jmatrix

1.1

Generated by Doxygen 1.9.1

Chapter 1

jmatlib: a matrix library to manipulate big 2D matrices

1.1 General explanation

Matrix indexes are 32-bit unsigned integers so matrix size is in practice limited by the amount of memory.
This library manages full, sparse and symmetric matrices using as less memory as possible.
Matrix elements can be of any data type.
Matrices can be read from/written to disk in a compact binary format.
Rows and columns may have names (any C++ string).
The characteristics of a matrix can be known without loading it into memory.
Rows and columns can be read from disk, either by their names or numbers, without loading the complete matrix in memory.

One example program is provided, jmat, which is a command-line interface that allows creation from/writing to jmatrices from CSV files, as long as any kind of matrix manipulation. See section Files.

This library is used by parpamlib, a library to implement in parallel the Partiioning Around Medoids (PAM) clustering method that can be found in https://github.com/JdMDE/ppamlib

Its code with interface modifications is also used inside the jmatrix R package (https://cran.↩
r-project.org/web/packages/jmatrix/index.html)

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:

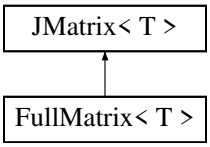
Chapter 5

Class Documentation

5.1 FullMatrix< T > Class Template Reference

`#include <fullmatrix.h>`

Inheritance diagram for FullMatrix< T >:



Public Member Functions

- FullMatrix ()
- FullMatrix (indextype nrows, indextype ncols)
- FullMatrix (indextype nrows, indextype ncols, bool warn)
- FullMatrix (const FullMatrix< T > &other)
- FullMatrix (std::string fname)
- FullMatrix (std::string fname, bool warn)
- FullMatrix (std::string fname, unsigned char vtype, char csep)
- void Resize (indextype newnr, indextype newnc)
- ~FullMatrix ()
- FullMatrix< T > & operator= (const FullMatrix< T > &other)
- FullMatrix< T > & operator!= (const FullMatrix< T > &other)
- T Get (indextype r, indextype c)
- void Set (indextype r, indextype c, T v)
- void GetRow (indextype r, T *v)
- void GetFullRow (indextype r, unsigned char *m, unsigned char s, T *v)
- void GetMarksOfFullRow (indextype r, unsigned char *m, unsigned char s)
- void SelfRowNorm (std::string ctype)
- void SelfColNorm (std::string ctype)
- void WriteCsv (std::string fname, char csep=',', bool withquotes=false)
- void WriteBin (std::string fname)
- float GetUsedMemoryMB ()

Additional Inherited Members

5.1.1 Detailed Description

template<typename T>
class FullMatrix< T >

@FullMatrix class to hold full matrices (all space booked in memory)

5.1.2 Constructor & Destructor Documentation

5.1.2.1 FullMatrix() [1/7]

```
template<typename T >
FullMatrix< T >::FullMatrix
```

Default constructor

5.1.2.2 FullMatrix() [2/7]

```
template<typename T >
FullMatrix< T >::FullMatrix (
            indextype nrows,
            indextype ncols )
```

Constructor with number of rows and columns

Parameters

in	*nrows*	Number of rows
in	*ncols*	Number of columns

5.1.2.3 FullMatrix() [3/7]

```
template<typename T >
FullMatrix< T >::FullMatrix (
            indextype nrows,
            indextype ncols,
            bool warn )
```

Constructor with number of rows and columns which gives memory warnings

Parameters

in	*nrows*	Number of rows
in	*ncols*	Number of columns
in	*warn*	Give memory warnings The idea is to use this form of constructor with warn=true, which internally calls MemoryWarnings, if one suspects the matrix to be constructed might provoke memory problems.

5.1.2.4 FullMatrix() [4/7]

```
template<typename T >
FullMatrix< T >::FullMatrix (
            const FullMatrix< T > & other )
```

Copy constructor

Parameters

in	*other*	Reference to the Matrix to be copied

5.1.2.5 FullMatrix() [5/7]

```
template<typename T >
FullMatrix< T >::FullMatrix (
            std::string fname )
```

Constructor to fill the matrix contents from a binary file
Binary file header as explained in the documentation to JMatrix::WriteBin

Parameters

in	*fname*	The name of the file to read

5.1.2.6 FullMatrix() [6/7]

```
template<typename T >
FullMatrix< T >::FullMatrix (
            std::string fname,
            bool warn )
```

Constructor to fill the matrix content from a binary file giving memory warnings
Binary file header as explained in the documentation to JMatrix::WriteBin

Parameters

in	*fname*	The name of the file to read
in	*warn*	Give memory warnings

5.1.2.7 FullMatrix() [7/7]

```
template<typename T >
FullMatrix< T >::FullMatrix (
            std::string fname,
            unsigned char vtype,
            char csep )
```

Constructor to fill the matrix content from a csv file
First line is supposed to have the field names which become the column names
First column of each line is supposed to have the field name, which becomes that row name
The passed character is the expected field sepparator (usually, comma or tab)

Parameters

in	*fname*	The name of the csv file to read
in	*vtype*	The data type to be stored (see constants at jmatrix.h)
in	*csep*	The character used as field sepparator

5.1.2.8 ~FullMatrix()

```
template<typename T >
FullMatrix< T >::~FullMatrix
```

Destructor

5.1.3 Member Function Documentation

5.1.3.1 Get()

```
template<typename T >
T FullMatrix< T >::Get (
            indextype r,
            indextype c )  [inline]
```

Function to get acess to an element

Parameters

in	*r*	The row to access
in	*c*	The columns to access

Returns

value at (r,c) of matrix with type T

5.1.3.2 GetFullRow()

```
template<typename T >
void FullMatrix< T >::GetFullRow (
            indextype r,
            unsigned char * m,
            unsigned char s,
            T * v )
```

Function to get a full row as a pointer to the content type plus a pointer to an array of marks. The content will have the values. The array of marks will be changed OR'ing the passed mark to each place where there is a value. The pointers to the values and marks are not returned since that way they do not need to be booked. They are passed as parameters and both are supposed to be properly allocated. This strange way of storing data has been choosen because it will be specially suitable to calculate distance between vectors

Parameters

in	*r*	The row to get
out	**m*	Pointer to the marks
in	*s*	The value to be OR'ed to each place at the mark array
out	**v*	Pointer to the values

5.1.3.3 GetMarksOfFullRow()

```
template<typename T >
void FullMatrix< T >::GetMarksOfFullRow (
            indextype r,
            unsigned char * m,
            unsigned char s )
```

Function to get from a full row a pointer to an array of marks signalling where the non-zero elements are. The array of marks will be changed OR'ing the passed mark to each place where there is a value. The pointer to the marks is not returned since that way it do not need to be booked. It is passed as a parameter and is supposed to be properly allocated.

Parameters

in	*r*	The row to get
out	**m*	Pointer to the marks
in	*s*	The value to be OR'ed to each place at the mark array

5.1.3.4 GetRow()

```
template<typename T >
void FullMatrix< T >::GetRow (
            indextype r,
            T * v )
```

Function to get a row as a pointer to the content type. The pointer is not returned since that way it does not need to be booked. The pointer to hold result is passed as parameter and it is supposed to be properly allocated.

Parameters

in	*r*	The row to get
out	**v*	Pointer to the result

5.1.3.5 GetUsedMemoryMB()

```
template<typename T >
float FullMatrix< T >::GetUsedMemoryMB
```

Function to get memory in MB used by this full matrix

Returns

The amount of memory in MB

5.1.3.6 operator"!=()

```
template<typename T >
FullMatrix< T > & FullMatrix< T >::operator!= (
            const FullMatrix< T > & other )
```

Transpose-assignment

Parameters

in	*other*	Reference to the Matrix to be assigned

Returns

Reference to the newly created Matrix, which is the transpose of the passed one

5.1.3.7 operator=()

```
template<typename T >
FullMatrix< T > & FullMatrix< T >::operator= (
            const FullMatrix< T > & other )
```

Assignment operator

Parameters

in	*other*	Reference to the Matrix to be assigned

Returns

Reference to the newly created Matrix

5.1.3.8 Resize()

```
template<typename T >
void FullMatrix< T >::Resize (
            indextype newnr,
            indextype newnc )
```

Function to resize the matrix
WARNING: previous content, if any, IS LOST (to be reviewed)

Parameters

in	*newnr*	New number of rows
in	*newnc*	New number of cols

5.1.3.9 SelfColNorm()

```
template<typename T >
void FullMatrix< T >::SelfColNorm (
            std::string ctype )
```

Function to alter the internal values of the matrix so that each column is normalized according to the requested normalization type The purpose of this function can be achieved with a loop using Set and Get, but using the internal structure makes the task much faster.
Normally, this function will not be used outside the context of bioinformatics where these normalizations are standard

Parameters

in	*ctype*	The requested type of normalization: rawn, log1 or log1n

5.1.3.10 SelfRowNorm()

```
template<typename T >
void FullMatrix< T >::SelfRowNorm (
            std::string ctype )
```

Function to alter the internal values of the matrix so that each row is normalized according to the requested normalization type The purpose of this function can be achieved with a loop using Set and Get, but using the internal structure makes the task much faster.
Normally, this function will not be used outside the context of bioinformatics where these normalizations are standard

Parameters

in	*ctype*	The requested type of normalization: rawn, log1 or log1n

5.1.3.11 Set()

```
template<typename T >
void FullMatrix< T >::Set (
            indextype r,
            indextype c,
            T v )  [inline]
```

Function to set an element

Parameters

in	*r*	The row to access
in	*c*	The column to access
in	*v*	The value to be set (of type T)

5.1.3.12 WriteBin()

```
template<typename T >
void FullMatrix< T >::WriteBin (
            std::string fname )
```

Function to write the matrix content to a binary file See format at documentation of JMatrix::WriteBin

Parameters

in	*fname*	The name of the file to write

5.1.3.13 WriteCsv()

```
template<typename T >
void FullMatrix< T >::WriteCsv (
            std::string fname,
            char csep = ',',
            bool withquotes = false )
```

Function to write the matrix content to a CSV file

Parameters

in	*fname*	The name of the file to write
in	*csep*	The separator character between fields (default: , (comma))
in	*withquotes*	Boolean value to indicate if field names in .csv must be written surrounded by quotes.

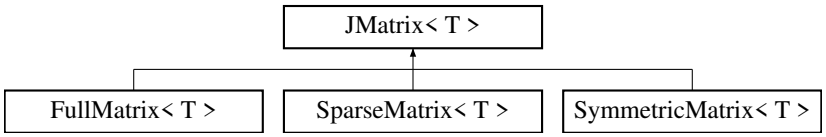
The documentation for this class was generated from the following files:

- src/headers/fullmatrix.h
- src/library/fullmatrix.cpp

5.2 JMatrix< T > Class Template Reference

```
#include <jmatrix.h>
```

Inheritance diagram for JMatrix< T >:



Public Member Functions

- JMatrix (unsigned char mtype)
- JMatrix (unsigned char mtype, indextype nrows, indextype ncols)
- JMatrix (std::string fname, unsigned char mtype)
- JMatrix (std::string fname, unsigned char mtype, unsigned char vtype, char csep)
- JMatrix (const JMatrix< T > &other)
- JMatrix< T > & operator= (const JMatrix< T > &other)
- JMatrix< T > & operator!= (const JMatrix< T > &other)
- indextype GetNRows ()
- indextype GetNCols ()
- void Resize (indextype newnr, indextype newnc)
- std::vector< std::string > GetColNames ()
- std::vector< std::string > GetRowNames ()

- void SetColNames (std::vector< std::string > cnames)
- void SetRowNames (std::vector< std::string > rnames)
- std::string GetComment ()
- void SetComment (std::string cm)
- void WriteCsv (std::string fname, char csep=',', bool withquotes=false)
- void WriteBin (std::string fname, unsigned char mtype)

Protected Member Functions

- unsigned char **TypeNameToId** ()
- bool **ProcessDataLineCsv** (std::string line, char csep, T *rowofdata)
- int **ReadMetadata** ()
- void **WriteMetadata** ()
- void **SetDataType** (unsigned char dtype)

Protected Attributes

- indextype **nr**
- indextype **nc**
- unsigned char **jctype**
- std::ifstream **ifile**
- std::ofstream **ofile**
- std::vector< std::string > **rownames**
- std::vector< std::string > **colnames**
- char **comment** [COMMENT_SIZE]

5.2.1 Detailed Description

template<typename T>
class JMatrix< T >

@JMatrix Wrapper class for all types of matrices. It is meant to hold some basic operations common to all of them Even the instances of this class may now hold real data (the metadata, row and column names), they don't do until an "authentic" matrix is constructed.

5.2.2 Constructor & Destructor Documentation

5.2.2.1 JMatrix() [1/5]

```
template<typename T >
JMatrix< T >::JMatrix (
            unsigned char mtype )
```

Default constructor

Parameters

in	*mtype*	The matrix type (see constants at jmatrix.h)

5.2.2.2 JMatrix() [2/5]

```
template<typename T >
JMatrix< T >::JMatrix (
            unsigned char mtype,
            indextype nrows,
            indextype ncols )
```

Constructor with number of rows and columns

Parameters

in	*mtype*	The matrix type (see constants at jmatrix.h)
in	*nrows*	Number of rows
in	*ncols*	Number of columns

5.2.2.3 JMatrix() [3/5]

```
template<typename T >
JMatrix< T >::JMatrix (
            std::string fname,
            unsigned char mtype )
```

Constructor to fill the matrix content from a binary file

Binary file header as explained in the documentation to WriteBin

TODO PRELIMINARY VERSION. ASSUMES SAME ENDIANESS FOR WRITER AND READER MACHINE

Parameters

in	*fname*	The name of the file to read
in	*mtype*	The matrix type (see constants at jmatrix.h)

5.2.2.4 JMatrix() [4/5]

```
template<typename T >
JMatrix< T >::JMatrix (
```

```
            std::string fname,
            unsigned char mtype,
            unsigned char vtype,
            char csep )
```

Constructor to fill the matrix content from a csv text file

Parameters

in	*fname*	The name of the file to read
in	*mtype*	The matrix type (see constants at jmatrix.h)
in	*vtype*	The data type to be contained in the matrix
in	*csep*	The character expected to be the sepparator

5.2.2.5 JMatrix() [5/5]

```
template<typename T >
JMatrix< T >::JMatrix (
            const JMatrix< T > & other )
```

Copy constructor

Parameters

in	*other*	Reference to the JMatrix to be copied

5.2.3 Member Function Documentation

5.2.3.1 GetColNames()

```
template<typename T >
std::vector< std::string > JMatrix< T >::GetColNames
```

Function to get the matrix column names, if present

Returns

: The vector of strings with the col names. Empty vector if not present

5.2.3.2 GetComment()

```
template<typename T >
std::string JMatrix< T >::GetComment
```

Function to get a string with the matrix comment, if any (or the empty string otherwise).

Returns

: A string with the content of the comment area.

5.2.3.3 GetNCols()

```
template<typename T >
indextype JMatrix< T >::GetNCols ( )  [inline]
```

Function to get number of columns

Returns

Number of columns of the matrix

5.2.3.4 GetNRows()

```
template<typename T >
indextype JMatrix< T >::GetNRows ( )  [inline]
```

Function to get number of rows

Returns

Number of rows of the matrix

5.2.3.5 GetRowNames()

```
template<typename T >
std::vector< std::string > JMatrix< T >::GetRowNames
```

Function to get the matrix row names, if present

Returns

: The vector of strings with the row names. Empty vector if not present

5.2.3.6 operator"!=()

```
template<typename T >
JMatrix< T > & JMatrix< T >::operator!= (
            const JMatrix< T > & other )
```

Transpose-assignment

Parameters

in	*other*	Reference to the JMatrix to be assigned

Returns

Reference to the newly created Matrix, which is the transpose of the passed one

5.2.3.7 operator=()

```
template<typename T >
JMatrix< T > & JMatrix< T >::operator= (
            const JMatrix< T > & other )
```

Assignment operator

Parameters

in	*other*	Reference to the JMatrix to be assigned

Returns

Reference to the newly created JMatrix

5.2.3.8 Resize()

```
template<typename T >
void JMatrix< T >::Resize (
            indextype newnr,
            indextype newnc )
```

Function to resize the matrix WARNING: previous content, if any, IS LOST (TODO: to be reviewed)

Parameters

in	*newnr*	New number of rows
in	*newnc*	New number of cols

5.2.3.9 SetColNames()

```
template<typename T >
```

```
void JMatrix< T >::SetColNames (
            std::vector< std::string > cnames )
```

Function to set the matrix column names.

Parameters

in	*cnames*	The std::vector of strings with the column names.

5.2.3.10 SetComment()

```
template<typename T >
void JMatrix< T >::SetComment (
            std::string cm )
```

Function to set the matrix comment

Parameters

in	*cm*	A std::string with the comment to be stored.

5.2.3.11 SetRowNames()

```
template<typename T >
void JMatrix< T >::SetRowNames (
            std::vector< std::string > rnames )
```

Function to set the matrix row names.

Parameters

in	*rnames*	The std::vector of strings with the row names.

5.2.3.12 WriteBin()

```
template<typename T >
void JMatrix< T >::WriteBin (
            std::string fname,
            unsigned char mtype )
```

Function to write the matrix content to a binary file

The binary header will contain:

- unsigned char t: matrix type (normal, sparse, symmetric). Other types might be added later
- unsigned char dt: the data type of the matrix elements, which is one of unsigned/signed char, unsigned signed short, unsigned/signed long, unsigned/signed longlong, float, double, longdouble in its lower 4 bits and the endianess in its upper 4 bits (big or little).
- indextype nr: number of rows
- indextype nc: number of columns
- unsigned char mdinfo: information on the presence/absence of metadata, currently row and/or column names and comment.

This means that the size of the header is 2+2∗sizeof(indextype)+1+empty_space. We have fixed the empty space so that total header size be 128 bytes.

After the header the binary file contains the matrix raw data, by rows. Internal representation is different according to the matrix type (full, sparse or symmetric).

- After the raw data come the row and/or column names (if any)
- Then, the row names, as character arrays separated by null character (0x00)
- Then, a separation mark (the succesion of bytes 0xFF 0x45 0x42 0xFF). 0x45 and 0x42 are ASCII characters EB, for End Block
- Then, the column names, as character arrays separated by null character (0x00)
- Then, a separation mark, as above
- After the row/column names comes the comment (if any)
- Then, a separation mark, as above

HEADER AND DATA ARE ALWAYS WRITTEN IN THE ENDIANESS OF THE MACHINE WHICH EXECUTES THIS CODE Nevertheless, other machines will know about it, since it is declared in the first byte of the written file.

Parameters

in	*string*	fname: The name of the file to write
in	*unsigned*	char mtype: The type identifier

5.2.3.13 WriteCsv()

```
template<typename T >
void JMatrix< T >::WriteCsv (
            std::string fname,
            char csep = ',',
            bool withquotes = false )
```

Function to write the matrix content to a CSV file

Parameters

in	*fname*	The name of the file to write
in	*csep*	The separator character between fields (default: , (comma))
in	*withquotes*	boolean value to indicate if field names in .csv must be written surrounded by quotes or not

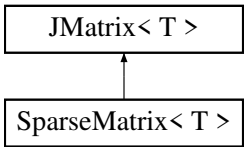
The documentation for this class was generated from the following files:

- src/headers/jmatrix.h
- src/library/jmatrix.cpp

5.3 SparseMatrix< T > Class Template Reference

`#include <sparsematrix.h>`

Inheritance diagram for SparseMatrix< T >:



Public Member Functions

- SparseMatrix ()
- SparseMatrix (indextype nrows, indextype ncols)
- SparseMatrix (std::string fname)
- SparseMatrix (std::string fname, TrMark)
- SparseMatrix (std::string fname, unsigned char vtype, char csep)
- void Resize (indextype newnr, indextype newnc)
- SparseMatrix (const SparseMatrix &other)
- ~SparseMatrix ()
- SparseMatrix< T > & operator= (const SparseMatrix< T > &other)
- SparseMatrix< T > & operator!= (const SparseMatrix< T > &other)
- T Get (indextype r, indextype c) const
- void Set (indextype r, indextype c, T v)
- void SetRow (indextype r, std::vector< indextype > vc, std::vector< T > v)
- void GetRow (indextype r, T *v)
- void GetSparseRow (indextype r, unsigned char *m, unsigned char s, T *v)
- void GetMarksOfSparseRow (indextype r, unsigned char *m, unsigned char s)
- void SelfRowNorm (std::string ctype)
- void SelfColNorm (std::string ctype)
- void WriteCsv (std::string fname, char csep=',', bool withquotes=false)
- void WriteBin (std::string fname)
- float GetUsedMemoryMB ()

Additional Inherited Members

5.3.1 Detailed Description

template<typename T>
class SparseMatrix< T >

@SparseMatrix Class to hold arbitrarily big sparse matrices. Elements are stored with column index + value in a vector associated to each row.
Time to set and get elements are of order O(log_2(Nc)) being Nc the number of columns.
Space is O(N*(sizeof(element)+sizeof(index)), being element the type of the matrix contents and index that of the matrix index (which is currently unsigned int).

5.3.2 Constructor & Destructor Documentation

5.3.2.1 SparseMatrix() [1/6]

```
template<typename T >
SparseMatrix< T >::SparseMatrix
```

Default constructor

5.3.2.2 SparseMatrix() [2/6]

```
template<typename T >
SparseMatrix< T >::SparseMatrix (
            indextype nrows,
            indextype ncols )
```

Constructor with number of rows and columns

Parameters

in	*nrows*	Number of rows
in	*ncols*	Number of columns

5.3.2.3 SparseMatrix() [3/6]

```
template<typename T >
SparseMatrix< T >::SparseMatrix (
            std::string fname )
```

Constructor to fill the matrix content from a binary file

Binary file header as explained in the documetation to WriteBin

PRELIMINARY VERSION. ASSUMES SAME ENDIANESS FOR WRITER AND READER MACHINE

Parameters

in	*fname*	The name of the file to read

5.3.2.4 SparseMatrix() [4/6]

```
template<typename T >
SparseMatrix< T >::SparseMatrix (
```

```
            std::string fname,
            TrMark  )
```

Constructor to fill the matrix content from the transpose of the matrix contained in a binary file

Binary file header as explained in the documentation to WriteBin

PRELIMINARY VERSION. ASSUMES SAME ENDIANESS FOR WRITER AND READER MACHINE

Parameters

in	*fname*	The name of the file to read

5.3.2.5 SparseMatrix() [5/6]

```
template<typename T >
SparseMatrix< T >::SparseMatrix (
            std::string fname,
            unsigned char vtype,
            char csep )
```

Constructor to fill the matrix content from a csv file First line is supposed to have the field names and is ignored. First column of each line is supposed to have the field name, and is ignored, too. The passed character is the expected field sepparator (usually, comma or tab)

Parameters

in	*fname*	The name of the csv file to read
in	*vtype*	The data type to be stored
in	*csep*	The character used as field sepparator

5.3.2.6 SparseMatrix() [6/6]

```
template<typename T >
SparseMatrix< T >::SparseMatrix (
            const SparseMatrix< T > & other )
```

Copy constructor

Parameters

in	*other*	Reference to the SparseMatrix to be copied

5.3.2.7 ~SparseMatrix()

```
template<typename T >
SparseMatrix< T >::~SparseMatrix
```

Destructor

5.3.3 Member Function Documentation

5.3.3.1 Get()

```
template<typename T >
T SparseMatrix< T >::Get (
            indextype r,
            indextype c ) const
```

Function to get acess to an element

Parameters

in	*r*	The row to access
in	*c*	The columns to access

Returns

value at (r,c) of matrix of type T

5.3.3.2 GetMarksOfSparseRow()

```
template<typename T >
void SparseMatrix< T >::GetMarksOfSparseRow (
            indextype r,
            unsigned char * m,
            unsigned char s )
```

Function to get from a sparse row a pointer to an array of marks signalling where the non-zero elements are. The array of marks will be changed OR'ing the passed mark to each place where there is a value. The pointer to the marks is not returned since that way it do not need to be booked. It is passed as a parameter and is supposed to be properly allocated.

Parameters

in	*r*	The row to get
out	**m*	Pointer to the marks
in	*s*	The value to be OR'ed to each place at the mark array

5.3.3.3 GetRow()

```
template<typename T >
void SparseMatrix< T >::GetRow (
            indextype r,
            T * v )
```

Function to get a row as a pointer to the content type. Row is not a sparse but a full vector with zeros when needed. The pointer is not returned since that way it does not need to be booked. The pointer to hold result is passed as parameter and it is supposed to be properly allocated.

Parameters

in	*r*	The row to get
out	*∗v*	Pointer to the result of type ∗T

5.3.3.4 GetSparseRow()

```
template<typename T >
void SparseMatrix< T >::GetSparseRow (
            indextype r,
            unsigned char * m,
            unsigned char s,
            T * v )
```

Function to get a sparse row as a pointer to the content type plus a pointer to an array of marks.
The content will have the values. The array of marks will be changed OR'ing the passed mark to each place where there is a value. The pointers to the values and marks are not returned since that way they do not need to be booked. They are passed as parameters and both are supposed to be properly allocated.
This strange way of storing data has been choosen because it will be specially suitable to calculate distance between sparse vectors

Parameters

in	*r*	The row to get
out	*∗v*	Pointer to the values
out	*∗m*	Pointer to the marks
in	*s*	The value to be OR'ed to each place at the mark array

5.3.3.5 GetUsedMemoryMB()

```
template<typename T >
float SparseMatrix< T >::GetUsedMemoryMB
```

Function to get memory in MB used by this sparse matrix (including values and additional indexes)

Returns

The amount of memory in MB

5.3.3.6 operator"!=()

```
template<typename T >
SparseMatrix< T > & SparseMatrix< T >::operator!= (
            const SparseMatrix< T > & other )
```

Transpose-assignment

Parameters

in	*other*	Reference to the SparseMatrix to be assigned

Returns

Reference to the newly created SparseMatrix, which is the transpose of the passed one

5.3.3.7 operator=()

```
template<typename T >
SparseMatrix< T > & SparseMatrix< T >::operator= (
            const SparseMatrix< T > & other )
```

Assignment operator

Parameters

in	*other*	Reference to the SparseMatrix to be assigned

Returns

Reference to the newly created SparseMatrix

5.3.3.8 Resize()

```
template<typename T >
void SparseMatrix< T >::Resize (
            indextype newnr,
            indextype newnc )
```

Function to resize the matrix
WARNING: previous content, if any, IS LOST (to be reviewed)

Parameters

in	*newnr*	New number of rows
in	*newnc*	New number of cols

5.3.3.9 SelfColNorm()

```
template<typename T >
void SparseMatrix< T >::SelfColNorm (
            std::string ctype )
```

Function to alter the internal values of the matrix so that each column is normalized according to the requested normalization type The purpose of this function can be achieved with a loop using Set and Get, but using the internal structure makes the task much faster
Normally, this function will not be used outside the context of bioinformatics where these normalizations are standard

Parameters

in	*ctype*	The requested type of normalization: rawn, log1 or log1n

5.3.3.10 SelfRowNorm()

```
template<typename T >
void SparseMatrix< T >::SelfRowNorm (
            std::string ctype )
```

Function to alter the internal values of the matrix so that each row is normalized according to the requested normalization type The purpose of this function can be achieved with a loop using Set and Get, but using the internal structure makes the task much faster
Normally, this function will not be used outside the context of bioinformatics where these normalizations are standard

Parameters

in	*ctype*	The requested type of normalization: rawn, log1 or log1n

5.3.3.11 Set()

```
template<typename T >
void SparseMatrix< T >::Set (
            indextype r,
            indextype c,
            T v )
```

Function to set an element

Parameters

in	*r*	The row of the element to be set
in	*c*	The column of the element to be set
in	*v*	The value to be set, of type T

5.3.3.12 SetRow()

```
template<typename T >
void SparseMatrix< T >::SetRow (
            indextype r,
            std::vector< indextype > vc,
            std::vector< T > v )
```

Function to set a row (as two vectors of locations and values)

Parameters

in	*r*	The row to be set
in	*vc*	The vector with the columns to be set
in	*v*	The vector with the corresponding values to be set. Must be the same length as vc

5.3.3.13 WriteBin()

```
template<typename T >
void SparseMatrix< T >::WriteBin (
            std::string fname )
```

Function to write the matrix content into a binary file
For the header format, see the documentation of JMatrix

After the header comes the content as raw data, by rows, with this content for each row:

- indextype ncr: number of non-zero entries of this row
- ncr values of indextype with the numbers of the columns of this row occupied by non-zero entries
- ncr elements of the current value type (the values of all non-zero entries of this row).

Parameters

in	*fname*	The name of the file to write

5.3.3.14 WriteCsv()

```
template<typename T >
void SparseMatrix< T >::WriteCsv (
            std::string fname,
            char csep = ',',
            bool withquotes = false )
```

Function to write the matrix content to a CSV file

Parameters

in	*fname*	The name of the file to write
in	*csep*	The separator character between fields (default: , (comma))
in	*withquotes*	Boolean value to indicate if field names in .csv must be written surrounded by quotes.

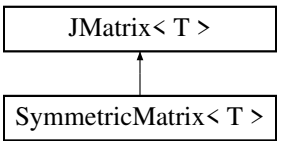
The documentation for this class was generated from the following files:

- src/headers/sparsematrix.h
- src/library/sparsematrix.cpp

5.4 SymmetricMatrix< T > Class Template Reference

`#include <symmetricmatrix.h>`

Inheritance diagram for SymmetricMatrix< T >:



Public Member Functions

- SymmetricMatrix ()
- SymmetricMatrix (indextype nrows)
- SymmetricMatrix (indextype nrows, bool warn)
- SymmetricMatrix (std::string fname)
- SymmetricMatrix (std::string fname, bool warn)
- void Resize (indextype newnr)
- SymmetricMatrix (const SymmetricMatrix< T > &other)
- ~SymmetricMatrix ()
- SymmetricMatrix< T > & operator= (const SymmetricMatrix< T > &other)
- bool TestDistDisMat ()
- T Get (indextype r, indextype c)
- void Set (indextype r, indextype c, T v)
- T GetRowSum (indextype r)
- void WriteCsv (std::string fname, char csep=',', bool withquotes=false)
- void WriteBin (std::string fname)
- float GetUsedMemoryMB ()

Additional Inherited Members

5.4.1 Detailed Description

template<typename T>
class SymmetricMatrix< T >

@SymmetricMatrix Class to hold arbitrarily big symmetric square matrices. For a matrix of size NxN, only Nx(N+1)/2 elements are stored.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 SymmetricMatrix() [1/6]

```
template<typename T >
SymmetricMatrix< T >::SymmetricMatrix
```

Default constructor

5.4.2.2 SymmetricMatrix() [2/6]

```
template<typename T >
SymmetricMatrix< T >::SymmetricMatrix (
            indextype nrows )
```

Constructor with number of rows/columns (the same, it is square)

Parameters

in	*nrows*	Number of rows

5.4.2.3 SymmetricMatrix() [3/6]

```
template<typename T >
SymmetricMatrix< T >::SymmetricMatrix (
            indextype nrows,
            bool warn )
```

Constructor with number of rows/columns (the same, it is square) giving memory warnings

Parameters

in	*nrows*	Number of rows

Parameters

in	*warn*	Boolean value to give memory warnings The idea is to use this form of constructor with warn=true, which internally calls MemoryWarnings, if one suspects the matrix to be constructed might provoke memory problems.

5.4.2.4 SymmetricMatrix() [4/6]

```
template<typename T >
SymmetricMatrix< T >::SymmetricMatrix (
            std::string fname )
```

Constructor to fill the matrix content from a binary file
Binary file header as explained in the documetation to WriteBin

PRELIMINARY VERSION. ASSUMES SAME ENDIANESS FOR WRITER AND READER MACHINE

Parameters

in	*fname*	The name of the file to read

5.4.2.5 SymmetricMatrix() [5/6]

```
template<typename T >
SymmetricMatrix< T >::SymmetricMatrix (
            std::string fname,
            bool warn )
```

Constructor to fill the matrix content from a binary file with warnings
Binary file header as explained in the documetation to WriteBin

PRELIMINARY VERSION. ASSUMES SAME ENDIANESS FOR WRITER AND READER MACHINE

Parameters

in	*fname*	The name of the file to read
in	*warn*	Boolean value to give memory warnings

5.4.2.6 SymmetricMatrix() [6/6]

```
template<typename T >
SymmetricMatrix< T >::SymmetricMatrix (
            const SymmetricMatrix< T > & other )
```

Copy constructor

Parameters

in	*other*	Reference to the SymmetricMatrix to be copied

5.4.2.7 ~SymmetricMatrix()

```
template<typename T >
SymmetricMatrix< T >::~SymmetricMatrix
```

Destructor

5.4.3 Member Function Documentation

5.4.3.1 Get()

```
template<typename T >
T SymmetricMatrix< T >::Get (
            indextype r,
            indextype c )
```

Function to get acess to an element

Parameters

in	*r*	The row to access
in	*c*	The columns to access

Returns

value at (r,c) of matrix, of type T

5.4.3.2 GetRowSum()

```
template<typename T >
T SymmetricMatrix< T >::GetRowSum (
            indextype r )
```

Function to get the sum of a row (used frequently by PAM)

Parameters

in	*r*	The row whose sum we want

Returns

The sum of all columns of row r, of type T

5.4.3.3 GetUsedMemoryMB()

```
template<typename T >
float SymmetricMatrix< T >::GetUsedMemoryMB
```

Function to get memory in MB used by this symmetric matrix

Returns

The amount of memory in MB

5.4.3.4 operator=()

```
template<typename T >
SymmetricMatrix< T > & SymmetricMatrix< T >::operator= (
            const SymmetricMatrix< T > & other )
```

Assignment operator

Parameters

in	*other*	Reference to the SymmetricMatrix to be assigned

Returns

Reference to the newly created SymmetricMatrix

5.4.3.5 Resize()

```
template<typename T >
void SymmetricMatrix< T >::Resize (
            indextype newnr )
```

Function to resize the matrix
WARNING: previous content, if any, IS LOST (to be reviewed)

Parameters

in	*newnr*	New number of rows (and columns)

5.4.3.6 Set()

```
template<typename T >
void SymmetricMatrix< T >::Set (
            indextype r,
            indextype c,
            T v )
```

Function to set an element

Parameters

in	*r*	The row to access
in	*c*	The columns to access
in	*T*	The value to be set, of type T

5.4.3.7 TestDistDisMat()

```
template<typename T >
bool SymmetricMatrix< T >::TestDistDisMat
```

Test of correctness
This is meant to test if the symmetric matrix is a distance or dissimilarity matrix. It checks that all elements in the main diagonal are 0 and all outside the main diagonal are stricty possitive

Returns

true if the matrix can be a distance or dissimilarity matrix. false otherwise.

5.4.3.8 WriteBin()

```
template<typename T >
void SymmetricMatrix< T >::WriteBin (
            std::string fname )
```

Function to write the matrix content to a binary file
See format at documentation of JMatrix::WriteBin

Parameters

in	*fname*	The name of the file to write

5.4.3.9 WriteCsv()

```
template<typename T >
void SymmetricMatrix< T >::WriteCsv (
            std::string fname,
            char csep = ',',
            bool withquotes = false )
```

Function to write the matrix content to a CSV file

Parameters

in	*fname*	The name of the file to write
in	*csep*	The separator character between fields (default: , (comma))
in	*withquotes*	Boolean value to indicate if field names in .csv must be written surrounded by quotes.

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

- src/headers/symmetricmatrix.h
- src/library/symmetricmatrix.cpp

Chapter 6

File Documentation

6.1 src/examples/jmat.cpp File Reference

```
#include "../headers/debugpar.h"
#include "../headers/jmatrix.h"
#include "../headers/apitocommands.h"
```

Functions

- int main (int argc, char ∗argv[ ])

6.1.1 Detailed Description

jmat

See program use in the documention to main() below

NOTE: The includes in this source file are for compilation of this program as an example together with the library,
before the library itself is installed. Once you have installed the library (assuming headers are in
/usr/local/include, lib is in /usr/local/lib or in other place included in your compiler search path)
you should substitute this by

#include <jmatrixlib/debugpar.h> etc...

and compile with something like

g++ -Wall jmat.cpp -o jmat -ljmatrix

6.1.2 Function Documentation

6.1.2.1 main()

```
int main (
            int argc,
            char * argv[] )
```

jmat

A program to manipulate matrices created with the jmatrix library and written to the disk as binary files.

The program must be called as

```
jmat command matrix_file other_options -o out_matrix file
```

where command is one of a predefined list (see below) which is followed by the matrix to be manipulated, other relevant options for the particular command and (optionally) the -o option with the result of the command.
If -o option is not given, the result is dumped to the console in ASCII
other_options are options dependent on the command (call 'jmat any_command' for specific information)
Also, remember that if this program is called as **jmatd** (symbolic link to jmat) you will get debugging messages in the console.

Possible commands are:

```
jmat info matrix_file [-o info_file.txt]
```

Dumps in ASCII to console or to info_file.txt information about the jmatrix in file matrix_file.

```
jmat rownum matrix_file n -o res_file
```

Creates a jmatrix file with the n-th row of the jmatrix in file matrix_file (index n from 0).

```
jmat rownums matrix_file n1,n2,... -o res_file
```

Creates a jmatrix file with the rows in the comma-sepparated list of the jmatrix in file matrix_file (indices n1,n2,... from 0).

```
jmat rowname matrix_file rname -o res_file
```

Creates a jmatrix file with the row named 'rname' of the jmatrix in file matrix_file (index n from 0).

```
jmat rownames matrix_file rn1,rn2,... -o res_file
```

Creates a jmatrix file, with the rows named 'rn1','rn2'... in the comma-sepparated list of the jmatrix n file matrix_file.

```
jmat colnum matrix_file n -o res_file
```

Creates a jmatrix file, with the n-th column of the jmatrix in file matrix_file (index n from 0).

```
jmat colnums matrix_file n1,n2,... -o res_file
```

Creates a jmatrix file, with the columns in the comma-sepparated list of the jmatrix in file matrix_file (indices form 0).

```
jmat colname matrix_file cname -o res_file
```

Creates a jmatrix file, with the column named 'cname' of the jmatrix in file matrix_file

```
jmat colnames matrix_file cn1,cn2,... -o res_file
```

Creates a jmatrix file, with the columns named 'cn1','cn2'... in the comma-sepparated list of the jmatrix in file matrix_file.

```
jmat subdiag symmetric_matrix_file -o res_file
```

Creates a jmatrix file with the lower-triangular part (not including the diagonal) of the symmetric matrix in the input.

```
jmat getrnames matrix_file -o res_file.txt
```

Dumps to ASCII file res_file.txt the names of the rows in the input matrix.

```
jmat getcnames matrix_file -o res_file.txt
```

Dumps to ASCII file res_file.txt the names of the columns in the input matrix.

```
jmat setrnames matrix_file rownames_file.txt -o res_file
```

Creates a copy of the original jmatrix setting or changing the row names to those given in file rownames_file.txt, which must have as many lines as rows in input matrix.

```
jmat setcnames matrix_file colnames_file.txt -o res_file
```

Creates a copy of the original jmatrix setting or changing the column names to those given in file colnames_file.txt, which must have as many lines as columns in input matrix.

```
jmat setrcnames matrix_file rownames_file.txt colnames_file.txt -o res_file
```

Creates a copy of the original jmatrix setting or changing the row and column names to those given in files rownames_file.txt and colnames_file.txt respectively, which must have as many lines as rows/columns in input matrix.

```
jmat csvdump matrix_file format -o res_file.csv
```

Dumps the content of the matrix in the input file to the .csv output file. Format must be one of these strings:
cn Comma as separator, row/column names not surrounded by double quotes.
cq Comma as separator, row/column names surrounded by double quotes.
tn Tab as separator, row/column names not surrounded by double quotes.
tq Tab as separator, row/column names surrounded by double quotes.

```
jmat csvread input_file.csv sepchar mtype valtype -o res_file
```

Reads the input file, which must be a csv file, and creates a binary jmatrix file with its content.
sepchar must be c or t to indicate that the expected field separator will be a comma or a tab, respectively.
mtype must be one of the strings 'full' or 'sparse'. Symmetric matrices cannot be read as such from a CSV file.
valtype must be one of the strings 'u8','s8','u16','s16','u32','s32','u64','s64','f','d' or 'ld'.
These stands for unsigned/signed integers of 8,16,32 or 64 bits, float, double, or long double datatypes, respectively.

6.2 src/headers/apitocommands.h File Reference

```
#include <string>
#include <vector>
#include "indextype.h"
```

Functions

- void JMatInfo (std::string iname, std::string oname)
- void JGetNumRow (std::string iname, std::string oname, indextype numrow)
- void JGetNumCol (std::string iname, std::string oname, indextype numcol)
- void JGetNameRow (std::string iname, std::string oname, std::string namerow)
- void JGetNameCol (std::string iname, std::string oname, std::string namecol)
- void JGetNumsRow (std::string iname, std::string oname, std::vector< indextype > lrows)
- void JGetNumsCol (std::string iname, std::string oname, std::vector< indextype > lcols)
- void JGetNamesRow (std::string iname, std::string oname, std::vector< std::string > lrows)
- void JGetNamesCol (std::string iname, std::string oname, std::vector< std::string > lcols)
- void JGetSubDiag (std::string iname, std::string oname)
- void JGetRowNames (std::string iname, std::string oname)
- void JGetColNames (std::string iname, std::string oname)
- void JSetRowNames (std::string iname, std::string oname, std::vector< std::string > rnames)
- void JSetColNames (std::string iname, std::string oname, std::vector< std::string > cnames)
- void JSetRowColNames (std::string iname, std::string oname, std::vector< std::string > rnames, std::vector< std::string > cnames)
- void JCsvDump (std::string iname, std::string oname, char sep, bool with_quotes)
- void JCsvToJMat (std::string iname, std::string oname, char sep, unsigned char mtype, unsigned char ctype)

6.2.1 Function Documentation

6.2.1.1 JCsvDump()

```
void JCsvDump (
            std::string iname,
            std::string oname,
            char sep,
            bool with_quotes )
```

Function to dump the JMatrix contained in a binary file to as ASCII csv file

Parameters

in	*iname*	Name of the JMatrix binary file with the matrix
in	*oname*	Name of the csv file to generat
in	*char*	The character to be used as sepparator
in	*withquotes*	Boolean value to indicate if field names in .csv must be written surrounded by quotes.

6.2.1.2 JCsvToJMat()

```
void JCsvToJMat (
            std::string iname,
            std::string oname,
            char sep,
            unsigned char mtype,
            unsigned char ctype )
```

Function to generate a binary JMatrix file from an ASCII csv file
First line of csv is supposed to contain the column names, starting with an empty field
First field of each line from the second line and on is supposed to contain the name of such row

Parameters

in	*iname*	CSV file with the data
in	*oname*	Name of the binary file to contain the created JMatrix
in	*sep*	The character that csv file uses as field sepparator
in	*mtype*	The type of the JMatrix. Possible values: 'full', 'sparse' or 'symmetric'
in	*ctype*	The data type to store the read values (value type of the JMatrix). Possible values: 'u8','s8','u16','s16','u32','s32','u64','s64','f','d' or 'ld'

6.2.1.3 JGetColNames()

```
void JGetColNames (
            std::string iname,
            std::string oname )
```

Function to write to an ASCII text file the names of the columns of the JMatrix stored in a binary file
The matrix is not loaded into memory

Parameters

in	*iname*	Name of the JMatrix binary file
in	*oname*	Name of the ASCII file to contain the column names, one at each line

6.2.1.4 JGetNameCol()

```
void JGetNameCol (
            std::string iname,
            std::string oname,
            std::string namecol )
```

Function to get a column by name and write it as a JMatrix in a binary file, assuming the input matrix has column names and such name exists

Parameters

in	*iname*	Name of the JMatrix binary file
in	*oname*	Name of the binary file to write the row contents
in	*namecol*	The name of the column we want to extract

6.2.1.5 JGetNameRow()

```
void JGetNameRow (
            std::string iname,
            std::string oname,
            std::string namerow )
```

Function to get a row by name and write it as a JMatrix in a binary file, assuming the input matrix has row names and such name exists

Parameters

in	*iname*	Name of the JMatrix binary file
in	*oname*	Name of the binary file to write the row contents
in	*namerow*	The name of the row we want to extract

6.2.1.6 JGetNamesCol()

```
void JGetNamesCol (
            std::string iname,
            std::string oname,
            std::vector< std::string > lcols )
```

Function to get several columns by their names and write them as a JMatrix in a binary file

Parameters

in	*iname*	Name of the JMatrix binary file
in	*oname*	Name of the binary file to write the rows as a matrix
in	*lcols*	A vector of strings with the names of the columns we want to extract

6.2.1.7 JGetNamesRow()

```
void JGetNamesRow (
            std::string iname,
```

```
            std::string oname,
            std::vector< std::string > lrows )
```

Function to get several rows by their names and write them as a JMatrix in a binary file

Parameters

in	*iname*	Name of the JMatrix binary file
in	*oname*	Name of the binary file to write the rows as a matrix
in	*lrows*	A vector of strings with the names of the rows we want to extract

6.2.1.8 JGetNumCol()

```
void JGetNumCol (
            std::string iname,
            std::string oname,
            indextype numcol )
```

Function to get a column by number and write it as a JMatrix in a binary file

Parameters

in	*iname*	Name of the JMatrix binary file
in	*oname*	Name of the binary file to write the column contents
in	*numrow*	The number of the column (0-based index) we want to extract

6.2.1.9 JGetNumRow()

```
void JGetNumRow (
            std::string iname,
            std::string oname,
            indextype numrow )
```

Function to get a row by number and write it as a JMatrix in a binary file

Parameters

in	*iname*	Name of the JMatrix binary file
in	*oname*	Name of the binary file to write the row contents
in	*numrow*	The number of the row (0-based index) we want to extract

6.2.1.10 JGetNumsCol()

```
void JGetNumsCol (
            std::string iname,
            std::string oname,
            std::vector< indextype > lcols )
```

Function to get several columns by number and write them as a JMatrix in a binary file

Parameters

in	*iname*	Name of the JMatrix binary file
in	*oname*	Name of the binary file to write the columns as a matrix
in	*lcols*	A vector with the numbers of the columns (0-based index) we want to extract

6.2.1.11 JGetNumsRow()

```
void JGetNumsRow (
            std::string iname,
            std::string oname,
            std::vector< indextype > lrows )
```

Function to get several rows by number and write them as a JMatrix in a binary file

Parameters

in	*iname*	Name of the JMatrix binary file
in	*oname*	Name of the binary file to write the rows as a matrix
in	*lrows*	A vector with the numbers of the rows (0-based index) we want to extract

6.2.1.12 JGetRowNames()

```
void JGetRowNames (
            std::string iname,
            std::string oname )
```

Function to write to an ASCII text file the names of the rows of the JMatrix stored in a binary file
The matrix is not loaded into memory

Parameters

in	*iname*	Name of the JMatrix binary file
in	*oname*	Name of the ASCII file to contain the row names, one at each line

6.2.1.13 JGetSubDiag()

```
void JGetSubDiag (
            std::string iname,
            std::string oname )
```

Function to get the subdiagonal of a SymmetricMatrix of size (n x n) stored in a binary file and write them as a vector of one row and n x (n-1)/2 columns (rows under the main diagonal, without the diagonal itself) stored in row-major order

Parameters

in	*iname*	Name of the SymmatricMatrix binary file
in	*oname*	Name of the binary file to write the subdiagonal elements as a 1-row matrix

6.2.1.14 JMatInfo()

```
void JMatInfo (
            std::string iname,
            std::string oname )
```

Function to get information about the JMatrix stored in a binary file and store such information in a text file

Parameters

in	*iname*	Name of the JMatrix binary file
in	*oname*	Name of the text file to write the information

6.2.1.15 JSetColNames()

```
void JSetColNames (
            std::string iname,
            std::string oname,
            std::vector< std::string > cnames )
```

Function to create a copy of the original jmatrix setting or changing the column names to those given in a vector of strings, which must have as many elements as columns in the input matrix

Parameters

in	*iname*	Name of the JMatrix binary file with the original matrix
in	*oname*	Name of the JMatrix binary file with the copy matrix with new column names
in	*cnames*	Vector of strings with the new column names

6.2.1.16 JSetRowColNames()

```
void JSetRowColNames (
            std::string iname,
            std::string oname,
            std::vector< std::string > rnames,
            std::vector< std::string > cnames )
```

Function to create a copy of the original jmatrix setting or changing the column and row names to those given in vectors of strings, which must have as many elements as rows and columns respectively in the input matrix

Parameters

in	*iname*	Name of the JMatrix binary file with the original matrix
in	*oname*	Name of the JMatrix binary file with the copy matrix with new column names
in	*rnames*	Vector of strings with the new row names
in	*cnames*	Vector of strings with the new column names

6.2.1.17 JSetRowNames()

```
void JSetRowNames (
            std::string iname,
            std::string oname,
            std::vector< std::string > rnames )
```

Function to create a copy of the original jmatrix setting or changing the row names to those given in a vector of strings, which must have as many elements as rows in the input matrix

Parameters

in	*iname*	Name of the JMatrix binary file with the original matrix
in	*oname*	Name of the JMatrix binary file with the copy matrix with new row names
in	*rnames*	Vector of strings with the new row names

6.3 src/headers/debugpar.h File Reference

```
#include <iostream>
#include <string>
```

Functions

- void JMatrixSetDebug (bool deb)
- void JMatrixStop (std::string errortext)
- void JMatrixWarning (std::string warntext)

Variables

- const unsigned char NODEBUG =0x0
- const unsigned char DEBJM =0x01

6.3.1 Function Documentation

6.3.1.1 JMatrixSetDebug()

```
void JMatrixSetDebug (
            bool deb )
```

Sets the debug state to get messages or not

Parameters

in	*deb*	true to get messages, false to suppress them. Default state is false.

6.3.1.2 JMatrixStop()

```
void JMatrixStop (
            std::string errortext )
```

Sends an error message to the console and stops the program that is using the library

Parameters

in	*errortext*	The text of the message to be shown. It will appear after a standard message saying that is comes from this library.

6.3.1.3 JMatrixWarning()

```
void JMatrixWarning (
            std::string warntext )
```

Sends a warning message to the console and goes on with the program that is using the library

Parameters

in	*errortext*	The text of the message to be shown. It will appear after a standard message saying that is comes from this library.

6.3.2 Variable Documentation

6.3.2.1 DEBJM

```
const unsigned char DEBJM =0x01
```

These are constants to allow selective debug by library. Each library will print messages or not using a test with logical AND between its particular constant and the DEB global variable. This allows the use of the system either in each separate package or in the global one

6.3.2.2 NODEBUG

```
const unsigned char NODEBUG =0x0
```

These are constants to allow selective debug by library. Each library will print messages or not using a test with logical AND between its particular constant and the DEB global variable. This allows the use of the system either in each separate package or in the global one

6.4 src/headers/fullmatrix.h File Reference

```
#include "jmatrix.h"
#include "memhelper.h"
```

Classes

- class FullMatrix< T >

6.5 src/headers/indextype.h File Reference

Typedefs

- typedef unsigned int indextype

6.5.1 Typedef Documentation

6.5.1.1 indextype

```
unsigned int indextype
```

This is the type of the indexes. It is left fixed, but using a typedef allows easy recompilation of the library if we decide otherwise in the future.
It is quite unlikely we need to change indextype, unless, to save some memory and if you know for sure that there will be no more than 65536 individuals, can be defined as unsigned short. But this might provoke hidden problems in other places so, please, think twice before...
Increasing it to 128 bits (unsigned long long) would be possible, but if you need such size of matrices, better don't use this library...

6.6 src/headers/intropage.h File Reference

6.7 src/headers/jmatrix.h File Reference

```
#include <iostream>
#include <string>
#include <cstring>
#include <cmath>
#include <sstream>
#include <fstream>
#include <vector>
#include <algorithm>
#include <type_traits>
#include <sys/stat.h>
#include "debugpar.h"
#include "indextype.h"
#include "matinfo.h"
```

Classes

- class JMatrix< T >

Macros

- #define WITH_CHECKS_MATRIX

Functions

- unsigned char ThisMachineEndianness ()
- unsigned long long GetFileSize (std::string fname)
- void PositionsInFile (std::string fname, unsigned long long *start_of_metadata, unsigned long long *start_of_comment)
- std::string MatrixTypeName (unsigned char matrixtypeident)

  Auxiliary functions to be used for error printing.
- std::string DataTypeName (unsigned char datatypeident)
- std::string MetadataInfo (unsigned char metadatainfo)
- int SizeOfType (unsigned char datatypeident)
- std::string **FixQuotes** (std::string s, bool withquotes)

Variables

- const unsigned short HEADER_SIZE =128

- const unsigned char MTYPENOTYPE =0x0F
- const unsigned char MTYPEFULL =0x00
- const unsigned char MTYPESPARSE =0x01
- const unsigned char MTYPESYMMETRIC =0x02

- const unsigned char NOTYPE =0x0F
- const unsigned char UCTYPE =0x00
- const unsigned char SCTYPE =0x01
- const unsigned char USTYPE =0x02
- const unsigned char SSTYPE =0x03
- const unsigned char UITYPE =0x04
- const unsigned char SITYPE =0x05
- const unsigned char ULTYPE =0x06
- const unsigned char SLTYPE =0x07
- const unsigned char ULLTYPE =0x08
- const unsigned char SLLTYPE =0x09
- const unsigned char FTYPE =0x0A
- const unsigned char DTYPE =0x0B
- const unsigned char LDTYPE =0x0C

- const unsigned char BIGEND =0x00
- const unsigned char LITEND =0xF0

- const int READ_OK =0
- const int ERROR_READING_STRINGS =1
- const int ERROR_READING_ROW_NAMES =2
- const int ERROR_READING_COL_NAMES =3
- const int ERROR_READING_SEP_MARK =4
- const unsigned int MAX_LEN_NAME =1023
- const unsigned int COMMENT_SIZE =1024
- const unsigned char NO_METADATA =0x00
- const unsigned char ROW_NAMES =0x01
- const unsigned char COL_NAMES =0x02
- const unsigned char COMMENT =0x04
- const unsigned int BLOCKSEP_LEN =4
- const unsigned char BLOCK_MARK =0xFF
- const unsigned char BLOCKSEP [BLOCKSEP_LEN] ={BLOCK_MARK,0x45,0x42,BLOCK_MARK}

6.7.1 Macro Definition Documentation

6.7.1.1 WITH_CHECKS_MATRIX

```
#define WITH_CHECKS_MATRIX
```

Constant defined to check access to matrix elements

The simple fact of being defined at compilation time adds a test in each matrix access to be sure we are not out of bound; if we are, a run-time error is raised.

This is obviously safer but at the expense of adding overhead and a slight increment of run time.
Comment these constant if you are absolutely sure your program does not make any Get or Set out of bounds.

6.7.2 Function Documentation

6.7.2.1 DataTypeName()

```
std::string DataTypeName (
            unsigned char datatypeident )
```

Returns the name of the data type whose type (as indentifier) is passed

Parameters

datatypeident.	The data type of the data in the matrix (unsigned char, char,...) as defined by the former constants.

Returns

A human-meaningful string decribing the data type

6.7.2.2 GetFileSize()

```
unsigned long long GetFileSize (
            std::string fname )
```

Returns the file size of a file in a sufficiently large number (unsigned long long) in a way (hopefully) independent of the operating system and of the architecture

Parameters

File	path

Returns

File size

6.7.2.3 MatrixTypeName()

```
std::string MatrixTypeName (
            unsigned char matrixtypeident )
```

Auxiliary functions to be used for error printing.

Returns the name of the matrix whose type (as indentifier) is passed

Parameters

matrixtypeident.	The type of the matrix (full, sparse, symmetric) as defined by the former constants.

Returns

A human-meaningful string decribing the type

6.7.2.4 MetadataInfo()

```
std::string MetadataInfo (
            unsigned char metadatainfo )
```

Returns a message to interpret the presence of metadata

Parameters

metadatainfo.	The constant for information on which metadata are present as defined by the former constants.

Returns

A human-meaningful string decribing the present metadata

6.7.2.5 PositionsInFile()

```
void PositionsInFile (
            std::string fname,
            unsigned long long * start_of_metadata,
            unsigned long long * start_of_comment )
```

Returns the positions of the start of metadata and start of comments (included inside metadata) as absolute positions measured in bytes from the beginning of the file

Parameters

File	path
∗*start_of_metadata*	
∗*start_of_comment*	

6.7.2.6 SizeOfType()

```
int SizeOfType (
            unsigned char datatypeident )
```

Returns the size in bytes of the data type whose type (as indentifier) is passed

Parameters

datatypeident.	The data type of the data in the matrix (unsigned char, char,...) as defined by the former constants.

Returns

The size in bytes of one element of the passed data type.

6.7.2.7 ThisMachineEndianness()

```
unsigned char ThisMachineEndianness ( )
```

Returns the endianness of the machine where this function is called

Returns

Either the constant BIGEND or the constant LITEND

6.7.3 Variable Documentation

6.7.3.1 BIGEND

```
const unsigned char BIGEND =0x00
```

Constants for the possible endianness of a machine Only two values allowed for big and little endian Big endian

6.7.3.2 BLOCK_MARK

```
const unsigned char BLOCK_MARK =0xFF
```

Constants for information about the metadata included with the matrix Possible metadata stored are currently names of rows and names of columns Errors are returned with these constants in case of bad reads

6.7.3.3 BLOCKSEP

```
const unsigned char BLOCKSEP[BLOCKSEP_LEN] ={BLOCK_MARK,0x45,0x42,BLOCK_MARK}
```

Constants for information about the metadata included with the matrix Possible metadata stored are currently names of rows and names of columns Errors are returned with these constants in case of bad reads

6.7.3.4 BLOCKSEP_LEN

```
const unsigned int BLOCKSEP_LEN =4
```

Constants for information about the metadata included with the matrix Possible metadata stored are currently names of rows and names of columns Errors are returned with these constants in case of bad reads

6.7.3.5 COL_NAMES

```
const unsigned char COL_NAMES =0x02
```

Constants for information about the metadata included with the matrix Possible metadata stored are currently names of rows and names of columns Errors are returned with these constants in case of bad reads

6.7.3.6 COMMENT

```
const unsigned char COMMENT =0x04
```

Constants for information about the metadata included with the matrix Possible metadata stored are currently names of rows and names of columns Errors are returned with these constants in case of bad reads

6.7.3.7 COMMENT_SIZE

```
const unsigned int COMMENT_SIZE =1024
```

Constants for information about the metadata included with the matrix Possible metadata stored are currently names of rows and names of columns Errors are returned with these constants in case of bad reads

6.7.3.8 DTYPE

```
const unsigned char DTYPE =0x0B
```

double

6.7.3.9 ERROR_READING_COL_NAMES

```
const int ERROR_READING_COL_NAMES =3
```

Constants for information about the metadata included with the matrix Possible metadata stored are currently names of rows and names of columns Errors are returned with these constants in case of bad reads

6.7.3.10 ERROR_READING_ROW_NAMES

```
const int ERROR_READING_ROW_NAMES =2
```

Constants for information about the metadata included with the matrix Possible metadata stored are currently names of rows and names of columns Errors are returned with these constants in case of bad reads

6.7.3.11 ERROR_READING_SEP_MARK

```
const int ERROR_READING_SEP_MARK =4
```

Constants for information about the metadata included with the matrix Possible metadata stored are currently names of rows and names of columns Errors are returned with these constants in case of bad reads

6.7.3.12 ERROR_READING_STRINGS

```
const int ERROR_READING_STRINGS =1
```

Constants for information about the metadata included with the matrix Possible metadata stored are currently names of rows and names of columns Errors are returned with these constants in case of bad reads

6.7.3.13 FTYPE

```
const unsigned char FTYPE =0x0A
```

float

6.7.3.14 HEADER_SIZE

```
const unsigned short HEADER_SIZE =128
```

The header size. We fix a header of 128 bytes. We don't need so much, but just in case in the future...

6.7.3.15 LDTYPE

```
const unsigned char LDTYPE =0x0C
```

long double

6.7.3.16 LITEND

```
const unsigned char LITEND =0xF0
```

Little endian

6.7.3.17 MAX_LEN_NAME

```
const unsigned int MAX_LEN_NAME =1023
```

Constants for information about the metadata included with the matrix Possible metadata stored are currently names of rows and names of columns Errors are returned with these constants in case of bad reads

6.7.3.18 MTYPEFULL

```
const unsigned char MTYPEFULL =0x00
```

Full matrix

6.7.3.19 MTYPENOTYPE

```
const unsigned char MTYPENOTYPE =0x0F
```

Constants for the possible matrix types Currently, they are no type (for errors), full matrix, sparse matrix and symmetric matrix. No matrix type

6.7.3.20 MTYPESPARSE

```
const unsigned char MTYPESPARSE =0x01
```

Sparse matrix

6.7.3.21 MTYPESYMMETRIC

```
const unsigned char MTYPESYMMETRIC =0x02
```

Symmetric matrix

6.7.3.22 NO_METADATA

```
const unsigned char NO_METADATA =0x00
```

Constants for information about the metadata included with the matrix Possible metadata stored are currently names of rows and names of columns Errors are returned with these constants in case of bad reads

6.7.3.23 NOTYPE

```
const unsigned char NOTYPE =0x0F
```

Constants for the possible data types a matrix can hold. These are (apart of the no type for errors) integer types: char (8 bits), short int (16 bits), int (32 bits), long (32 bits), long long (64 bits) with their signed versions and float types in IEEE-754 format: float (32 bits), double (64 bits) and long double (128 bits) No data type

6.7.3.24 READ_OK

```
const int READ_OK =0
```

Constants for information about the metadata included with the matrix Possible metadata stored are currently names of rows and names of columns Errors are returned with these constants in case of bad reads

6.7.3.25 ROW_NAMES

```
const unsigned char ROW_NAMES =0x01
```

Constants for information about the metadata included with the matrix Possible metadata stored are currently names of rows and names of columns Errors are returned with these constants in case of bad reads

6.7.3.26 SCTYPE

```
const unsigned char SCTYPE =0x01
```

char

6.7.3.27 SITYPE

```
const unsigned char SITYPE =0x05
```

int

6.7.3.28 SLLTYPE

```
const unsigned char SLLTYPE =0x09
```

long long

6.7.3.29 SLTYPE

```
const unsigned char SLTYPE =0x07
```

long

6.7.3.30 SSTYPE

```
const unsigned char SSTYPE =0x03
```

short

6.7.3.31 UCTYPE

```
const unsigned char UCTYPE =0x00
```

unsigned char

6.7.3.32 UITYPE

```
const unsigned char UITYPE =0x04
```

unsigned int

6.7.3.33 ULLTYPE

```
const unsigned char ULLTYPE =0x08
```

unsigned long long

6.7.3.34 ULTYPE

```
const unsigned char ULTYPE =0x06
```

unsigned long

6.7.3.35 USTYPE

```
const unsigned char USTYPE =0x02
```

unsigned short

6.8 src/headers/matinfo.h File Reference

```
#include <iostream>
#include <string>
#include "indextype.h"
```

Functions

- void JMatInfo (std::string fname, std::string fres="")

- void MatrixType (std::string fname, unsigned char &mtype)
- void MatrixType (std::string fname, unsigned char &mtype, unsigned char &ctype)
- void MatrixType (std::string fname, unsigned char &mtype, unsigned char &ctype, unsigned char &endianness)
- void MatrixType (std::string fname, unsigned char &mtype, unsigned char &ctype, unsigned char &endianness, unsigned char &mdinf)
- void MatrixType (std::string fname, unsigned char &mtype, unsigned char &ctype, unsigned char &endianness, unsigned char &mdinf, indextype &nrows, indextype &ncols)

6.8.1 Function Documentation

6.8.1.1 JMatInfo()

```
void JMatInfo (
            std::string fname,
            std::string fres = "" )
```

Gives information about the JMatrix stored in a file

Parameters

in	*fname*	The name of the binary file that contains the matrix
in	*fres*	The name of the text file that will contain the information, or the empty string to show the information in the console

Function to get information about the JMatrix stored in a binary file and store such information in a text file

Parameters

in	*iname*	Name of the JMatrix binary file
in	*oname*	Name of the text file to write the information

6.8.1.2 MatrixType() [1/5]

```
void MatrixType (
            std::string fname,
            unsigned char & mtype )
```

Auxiliary funcions to check characteristics of matrix stored in binary file looking only at its header. Matrix in file is NOT loaded into memory.

Parameters

in	*fname*	Name of the binary file
out	*mtype*	Returns matrix type (full,sparse,symmetric)
out	*ctype*	Returns matrix data type
out	*endianness*	Returns endianness (big,little) of the data stored in the matrix
out	*nrows*	Returns the number of rows
out	*ncols*	Returns the number of columns
out	*mdinf*	Returns the signal for the presence of metadata

6.8.1.3 MatrixType() [2/5]

```
void MatrixType (
            std::string fname,
            unsigned char & mtype,
            unsigned char & ctype )
```

Auxiliary funcions to check characteristics of matrix stored in binary file looking only at its header. Matrix in file is NOT loaded into memory.

Parameters

in	*fname*	Name of the binary file
out	*mtype*	Returns matrix type (full,sparse,symmetric)
out	*ctype*	Returns matrix data type
out	*endianness*	Returns endianness (big,little) of the data stored in the matrix
out	*nrows*	Returns the number of rows
out	*ncols*	Returns the number of columns
out	*mdinf*	Returns the signal for the presence of metadata

6.8.1.4 MatrixType() [3/5]

```
void MatrixType (
            std::string fname,
```

```
            unsigned char & mtype,
            unsigned char & ctype,
            unsigned char & endianness )
```

Auxiliary funcions to check characteristics of matrix stored in binary file looking only at its header. Matrix in file is NOT loaded into memory.

Parameters

in	*fname*	Name of the binary file
out	*mtype*	Returns matrix type (full,sparse,symmetric)
out	*ctype*	Returns matrix data type
out	*endianness*	Returns endianness (big,little) of the data stored in the matrix
out	*nrows*	Returns the number of rows
out	*ncols*	Returns the number of columns
out	*mdinf*	Returns the signal for the presence of metadata

6.8.1.5 MatrixType() [4/5]

```
void MatrixType (
            std::string fname,
            unsigned char & mtype,
            unsigned char & ctype,
            unsigned char & endianness,
            unsigned char & mdinf )
```

Auxiliary funcions to check characteristics of matrix stored in binary file looking only at its header. Matrix in file is NOT loaded into memory.

Parameters

in	*fname*	Name of the binary file
out	*mtype*	Returns matrix type (full,sparse,symmetric)
out	*ctype*	Returns matrix data type
out	*endianness*	Returns endianness (big,little) of the data stored in the matrix
out	*nrows*	Returns the number of rows
out	*ncols*	Returns the number of columns
out	*mdinf*	Returns the signal for the presence of metadata

6.8.1.6 MatrixType() [5/5]

```
void MatrixType (
            std::string fname,
            unsigned char & mtype,
            unsigned char & ctype,
```

```
            unsigned char & endianness,
            unsigned char & mdinf,
            indextype & nrows,
            indextype & ncols )
```

Auxiliary funcions to check characteristics of matrix stored in binary file looking only at its header. Matrix in file is NOT loaded into memory.

Parameters

in	*fname*	Name of the binary file
out	*mtype*	Returns matrix type (full,sparse,symmetric)
out	*ctype*	Returns matrix data type
out	*endianness*	Returns endianness (big,little) of the data stored in the matrix
out	*nrows*	Returns the number of rows
out	*ncols*	Returns the number of columns
out	*mdinf*	Returns the signal for the presence of metadata

6.9 src/headers/matmetadata.h File Reference

```
#include "fullmatrix.h"
#include "sparsematrix.h"
#include "symmetricmatrix.h"
#include <cmath>
```

Functions

- std::vector< std::string > JGetRowNames (std::string fname)
- std::vector< std::string > JGetColNames (std::string fname)

6.9.1 Function Documentation

6.9.1.1 JGetColNames()

```
std::vector<std::string> JGetColNames (
            std::string fname )
```

Function to get the names of the columns of the JMatrix stored in a binary file
The matrix is not loaded into memory

Parameters

in	*fname*	Name of the binary file containing the matrix

Returns

A vector of strings with the column names

6.9.1.2 JGetRowNames()

```
std::vector<std::string> JGetRowNames (
            std::string fname )
```

Function to get the names of the rows of the JMatrix stored in a binary file
The matrix is not loaded into memory

Parameters

in	*fname*	Name of the binary file containing the matrix

Returns

A vector of strings with the row names

6.10 src/headers/memhelper.h File Reference

```
#include <iostream>
```

Functions

- void GetAvailableMemAndSwap (unsigned long &avmem, unsigned long &avswap)
- void MemoryWarnings (unsigned long nr, unsigned long nc, int s)
- void MemoryWarnings (unsigned long nr, int s)

6.10.1 Function Documentation

6.10.1.1 GetAvailableMemAndSwap()

```
void GetAvailableMemAndSwap (
            unsigned long & avmem,
            unsigned long & avswap )
```

Finds how much memory is available and how much swap at call time.

Parameters

out	*avmem*	Available memory in bytes
out	*avswap*	Available swap in bytes ========== WARNING: currently this funcion work only on Linux systems. Calling in other systems simply sets the variables to 0 ==========

6.10.1.2 MemoryWarnings() [1/2]

```
void MemoryWarnings (
            unsigned long nr,
            int s )
```

Shows memory warnings if the memory needed to declare a SymmetricMatrix is above 75% of available memory or directly shows an error and stops the program is it is avobe the available memory
The idea is to call this function before going into an scope where a SymmetricMatrix will be declared if you think it might be too large, or if you don't know its size in advance

Parameters

in	*nr*	Number of rows of the prospective SymmetricMatrix (obviously, it will be square)
in	*s*	Size in bytes of each element (normally obtained with sizeof(Type)) ========== WARNING: currently this funcion work only on Linux systems. Calling in other systems simply sets the variables to 0 ==========

6.10.1.3 MemoryWarnings() [2/2]

```
void MemoryWarnings (
            unsigned long nr,
            unsigned long nc,
            int s )
```

Shows memory warnings if the memory needed to declare a FullMatrix is above 75% of available memory or directly shows an error and stops the program is it is avobe the available memory
The idea is to call this function before going into an scope where a FullMatrix will be declared if you think it might be too large, or if you don't know its size in advance

Parameters

in	*nr*	Number of rows of the prospective FullMatrix
in	*nc*	Number of columns of the prospective FullMatrix

Parameters

in	*s*	Size in bytes of each element (normally obtained with sizeof(Type)) ========== WARNING: currently this funcion work only on Linux systems. Calling in other systems simply sets the variables to 0 ==========

6.11 src/headers/sparsematrix.h File Reference

```
#include "jmatrix.h"
#include <algorithm>
```

Classes

- class SparseMatrix< T >

Enumerations

- enum **TrMark** { **transpose** =0 }

Functions

- template<typename T >
  void **sort_indexes_and_value** (const std::vector< T > &v, std::vector< size_t > &idx, std::vector< indextype > &idv)

6.12 src/headers/symmetricmatrix.h File Reference

```
#include "jmatrix.h"
#include "memhelper.h"
```

Classes

- class SymmetricMatrix< T >

Index