Matrix View

Generated by Doxygen 1.9.1

Chapter 1

Overview

This library implements a class similar to the GNU Scientific Library matrix_view. It points to a section of a C++ vector and interprets that section as a matrix. Matrix manipulations can then be done as usual. The matrices are column-major. See documentation for the available methods. Matrix dimensions are checked when necessary, unless the -DPKG_DEBUG_OFF compiler flag is set.

1.1 Dependencies

1.1.1 BLAS/LAPACK

Interfaces to a subset of BLAS and LAPACK routines is available. People prefer a variety of implementations of these libraries. I have two built-in possibilities:

- Using the libraries included with R, useful for writing R extensions.
- Using a stand-alone library, in particular Intel's Math Kernel Library. To use this option, include the -DNO_REXT compilation flag and make sure MKL is visible to the compiler.

Using other libraries is also possible, but you may need to modify the #include statement for BLAS and LAPACK. If you want to use CBLAS, you may also need to modify routine names.

1.1.2 Utilities

The implementation depends on a set of numerical utilities that I collected in the bayesicUtilities repository. I assume that the utilities are available in a bayesicUtilities directory at the same level as bayesicMatrix. This can be changed by modifying the #include path.

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 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**

4.1.1 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("")`.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 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

4.1.2.2 MatrixView() [2/2]

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

Move constructor.

Parameters

in	*inMat*	object to be moved

4.1.3 Member Function Documentation

4.1.3.1 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

4.1.3.2 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.

4.1.3.3 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

4.1.3.4 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.

4.1.3.5 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

4.1.3.6 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

4.1.3.7 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

4.1.3.8 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

4.1.3.9 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

4.1.3.10 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`

4.1.3.11 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`

4.1.3.12 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`

4.1.3.13 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

4.1.3.14 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

4.1.3.15 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

4.1.3.16 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

4.1.3.17 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

4.1.3.18 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

4.1.3.19 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`

4.1.3.20 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`

4.1.3.21 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

4.1.3.22 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

4.1.3.23 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

4.1.3.24 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

4.1.3.25 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

4.1.3.26 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

4.1.3.27 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

4.1.3.28 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

4.1.3.29 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

4.1.3.30 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

4.1.3.31 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

4.1.3.32 getNcols()

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

Access to number of columns.

Returns

size_t number of columns

4.1.3.33 getNrows()

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

Access to number of rows.

Returns

size_t number of rows

4.1.3.34 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

4.1.3.35 operator*=()

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

MatrixView-scalar compound product.

Parameters

in	*scal*	scalar

Returns

MatrixView result

4.1.3.36 operator+=()

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

MatrixView-scalar compound addition.

Parameters

in	*scal*	scalar

Returns

MatrixView result

4.1.3.37 operator-=()

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

MatrixView-scalar compound subtraction.

Parameters

in	*scal*	scalar

Returns

MatrixView result

4.1.3.38 operator/=()

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

MatrixView-scalar compound division.

Parameters

in	*scal*	scalar

Returns

MatrixView result

4.1.3.39 operator=()

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

Move assignment operator.

Parameters

in	*inMat*	object to be moved

Returns

MatrixView target object

4.1.3.40 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

4.1.3.41 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

4.1.3.42 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.

4.1.3.43 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

4.1.3.44 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

4.1.3.45 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

4.1.3.46 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

4.1.3.47 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

4.1.3.48 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

4.1.3.49 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

4.1.3.50 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

4.1.3.51 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

4.1.3.52 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

4.1.3.53 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

4.1.3.54 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

4.1.3.55 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

4.1.3.56 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

4.1.3.57 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

4.1.3.58 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

4.1.3.59 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`

4.1.3.60 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`

4.1.3.61 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

4.1.3.62 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

4.1.3.63 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

4.1.3.64 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

4.1.3.65 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

4.1.3.66 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

4.1.3.67 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

4.1.3.68 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

4.1.3.69 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

4.1.3.70 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

4.1.3.71 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

4.1.3.72 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

4.1.3.73 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

4.1.3.74 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

4.1.3.75 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:

- matrixView.hpp
- matrixView.cpp

4.2 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**

4.2.1 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.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 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

4.2.2.2 MatrixViewConst() [2/3]

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

Move constructor.

Parameters

in	*inMat*	object to be moved

4.2.2.3 MatrixViewConst() [3/3]

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

Move constructor from MatrixView

Parameters

in	*inMat*	object to be moved

4.2.3 Member Function Documentation

4.2.3.1 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

4.2.3.2 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

4.2.3.3 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

4.2.3.4 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

4.2.3.5 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`

4.2.3.6 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

4.2.3.7 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

4.2.3.8 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`

4.2.3.9 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`

4.2.3.10 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

4.2.3.11 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

4.2.3.12 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

4.2.3.13 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

4.2.3.14 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

4.2.3.15 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

4.2.3.16 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

4.2.3.17 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

4.2.3.18 getNcols()

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

Access to number of columns.

Returns

size_t number of columns

4.2.3.19 getNrows()

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

Access to number of rows.

Returns

size_t number of rows

4.2.3.20 operator=() [1/2]

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

Move assignment operator from MatrixView

Parameters

in	*inMat*	object to be moved

Returns

MatrixViewConst target object

4.2.3.21 operator=() [2/2]

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

Move assignment operator.

Parameters

in	*inMat*	object to be moved

Returns

MatrixViewConst target object

4.2.3.22 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

4.2.3.23 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

4.2.3.24 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`

4.2.3.25 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`

4.2.3.26 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`

4.2.3.27 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

4.2.3.28 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

4.2.3.29 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`

4.2.3.30 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`

4.2.3.31 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

4.2.3.32 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

4.2.3.33 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

4.2.3.34 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

4.2.3.35 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

4.2.3.36 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

4.2.3.37 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

4.2.3.38 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:

- matrixView.hpp
- matrixView.cpp

Chapter 5

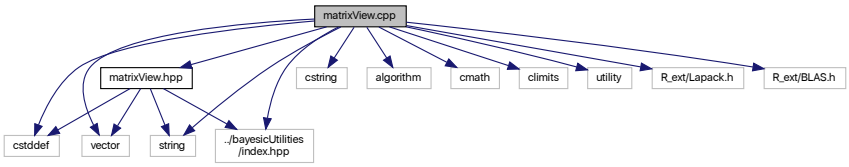
File Documentation

5.1 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 "../bayesicUtilities/index.hpp"
```

Include dependency graph for matrixView.cpp:



Variables

- static const double **ZEROTOL** = 100.0 ∗ numeric_limits<double>::epsilon()

5.1.1 Detailed Description

C++ matrix class that wraps pointers.

Author

Anthony J. Greenberg

Copyright

Version

0.1

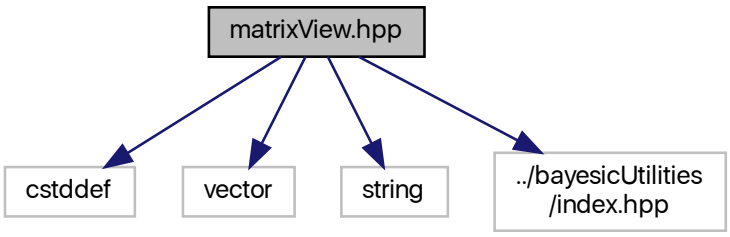
This is the class implementation file for the experimental MatrixView class. This version is for including in R packages, so it uses the R BLAS and LAPACK interfaces.

5.2 matrixView.hpp File Reference

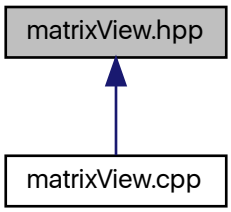
C++ matrix class that wraps pointers.

```
#include <cstddef>
#include <vector>
#include <string>
#include "../bayesicUtilities/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`

5.2.1 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.

Index