

Hierarchical Matrices Lib

Generated by Doxygen 1.9.4

<b>1 hierarchical-matrices-lib</b>	<b>1</b>
1.1 Available functions:	2
1.2 Used sources/tools for development:	2
<b>2 Hierarchical Index</b>	<b>2</b>
2.1 Class Hierarchy	2
<b>3 Class Index</b>	<b>3</b>
3.1 Class List	3
<b>4 File Index</b>	<b>3</b>
4.1 File List	3
<b>5 Class Documentation</b>	<b>3</b>
5.1 Block< datatype > Class Template Reference	3
5.1.1 Detailed Description	4
5.1.2 Member Function Documentation	4
5.2 DivByZeroException Class Reference	6
5.3 EntrywiseBlock< datatype > Class Template Reference	6
5.3.1 Detailed Description	7
5.3.2 Member Function Documentation	7
5.4 HierarchicalMatrix< datatype > Class Template Reference	8
5.4.1 Detailed Description	9
5.4.2 Constructor & Destructor Documentation	9
5.4.3 Member Function Documentation	9
5.5 NonInvertibleMatrixException Class Reference	11
5.6 OuterProductBlock< datatype > Class Template Reference	11
5.6.1 Detailed Description	12
5.6.2 Member Function Documentation	12
<b>6 File Documentation</b>	<b>14</b>
6.1 Block.hpp	14
6.2 EntrywiseBlock.hpp	14
6.3 Exceptions.hpp	15
6.4 HierarchicalMatrix.hpp	15
6.5 OuterProductBlock.hpp	16
6.6 user_settings.hpp	16
<b>Index</b>	<b>17</b>

## 1 hierarchical-matrices-lib

This is an incomplete C++ library for efficiently computing approximations of the most common matrix calculations via hierarchical matrices.

Note that input matrices can only be square!

The only file that should be changed is "user\_settings", in which you will be able to switch between the datatypes float, double and complex floats. If you are only using float, there is no need to change the settings since it is the default datatype.

Please see the included documentation file for further instructions on how to use this library.

Any mentionings of pages refer to the book "Hierarchical Matrices", see below.

## 1.1 Available functions:

- Constructor
- Matrix-matrix addition
- Matrix-vector multiplication

## 1.2 Used sources/tools for development:

- "Hierarchical Matrices" by Mario Bebendorf from 2008  
(see <https://link.springer.com/book/10.1007/978-3-540-77147-0>)  
Specifically the first two chapters about "Low-Rank Matrices and Matrix Partitioning" and "Hierarchical Matrices"
- OS: Windows
- Compiler: MinGW 32
- Libraries: LAPACK 3.10 (Instructions: <https://icl.utk.edu/lapack-for-windows/lapack/#lapacke>),  
--> Needs C, C++ and Fortran compiler (e.g. gcc, g++ and gfortran included in MinGW)
- Documentation: Doxygen

# 2 Hierarchical Index

## 2.1 Class Hierarchy

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

<b>Block</b> < datatype >	<b>3</b>
<b>EntrywiseBlock</b> < datatype >	<b>6</b>
<b>HierarchicalMatrix</b> < datatype >	<b>8</b>
<b>OuterProductBlock</b> < datatype >	<b>11</b>
std::exception	
<b>DivByZeroException</b>	<b>6</b>
<b>NonInvertibleMatrixException</b>	<b>11</b>

## 3 Class Index

### 3.1 Class List

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

<a href="#">Block&lt; datatype &gt;</a>	
Abstract class to depict all parts of the matrix	3
<a href="#">DivByZeroException</a>	6
<a href="#">EntrywiseBlock&lt; datatype &gt;</a>	
Depicts a non-admissible or full rank part of the original matrix	6
<a href="#">HierarchicalMatrix&lt; datatype &gt;</a>	
Transforms a square entrywise matrix into a hierarchical matrix to be calculated with	8
<a href="#">NonInvertibleMatrixException</a>	11
<a href="#">OuterProductBlock&lt; datatype &gt;</a>	
Depicts an admissible or low-rank part of the original matrix	11

## 4 File Index

### 4.1 File List

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

hierarchical-matrices-lib/ <a href="#">Block.hpp</a>	14
hierarchical-matrices-lib/ <a href="#">EntrywiseBlock.hpp</a>	14
hierarchical-matrices-lib/ <a href="#">Exceptions.hpp</a>	15
hierarchical-matrices-lib/ <a href="#">HierarchicalMatrix.hpp</a>	15
hierarchical-matrices-lib/ <a href="#">OuterProductBlock.hpp</a>	16
hierarchical-matrices-lib/ <a href="#">user_settings.hpp</a>	16

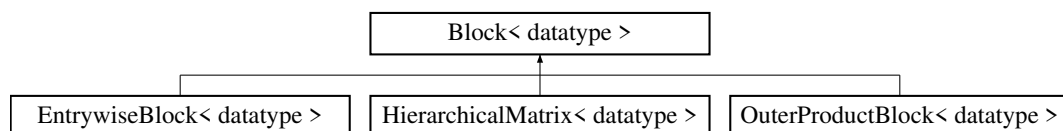
## 5 Class Documentation

### 5.1 Block< datatype > Class Template Reference

Abstract class to depict all parts of the matrix.

```
#include <Block.hpp>
```

Inheritance diagram for Block< datatype >:



## Public Member Functions

- **Block** (unsigned int m, unsigned int n)  
*Automatic initialization of attributes from subclasses.*
- virtual **Block** \* **coarse** (double accuracy)=0
- virtual datatype \* **operator\*** (const datatype vector[])=0
- virtual **Block** \* **operator+** (**Block** \*addedBlock)=0  
*First layer of polymorphic addition, detects the first operand and calls swapped addition to detect the second operand through the second layer.*
- virtual **Block** \* **operator+** (**HierarchicalMatrix**< datatype > \*addedBlock)=0  
*Second layer of polymorphic addition, detects the second operand and actually calculates the corresponding sum.*
- virtual **Block** \* **operator+** (**OuterProductBlock**< datatype > \*addedBlock)=0  
*Second layer of polymorphic addition, detects the second operand and actually calculates the corresponding sum.*
- virtual **Block** \* **operator+** (**EntrywiseBlock**< datatype > \*addedBlock)=0  
*Second layer of polymorphic addition, detects the second operand and actually calculates the corresponding sum.*

## Protected Attributes

- unsigned int **mDim**  
*Number of rows from the original matrix depicted within this block.*
- unsigned int **nDim**  
*Number of columns from the original matrix depicted within this block.*
- unsigned int **indiceRange** [2][2]  
*Lower and upper bound of row and column indices from the original matrix depicted within this block, see enum InndiceOrientation.*

### 5.1.1 Detailed Description

```
template<class datatype>
class Block< datatype >
```

Abstract class to depict all parts of the matrix.

### 5.1.2 Member Function Documentation

```
5.1.2.1 coarse()  template<class datatype >
virtual Block * Block< datatype >::coarse (
    double accuracy )  [pure virtual]
```

Implemented in **HierarchicalMatrix**< datatype >.

**5.1.2.2 operator\*()** `template<class datatype >`  
`virtual datatype * Block< datatype >::operator* (`  
`const datatype vector[] ) [pure virtual]`

Implemented in [HierarchicalMatrix< datatype >](#).

**5.1.2.3 operator+()** [1/4] `template<class datatype >`  
`virtual Block * Block< datatype >::operator+ (`  
`Block< datatype > * addedBlock ) [pure virtual]`

First layer of polymorphic addition, detects the first operand and calls swapped addition to detect the second operand through the second layer.

Implemented in [EntrywiseBlock< datatype >](#), and [OuterProductBlock< datatype >](#).

**5.1.2.4 operator+()** [2/4] `template<class datatype >`  
`virtual Block * Block< datatype >::operator+ (`  
`EntrywiseBlock< datatype > * addedBlock ) [pure virtual]`

Second layer of polymorphic addition, detects the second operand and actually calculates the corresponding sum.

Implemented in [EntrywiseBlock< datatype >](#), and [OuterProductBlock< datatype >](#).

**5.1.2.5 operator+()** [3/4] `template<class datatype >`  
`virtual Block * Block< datatype >::operator+ (`  
`HierarchicalMatrix< datatype > * addedBlock ) [pure virtual]`

Second layer of polymorphic addition, detects the second operand and actually calculates the corresponding sum.

Implemented in [EntrywiseBlock< datatype >](#), and [OuterProductBlock< datatype >](#).

**5.1.2.6 operator+()** [4/4] `template<class datatype >`  
`virtual Block * Block< datatype >::operator+ (`  
`OuterProductBlock< datatype > * addedBlock ) [pure virtual]`

Second layer of polymorphic addition, detects the second operand and actually calculates the corresponding sum.

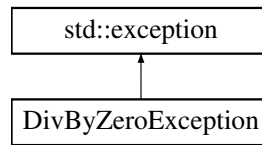
Implemented in [OuterProductBlock< datatype >](#), and [EntrywiseBlock< datatype >](#).

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

- hierarchical-matrices-lib/Block.hpp

## 5.2 DivByZeroException Class Reference

Inheritance diagram for DivByZeroException:



### Public Member Functions

- virtual const char \* **what** () const noexcept

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

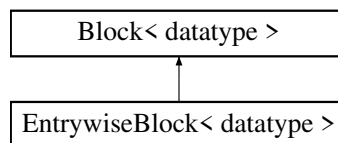
- hierarchical-matrices-lib/Exceptions.hpp

## 5.3 EntrywiseBlock< datatype > Class Template Reference

Depicts a non-admissible or full rank part of the original matrix.

```
#include <EntrywiseBlock.hpp>
```

Inheritance diagram for EntrywiseBlock< datatype >:



### Public Member Functions

- **EntrywiseBlock** (datatype \*\*originalBlock, unsigned int `mDim`, unsigned int `nDim`, std::vector< unsigned int > `ilnd`, std::vector< unsigned int > `jlnd`)  
*Copies part of the matrix to be stored in a block.*
- `Block< datatype > * coarse` (double accuracy) final
- datatype \* **operator\*** (const datatype vector[ ])
- `Block< datatype > * operator+ (Block< datatype > *addedBlock)`  
*First layer of polymorphic addition, detects the first operand and calls swapped addition to detect the second operand through the second layer.*
- `Block< datatype > * operator+ (HierarchicalMatrix< datatype > *addedBlock)`  
*Second layer of polymorphic addition, detects the second operand and actually calculates the corresponding sum.*
- `Block< datatype > * operator+ (OuterProductBlock< datatype > *addedBlock)`  
*Second layer of polymorphic addition, detects the second operand and actually calculates the corresponding sum.*
- `Block< datatype > * operator+ (EntrywiseBlock *addedBlock)`  
*Second layer of polymorphic addition, detects the second operand and actually calculates the corresponding sum.*
- **~EntrywiseBlock** ()  
*Frees all memory allocated for the block array.*

### Protected Attributes

- datatype \*\* **block**  
*mDim \* nDim array / Exact part of the original matrix*

#### 5.3.1 Detailed Description

```
template<class datatype>
class EntrywiseBlock< datatype >
```

Depicts a non-admissible or full rank part of the original matrix.

#### 5.3.2 Member Function Documentation

**5.3.2.1 coarse()** `template<class datatype >`  
`Block< datatype > * EntrywiseBlock< datatype >::coarse (`  
     `double accuracy ) [final], [virtual]`

Implements [Block< datatype >](#).

**5.3.2.2 operator\*()** `template<class datatype >`  
`datatype * EntrywiseBlock< datatype >::operator* (`  
     `const datatype vector[] ) [virtual]`

Implements [Block< datatype >](#).

**5.3.2.3 operator+() [1/4]** `template<class datatype >`  
`Block< datatype > * EntrywiseBlock< datatype >::operator+ (`  
     `Block< datatype > * addedBlock ) [virtual]`

First layer of polymorphic addition, detects the first operand and calls swapped addition to detect the second operand through the second layer.

Implements [Block< datatype >](#).

**5.3.2.4 operator+() [2/4]** `template<class datatype >`  
`Block< datatype > * EntrywiseBlock< datatype >::operator+ (`  
     `EntrywiseBlock< datatype > * addedBlock ) [virtual]`

Second layer of polymorphic addition, detects the second operand and actually calculates the corresponding sum.

Implements [Block< datatype >](#).



```

5.3.2.5 operator+() [3/4] template<class datatype >
Block< datatype > * EntrywiseBlock< datatype >::operator+ (
    HierarchicalMatrix< datatype > * addedBlock ) [virtual]

```

Second layer of polymorphic addition, detects the second operand and actually calculates the corresponding sum.

Implements `Block< datatype >`.

```

5.3.2.6 operator+() [4/4] template<class datatype >
Block< datatype > * EntrywiseBlock< datatype >::operator+ (
    OuterProductBlock< datatype > * addedBlock ) [virtual]

```

Second layer of polymorphic addition, detects the second operand and actually calculates the corresponding sum.

Implements `Block< datatype >`.

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

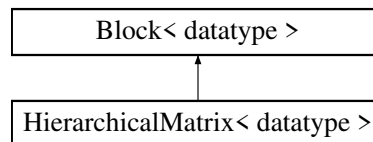
- hierarchical-matrices-lib/Block.hpp
- hierarchical-matrices-lib/EntrywiseBlock.hpp

## 5.4 HierarchicalMatrix< datatype > Class Template Reference

Transforms a square entrywise matrix into a hierarchical matrix to be calculated with.

```
#include <HierarchicalMatrix.hpp>
```

Inheritance diagram for HierarchicalMatrix< datatype >:



### Public Member Functions

- `HierarchicalMatrix` (datatype \*\*originalMatrix, std::list< std::vector< unsigned int > > \*originalIndices, unsigned int dim, double clusterParamEta=0.5)
- `Block< datatype > * coarse` (double accuracy) final
- `HierarchicalMatrix * operator+` (const `HierarchicalMatrix` &addedMatrix)
- `HierarchicalMatrix * operator+=` (const `HierarchicalMatrix` &addedMatrix)
- `datatype * operator*` (const datatype vector[ ])
- `~HierarchicalMatrix` ()  
*Recursively frees all Blocks saved in the matrix-attribute.*

### Protected Attributes

- `Block< datatype > * matrix` [2][2]  
*Hierarchical matrix, recursively divided into quadrants until one of the other `Block` types is reached, can also hold nullptr if a 2x1 or 1x2 division is reached.*

## Friends

- class **OuterProductBlock**< datatype >
- class **EntrywiseBlock**< datatype >

### 5.4.1 Detailed Description

```
template<class datatype>
class HierarchicalMatrix< datatype >
```

Transforms a square entrywise matrix into a hierarchical matrix to be calculated with.

### 5.4.2 Constructor & Destructor Documentation

**5.4.2.1 HierarchicalMatrix()** `template<class datatype >`  
`HierarchicalMatrix< datatype >::HierarchicalMatrix (`  
     `datatype ** originalMatrix,`  
     `std::list< std::vector< unsigned int > > * originalIndices,`  
     `unsigned int dim,`  
     `double clusterParamEta = 0.5 )`

Transforms an entrywise matrix with float, double or complex float entries into a hierarchical matrix Example↔  
: HierarchicalMatrix<double> exampleMatrix(data, &indices, 100)

#### Parameters

<i>originalMatrix</i>	A two-dimensional array containing the square entrywise matrix to be transformed and calculated with
<i>originalIndices</i>	List containing the admissible partition into blocks for column and row indices of the matrix, each vector contains all the indices in ascending order, vectors are listed in ascending order so the first vector always s withh 0 and the last ends with dim-1, see page 31: Fig 1.5
<i>dim</i>	Number of columns/rows of the input matrix
<i>clusterParamEta</i>	Optional cluster parameter between 0 and 1, will be defaulted to 0.5 if no value is given, see page 24: (1.13)

### 5.4.3 Member Function Documentation

**5.4.3.1 coarse()** `template<class datatype >`  
`Block< datatype > * HierarchicalMatrix< datatype >::coarse (`  
     `double accuracy ) [final], [virtual]`

Coarsens the given hierarchical matrix until the given accuracy can no longer be held, usually keeping the same storage size while reducing the number of branches/blocks

**Parameters**

<i>accuracy</i>	Accuracy > 0 to be satisfied in each coarsening step, see page 72: (2.13)
-----------------	---------------------------------------------------------------------------

**Returns**

The coarsened hierarchical matrix, nullptr if matrix couldn't be coarsened

Implements [Block< datatype >](#).

**5.4.3.2 operator\*()** `template<class datatype >`  
`datatype * HierarchicalMatrix< datatype >::operator* (`  
`const datatype vector[] ) [virtual]`

Matrix-vector multiplication

**Parameters**

<i>vector</i>	One-dimensional array with size equaling the number of columns(=rows) of the input matrix
---------------	-------------------------------------------------------------------------------------------

**Returns**

Resulting vector with size equaling the number of rows(=columns) of the input matrix, nullptr if vector couldn't be calculated

Implements [Block< datatype >](#).

**5.4.3.3 operator+()** `template<class datatype >`  
`HierarchicalMatrix * HierarchicalMatrix< datatype >::operator+ (`  
`const HierarchicalMatrix< datatype > & addedMatrix )`

Rounded addition of two Hierarchical Matrices of the same size

**Parameters**

<i>addedMatrix</i>	Second matrix to be added with the same number of rows and columns as the first matrix
--------------------	----------------------------------------------------------------------------------------

**Returns**

Sum of the two matrices with the same number of rows and columns as the input matrices, nullptr if matrices couldn't be added

**5.4.3.4 operator+=()** `template<class datatype >`  
`HierarchicalMatrix * HierarchicalMatrix< datatype >::operator+= (`  
`const HierarchicalMatrix< datatype > & addedMatrix )`

Rounded addition of two Hierarchical Matrices of the same size

#### Parameters

<i>addedMatrix</i>	Second matrix to be added with the same number of rows and columns as the first matrix
--------------------	----------------------------------------------------------------------------------------

#### Returns

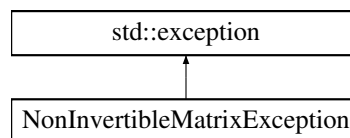
Sum of the two matrices with the same number of rows and columns as the input matrices, nullptr if matrices couldn't be added

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

- hierarchical-matrices-lib/Block.hpp
- hierarchical-matrices-lib/HierarchicalMatrix.hpp

## 5.5 NonInvertibleMatrixException Class Reference

Inheritance diagram for NonInvertibleMatrixException:



#### Public Member Functions

- virtual const char \* **what** () noexcept

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

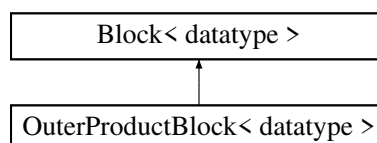
- hierarchical-matrices-lib/Exceptions.hpp

## 5.6 OuterProductBlock< datatype > Class Template Reference

Depicts an admissible or low-rank part of the original matrix.

```
#include <OuterProductBlock.hpp>
```

Inheritance diagram for OuterProductBlock< datatype >:



## Public Member Functions

- **OuterProductBlock** (datatype \*\*originalBlock, unsigned int mDim, unsigned int nDim, std::vector< unsigned int > iInd, std::vector< unsigned int > jInd, unsigned int rank)  
*Transforms an entrywise part of the matrix into its outer product form.*
- **Block**< datatype > \* **coarse** (double accuracy) final
- datatype \* **operator\*** (const datatype vector[ ])
- **Block**< datatype > \* **operator+** (**Block**< datatype > \*addedBlock)  
*First layer of polymorphic addition, detects the first operand and calls swapped addition to detect the second operand through the second layer.*
- **Block**< datatype > \* **operator+** (**HierarchicalMatrix**< datatype > \*addedBlock)  
*Second layer of polymorphic addition, detects the second operand and actually calculates the correponding sum.*
- **Block**< datatype > \* **operator+** (**OuterProductBlock** \*addedBlock)  
*Second layer of polymorphic addition, detects the second operand and actually calculates the correponding sum.*
- **Block**< datatype > \* **operator+** (**EntrywiseBlock**< datatype > \*addedBlock)  
*Second layer of polymorphic addition, detects the second operand and actually calculates the correponding sum.*
- **~OuterProductBlock** ()  
*Frees all memory allocated for the u, x and v arrays.*

## Protected Attributes

- datatype \*\* **u**  
*mDim \* k array / U*
- datatype \*\* **x**  
*k \* k array / X, similar to S*
- datatype \*\* **v**  
*nDim \* k array / V<sup>H</sup> H with complex entries, V<sup>H</sup> T with real entries*
- unsigned int **k**  
*Rank of original matrix part.*

### 5.6.1 Detailed Description

```
template<class datatype>
class OuterProductBlock< datatype >
```

Depicts an admissible or low-rank part of the original matrix.

### 5.6.2 Member Function Documentation

**5.6.2.1 coarse()** template<class datatype >  
**Block**< datatype > \* **OuterProductBlock**< datatype >::coarse (   
     double accuracy ) [final], [virtual]

Implements **Block**< datatype >.

**5.6.2.2 operator\*()** `template<class datatype >`  
`datatype * OuterProductBlock< datatype >::operator* (`  
`const datatype vector[] ) [virtual]`

Implements [Block< datatype >](#).

**5.6.2.3 operator+()** [1/4] `template<class datatype >`  
`Block< datatype > * OuterProductBlock< datatype >::operator+ (`  
`Block< datatype > * addedBlock ) [virtual]`

First layer of polymorphic addition, detects the first operand and calls swapped addition to detect the second operand through the second layer.

Implements [Block< datatype >](#).

**5.6.2.4 operator+()** [2/4] `template<class datatype >`  
`Block< datatype > * OuterProductBlock< datatype >::operator+ (`  
`EntrywiseBlock< datatype > * addedBlock ) [virtual]`

Second layer of polymorphic addition, detects the second operand and actually calculates the corresponding sum.

Implements [Block< datatype >](#).

**5.6.2.5 operator+()** [3/4] `template<class datatype >`  
`Block< datatype > * OuterProductBlock< datatype >::operator+ (`  
`HierarchicalMatrix< datatype > * addedBlock ) [virtual]`

Second layer of polymorphic addition, detects the second operand and actually calculates the corresponding sum.

Implements [Block< datatype >](#).

**5.6.2.6 operator+()** [4/4] `template<class datatype >`  
`Block< datatype > * OuterProductBlock< datatype >::operator+ (`  
`OuterProductBlock< datatype > * addedBlock ) [virtual]`

Second layer of polymorphic addition, detects the second operand and actually calculates the corresponding sum.

Implements [Block< datatype >](#).

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

- hierarchical-matrices-lib/Block.hpp
- hierarchical-matrices-lib/OuterProductBlock.hpp

## 6 File Documentation

### 6.1 Block.hpp

```

1 #ifndef HIERARCHICAL_MATRICES_BLOCK_H
2 #define HIERARCHICAL_MATRICES_BLOCK_H
3
4 template <class datatype> class HierarchicalMatrix;
5 template <class datatype> class OuterProductBlock;
6 template <class datatype> class EntrywiseBlock;
7
9 enum IndiceOrientation {kRangeI=0, kRangeJ=1, kBottom=0, kTop=1};
10
11
13 template <class datatype>
14 class Block {
15
16 protected:
17
18     unsigned int mDim;
19     unsigned int nDim;
20     unsigned int indiceRange[2][2];
21
22 public:
24     Block(unsigned int m, unsigned int n): mDim(m), nDim(n) {}
25
26     virtual Block* coarse( double accuracy ) =0;
27
28     virtual datatype* operator*( const datatype vector[] ) =0;
29
30     // virtual Block& operator*(const Block& multBlock) =0;
31
32     virtual Block* operator+( Block* addedBlock ) =0;
33
34     virtual Block* operator+( HierarchicalMatrix<datatype>* addedBlock ) =0;
35     virtual Block* operator+( OuterProductBlock<datatype>* addedBlock ) =0;
36     virtual Block* operator+( EntrywiseBlock<datatype>* addedBlock ) =0;
37
38     virtual ~Block(){};
39 };
40
41 #endif // HIERARCHICAL_MATRICES_BLOCK_H

```

### 6.2 EntrywiseBlock.hpp

```

1 #ifndef HIERARCHICAL_MATRICES_ENTRYWISEBLOCK_H
2 #define HIERARCHICAL_MATRICES_ENTRYWISEBLOCK_H
3
4 #include "Block.hpp"
5
6 #include <list>
7 #include <vector>
8
9
11 template <class datatype>
12 class EntrywiseBlock: public Block<datatype>{
13
14 protected:
15     datatype ** block;
16
17 public:
19     EntrywiseBlock(datatype ** originalBlock, unsigned int mDim, unsigned int nDim,
20         std::vector<unsigned int> iInd, std::vector<unsigned int> jInd);
21
22     Block<datatype>* coarse( double accuracy ) final;
23
24     datatype* operator*( const datatype vector[] );
25
26     // Block<datatype>& operator*( const Block<datatype>& multBlock );
27
28     Block<datatype>* operator+( Block<datatype>* addedBlock );
29     Block<datatype>* operator+( HierarchicalMatrix<datatype>* addedBlock );
30     Block<datatype>* operator+( OuterProductBlock<datatype>* addedBlock );
31     Block<datatype>* operator+( EntrywiseBlock* addedBlock );
32
33     ~EntrywiseBlock();
34 };
35
36 #endif // HIERARCHICAL_MATRICES_ENTRYWISEBLOCK_H

```

## 6.3 Exceptions.hpp

```

1 #ifndef HIERARCHICAL_MATRICES_EXCEPTIONS_H
2 #define HIERARCHICAL_MATRICES_EXCEPTIONS_H
3
4 #include <stdexcept>
5 #include <exception> //enthält what()
6
7 //fkt()noexcept{} ==in fkt darf/wird keine Exception geworfen
8
9 // try{
10 //     if(l==1) {
11 //         throw( DivByZeroException() );
12 //     }
13 // } catch( DivByZeroException exc ){
14 //     std::cerr « exc.what();
15 // }
16
17
18 // Exception for divisions by zero
19 class DivByZeroException: public std::exception{
20 public:
21     virtual const char* what() const noexcept{
22         return "Division by zero\n";
23     }
24 };
25
26
27 // Exception for when the matrix inversion fails due to unchecked invertibility
28 class NonInvertibleMatrixException: public std::exception{
29 public:
30     virtual const char* what() noexcept{
31         return "Given matrix cannot be inverted\n";
32     }
33 };
34
35 #endif // HIERARCHICAL_MATRICES_EXCEPTIONS_H

```

## 6.4 HierarchicalMatrix.hpp

```

1 #ifndef HIERARCHICAL_MATRICES_HIERARCHICALMATRIX_H
2 #define HIERARCHICAL_MATRICES_HIERARCHICALMATRIX_H
3
4 #include "Block.hpp"
5 #include "EntrywiseBlock.hpp"
6 #include "OuterProductBlock.hpp"
7
8 #include <array>
9 #include <list>
10 #include <vector>
11
12
13 template <class datatype>
14 class HierarchicalMatrix: public Block<datatype> {
15
16
17 friend class OuterProductBlock<datatype>;
18 friend class EntrywiseBlock<datatype>;
19
20 protected:
21     // [0][0] = top left, [0][1] = top right / [1][0] = bottom left, [1][1] = bottom right
22     Block<datatype>* matrix[2][2];
23
24 public:
25     HierarchicalMatrix(datatype ** originalMatrix, std::list<std::vector<unsigned int>*>
originalIndices, unsigned int dim, double clusterParamEta =0.5);
26
27
28     Block<datatype>* coarse( double accuracy ) final;
29
30     HierarchicalMatrix* operator+( const HierarchicalMatrix& addedMatrix );
31     HierarchicalMatrix* operator+=( const HierarchicalMatrix& addedMatrix );
32
33
34     datatype* operator*( const datatype vector[] );
35
36     // Matrix-matrix multiplication
37     // Block<datatype>& operator*( const Block<datatype>& multMatrix );
38     // Block<datatype>& operator*=( const HierarchicalMatrix& multMatrix );
39
40     // Inversion
41     // HierarchicalMatrix& invert();
42
43     // LU-decomposition
44     // std::array<HierarchicalMatrix*,2> luDecomposition();

```



```

71
72     ~HierarchicalMatrix();
73
74 private:
75     HierarchicalMatrix(datatype ** originalMatrix, std::list<std::vector<unsigned int>>*
76         originalIndices, unsigned int indices[2][2], double clusterParamEta, unsigned int ** distances);
77     void constructHierarchicalMatrix(datatype ** originalMatrix, std::list<std::vector<unsigned int>>*
78         originalIndices, unsigned int indices[2][2], double clusterParamEta, unsigned int ** distances);
79
80     Block<datatype>* operator+( Block<datatype>* addedBlock );
81     Block<datatype>* operator+( HierarchicalMatrix* addedBlock );
82     Block<datatype>* operator+( OuterProductBlock<datatype>* addedBlock );
83     Block<datatype>* operator+( EntrywiseBlock<datatype>* addedBlock );
84 };
85
86
87 #endif // HIERARCHICAL_MATRICES_HIERARCHICALMATRIX_H

```

## 6.5 OuterProductBlock.hpp

```

1 #ifndef HIERARCHICAL_MATRICES_OUTERPRODUCTBLOCK_H
2 #define HIERARCHICAL_MATRICES_OUTERPRODUCTBLOCK_H
3
4 #include "Block.hpp"
5 #include <list>
6 #include <vector>
7
8
9 template <class datatype>
10 class OuterProductBlock: public Block<datatype> {
11 protected:
12     datatype ** u;
13     datatype ** x;
14     datatype ** v;
15
16     unsigned int k;
17
18 public:
19     OuterProductBlock(datatype ** originalBlock, unsigned int mDim, unsigned int nDim,
20         std::vector<unsigned int> iInd, std::vector<unsigned int> jInd, unsigned int rank);
21
22     Block<datatype>* coarse( double accuracy ) final;
23
24     datatype* operator*( const datatype vector[] );
25
26     // Block<datatype>& operator*( const Block<datatype>& multBlock );
27
28     Block<datatype>* operator+( Block<datatype>* addedBlock );
29     Block<datatype>* operator+( HierarchicalMatrix<datatype>* addedBlock );
30     Block<datatype>* operator+( OuterProductBlock* addedBlock );
31     Block<datatype>* operator+( EntrywiseBlock<datatype>* addedBlock );
32
33     ~OuterProductBlock();
34 };
35
36
37 #endif // HIERARCHICAL_MATRICES_OUTERPRODUCTBLOCK_H

```

## 6.6 user\_settings.hpp

```

1 // This file includes necessary settings for you, the user
2 // ONLY CHANGE THE PARTS THAT ARE EXPLAINED IN THE COMMENTS
3 // Otherwise, being able to run the code cannot be guaranteed
4
5 // Change the number according to the format of the numbers in your matrices
6 // The default is float
7 // 0 = float
8 // 1 = double
9 // 2 = complex float
10 #define USED_DATATYPE 1
11
12
13
14
15 // DO NOT CHANGE ANYTHING BEYOND THIS POINT
16
17 #if USED_DATATYPE == 1
18     #define CALC_SVD LAPACKE_dgesvd_work
19 #elif USED_DATATYPE == 2
20     #define CALC_SVD LAPACKE_cgesvd_work
21 #else
22     #define CALC_SVD LAPACKE_sgesvd_work
23 #endif
24

```

## Index

Block< datatype >, [3](#)

coarse, [4](#)

operator\*, [4](#)

operator+, [5](#)

coarse

Block< datatype >, [4](#)

EntrywiseBlock< datatype >, [7](#)

HierarchicalMatrix< datatype >, [9](#)

OuterProductBlock< datatype >, [12](#)

DivByZeroException, [6](#)

EntrywiseBlock< datatype >, [6](#)

coarse, [7](#)

operator\*, [7](#)

operator+, [7](#), [8](#)

hierarchical-matrices-lib/Block.hpp, [14](#)

hierarchical-matrices-lib/EntrywiseBlock.hpp, [14](#)

hierarchical-matrices-lib/Exceptions.hpp, [15](#)

hierarchical-matrices-lib/HierarchicalMatrix.hpp, [15](#)

hierarchical-matrices-lib/OuterProductBlock.hpp, [16](#)

hierarchical-matrices-lib/user\_settings.hpp, [16](#)

HierarchicalMatrix

HierarchicalMatrix< datatype >, [9](#)

HierarchicalMatrix< datatype >, [8](#)

coarse, [9](#)

HierarchicalMatrix, [9](#)

operator\*, [10](#)

operator+, [10](#)

operator+=, [10](#)

NonInvertibleMatrixException, [11](#)

operator\*

Block< datatype >, [4](#)

EntrywiseBlock< datatype >, [7](#)

HierarchicalMatrix< datatype >, [10](#)

OuterProductBlock< datatype >, [12](#)

operator+

Block< datatype >, [5](#)

EntrywiseBlock< datatype >, [7](#), [8](#)

HierarchicalMatrix< datatype >, [10](#)

OuterProductBlock< datatype >, [13](#)

operator+=

HierarchicalMatrix< datatype >, [10](#)

OuterProductBlock< datatype >, [11](#)

coarse, [12](#)

operator\*, [12](#)

operator+, [13](#)