplerNUTS &in)=delete

  Copy constructor (deleted)
- SamplerNUTS & operator= (const SamplerNUTS &in)=delete

  Copy assignement operator (deleted)
- SamplerNUTS (SamplerNUTS &&in)

  Move constructor.
- SamplerNUTS & operator= (SamplerNUTS &&in)

  Move assignement operator.
- ~SamplerNUTS ()

  Destructor.
- double getEpsilon () const

  Get the current step size ϵ.
- int16_t adapt () override

  Adaptation phase of the NUTS updating.
- int16_t update () override

  NUTS update of parameters.

Protected Member Functions

- void findInitialEpsilon_ ()

  Initialize step size.
- void leapfrog_ (vector< double > &theta, vector< double > &r, const double &epsilon)

  Single leapfrog step.
- void buildTreePos_ (const vector< double > &theta, const vector< double > &r, const double &lu, const double &epsilon, const uint16_t &j, vector< double > &thetaPlus, vector< double > &rPlus, const vector< double > &thetaMinus, const vector< double > &rMinus, vector< double > &thetaPrime, double &nPrime, char &s)

Positive tree building function for the NUTS algorithm.

- void buildTreeNeg_ (const vector< double > &theta, const vector< double > &r, const double &lu, const double &epsilon, const uint16_t &j, const vector< double > &thetaPlus, const vector< double > &rPlus, vector< double > &thetaMinus, vector< double > &rMinus, vector< double > &thetaPrime, double &nPrime, char &s)

  Negative tree building function for the NUTS algorithm.

- void buildTreePos_ (const vector< double > &theta, const vector< double > &r, const double &lu, const double &epsilon, const uint16_t &j, vector< double > &thetaPlus, vector< double > &rPlus, const vector< double > &thetaMinus, const vector< double > &rMinus, vector< double > &thetaPrime, double &nPrime, char &s, double &alphaPrime, double &nAlphaPrime)

  Positive tree building function for the NUTS dual-averaging algorithm.

- void buildTreeNeg_ (const vector< double > &theta, const vector< double > &r, const double &lu, const double &epsilon, const uint16_t &j, const vector< double > &thetaPlus, const vector< double > &rPlus, vector< double > &thetaMinus, vector< double > &rMinus, vector< double > &thetaPrime, double &nPrime, char &s, double &alphaPrime, double &nAlphaPrime)

  Negative tree building function for the NUTS dual-averaging algorithm.

Protected Attributes

- RanDraw rng_

  Random numbers.

- NumerUtil nuc_

  Numerical method collection.

- double epsilon_

  HMC step size parameter ϵ.

- double mu_

  Shrinkage point μ.

- double nH0_

  Store the $-H(\theta^0, r^0)$ for each DA step here.

- double m_

  Warm-up step number.

- double Hprevious_

  The value $\bar{H}_{m-1}$ of the H_t statistic from the previous warm-up step.

- double logEpsBarPrevious_

  The value $\log \bar{\epsilon}_{m-1}$ of ϵ being optimized, from the previous warm-up step.

- double lastEpsilons_ [20]

  Last 20 ϵ values from the adaptation phase.

- bool firstAdapt_

  Has the first adaptation step been run?

- bool firstUpdate_

  Has the first post-adaptation update been run?

- const Model * model_

  Pointer to a model object.

- vector< double > * theta_

  Pointer to the parameter vector.

Static Protected Attributes

- static const double deltaMax_ = 1000.0

  The Δ_{max} value for the NUTS sampler.
- static const double delta_ = 0.6

  Target acceptance rate δ.
- static const double t0_ = 10.0

  Stabilization parameter t_0.
- static const double gamma_ = 0.05

  Shrinkage parameter γ.
- static const double negKappa_ = -0.75

  Step size schedule power $-\kappa$.
- static const uint64_t mask_ = static_cast<uint64_t>(0x01)

  Bit mask for the U{1,1} test.

5.20.1 Detailed Description

NUTS sampler class.

MCMC sampler class that implements the No-U-Turn Sampling with a dual-averaging algorithm to autimatically set the Hamiltonian step size ϵ. A class that implements a statistical model has to provide function to calculate a log-posterior and its gradient, as well as a pointer to the parameter vector.

5.20.2 Constructor & Destructor Documentation

5.20.2.1 SamplerNUTS() [1/2]

```
BayesicSpace::SamplerNUTS::SamplerNUTS (
            const Model * model,
            vector< double > * theta )  [inline]
```

Constructor.

Sets up the necessary functions and the pointer to the calling class parameter vector.

Parameters

in	*model*	pointer to a Model object that implements a particular statistical model
in	*theta*	pointer to the vector of parameters

5.20.2.2 SamplerNUTS() [2/2]

```
SamplerNUTS::SamplerNUTS (
            SamplerNUTS && in )
```

Move constructor.

Parameters

in	*in*	object to be moved

5.20.3 Member Function Documentation

5.20.3.1 adapt()

```
int16_t SamplerNUTS::adapt ( )  [override], [virtual]
```

Adaptation phase of the NUTS updating.

Uses Algorithm 6 from Hoffman and Gelman to settle on a good value for ϵ.

Checks the output of the log-posterior function and throws an exception if it evaluates to NaN or $+\infty$.

Returns

Number of leapfrog steps performed or -1 if log-posterior is $-\infty$

Implements BayesicSpace::Sampler.

5.20.3.2 buildTreeNeg_() [1/2]

```
void SamplerNUTS::buildTreeNeg_ (
            const vector< double > & theta,
            const vector< double > & r,
            const double & lu,
            const double & epsilon,
            const uint16_t & j,
            const vector< double > & thetaPlus,
            const vector< double > & rPlus,
            vector< double > & thetaMinus,
            vector< double > & rMinus,
            vector< double > & thetaPrime,
            double & nPrime,
            char & s )  [protected]
```

Negative tree building function for the NUTS algorithm.

As described in Algorithm 3 of Hoffman and Gelman, but the negative direction only. Instead of v, I use a signed ϵ.

Checks the output of the log-posterior function and throws an exception if it evaluates to NaN or $+\infty$.

Parameters

in	*theta*	input parameter vector θ
in	*r*	input momentum variables r
in	*lu*	log of the slice variable u
in	*epsilon*	step size ϵ
in	*j*	tree height j
in	*thetaPlus*	positive direction parameter vector θ^+
in	*rPlus*	positive direction momentum variable vector r^+
in,out	*thetaMinus*	negative direction parameter vector θ^-
in,out	*rMinus*	negative direction momentum variable vector r^-
out	*thetaPrime*	proposed move θ' to θ^m
out	*nPrime*	size n' of the implicit tree $\mathcal{C}'$
out	*s*	stopping condition s

5.20.3.3 buildTreeNeg_() [2/2]

```
void SamplerNUTS::buildTreeNeg_ (
            const vector< double > & theta,
            const vector< double > & r,
            const double & lu,
            const double & epsilon,
            const uint16_t & j,
            const vector< double > & thetaPlus,
            const vector< double > & rPlus,
            vector< double > & thetaMinus,
            vector< double > & rMinus,
            vector< double > & thetaPrime,
            double & nPrime,
            char & s,
            double & alphaPrime,
            double & nAlphaPrime )  [protected]
```

Negative tree building function for the NUTS dual-averaging algorithm.

This is for the adaptation phase to find optimal ϵ, as described in Algorithm 6 of Hoffman and Gelman, but for the negative direction only. Instead of v, I use a signed ϵ. All other variables follow the notation in the paper. To be used in the warm-up phase to tune step size ϵ.

Checks the output of the log-posterior function and throws an exception if it evaluates to NaN or $+\infty$.

Parameters

in	*theta*	input parameter vector θ
in	*r*	input momentum variables r
in	*lu*	log of the slice variable u

Parameters

in	*epsilon*	step size ϵ
in	*j*	tree height j
in	*thetaPlus*	positive direction parameter vector θ^+
in	*rPlus*	positive direction momentum variable vector r^+
in,out	*thetaMinus*	negative direction parameter vector θ^-
in,out	*rMinus*	negative direction momentum variable vector r^-
out	*thetaPrime*	proposed move θ' to θ^m
out	*nPrime*	size n' of the implicit tree $\mathcal{C}'$
out	*s*	stopping condition s
out	*alphaPrime*	acceptance probability α' to be optimized
out	*nAlphaPrime*	tree size after last doubling n'_α

5.20.3.4 buildTreePos_() [1/2]

```
void SamplerNUTS::buildTreePos_ (
            const vector< double > & theta,
            const vector< double > & r,
            const double & lu,
            const double & epsilon,
            const uint16_t & j,
            vector< double > & thetaPlus,
            vector< double > & rPlus,
            const vector< double > & thetaMinus,
            const vector< double > & rMinus,
            vector< double > & thetaPrime,
            double & nPrime,
            char & s )  [protected]
```

Positive tree building function for the NUTS algorithm.

As described in Algorithm 3 of Hoffman and Gelman, but the positive direction only. Instead of v, I use a signed ϵ.

Parameters

in	*theta*	input parameter vector θ
in	*r*	input momentum variables r
in	*lu*	log of the slice variable u
in	*epsilon*	step size ϵ
in	*j*	tree height j
in,out	*thetaPlus*	positive direction parameter vector θ^+
in,out	*rPlus*	positive direction momentum variable vector r^+
in	*thetaMinus*	negative direction parameter vector θ^-

Parameters

in	*rMinus*	negative direction momentum variable vector r^-
out	*thetaPrime*	proposed move θ' to θ^m
out	*nPrime*	size n' of the implicit tree $\mathcal{C}'$
out	*s*	stopping condition s

5.20.3.5 buildTreePos_() [2/2]

```
void SamplerNUTS::buildTreePos_ (
            const vector< double > & theta,
            const vector< double > & r,
            const double & lu,
            const double & epsilon,
            const uint16_t & j,
            vector< double > & thetaPlus,
            vector< double > & rPlus,
            const vector< double > & thetaMinus,
            const vector< double > & rMinus,
            vector< double > & thetaPrime,
            double & nPrime,
            char & s,
            double & alphaPrime,
            double & nAlphaPrime )  [protected]
```

Positive tree building function for the NUTS dual-averaging algorithm.

This is for the adaptation phase to find optimal ϵ, as described in Algorithm 6 of Hoffman and Gelman, but for the positive direction only. Instead of v, I use a signed ϵ. All other variables follow the notation in the paper. To be used in the warm-up phase to tune step size ϵ.

Checks the output of the log-posterior function and throws an exception if it evaluates to $\mathtt{NaN}$ or $+\infty$.

Parameters

in	*theta*	input parameter vector θ
in	*r*	input momentum variables r
in	*lu*	log of the slice variable u
in	*epsilon*	step size ϵ
in	*j*	tree height j
in,out	*thetaPlus*	positive direction parameter vector θ^+
in,out	*rPlus*	positive direction momentum variable vector r^+
in	*thetaMinus*	negative direction parameter vector θ^-
in	*rMinus*	negative direction momentum variable vector r^-
out	*thetaPrime*	proposed move θ' to θ^m
out	*nPrime*	size n' of the implicit tree $\mathcal{C}'$
out	*s*	stopping condition s
out	*alphaPrime*	acceptance probability α' to be optimized
out	*nAlphaPrime*	tree size after last doubling n'_α

5.20.3.6 findInitialEpsilon_()

```
void SamplerNUTS::findInitialEpsilon_ ( )  [protected]
```

Initialize step size.

Picks a reasonable initial value for the HMC/NUTS step size ϵ. Uses Algorithm 4 from Hoffman and Gelman.

5.20.3.7 getEpsilon()

```
double BayesicSpace::SamplerNUTS::getEpsilon ( ) const  [inline]
```

Get the current step size ϵ.

Returns

Current ϵ

5.20.3.8 leapfrog_()

```
void SamplerNUTS::leapfrog_ (
            vector< double > & theta,
            vector< double > & r,
            const double & epsilon )  [protected]
```

Single leapfrog step.

Takes a single leapfrog step, modifying θ and r; ϵ can be negative, in which case the step is in the reverse direction.

Checks the output of the log-posterior function and throws an exception if it evaluates to NaN or $+\infty$.

Parameters

in,out	*theta*	the θ vector
in,out	*r*	the r vector
in	*epsilon*	the step size ϵ, possibly negative

5.20.3.9 operator=()

```
SamplerNUTS & SamplerNUTS::operator= (
            SamplerNUTS && in )
```

Move assignement operator.

Parameters

in	*in*	object to be moved

Returns

SamplerNUTS object

5.20.3.10 update()

```
int16_t SamplerNUTS::update ( )  [override], [virtual]
```

NUTS update of parameters.

The step size ϵ set during the adaptation phase.

Checks the output of the log-posterior function and throws an exception if it evaluates to NaN or $+\infty$.

Returns

Number of leapfrog steps performed or -1 if log-posterior is $-\infty$

Implements BayesicSpace::Sampler.

5.20.4 Member Data Documentation

5.20.4.1 model_

```
const Model* BayesicSpace::SamplerNUTS::model_  [protected]
```

Pointer to a model object.

Derived classes of this object implement particular statistical models.

5.20.4.2 theta_

```
vector<double>* BayesicSpace::SamplerNUTS::theta_  [protected]
```

Pointer to the parameter vector.

Points to the parameters of the calling model class.

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

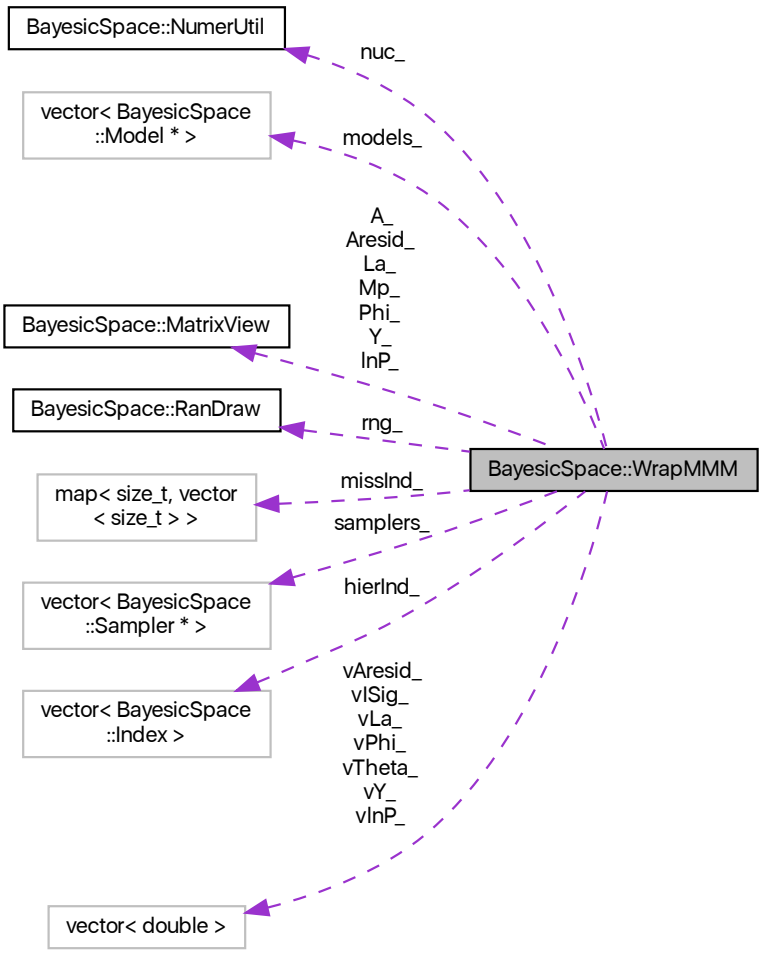
- src/danuts.hpp
- src/danuts.cpp

5.21 BayesicSpace::WrapMMM Class Reference

Replicated mixture model analysis.

```
#include <mumimo.hpp>
```

Collaboration diagram for BayesicSpace::WrapMMM:



Public Member Functions

- WrapMMM ()

  Default constructor.
- WrapMMM (const vector< double > &vY, const size_t &d, const uint32_t &Npop, const double &alphaPr, const double &tau0, const double &nu0, const double &invAsq)

  Constructor for a model with no replication.
- WrapMMM (const vector< double > &vY, const vector< size_t > &y2line, const uint32_t &Npop, const double &tauPrPhi, const double &alphaPr, const double &tau0, const double &nu0, const double &invAsq)

  Constructor for a one-level hierarchical model.
- WrapMMM (const vector< double > &vY, const vector< size_t > &y2line, const vector< int32_t > &missIDs, const uint32_t &Npop, const double &tauPrPhi, const double &alphaPr, const double &tau0, const double &nu0, const double &invAsq)

Constructor for a one-level hierarchical model with missing data.

- WrapMMM (WrapMMM &in)=delete

  Copy constructor (deleted)
- WrapMMM (WrapMMM &&in)=delete

  Move constructor (deleted)
- ∼WrapMMM ()

  Destructor.
- void runSampler (const uint32_t &Nadapt, const uint32_t &Nsample, const uint32_t &Nthin, vector< double > &thetaChain, vector< double > &isigChain, vector< double > &piChain)

  Sampler.
- void runSampler (const uint32_t &Nadapt, const uint32_t &Nsample, const uint32_t &Nthin, vector< double > &thetaChain, vector< double > &isigChain, vector< double > &piChain, vector< double > &impYchain)

  Sampler with missing data.

Protected Member Functions

- void imputeMissing_ ()

  Impute missing phenotype data.
- void sortPops_ ()

  Sort the populations.
- void calibratePhi_ ()

  Calibrate rows of `Phi_`
- void lnp2phi_ ()

  Convert log-probabilities to logit scores.
- void expandLa_ ()

  Expand lower triangle of the $\boldsymbol{L}_A$ *matrix.*
- double rowDistance_ (const MatrixView &m1, const size_t &row1, const MatrixView &m2, const size_t &row2)

  Euclidean distance between matrix rows.
- void kMeans_ (const MatrixView &X, const size_t &Kclust, const uint32_t &maxIt, Index &x2m, MatrixView &M)

  K-means clustering.

Protected Attributes

- vector< double > vY_

  Vectorized data matrix.
- MatrixView Y_

  Data matrix view.
- vector< Index > hierInd_

  Vector of indexes connecting hierarchy levels.
- vector< double > vTheta_

  Model paramaters.
- vector< double > vISig_

  Inverse-covariances.
- vector< double > vPhi_

  Transformed population assignment probabilities.
- vector< double > vlnP_

Population assignment log-probabilities.

- MatrixView A_

  Matrix view of line (accession) means.

- MatrixView Mp_

  Matrix view of population means.

- size_t PhiBegInd_

  Index of the first probability element.

- MatrixView Phi_

  Matrix view of logit-population assignment probabilities.

- MatrixView lnP_

  Matrix view of population assignment log-probabilies.

- size_t fTeInd_

  Index of the first $\boldsymbol{T}_E$ element.

- size_t fLaInd_

  Index of the first $\boldsymbol{L}_A$ element.

- size_t fTaInd_

  Index of the first $\boldsymbol{T}_A$ element.

- vector< double > vLa_

  Expanded vectorized $\boldsymbol{L}_A$ matrix.

- MatrixView La_

  Matrix view of the $\boldsymbol{L}_A$ matrix.

- vector< double > vAresid_

  Vectorized $\boldsymbol{A} - \boldsymbol{\mu}_{p\cdot}$ residual.

- MatrixView Aresid_

  Matrix view of the residual.

- map< size_t, vector< size_t > > missInd_

  Missingness index.

- vector< Model * > models_

  Models.

- vector< Sampler * > samplers_

  Vector of pointers to samplers.

- RanDraw rng_

  Random number generator.

- NumerUtil nuc_

  Numerical utility collection.

Static Protected Attributes

- static const double phiMin_ = -5.0

  `Phi_` *recalibration trigger value*

- static const double addVal_ = 3.0

  Value to add for calibration.

5.21.1 Detailed Description

Replicated mixture model analysis.

Builds a daNUTS within Gibbs sampler to fit a Gaussian mixture model for replicated data on multiple traits. Takes the data and factors for parameters, sets the initial values, and performs the sampling.

5.21.2 Constructor & Destructor Documentation

5.21.2.1 WrapMMM() [1/3]

```
WrapMMM::WrapMMM (
            const vector< double > & vY,
            const size_t & d,
            const uint32_t & Npop,
            const double & alphaPr,
            const double & tau0,
            const double & nu0,
            const double & invAsq )
```

Constructor for a model with no replication.

Establishes the initial parameter values and the sampler kind.

Parameters

in	*vY*	vectorized data matrix
in	*d*	number of traits
in	*Npop*	number of populations
in	*alphaPr*	α prior parameter for population assignment probabilities
in	*tau0*	prior precision for the "fixed" effects
in	*nu0*	prior degrees of freedom for precision matrices
in	*invAsq*	prior inverse variance for precision matrices

5.21.2.2 WrapMMM() [2/3]

```
WrapMMM::WrapMMM (
            const vector< double > & vY,
            const vector< size_t > & y2line,
            const uint32_t & Npop,
            const double & tauPrPhi,
```

```
            const double & alphaPr,
            const double & tau0,
            const double & nu0,
            const double & invAsq )
```

Constructor for a one-level hierarchical model.

Establishes the initial parameter values and the sampler kind. Input to the factor vector must be non-negative. This should be checked in the calling function.

Parameters

in	*vY*	vectorized data matrix
in	*y2line*	factor connecting data to lines (accessions)
in	*Npop*	number of populations
in	*tauPrPhi*	prior precision for ϕ
in	*alphaPr*	α prior parameter for population assignment probabilities
in	*tau0*	prior precision for the "fixed" effects
in	*nu0*	prior degrees of freedom for precision matrices
in	*invAsq*	prior inverse variance for precision matrices

5.21.2.3 WrapMMM() [3/3]

```
WrapMMM::WrapMMM (
            const vector< double > & vY,
            const vector< size_t > & y2line,
            const vector< int32_t > & missIDs,
            const uint32_t & Npop,
            const double & tauPrPhi,
            const double & alphaPr,
            const double & tau0,
            const double & nu0,
            const double & invAsq )
```

Constructor for a one-level hierarchical model with missing data.

Establishes the initial parameter values and the sampler kind. Input to the factor vector must be non-negative. This should be checked in the calling function.

Parameters

in	*vY*	vectorized data matrix
in	*y2line*	factor connecting data to lines (accessions)
in	*missIDs*	vectorized matrix (same dimensions as vY) with 1 corresponding to a missing data point and 0 otherwise
in	*Npop*	number of populations
in	*tauPrPhi*	prior precision for ϕ

Parameters

in	*alphaPr*	α prior parameter for population assignment probabilities
in	*tau0*	prior precision for the "fixed" effects
in	*nu0*	prior degrees of freedom for precision matrices
in	*invAsq*	prior inverse variance for precision matrices

5.21.3 Member Function Documentation

5.21.3.1 calibratePhi_()

```
void WrapMMM::calibratePhi_ ( )  [protected]
```

Calibrate rows of `Phi_`

Looks for rows of `Phi_` with all elements negative and below a threshold (currently - 5.0, determined empirically) and adds a constant to increase numerical stability. This does not affect population assignment probabilities.

5.21.3.2 expandLa_()

```
void WrapMMM::expandLa_ ( )  [protected]
```

Expand lower triangle of the $\boldsymbol{L}_A$ matrix.

Expands the triangular $\boldsymbol{L}_A$ matrix and multiplies its columns by the square root of $\boldsymbol{T}_A$. The input vector `vISig_` stores only the non-zero elements of these matrices.

5.21.3.3 kMeans_()

```
void WrapMMM::kMeans_ (
            const MatrixView & X,
            const size_t & Kclust,
            const uint32_t & maxIt,
            Index & x2m,
            MatrixView & M )  [protected]
```

K-means clustering.

Performs k-means clustering on a matrix of values. Each row of the input matrix is an item with observed values in columns.

Parameters

in	*X*	matrix of observations to be clustered
in	*Kclust*	number of clusters
in	*max↩ It*	maximum number of iterations
out	*x2m*	Index relating clusters to values
out	*M*	matrix of cluster means (clusters in rows)

5.21.3.4 lnp2phi_()

```
void WrapMMM::lnp2phi_ ( )  [protected]
```

Convert log-probabilities to logit scores.

Uses the Betancourt (2012) method of transforming population assignment log-probabilities to hyper-spherical coordinates. I then apply a logit transformation to further map scores to the $(-\infty, \infty)$ interval.

5.21.3.5 rowDistance_()

```
double WrapMMM::rowDistance_ (
            const MatrixView & m1,
            const size_t & row1,
            const MatrixView & m2,
            const size_t & row2 )  [protected]
```

Euclidean distance between matrix rows.

Parameters

in	*m1*	first matrix
in	*row1*	index of the first matrix row
in	*m2*	second matrix
in	*row2*	index of the second matrix row

Returns

euclidean distance between the rows

5.21.3.6 runSampler() [1/2]

```
void WrapMMM::runSampler (
            const uint32_t & Nadapt,
```

```
            const uint32_t & Nsample,
            const uint32_t & Nthin,
            vector< double > & thetaChain,
            vector< double > & isigChain,
            vector< double > & piChain )
```

Sampler.

Runs the chosen sampler with given parameters and outputs the chain.

Parameters

in	*Nadapt*	number of adaptation (burn-in) steps
in	*Nsample*	number of sampling steps
in	*Nthin*	thinning number
out	*thetaChain*	MCMC chain of location parameters
out	*isigChain*	MCMC chain of inverse-covariance parameters
out	*piChain*	MCMC chain of p_{ip}

5.21.3.7 runSampler() [2/2]

```
void WrapMMM::runSampler (
            const uint32_t & Nadapt,
            const uint32_t & Nsample,
            const uint32_t & Nthin,
            vector< double > & thetaChain,
            vector< double > & isigChain,
            vector< double > & piChain,
            vector< double > & impYchain )
```

Sampler with missing data.

Runs the sampler with given parameters, imputes missing data, and outputs chains. Imputed values for the missing data points are in the `impYchain` variable. The imputed data are arranged by row first, then by trait index within the row.

Parameters

in	*Nadapt*	number of adaptation (burn-in) steps
in	*Nsample*	number of sampling steps
in	*Nthin*	thinning number
out	*thetaChain*	MCMC chain of model parameters
out	*isigChain*	MCMC chain of inverse-covariance parameters
out	*piChain*	MCMC chain of p_{ip}
out	*impYchain*	MCMC chain tracking imputed data

5.21.3.8 sortPops_()

```
void WrapMMM::sortPops_ ( )  [protected]
```

Sort the populations.

Populations are sorted according to the index of the first individual that belongs to a population with high probability. This scheme is also known as left-ordering.

5.21.4 Member Data Documentation

5.21.4.1 hierInd_

```
vector<Index> BayesicSpace::WrapMMM::hierInd_  [protected]
```

Vector of indexes connecting hierarchy levels.

First element connects replicates (data) to line means, second connects lines to populations. The second index is updated as part of the mixture model.

5.21.4.2 missInd_

```
map<size_t, vector<size_t> > BayesicSpace::WrapMMM::missInd_  [protected]
```

Missingness index.

The key values are indexes of rows that have missing data, the mapped value is a vector with indexes of missing phenotypes for that row.

5.21.4.3 models_

```
vector<Model*> BayesicSpace::WrapMMM::models_  [protected]
```

Models.

The location parameter model first, then the inverse-covariance model.

5.21.4.4 samplers_

```
vector<Sampler*> BayesicSpace::WrapMMM::samplers_  [protected]
```

Vector of pointers to samplers.

Will point to the chosen derived sampler class(es).

5.21.4.5 vY_

```
vector<double> BayesicSpace::WrapMMM::vY_  [protected]
```

Vectorized data matrix.

Vectorized matrix of responses.

5.21.4.6 Y_

```
MatrixView BayesicSpace::WrapMMM::Y_  [protected]
```

Data matrix view.

Points to `vY_`.

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

- src/mumimo.hpp
- src/mumimo.cpp

Chapter 6

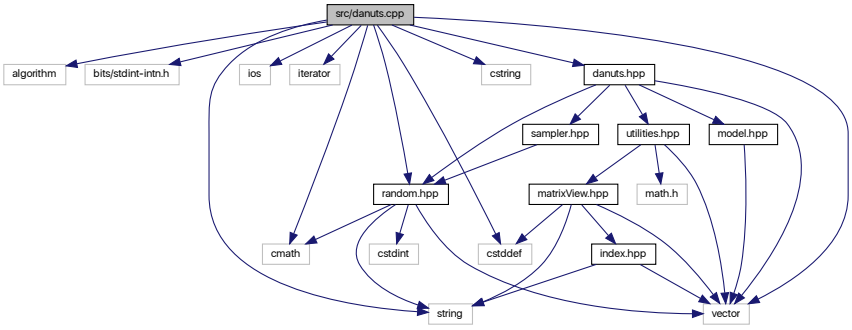
File Documentation

6.1 src/danuts.cpp File Reference

NUTS with dual averaging.

```
#include <algorithm>
#include <bits/stdint-intn.h>
#include <cstddef>
#include <ios>
#include <iterator>
#include <vector>
#include <string>
#include <cmath>
#include <cstring>
#include "danuts.hpp"
#include "random.hpp"
```

Include dependency graph for danuts.cpp:



6.1.1 Detailed Description

NUTS with dual averaging.

Author

Anthony J. Greenberg

Copyright

Version

1.0

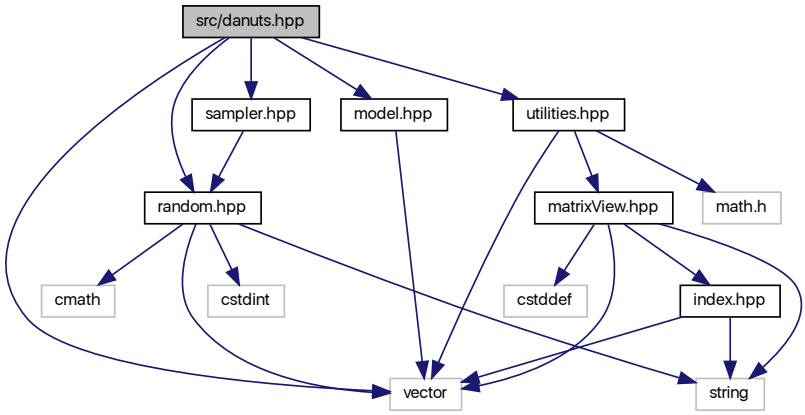
Class implementation for the No-U-Turn Sampler with dual averaging.

6.2 src/danuts.hpp File Reference

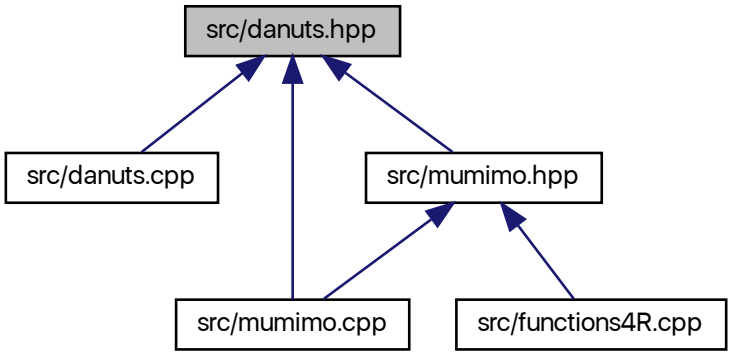
NUTS with dual averaging.

```
#include <vector>
#include "random.hpp"
#include "model.hpp"
#include "sampler.hpp"
#include "utilities.hpp"
```

Include dependency graph for danuts.hpp:



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



Classes

- class BayesicSpace::SamplerNUTS

  NUTS sampler class.
- class BayesicSpace::SamplerMetro

  Metropolis sampler.

6.2.1 Detailed Description

NUTS with dual averaging.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2018 Anthony J. Greenberg

Version

1.0

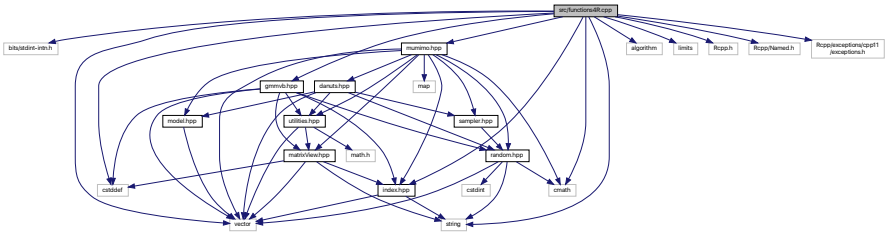
Class definition for an implementation of the No-U-Turn Sampler with dual averaging.

6.3 src/functions4R.cpp File Reference

R interface functions.

```
#include <bits/stdint-intn.h>
#include <cstddef>
#include <vector>
#include <cmath>
#include <algorithm>
#include <string>
#include <limits>
#include <Rcpp.h>
#include <Rcpp/Named.h>
#include <Rcpp/exceptions/cpp11/exceptions.h>
#include "mumimo.hpp"
#include "gmmvb.hpp"
#include "index.hpp"
```

Include dependency graph for functions4R.cpp:



Functions

- Rcpp::List **testLpostNR** (const std::vector< double > &yVec, const int32_t &d, const int32_t &Npop, std::vector< double > &theta, const std::vector< double > &P, const int32_t &ind, const double &limit, const double &incr)
- Rcpp::List **testLpostP** (const std::vector< double > &yVec, const int32_t &d, const int32_t &Npop, const std::vector< double > &theta, std::vector< double > &Phi, const int32_t &ind, const double &limit, const double &incr)
- Rcpp::List **testLpostLocNR** (const std::vector< double > &yVec, const int32_t &d, const int32_t &Npop, std::vector< double > &theta, const std::vector< double > &iSigTheta, const int32_t &ind, const double &limit, const double &incr)
- Rcpp::List **testGradNR** (const std::vector< double > &yVec, const int32_t &d, const int32_t &Npop, std::vector< double > &theta, const std::vector< double > &P, const int32_t &ind, const double &limit, const double &incr)
- Rcpp::List **testGradP** (const std::vector< double > &yVec, const int32_t &d, const int32_t &Npop, const std::vector< double > &theta, std::vector< double > &Phi, const int32_t &ind, const double &limit, const double &incr)
- Rcpp::List **testGradLocNR** (const std::vector< double > &yVec, const int32_t &d, const int32_t &Npop, std::vector< double > &theta, const std::vector< double > &iSigTheta, const int32_t &ind, const double &limit, const double &incr)
- Rcpp::List **testLpostSigNR** (const std::vector< double > &yVec, const int32_t &d, const int32_t &Npop, const std::vector< double > &theta, std::vector< double > &iSigTheta, const int32_t &ind, const double &limit, const double &incr)

- double **testLpostLoc** (const std::vector< double > &yVec, const std::vector< int32_t > &lnFac, const int32_t &Npop, const std::vector< double > &theta, const std::vector< double > &iSigTheta)
- Rcpp::List **lpTestLl** (const std::vector< double > &yVec, const std::vector< int32_t > &lnFac, const int32_t &Npop, std::vector< double > &theta, const std::vector< double > &iSigTheta, const int32_t &ind, const double &limit, const double &incr)
- Rcpp::List **gradTestLl** (const std::vector< double > &yVec, const std::vector< int32_t > &lnFac, const int32_t &Npop, std::vector< double > &theta, const std::vector< double > &iSigTheta, const int32_t &ind, const double &limit, const double &incr)
- Rcpp::List **lpTestSl** (const std::vector< double > &yVec, const std::vector< int32_t > &lnFac, const int32_t &Npop, const std::vector< double > &theta, std::vector< double > &iSigTheta, const int32_t &ind, const double &limit, const double &incr)
- Rcpp::List **gradTestSlnr** (const std::vector< double > &yVec, const int32_t &d, const int32_t &Npop, const std::vector< double > &theta, std::vector< double > &iSigTheta, const int32_t &ind, const double &limit, const double &incr)
- Rcpp::List **gradTestSl** (const std::vector< double > &yVec, const std::vector< int32_t > &lnFac, const int32_t &Npop, const std::vector< double > &theta, std::vector< double > &iSigTheta, const int32_t &ind, const double &limit, const double &incr)
- Rcpp::List **vbFit** (const std::vector< double > &yVec, const int32_t &d, const int32_t &nGroups, const double &alphaPr, const double &sigSqPr, const double &covRatio, const int32_t nReps)
- Rcpp::List **vbFitMiss** (std::vector< double > &yVec, const int32_t &d, const int32_t &nGroups, const double &alphaPr, const double &sigSqPr, const double &covRatio, const int32_t nReps)
- Rcpp::List **runSamplerNR** (const std::vector< double > &yVec, const int32_t &d, const int32_t &Npop, const int32_t &Nadapt, const int32_t &Nsamp, const int32_t &Nthin, const int32_t &Nchains)
- Rcpp::List **runSampler** (const std::vector< double > &yVec, const std::vector< int32_t > &lnFac, const int32_t &Npop, const int32_t &Nadapt, const int32_t &Nsamp, const int32_t &Nthin, const int32_t &Nchains)
- Rcpp::List **runSamplerMiss** (const std::vector< double > &yVec, const std::vector< int32_t > &lnFac, const std::vector< int32_t > &missIDs, const int32_t &Npop, const int32_t &Nadapt, const int32_t &Nsamp, const int32_t &Nthin, const int32_t &Nchains)

6.3.1 Detailed Description

R interface functions.

Author

Anthony J. Greenberg

Copyright

Version

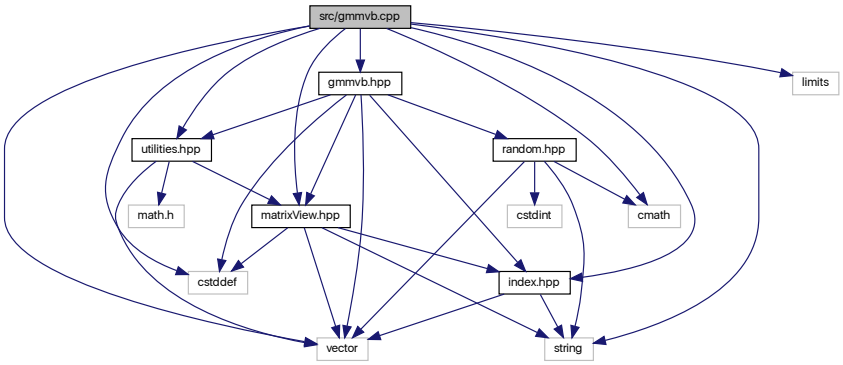
1.0

6.4 src/gmmvb.cpp File Reference

Variational inference of Gaussian mixture models.

```
#include <cstddef>
#include <vector>
#include <string>
#include <cmath>
#include <limits>
#include "gmmvb.hpp"
#include "matrixView.hpp"
#include "utilities.hpp"
#include "index.hpp"
```

Include dependency graph for gmmvb.cpp:



6.4.1 Detailed Description

Variational inference of Gaussian mixture models.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2020 Anthony J. Greenberg

Version

1.0

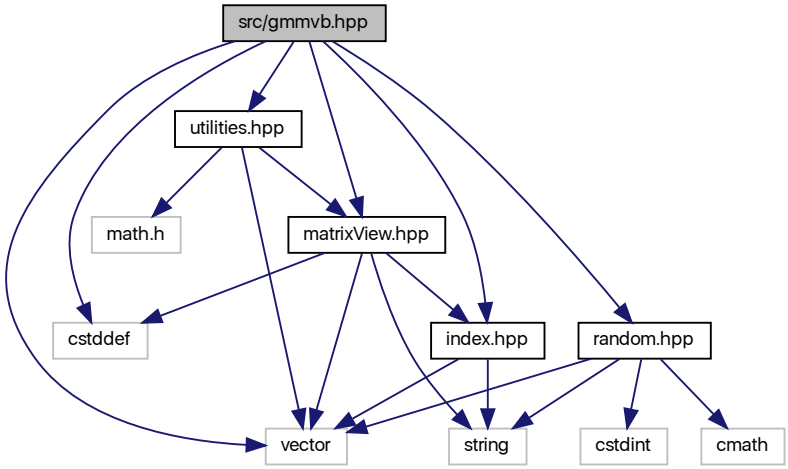
Implementation of variational Bayes inference of Gaussian mixture models.

6.5 src/gmmvb.hpp File Reference

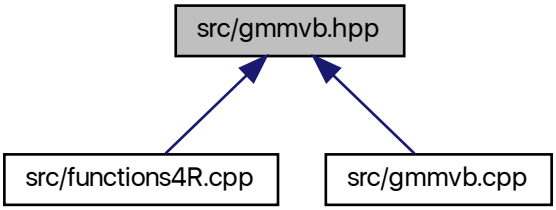
Variational inference of Gaussian mixture models.

```
#include <cstddef>
#include <vector>
#include "matrixView.hpp"
#include "utilities.hpp"
#include "random.hpp"
#include "index.hpp"
```

Include dependency graph for gmmvb.hpp:



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



Classes

- class BayesicSpace::GmmVB

  Variational Bayes class.
- class BayesicSpace::GmmVBmiss

  Variational Bayes with missing data.

6.5.1 Detailed Description

Variational inference of Gaussian mixture models.

Author

Anthony J. Greenberg

Copyright

Version

1.0

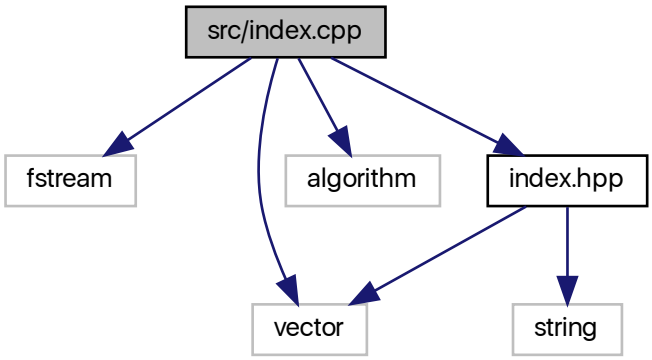
Class definition and interface documentation for variational Bayes inference of Gaussian mixture models.

6.6 src/index.cpp File Reference

Connect lines with populations.

```
#include <fstream>
#include <vector>
#include <algorithm>
#include "index.hpp"
```

Include dependency graph for index.cpp:



6.6.1 Detailed Description

Connect lines with populations.

Author

Anthony J. Greenberg

Copyright

Version

1.0

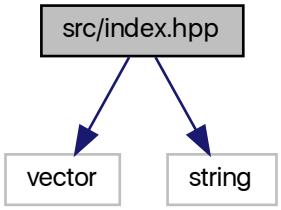
Implementation of a class that relates individuals to groups, similar to an factor in R.

6.7 src/index.hpp File Reference

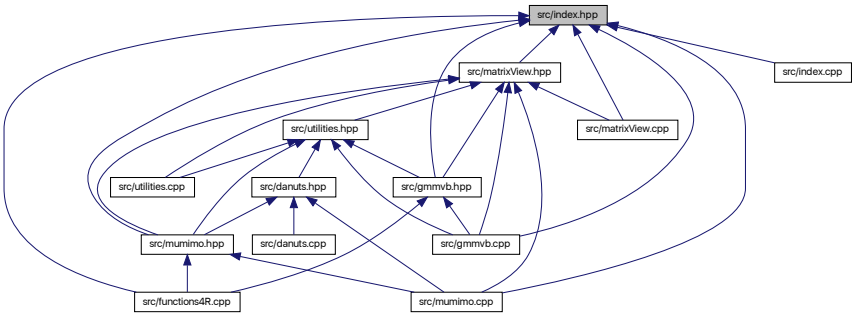
Connect lines with groups.

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

Include dependency graph for index.hpp:



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



Classes

- class BayesicSpace::Index

  Group index.

6.7.1 Detailed Description

Connect lines with groups.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2017 Anthony J. Greenberg

Version

1.0

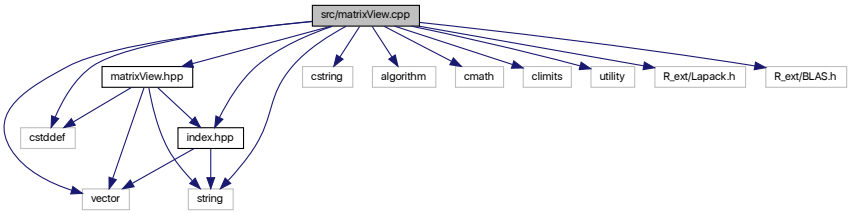
Definitions and interface documentation for a class that relates individuals to groups, similar to an factor in R.

6.8 src/matrixView.cpp File Reference

C++ matrix class that wraps pointers.

```
#include <cstddef>
#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 "matrixView.hpp"
#include "index.hpp"
```

Include dependency graph for matrixView.cpp:



Variables

- static const double **ZEROTOL** = 100.0 ∗ numeric_limits<double>::epsilon()

6.8.1 Detailed Description

C++ matrix class that wraps pointers.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2019 Anthony J. Greenberg

Version

0.1

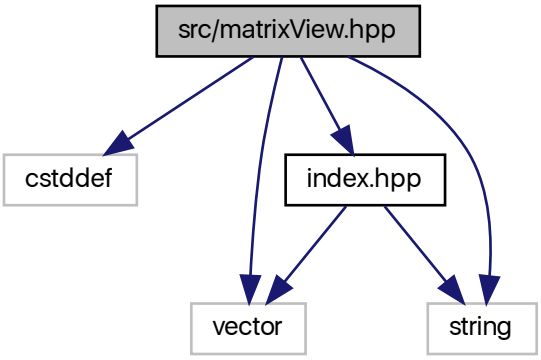
This is the class implementation file for the experimental MatrixView class. This version is for including in R packages, so it uses the R BLAS and LAPACK interfaces.

6.9 src/matrixView.hpp File Reference

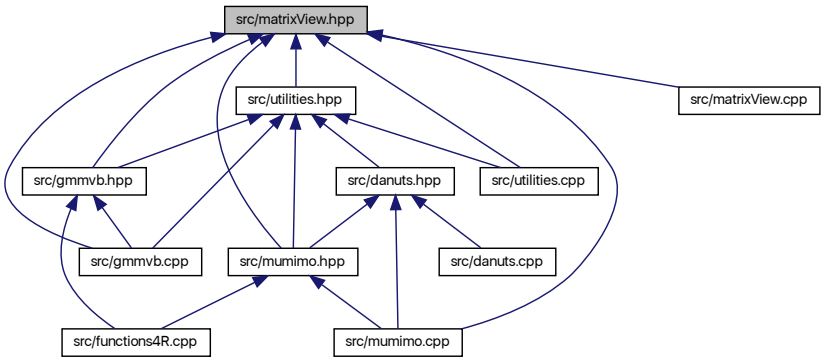
C++ matrix class that wraps pointers.

```
#include <cstddef>
#include <vector>
#include <string>
#include "index.hpp"
```

Include dependency graph for matrixView.hpp:



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



Classes

- class BayesicSpace::MatrixView

  Matrix view of a `vector`

- class BayesicSpace::MatrixViewConst

  A `const` *version of* `MatrixView`

6.9.1 Detailed Description

C++ matrix class that wraps pointers.

Author

Anthony J. Greenberg

Copyright

Version

0.1

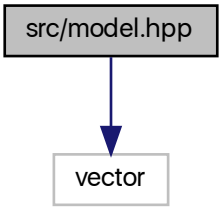
This is the project header file containing class definitions and interface documentation.

6.10 src/model.hpp File Reference

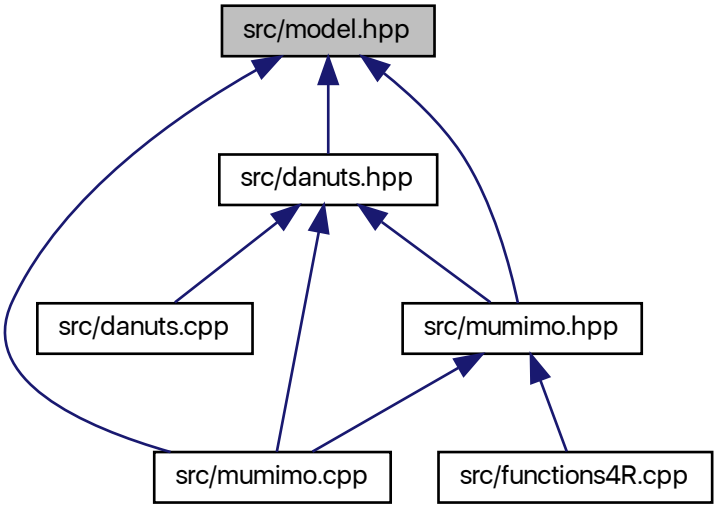
Abstract base statistical model class.

```
#include <vector>
```

Include dependency graph for model.hpp:



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



Classes

- class BayesicSpace::Model
  Model class.

6.10.1 Detailed Description

Abstract base statistical model class.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2018 Anthony J. Greenberg

Version

1.0

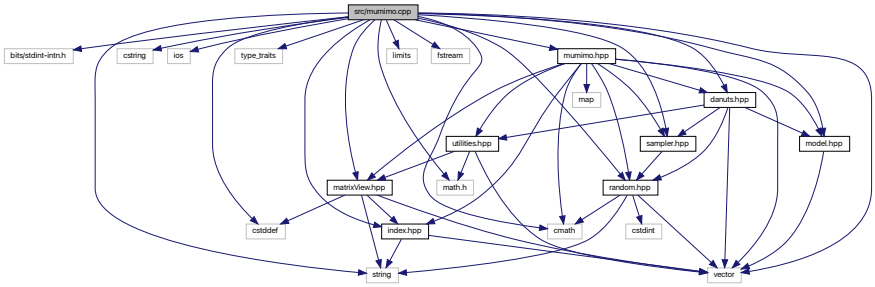
Class definition for an abstract base class for statistical models. Surfaces the log-posterior and its gradient for a given model.

6.11 src/mumimo.cpp File Reference

Gaussian mixture models.

```
#include <bits/stdint-intn.h>
#include <cstddef>
#include <cstring>
#include <ios>
#include <math.h>
#include <type_traits>
#include <vector>
#include <string>
#include <cmath>
#include <limits>
#include <fstream>
#include "index.hpp"
#include "random.hpp"
#include "model.hpp"
#include "mumimo.hpp"
#include "sampler.hpp"
#include "danuts.hpp"
#include "matrixView.hpp"
```

Include dependency graph for mumimo.cpp:



6.11.1 Detailed Description

Gaussian mixture models.

Author

Anthony J. Greenberg

Copyright

Version

1.0

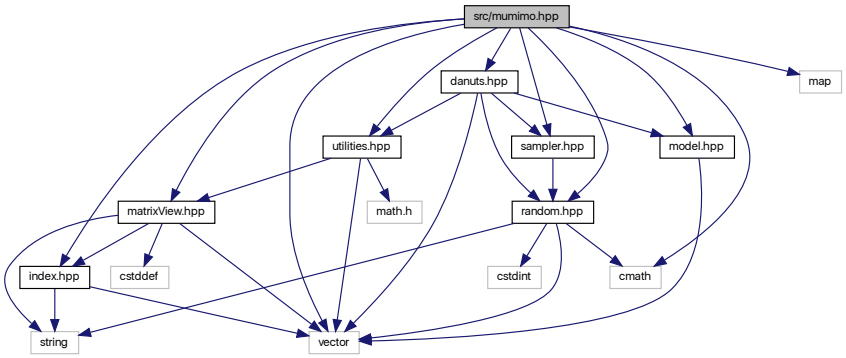
Class implementation to generate Markov chains for inference of Gaussian mixture models. Dual-averaging NUTS and Metropolis samplers for parameters groups are included within a Gibbs sampler.

6.12 src/mumimo.hpp File Reference

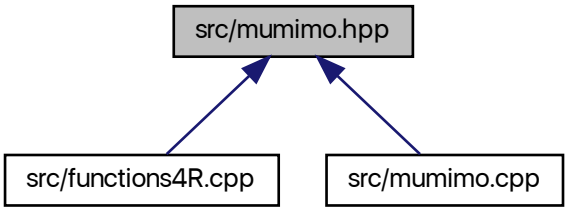
Gaussia mixture models.

```
#include <vector>
#include <cmath>
#include <map>
#include "index.hpp"
#include "random.hpp"
#include "model.hpp"
#include "sampler.hpp"
#include "danuts.hpp"
#include "matrixView.hpp"
#include "utilities.hpp"
```

Include dependency graph for mumimo.hpp:



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



Classes

- class BayesicSpace::MumiNR

No-replication multiplicative mixture model parameter component.

- class BayesicSpace::MumiPNR

  Population assignement probability component.

- class BayesicSpace::MumiLocNR

  Mixture model for location parameters, no replication.

- class BayesicSpace::MumiLoc

  Mixture model for location parameters.

- class BayesicSpace::MumiISigNR

  Model for inverse covariances with no replication.

- class BayesicSpace::MumiISig

  Model for inverse covariances.

- class BayesicSpace::WrapMMM

  Replicated mixture model analysis.

6.12.1 Detailed Description

Gaussia mixture models.

Author

Anthony J. Greenberg

Copyright

Version

1.0

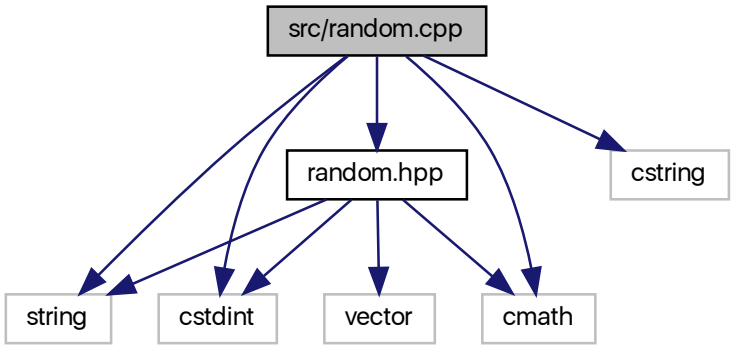
Class definition and interface documentation to generate Markov chains for inference of Gaussian mixture models. Dual-averaging NUTS and Metropolis samplers for parameters groups are included within a Gibbs sampler.

6.13 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:



6.13.1 Detailed Description

Random number generation.

Author

Anthony J. Greenberg

Copyright

Version

1.0

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

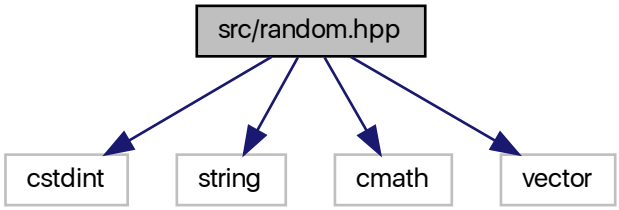
6.14 src/random.hpp File Reference

Random number generation.

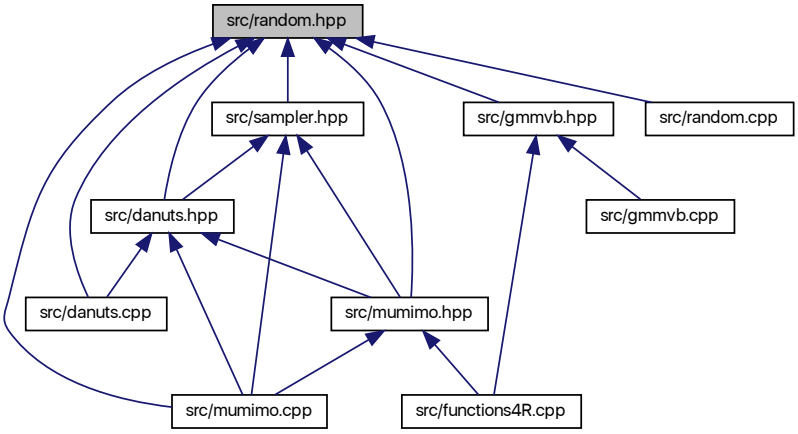
```
#include <cstdint>
#include <string>
#include <cmath>
```

```
#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.

6.14.1 Detailed Description

Random number generation.

Author

Anthony J. Greenberg

Copyright

Version

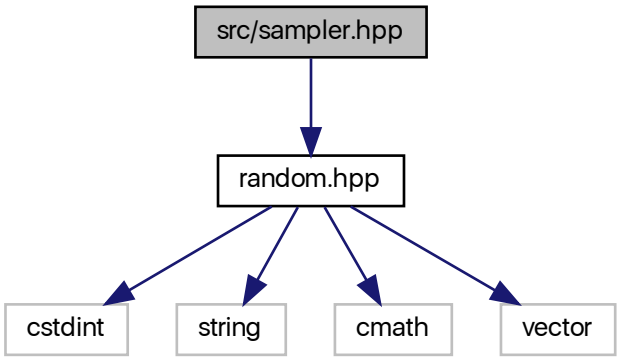
1.0

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

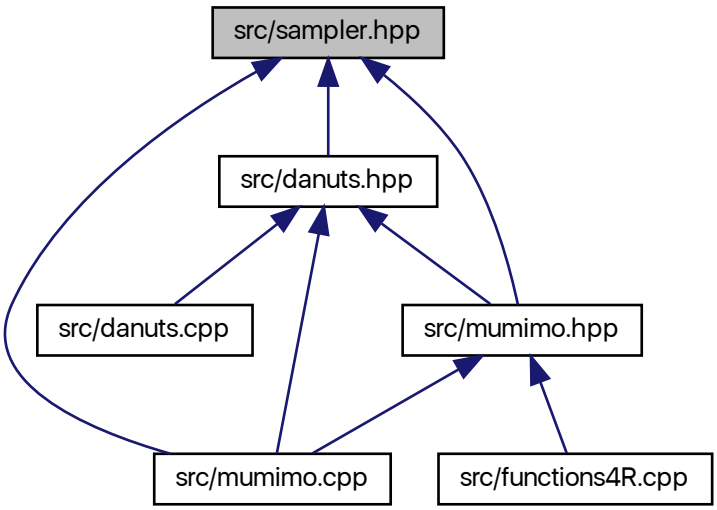
6.15 src/sampler.hpp File Reference

```
#include "random.hpp"
```

Include dependency graph for sampler.hpp:



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



Classes

- class BayesicSpace::Sampler

  Sampler abstract base class.

6.15.1 Detailed Description

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2018 Anthony J. Greenberg

Version

1.0

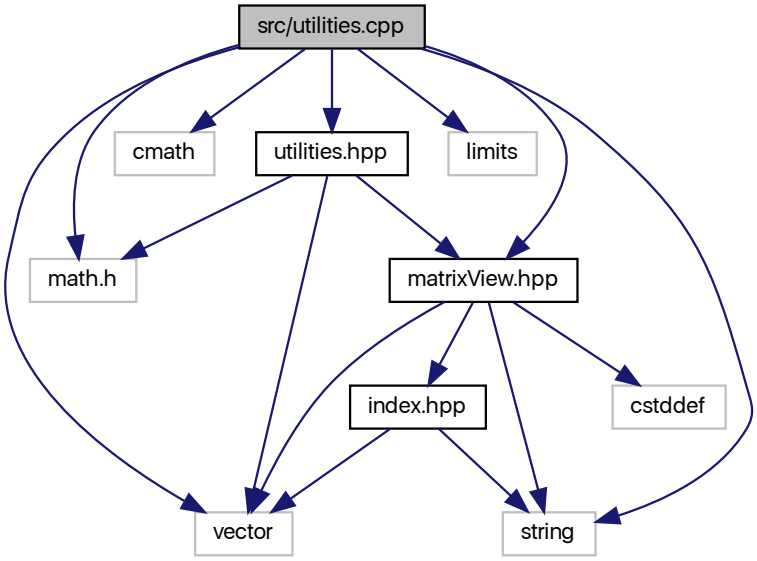
Class definition and interface documentation for the abstract base MCMC sampler class. The derived classes must surface an adapt() and update() functions.

6.16 src/utilities.cpp File Reference

Numerical utilities implementation.

```
#include <math.h>
#include <vector>
#include <cmath>
#include <string>
#include <limits>
#include "utilities.hpp"
#include "matrixView.hpp"
```

Include dependency graph for utilities.cpp:



6.16.1 Detailed Description

Numerical utilities implementation.

Author

Anthony J. Greenberg

Copyright

Copyright (c) 2020 Anthony J. Greenberg

Version

1.0

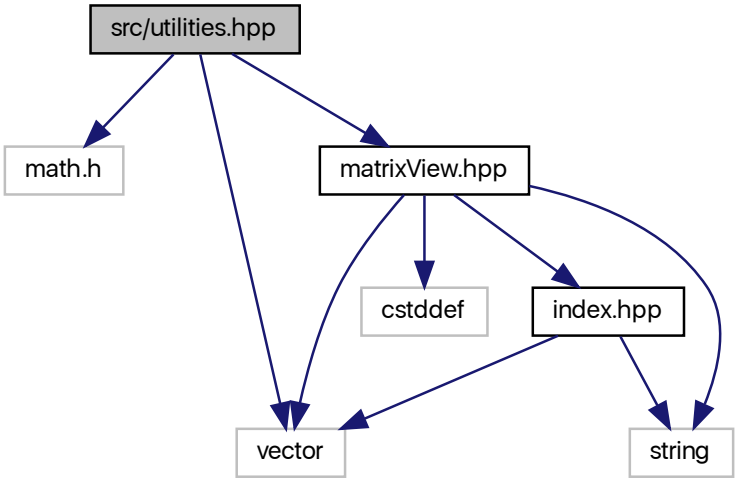
Class implementation for a set of numerical utilities. Implemented as a class because this seems to be the only way for these methods to be included using Rcpp with no compilation errors.

6.17 src/utilities.hpp File Reference

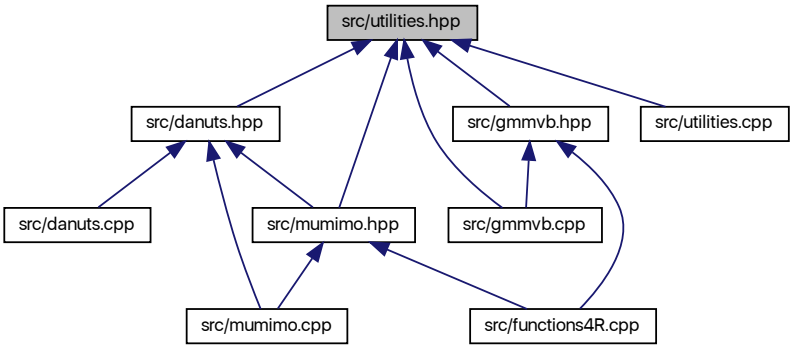
Numerical utilities.

```
#include <math.h>
#include <vector>
#include "matrixView.hpp"
```

Include dependency graph for utilities.hpp:



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



Classes

- class BayesicSpace::NumerUtil

  Numerical utilities collection.

6.17.1 Detailed Description

Numerical utilities.

Author

Anthony J. Greenberg

Copyright

Version

1.0

Class definition for a set of numerical utilities. Implemented as a class because this seems to be the only way for these methods to be included using Rcpp with no compilation errors.

Index