Heidelberg Educational Numerics Library

Version 0.24 (from 9 September 2011)

Generated by Doxygen 1.8.6

Wed Oct 21 2015 11:46:59

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

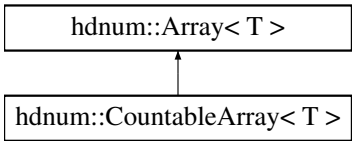
Class Documentation

4.1 hdnum::Array< T > Class Template Reference

A basic dynamic array class.

`#include <array.hh>`

Inheritance diagram for hdnum::Array< T >:



Public Types

- typedef T value_type

  Remember the storage type.
- typedef value_type & reference

  Reference to an object.
- typedef const value_type & const_reference

  Const reference to an object.
- typedef std::size_t size_type

  Type used for array indices.
- typedef std::ptrdiff_t difference_type

  Difference type.

Public Member Functions

- Array ()

  make empty array
- Array (size_type _n)

  make array with _n uninitialized components
- Array (size_type _n, const T &_t)

  make array with _n initialized components
- Array (const Array &a)

  copy constructor

- ~Array ()

  destructor, free dynamic memory
- void resize (size_type _n)

  reallocate array to given size, any data is lost
- void resize (size_type _n, const T &_t)

  reallocate array to given size, any data is lost
- Array & operator= (const Array &a)

  assignment
- reference operator[] (size_type i)

  Component access.
- const_reference operator[] (size_type i) const

  Const component access.
- size_type size () const

  get array size

4.1.1 Detailed Description

template<class T>class hdnum::Array< T >

A basic dynamic array class.

Provides a dyamically allocated array with access operator, resizing and size method.

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

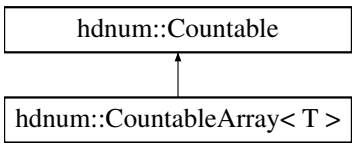
- src/array.hh

4.2 hdnum::Countable Class Reference

Base class for object pointed to by CP.

`#include <countingptr.hh>`

Inheritance diagram for hdnum::Countable:



Public Member Functions

- Countable ()

  Default constructor.
- Countable (const Countable &)

  copy constructor: new object, no pointer exists
- Countable & operator= (const Countable &)

  number of pointers does not change
- void reference_counter_increment () const

  increment reference counter
- void reference_counter_decrement () const

decrement reference counter

- bool reference_counter_zero () const

  check wether the reference counter is zero

- int get_reference_counter () const

  get value of reference counter

- ~Countable ()

  Destructor.

4.2.1 Detailed Description

Base class for object pointed to by CP.

This provides the necessary functionality in the target object for the CP template class to work.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 hdnum::Countable::~Countable ( ) [inline]

Destructor.

Warn if any CP is still pointing to us.

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

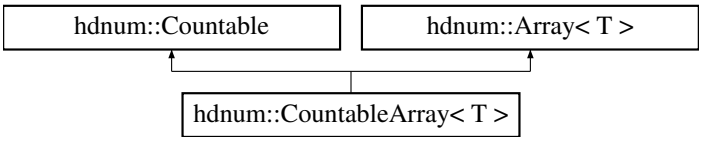
- src/countingptr.hh

4.3 hdnum::CountableArray< T > Class Template Reference

Dynamic array that can be used with the reference counting pointer.

`#include <countablearray.hh>`

Inheritance diagram for hdnum::CountableArray< T >:



Public Types

- typedef std::size_t size_type

  Type used for array indices.

Public Member Functions

- CountableArray ()

  make empty array

- CountableArray (size_type _n)

  make array with _n uninitialized components

- CountableArray (size_type _n, const T &_t)

  make array with _n initialized components

4.3.1 Detailed Description

template<class T>class hdnum::CountableArray< T >

Dynamic array that can be used with the reference counting pointer.

Provides a dyamically allocated array with access operator, resizing and size method.

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

- src/countablearray.hh

4.4 hdnum::CountableException Class Reference

Public Member Functions

- **CountableException** (int i)
- int **get_counter** () const

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

- src/countingptr.hh

4.5 hdnum::CP< T, P > Class Template Reference

Pointer with a reference count in the pointed-to object.

`#include <countingptr.hh>`

Public Member Functions

- CP ()
  Construct a CP object which points to 0.
- CP (T *p_)
  Construct a CP object which points to p_ (which may be 0)
- CP (const CP< T > &cp)
  Copy constructor.
- ~CP ()
  Destructor.
- CP< T > & operator= (T *p_)
  assignment from a C pointer
- CP< T > & operator= (const CP< T > &cp)
  copy operator
- T * operator-> () const
  target element access
- T & operator* () const
  dereference operator
- bool operator== (const CP< T > &cp) const
  check whether both point to same target
- bool operator!= (const CP< T > &cp) const
  check whether target have different adress

4.5.1 Detailed Description

template<typename T, typename P = DeletingMemoryManagementPolicy>class hdnum::CP< T, P >

Pointer with a reference count in the pointed-to object.

Template Parameters

T	The type of the pointed-to object. Must be derived from Countable.
P	What to do when the reference count reaches 0. Two predefined policy classes are available: NondeletingMemoryManagementPolicy and DeletingMemoryManagementPolicy (the default).

An object cp of class CP points to another object of a class derived from Countable, or to 0. If it does not point to 0, it will keep track of how many CP objects point to the same object. If cp stops pointing to the target object, it will decrement its reference count, and if the reference count reaches zero may or may not delete the target object, depending on what the memory managment policy dictates.

cp may be set via assingment from an apropriate C pointer or another CP of the same type. To access the pointed to object, the expressions ∗cp and cp->member may be used, where member is a member of the pointed to object. Finally, CP objects may be compared using == and != to find out whether they point to the same object.

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

- src/countingptr.hh

4.6 hdnum::DeletingMemoryManagementPolicy Class Reference

Delete target if reference count reaches zero.

`#include <countingptr.hh>`

Static Public Member Functions

- template<typename T >
  static void **delete_action** (T ∗p)

4.6.1 Detailed Description

Delete target if reference count reaches zero.

If this class is given to CP as the memory management policy, the CP objects will delete the pointed to object if the reference count reaches zero.

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

- src/countingptr.hh

4.7 hdnum::DenseMatrix< REAL > Class Template Reference

Class with mathematical matrix operations.

`#include <densematrix.hh>`

Public Types

- typedef std::size_t size_type
  Type used for array indices.
- typedef std::vector< REAL > **VType**
- typedef VType::const_iterator **ConstVectorIterator**
- typedef VType::iterator **VectorIterator**

Public Member Functions

- **DenseMatrix** (const std::size_t _rows, const std::size_t _cols, const REAL def_val=0)
- void **addNewRow** (const hdnum::Vector< REAL > &rowvector)
- size_t rowsize () const

  get number of rows of the matrix
- size_t colsize () const

  get number of columns of the matrix
- bool **scientific** () const
- void scientific (bool b) const

  Switch between floating point (default=true) and fixed point (false) display.
- std::size_t iwidth () const

  get index field width for pretty-printing
- std::size_t width () const

  get data field width for pretty-printing
- std::size_t precision () const

  get data precision for pretty-printing
- void iwidth (std::size_t i) const

  set index field width for pretty-printing
- void width (std::size_t i) const

  set data field width for pretty-printing
- void precision (std::size_t i) const

  set data precision for pretty-printing
- REAL & operator() (const std::size_t row, const std::size_t col)

  (i,j)-operator for accessing entries of a (m x n)-matrix directly
- const REAL & **operator()** (const std::size_t row, const std::size_t col) const
- const ConstVectorIterator **operator[]** (const std::size_t row) const
- VectorIterator operator[] (const std::size_t row)

  [i][j]-operator for accessing entries of a (m x n)-matrix directly
- DenseMatrix & operator= (const DenseMatrix &A)

  assignment operator
- DenseMatrix & operator= (const REAL value)

  assignment from a scalar value
- DenseMatrix sub (size_type i, size_type j, size_type rows, size_type cols)

  Submatrix extraction.
- DenseMatrix & operator+= (const DenseMatrix &B)

  Addition assignment.
- DenseMatrix & operator-= (const DenseMatrix &B)

  Subtraction assignment.
- DenseMatrix & operator*= (const REAL s)

  Scalar multiplication assignment.
- DenseMatrix & operator/= (const REAL s)

  Scalar division assignment.
- void update (const REAL s, const DenseMatrix &B)

  Scaled update of a Matrix.
- template<class V >
  void mv (Vector< V > &y, const Vector< V > &x) const

  *matrix vector product y = A*x*
- template<class V >
  void umv (Vector< V > &y, const Vector< V > &x) const

  *update matrix vector product y += A*x*

- template<class V >
  void umv (Vector< V > &y, const V &s, const Vector< V > &x) const

  update matrix vector product y += sA∗x
- void mm (const DenseMatrix< REAL > &A, const DenseMatrix< REAL > &B)

  assign to matrix product C = A∗B to matrix C
- void umm (const DenseMatrix< REAL > &A, const DenseMatrix< REAL > &B)

  add matrix product A∗B to matrix C
- void sc (const Vector< REAL > &x, std::size_t k)

  set column: make x the k'th column of A
- void sr (const Vector< REAL > &x, std::size_t k)

  set row: make x the k'th row of A
- REAL norm_infty () const

  compute row sum norm
- REAL norm_1 () const

  compute column sum norm
- Vector< REAL > operator∗ (const Vector< REAL > &x)

  vector = matrix ∗ vector
- DenseMatrix operator∗ (const DenseMatrix &x) const

  matrix = matrix ∗ matrix
- DenseMatrix operator+ (const DenseMatrix &x) const

  matrix = matrix + matrix
- DenseMatrix operator- (const DenseMatrix &x) const

  matrix = matrix - matrix

Related Functions

(Note that these are not member functions.)

- template<class T >
  void identity (DenseMatrix< T > &A)
- template<typename REAL >
  void spd (DenseMatrix< REAL > &A)
- template<typename REAL >
  void vandermonde (DenseMatrix< REAL > &A, const Vector< REAL > x)
- template<typename REAL >
  void readMatrixFromFile (const std::string &filename, DenseMatrix< REAL > &A)

  Read matrix from a text file.

4.7.1 Detailed Description

template<typename REAL>class hdnum::DenseMatrix< REAL >

Class with mathematical matrix operations.

4.7.2 Member Function Documentation

4.7.2.1 template<typename REAL> size_t hdnum::DenseMatrix< REAL >::colsize ( ) const [inline]

get number of columns of the matrix

Example:

```
hdnum::DenseMatrix<double> A(4,5);
size_t nColumns = A.colsize();
std::cout << "Matrix A has " << nColumns << " columns." << std::endl;
```

Output:

```
Matrix A has 5 columns.
```

4.7.2.2 template<typename REAL> void hdnum::DenseMatrix< REAL >::mm (const DenseMatrix< REAL > & *A*, const DenseMatrix< REAL > & *B*) [inline]

assign to matrix product C = A∗B to matrix C

Implements C = A∗B where A and B are matrices

Parameters

in	*x*	constant reference to a DenseMatrix

in	*x*	constant reference to a DenseMatrix
		
		\b Example:
		
		`hdnum::DenseMatrix<double> A(2,6,1.0);`
		`hdnum::DenseMatrix<double> B(6,3,-1.0);`
		
		`A.scientific(false); // fixed point representation for all DenseMatrix objects`
		`A.width(6); // use at least 6 columns for displaying matrix entries`
		`A.precision(3); // display 3 digits behind the point`
		
		`std::cout << "A =" << A << std::endl;`
		`std::cout << "B =" << B << std::endl;`
		
		`hdnum::DenseMatrix<double> C(2,3);`
		`C.mm(A,B);`
		`std::cout << "C = A*B =" << C << std::endl;`

Output:

```
A =
0      1      2      3      4      5
0   1.000  1.000  1.000  1.000  1.000  1.000
1   1.000  1.000  1.000  1.000  1.000  1.000

B =
0      1      2
0  -1.000 -1.000 -1.000
1  -1.000 -1.000 -1.000
2  -1.000 -1.000 -1.000
3  -1.000 -1.000 -1.000
4  -1.000 -1.000 -1.000
5  -1.000 -1.000 -1.000

C = A*B =
0      1      2
0  -6.000 -6.000 -6.000
1  -6.000 -6.000 -6.000
```

4.7.2.3 template<typename REAL> template<class V > void hdnum::DenseMatrix< REAL >::mv (Vector< V > & *y*, const Vector< V > & *x*) const [inline]

matrix vector product y = A∗x

Implements y = A∗x where x and y are a vectors and A is a matrix x = A∗x is not allowed

Parameters

in	*y*	reference to the resulting Vector

in	*x*	constant reference to a Vector
		
		\b Example:
		
		hdnum::Vector<double> x(3,10.0);
		hdnum::Vector<double> y(2);
		hdnum::DenseMatrix<double> A(2,3,1.0);
		
		x.scientific(false); // fixed point representation for all Vector objects
		A.scientific(false); // fixed point representation for all DenseMatrix objects
		
		std::cout << "A =" << A << std::endl;
		std::cout << "x =" << x << std::endl;
		A.mv(y,x);
		std::cout << "y = A*x =" << y << std::endl;

Output:

```
A =
0           1           2
0       1.000       1.000       1.000
1       1.000       1.000       1.000

x =
[ 0]     10.0000000
[ 1]     10.0000000
[ 2]     10.0000000

y = A*x =
[ 0]     30.0000000
[ 1]     30.0000000
```

4.7.2.4 template<typename REAL> REAL& hdnum::DenseMatrix< REAL >::operator() (const std::size_t *row*, const std::size_t *col*) [inline]

(i,j)-operator for accessing entries of a (m x n)-matrix directly

Parameters

in	*i*	row index (0...m-1)

in	*j*	column index (0...n-1)
		
		\b Example:
		
		hdnum::DenseMatrix<double> A(4,4);
		A.scientific(false); // fixed point representation for all DenseMatrix objects
		A.width(8);
		A.precision(3);
		
		identity(A); // Defines the identity matrix of the same dimension
		std::cout << "A=" << A << std::endl;
		
		std::cout << "reading A(0,0)=" << A(0,0) << std::endl;
		
		std::cout << "resetting A(0,0) and A(2,3)..." << std::endl;
		A(0,0) = 1.234;
		A(2,3) = 432.1;
		
		std::cout << "A=" << A << std::endl;

Output:

```
A=
0         1         2         3
0     1.000     0.000     0.000     0.000
1     0.000     1.000     0.000     0.000
2     0.000     0.000     1.000     0.000
3     0.000     0.000     0.000     1.000
```

```
reading A(0,0)=1.000
resetting A(0,0) and A(2,3)...
A=
0        1        2        3
0    1.234    0.000    0.000    0.000
1    0.000    1.000    0.000    0.000
2    0.000    0.000    1.000  432.100
3    0.000    0.000    0.000    1.000
```

4.7.2.5 template<typename REAL> Vector<REAL> hdnum::DenseMatrix< REAL >::operator* (const Vector< REAL > & *x*) [inline]

vector = matrix * vector

Parameters

in	*x*	constant reference to a Vector
		
		\b Example:
		
		`hdnum::Vector<double> x(3,4.0);`
		`hdnum::DenseMatrix<double> A(3,3,2.0);`
		`hdnum::Vector<double> y(3);`
		
		`A.scientific(false); // fixed point representation for all DenseMatrix objects`
		`A.width(8);`
		`A.precision(1);`
		
		`x.scientific(false); // fixed point representation for all Vector objects`
		`x.width(8);`
		`x.precision(1);`
		
		`std::cout << "A=" << A << std::endl;`
		`std::cout << "x=" << x << std::endl;`
		`y=A*x;`
		`std::cout << "y=A*x" << y << std::endl;`

Output:

```
A=
0        1        2
0      2.0      2.0      2.0
1      2.0      2.0      2.0
2      2.0      2.0      2.0

x=
[ 0]     4.0
[ 1]     4.0
[ 2]     4.0

y=A*x
[ 0]    24.0
[ 1]    24.0
[ 2]    24.0
```

4.7.2.6 template<typename REAL> DenseMatrix hdnum::DenseMatrix< REAL >::operator* (const DenseMatrix< REAL > & *x*) const [inline]

matrix = matrix * matrix

Parameters

in	*x*	constant reference to a DenseMatrix
		
		\b Example:
		
		`hdnum::DenseMatrix<double> A(3,3,2.0);`
		`hdnum::DenseMatrix<double> B(3,3,4.0);`
		`hdnum::DenseMatrix<double> C(3,3);`
		
		`A.scientific(false); // fixed point representation for all DenseMatrix objects`
		`A.width(8);`
		`A.precision(1);`
		
		`std::cout << "A=" << A << std::endl;`
		`std::cout << "B=" << B << std::endl;`
		`C=A*B;`
		`std::cout << "C=A*B=" << C << std::endl;`

Output:

```
A=
0         1         2
0       2.0       2.0       2.0
1       2.0       2.0       2.0
2       2.0       2.0       2.0

B=
0         1         2
0       4.0       4.0       4.0
1       4.0       4.0       4.0
2       4.0       4.0       4.0

C=A*B=
0         1         2
0      24.0      24.0      24.0
1      24.0      24.0      24.0
2      24.0      24.0      24.0
```

4.7.2.7 template<typename REAL> DenseMatrix& hdnum::DenseMatrix< REAL >::operator*= (const REAL *s*) [inline]

Scalar multiplication assignment.

Implements A *= s where s is a scalar

Parameters

in	*s*	scalar value to multiply with
		
		\b Example:
		
		`double s = 0.5;`
		`hdnum::DenseMatrix<double> A(2,3,1.0);`
		`std::cout << "A=" << A << std::endl;`
		`A *= s;`
		`std::cout << "A=" << A << std::endl;`

Output:

```
A=
0           1           2
0   1.000e+00   1.000e+00   1.000e+00
1   1.000e+00   1.000e+00   1.000e+00

0.5*A =
0           1           2
0   5.000e-01   5.000e-01   5.000e-01
1   5.000e-01   5.000e-01   5.000e-01
```

4.7.2.8 template<typename REAL> DenseMatrix hdnum::DenseMatrix< REAL >::operator+ (const DenseMatrix< REAL > & *x*) const [inline]

matrix = matrix + matrix

Parameters

in	*x*	constant reference to a DenseMatrix
		
		\b Example:
		
		hdnum::DenseMatrix<double> A(3,3,2.0);
		hdnum::DenseMatrix<double> B(3,3,4.0);
		hdnum::DenseMatrix<double> C(3,3);
		
		A.scientific(false); // fixed point representation for all DenseMatrix objects
		A.width(8);
		A.precision(1);
		
		std::cout << "A=" << A << std::endl;
		std::cout << "B=" << B << std::endl;
		C=A+B;
		std::cout << "C=A+B=" << C << std::endl;

Output:

```
A=
0         1         2
0       2.0       2.0       2.0
1       2.0       2.0       2.0
2       2.0       2.0       2.0

B=
0         1         2
0       4.0       4.0       4.0
1       4.0       4.0       4.0
2       4.0       4.0       4.0

C=A+B=
0         1         2
0       6.0       6.0       6.0
1       6.0       6.0       6.0
2       6.0       6.0       6.0
```

4.7.2.9 template<typename REAL> DenseMatrix& hdnum::DenseMatrix< REAL >::operator+= (const DenseMatrix< REAL > & *B*) [inline]

Addition assignment.

Implements A += B matrix addition

Parameters

in	*B*	another Matrix

4.7.2.10 template<typename REAL> DenseMatrix hdnum::DenseMatrix< REAL >::operator- (const DenseMatrix< REAL > & *x*) const [inline]

matrix = matrix - matrix

Parameters

in	*x*	constant reference to a DenseMatrix
		
		\b Example:
		
		`hdnum::DenseMatrix<double> A(3,3,2.0);`
		`hdnum::DenseMatrix<double> B(3,3,4.0);`
		`hdnum::DenseMatrix<double> C(3,3);`
		
		`A.scientific(false); // fixed point representation for all DenseMatrix objects`
		`A.width(8);`
		`A.precision(1);`
		
		`std::cout << "A=" << A << std::endl;`
		`std::cout << "B=" << B << std::endl;`
		`C=A-B;`
		`std::cout << "C=A-B=" << C << std::endl;`

Output:

```
A=
0         1         2
0       2.0       2.0       2.0
1       2.0       2.0       2.0
2       2.0       2.0       2.0

B=
0         1         2
0       4.0       4.0       4.0
1       4.0       4.0       4.0
2       4.0       4.0       4.0

C=A-B=
0         1         2
0      -2.0      -2.0      -2.0
1      -2.0      -2.0      -2.0
2      -2.0      -2.0      -2.0
```

4.7.2.11 template<typename REAL> DenseMatrix& hdnum::DenseMatrix< REAL >::operator-= (const DenseMatrix< REAL > & *B*) [inline]

Subtraction assignment.

Implements A -= B matrix subtraction

Parameters

in	*B*	another matrix

4.7.2.12 template<typename REAL> DenseMatrix& hdnum::DenseMatrix< REAL >::operator/= (const REAL *s*) [inline]

Scalar division assignment.

Implements A /= s where s is a scalar

Parameters

in	*s*	scalar value to multiply with
		
		\b Example:
		
		`double s = 0.5;`
		`hdnum::DenseMatrix<double> A(2,3,1.0);`
		`std::cout << "A=" << A << std::endl;`
		`A /= s;`
		`std::cout << "A=" << A << std::endl;`

Output:

```
A=
0           1           2
0   1.000e+00   1.000e+00   1.000e+00
1   1.000e+00   1.000e+00   1.000e+00

A/0.5 =
0           1           2
0   2.000e+00   2.000e+00   2.000e+00
1   2.000e+00   2.000e+00   2.000e+00
```

4.7.2.13 template<typename REAL> DenseMatrix& hdnum::DenseMatrix< REAL >::operator= (const DenseMatrix< REAL > & *A*) [inline]

assignment operator

Example:

```
hdnum::DenseMatrix<double> A(4,4);
spd(A);
hdnum::DenseMatrix<double> B(4,4);
B = A;
std::cout << "B=" << B << std::endl;
```

Output:

```
B=
0           1           2           3
0   4.000e+00 -1.000e+00 -2.500e-01 -1.111e-01
1  -1.000e+00  4.000e+00 -1.000e+00 -2.500e-01
2  -2.500e-01 -1.000e+00  4.000e+00 -1.000e+00
3  -1.111e-01 -2.500e-01 -1.000e+00  4.000e+00
```

4.7.2.14 template<typename REAL> DenseMatrix& hdnum::DenseMatrix< REAL >::operator= (const REAL *value*) [inline]

assignment from a scalar value

Example:

```
hdnum::DenseMatrix<double> A(2,3);
A = 5.432;
A.scientific(false); // fixed point representation for all DenseMatrix objects
A.width(8);
A.precision(3);
std::cout << "A=" << A << std::endl;
```

Output:

```
A=
0        1        2
0    5.432    5.432    5.432
1    5.432    5.432    5.432
```

4.7.2.15 template<typename REAL> VectorIterator hdnum::DenseMatrix< REAL >::operator[] (const std::size_t *row*) [inline]

[i][j]-operator for accessing entries of a (m x n)-matrix directly

Parameters

in	*i*	row index (0...m-1)
in	*j*	column index (0...n-1)

\b Example:

```
hdnum::DenseMatrix<double> A(4,4);
A.scientific(false); // fixed point representation for all DenseMatrix objects
A.width(8);
A.precision(3);

identity(A);  // Defines the identity matrix of the same dimension
std::cout << "A=" << A << std::endl;

std::cout << "reading A[0][0]=" << A[0][0] << std::endl;
std::cout << "resetting A[0][0] and A[2][3]..." << std::endl;
A[0][0] = 1.234;
A[2][3] = 432.1;

std::cout << "A=" << A << std::endl;
```

Output:

```
A=
0         1         2         3
0     1.000     0.000     0.000     0.000
1     0.000     1.000     0.000     0.000
2     0.000     0.000     1.000     0.000
3     0.000     0.000     0.000     1.000

reading A[0][0]=1.000
resetting A[0][0] and A[2][3]...
A=
0         1         2         3
0     1.234     0.000     0.000     0.000
1     0.000     1.000     0.000     0.000
2     0.000     0.000     1.000   432.100
3     0.000     0.000     0.000     1.000
```

4.7.2.16 template<typename REAL> size_t hdnum::DenseMatrix< REAL >::rowsize ( ) const [inline]

get number of rows of the matrix

Example:

```
hdnum::DenseMatrix<double> A(4,5);
size_t nRows = A.rowsize();
std::cout << "Matrix A has " << nRows << " rows." << std::endl;
```

Output:

```
Matrix A has 4 rows.
```

4.7.2.17 template<typename REAL> void hdnum::DenseMatrix< REAL >::sc (const Vector< REAL > & *x*, std::size_t *k*) [inline]

set column: make x the k'th column of A

Parameters

in	*x*	constant reference to a Vector

in	*k*	number of the column of A to be set
		
		\b Example:
		
		hdnum::Vector<double> x(2,434.0);
		hdnum::DenseMatrix<double> A(2,6);
		
		A.scientific(false); // fixed point representation for all DenseMatrix objects
		A.width(8);
		A.precision(1);
		
		std::cout << "original A=" << A << std::endl;
		A.sc(x,3); // redefine fourth column of the matrix
		std::cout << "modified A=" << A << std::endl;

Output:

```
original A=
0        1        2        3        4        5
0      0.0      0.0      0.0      0.0      0.0      0.0
1      0.0      0.0      0.0      0.0      0.0      0.0

modified A=
0        1        2        3        4        5
0      0.0      0.0      0.0    434.0      0.0      0.0
1      0.0      0.0      0.0    434.0      0.0      0.0
```

4.7.2.18 template<typename REAL> void hdnum::DenseMatrix< REAL >::scientific (bool *b*) const [inline]

Switch between floating point (default=true) and fixed point (false) display.

Example:

```
hdnum::DenseMatrix<double> A(4,4);
A.scientific(false); // fixed point representation for all DenseMatrix objects
A.width(8);
A.precision(3);
identity(A);  // Defines the identity matrix of the same dimension
std::cout << "A=" << A << std::endl;
```

Output:

```
A=
0        1        2        3
0    1.000    0.000    0.000    0.000
1    0.000    1.000    0.000    0.000
2    0.000    0.000    1.000    0.000
3    0.000    0.000    0.000    1.000
```

4.7.2.19 template<typename REAL> void hdnum::DenseMatrix< REAL >::sr (const Vector< REAL > & *x*, std::size_t *k*) [inline]

set row: make x the k'th row of A

Parameters

in	*x*	constant reference to a Vector

in	*k*	number of the row of A to be set
		
		\b Example:
		
		hdnum::Vector<double> x(3,434.0);
		hdnum::DenseMatrix<double> A(3,3);
		
		A.scientific(false); // fixed point representation for all DenseMatrix objects
		A.width(8);
		A.precision(1);
		
		std::cout << "original A=" << A << std::endl;
		A.sr(x,1); // redefine second row of the matrix
		std::cout << "modified A=" << A << std::endl;

Output:

```
original A=
0         1         2
0       0.0       0.0       0.0
1       0.0       0.0       0.0
2       0.0       0.0       0.0

modified A=
0         1         2
0       0.0       0.0       0.0
1     434.0     434.0     434.0
2       0.0       0.0       0.0
```

4.7.2.20 template<typename REAL> DenseMatrix hdnum::DenseMatrix< REAL >::sub (size_type *i*, size_type *j*, size_type *rows*, size_type *cols*) [inline]

Submatrix extraction.

Returns a new matrix that is a subset of the components of the given matrix.

Parameters

in	*i*	first row index of the new matrix
in	*j*	first column index of the new matrix
in	*rows*	row size of the new matrix, i.e. it has components [i,i+rows-1]
in	*cols*	column size of the new matrix, i.e. it has components [j,j+m-1]

4.7.2.21 template<typename REAL> void hdnum::DenseMatrix< REAL >::umm (const DenseMatrix< REAL > & *A*, const DenseMatrix< REAL > & *B*) [inline]

add matrix product A∗B to matrix C

Implements C += A∗B where A, B and C are matrices

Parameters

in	*x*	constant reference to a DenseMatrix

in	*x*	constant reference to a DenseMatrix
		
		\b Example:
		
		hdnum::DenseMatrix<double> A(2,6,1.0);
		hdnum::DenseMatrix<double> B(6,3,-1.0);
		hdnum::DenseMatrix<double> C(2,3,0.5);
		
		A.scientific(false); // fixed point representation for all DenseMatrix objects
		A.width(6);
		A.precision(3);
		
		std::cout << "C =" << C << std::endl;
		std::cout << "A =" << A << std::endl;
		std::cout << "B =" << B << std::endl;
		
		C.umm(A,B);
		std::cout << "C + A*B =" << C << std::endl;

Output:

```
C =
0       1       2
0   0.500   0.500   0.500
1   0.500   0.500   0.500

A =
0       1       2       3       4       5
0   1.000   1.000   1.000   1.000   1.000   1.000
```

```
1   1.000  1.000  1.000  1.000  1.000  1.000

B =
0      1      2
0  -1.000 -1.000 -1.000
1  -1.000 -1.000 -1.000
2  -1.000 -1.000 -1.000
3  -1.000 -1.000 -1.000
4  -1.000 -1.000 -1.000
5  -1.000 -1.000 -1.000

C + A*B =
0      1      2
0  -5.500 -5.500 -5.500
1  -5.500 -5.500 -5.500
```

4.7.2.22 template<typename REAL> template<class V > void hdnum::DenseMatrix< REAL >::umv (Vector< V > & *y*, const Vector< V > & *x*) const [inline]

update matrix vector product y += A∗x

Implements y += A∗x where x and y are a vectors and A is a matrix

Parameters

in	*y*	reference to the resulting Vector

in	*x*	constant reference to a Vector
		
		\b Example:
		
		hdnum::Vector<double> x(3,10.0);
		hdnum::Vector<double> y(2,5.0);
		hdnum::DenseMatrix<double> A(2,3,1.0);
		
		x.scientific(false); // fixed point representation for all Vector objects
		A.scientific(false); // fixed point representation for all DenseMatrix objects
		
		std::cout << "y =" << y << std::endl;
		std::cout << "A =" << A << std::endl;
		std::cout << "x =" << x << std::endl;
		A.umv(y,x);
		std::cout << "y = A*x =" << y << std::endl;

Output:

```
y =
[ 0]      5.0000000
[ 1]      5.0000000

A =
0          1          2
0       1.000      1.000      1.000
1       1.000      1.000      1.000

x =
[ 0]     10.0000000
[ 1]     10.0000000
[ 2]     10.0000000

y + A*x =
[ 0]     35.0000000
[ 1]     35.0000000
```

4.7.2.23 template<typename REAL> template<class V > void hdnum::DenseMatrix< REAL >::umv (Vector< V > & *y*, const V & *s*, const Vector< V > & *x*) const [inline]

update matrix vector product y += sA∗x

Implements y += sA∗x where s is a scalar value, x and y are a vectors and A is a matrix

Parameters

in	*y*	reference to the resulting Vector
in	*s*	constant reference to a number type

in	*x*	constant reference to a Vector
		
		\b Example:
		
		`double s=0.5;`
		`hdnum::Vector<double> x(3,10.0);`
		`hdnum::Vector<double> y(2,5.0);`
		`hdnum::DenseMatrix<double> A(2,3,1.0);`
		
		`x.scientific(false); // fixed point representation for all Vector objects`
		`A.scientific(false); // fixed point representation for all DenseMatrix objects`
		
		`std::cout << "y =" << y << std::endl;`
		`std::cout << "A =" << A << std::endl;`
		`std::cout << "x =" << x << std::endl;`
		`A.umv(y,s,x);`
		`std::cout << "y = s*A*x =" << y << std::endl;`

Output:

```
y =
[ 0]      5.0000000
[ 1]      5.0000000

A =
0           1           2
0       1.000       1.000       1.000
1       1.000       1.000       1.000

x =
[ 0]     10.0000000
[ 1]     10.0000000
[ 2]     10.0000000

y = s*A*x =
[ 0]     20.0000000
[ 1]     20.0000000
```

4.7.2.24 template<typename REAL> void hdnum::DenseMatrix< REAL >::update (const REAL *s*, const DenseMatrix< REAL > & *B*) [inline]

Scaled update of a Matrix.

Implements A += s∗B where s is a scalar and B a matrix

Parameters

in	*s*	scalar value to multiply with

in	*B*	another matrix
		
		\b Example:
		
		`double s = 0.5;`
		`hdnum::DenseMatrix<double> A(2,3,1.0);`
		`hdnum::DenseMatrix<double> B(2,3,2.0);`
		`A.update(s,B);`
		`std::cout << "A + s*B =" << A << std::endl;`

Output:

```
A + s*B =
0           1           2
0       1.500       1.500       1.500
1       1.500       1.500       1.500
```

4.7.3 Friends And Related Function Documentation

4.7.3.1 template<class T > void identity (DenseMatrix< T > & *A*) [related]

Function: make identity matrix

```
template<class T>
inline void identity (DenseMatrix<T> &A)
```

Parameters

in	*A*	reference to a DenseMatrix that shall be filled with entries

Example:

```
hdnum::DenseMatrix<double> A(4,4);
identity(A);

A.scientific(false); // fixed point representation for all DenseMatrix objects
A.width(10);
A.precision(5);

std::cout << "A=" << A << std::endl;
```

Output:

```
A=
0          1          2          3
0    1.00000    0.00000    0.00000    0.00000
1    0.00000    1.00000    0.00000    0.00000
2    0.00000    0.00000    1.00000    0.00000
3    0.00000    0.00000    0.00000    1.00000
```

4.7.3.2 template<typename REAL > void readMatrixFromFile (const std::string & *filename*, DenseMatrix< REAL > & *A*) [related]

Read matrix from a text file.

Parameters

in	*filename*	name of the text file

in,out	*A*	reference to a DenseMatrix
		
		\b Example:
		
		hdnum::DenseMatrix<number> L;
		readMatrixFromFile("matrixL.dat", L);
		std::cout << "L=" << L << std::endl;

Output:

```
Contents of "matrixL.dat":
1.000e+00  0.000e+00  0.000e+00
2.000e+00  1.000e+00  0.000e+00
3.000e+00  2.000e+00  1.000e+00

would give:
L=
0          1          2
0   1.000e+00  0.000e+00  0.000e+00
1   2.000e+00  1.000e+00  0.000e+00
2   3.000e+00  2.000e+00  1.000e+00
```

4.7.3.3 template<typename REAL > void spd (DenseMatrix< REAL > & *A*) [related]

Function: make a symmetric and positive definite matrix

```
template<typename REAL>
inline void spd (DenseMatrix<REAL> &A)
```

Parameters

in	*A*	reference to a DenseMatrix that shall be filled with entries

Example:

```
hdnum::DenseMatrix<double> A(4,4);
spd(A);

A.scientific(false); // fixed point representation for all DenseMatrix objects
A.width(10);
A.precision(5);

std::cout << "A=" << A << std::endl;
```

Output:

```
A=
0           1           2           3
0     4.00000    -1.00000    -0.25000    -0.11111
1    -1.00000     4.00000    -1.00000    -0.25000
2    -0.25000    -1.00000     4.00000    -1.00000
3    -0.11111    -0.25000    -1.00000     4.00000
```

4.7.3.4 template<typename REAL > void vandermonde (DenseMatrix< REAL > & *A*, const Vector< REAL > *x*) [related]

Function: make a vandermonde matrix

```
template<typename REAL>
inline void vandermonde (DenseMatrix<REAL> &A, const Vector<REAL> x)
```

Parameters

in	*A*	reference to a DenseMatrix that shall be filled with entries
in	*x*	constant reference to a Vector

Example:

```
hdnum::Vector<double> x(4);
fill(x,2.0,1.0);
hdnum::DenseMatrix<double> A(4,4);
vandermonde(A,x);

A.scientific(false); // fixed point representation for all DenseMatrix objects
A.width(10);
A.precision(5);

x.scientific(false); // fixed point representation for all Vector objects
x.width(10);
x.precision(5);

std::cout << "x=" << x << std::endl;
std::cout << "A=" << A << std::endl;
```

Output:

```
x=
[ 0]    2.00000
[ 1]    3.00000
[ 2]    4.00000
[ 3]    5.00000

A=
0           1           2           3
0     1.00000     2.00000     4.00000     8.00000
1     1.00000     3.00000     9.00000    27.00000
2     1.00000     4.00000    16.00000    64.00000
3     1.00000     5.00000    25.00000   125.00000
```

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

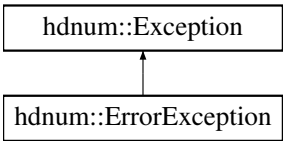
- src/densematrix.hh

4.8 hdnum::ErrorException Class Reference

General Error.

`#include <exceptions.hh>`

Inheritance diagram for hdnum::ErrorException:



Additional Inherited Members

4.8.1 Detailed Description

General Error.

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

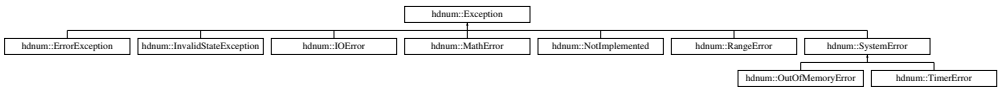
- src/exceptions.hh

4.9 hdnum::Exception Class Reference

Base class for Exceptions.

`#include <exceptions.hh>`

Inheritance diagram for hdnum::Exception:



Public Member Functions

- void message (const std::string &message)

  store string in internal message buffer
- const std::string & what () const

  output internal message buffer

4.9.1 Detailed Description

Base class for Exceptions.

all HDNUM exceptions are derived from this class via trivial subclassing:

```
class MyException : public Dune::Exception {};
```

You should not `throw` a Dune::Exception directly but use the macro DUNE_THROW() instead which fills the message-buffer of the exception in a standard way and features a way to pass the result in the operator<<-style

See Also

HDNUM_THROW, IOError, MathError

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

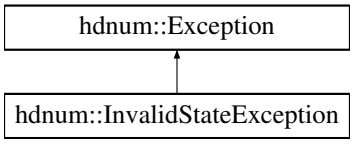
- src/exceptions.hh

4.10 hdnum::InvalidStateException Class Reference

Default exception if a function was called while the object is not in a valid state for that function.

`#include <exceptions.hh>`

Inheritance diagram for hdnum::InvalidStateException:



Additional Inherited Members

4.10.1 Detailed Description

Default exception if a function was called while the object is not in a valid state for that function.

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

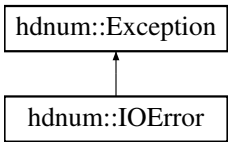
- src/exceptions.hh

4.11 hdnum::IOError Class Reference

Default exception class for I/O errors.

`#include <exceptions.hh>`

Inheritance diagram for hdnum::IOError:



Additional Inherited Members

4.11.1 Detailed Description

Default exception class for I/O errors.

This is a superclass for any errors dealing with file/socket I/O problems like

- file not found
- could not write file
- could not connect to remote socket

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

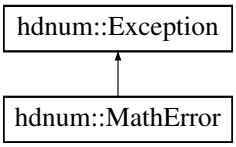
- src/exceptions.hh

4.12 hdnum::MathError Class Reference

Default exception class for mathematical errors.

`#include <exceptions.hh>`

Inheritance diagram for hdnum::MathError:



Additional Inherited Members

4.12.1 Detailed Description

Default exception class for mathematical errors.

This is the superclass for all errors which are caused by mathematical problems like

- matrix not invertible
- not convergent

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

- src/exceptions.hh

4.13 hdnum::NondeletingMemoryManagementPolicy Class Reference

Don't delete target if reference count reaches zero.

`#include <countingptr.hh>`

Static Public Member Functions

- template<typename T >
  static void **delete_action** (T *p)

4.13.1 Detailed Description

Don't delete target if reference count reaches zero.

If this class is given to CP as the memory management policy, the CP objects won't delete the pointed to object if the reference count reaches zero.

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

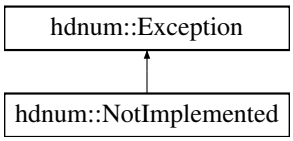
- src/countingptr.hh

4.14 hdnum::NotImplemented Class Reference

Default exception for dummy implementations.

`#include <exceptions.hh>`

Inheritance diagram for hdnum::NotImplemented:



Additional Inherited Members

4.14.1 Detailed Description

Default exception for dummy implementations.

This exception can be used for functions/methods

- that have to be implemented but should never be called
- that are missing

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

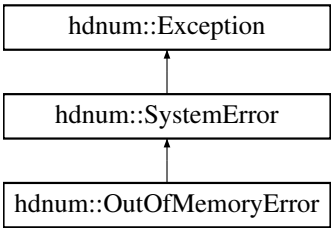
- src/exceptions.hh

4.15 hdnum::OutOfMemoryError Class Reference

Default exception if memory allocation fails.

`#include <exceptions.hh>`

Inheritance diagram for hdnum::OutOfMemoryError:



Additional Inherited Members

4.15.1 Detailed Description

Default exception if memory allocation fails.

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

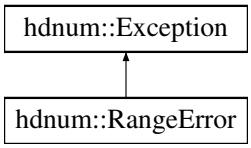
- src/exceptions.hh

4.16 hdnum::RangeError Class Reference

Default exception class for range errors.

`#include <exceptions.hh>`

Inheritance diagram for hdnum::RangeError:



Additional Inherited Members

4.16.1 Detailed Description

Default exception class for range errors.

This is the superclass for all errors which are caused because the user tries to access data that was not allocated before. These can be problems like

- accessing array entries behind the last entry
- adding the fourth non zero entry in a sparse matrix with only three non zero entries per row

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

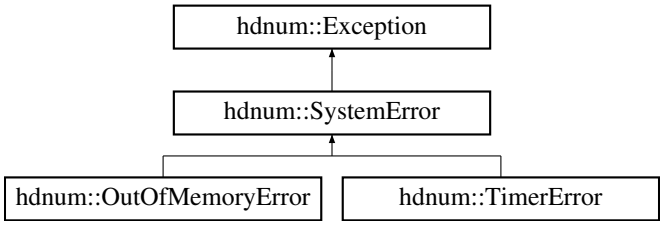
- src/exceptions.hh

4.17 hdnum::SystemError Class Reference

Default exception class for OS errors.

`#include <exceptions.hh>`

Inheritance diagram for hdnum::SystemError:



Additional Inherited Members

4.17.1 Detailed Description

Default exception class for OS errors.

This class is thrown when a system-call is used and returns an error.

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

- src/exceptions.hh

4.18 hdnum::Timer Class Reference

A simple stop watch.

`#include <timer.hh>`

Public Member Functions

- Timer () throw (TimerError)

  A new timer, start immediately.
- void reset () throw (TimerError)

  Reset timer.
- double elapsed () const throw (TimerError)

  Get elapsed user-time in seconds.

4.18.1 Detailed Description

A simple stop watch.

This class reports the elapsed user-time, i.e. time spent computing, after the last call to Timer::reset(). The results are seconds and fractional seconds. Note that the resolution of the timing depends on your OS kernel which should be somewhere in the milisecond range.

The class is basically a wrapper for the libc-function getrusage()

Taken from the DUNE project www.dune-project.org

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

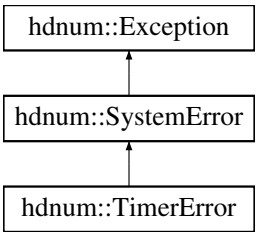
- src/timer.hh

4.19 hdnum::TimerError Class Reference

Exception thrown by the Timer class

`#include <timer.hh>`

Inheritance diagram for hdnum::TimerError:



Additional Inherited Members

4.19.1 Detailed Description

Exception thrown by the Timer class

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

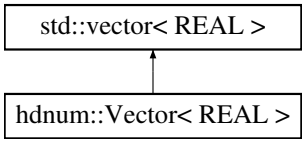
- src/timer.hh

4.20 hdnum::Vector< REAL > Class Template Reference

Class with mathematical vector operations.

`#include <vector.hh>`

Inheritance diagram for hdnum::Vector< REAL >:



Public Types

- typedef std::size_t size_type

  Type used for array indices.

Public Member Functions

- **Vector** (const size_t size, const REAL defaultvalue_=0)
- Vector & operator= (const REAL value)

  Assign all values of the Vector from one scalar value: x = value.
- Vector sub (size_type i, size_type m)

  Subvector extraction.
- Vector & operator*= (const REAL value)

  Assigning a vector from a given vector: x = y.
- Vector & **operator/=** (const REAL value)
- Vector & **operator+=** (const Vector &y)
- Vector & **operator-=** (const Vector &y)
- Vector & **update** (const REAL alpha, const Vector &y)
- REAL operator* (Vector &x) const

  Inner product with another vector.

- Vector operator+ (Vector &x) const

  Adding two vectors x+y.
- Vector operator- (Vector &x) const

  vector subtraction x-y
- REAL two_norm_2 () const

  Square of the Euclidean norm.
- REAL two_norm () const

  Euclidean norm of a vector.
- bool scientific () const

  pretty-print output property: true = scientific, false = fixed point representation
- void scientific (bool b) const

  scientific(true) is the default, scientific(false) switches to the fixed point representation
- std::size_t **width** () const
- void **width** (std::size_t i) const
- std::size_t **iwidth** () const
- void **iwidth** (std::size_t i) const
- std::size_t **precision** () const
- void **precision** (std::size_t i) const

Related Functions

(Note that these are not member functions.)

- template<typename REAL >
  std::ostream & operator<< (std::ostream &os, const Vector< REAL > &x)

  Output operator for Vector.
- template<typename REAL >
  void gnuplot (const std::string &fname, const Vector< REAL > x)

  Output contents of a Vector x to a text file named fname.
- template<typename REAL >
  void readVectorFromFile (const std::string &filename, Vector< REAL > &vector)

  Read vector from a text file.
- template<class REAL >
  void fill (Vector< REAL > &x, const REAL &t, const REAL &dt)

  Fill vector, with entries starting at t, consecutively shifted by dt.
- template<class REAL >
  void unitvector (Vector< REAL > &x, std::size_t j)

  Defines j-th unitvector (j=0,...,n-1) where n = length of the vector.

4.20.1 Detailed Description

template<typename REAL>class hdnum::Vector< REAL >

Class with mathematical vector operations.

4.20.2 Member Function Documentation

4.20.2.1 template<typename REAL> REAL hdnum::Vector< REAL >::operator* (Vector< REAL > & *x*) const [inline]

Inner product with another vector.

Example:

```
hdnum::Vector<double> x(2);
x.scientific(false); // set fixed point display mode
x[0] = 12.0;
x[1] = 3.0;
std::cout << "x=" << x << std::endl;
hdnum::Vector<double> y(2);
y[0] = 4.0;
y[1] = -1.0;
std::cout << "y=" << y << std::endl;
double s = x*y;
std::cout << "s = x*y = " << s << std::endl;
```

Output:

```
x=
[ 0]     12.0000000
[ 1]      3.0000000

y=
[ 0]      4.0000000
[ 1]     -1.0000000

s = x*y = 45.0000000
```

4.20.2.2 template<typename REAL> Vector& hdnum::Vector< REAL >::operator*= (const REAL *value*) [inline]

Assigning a vector from a given vector: x = y.

Example:

```
hdnum::Vector<double> x(4);
hdnum::Vector<double> y(4);
x[0] = 1.23;
x[1] = 2.31;
x[2] = 4.54;
x[3] = 9.98;
std::cout << "x=" << x << std::endl;
y = x;
std::cout << "y=" << y << std::endl;
```

Output:

```
x=
[ 0]  1.2300000e+00
[ 1]  2.3100000e+00
[ 2]  4.5400000e+00
[ 3]  9.9800000e+00

y=
[ 0]  1.2300000e+00
[ 1]  2.3100000e+00
[ 2]  4.5400000e+00
[ 3]  9.9800000e+00
```

4.20.2.3 template<typename REAL> Vector hdnum::Vector< REAL >::operator+ (Vector< REAL > & *x*) const [inline]

Adding two vectors x+y.

Example:

```
hdnum::Vector<double> x(2);
x.scientific(false); // set fixed point display mode
x[0] = 12.0;
x[1] = 3.0;
std::cout << "x=" << x << std::endl;
hdnum::Vector<double> y(2);
y[0] = 4.0;
y[1] = -1.0;
std::cout << "y=" << y << std::endl;
std::cout << "x+y = " << x+y << std::endl;
```

Output:

```
x=
[ 0]      12.0000000
[ 1]       3.0000000

y=
[ 0]       4.0000000
[ 1]      -1.0000000

x+y =
[ 0]      16.0000000
[ 1]       2.0000000
```

4.20.2.4 template<typename REAL> Vector hdnum::Vector< REAL >::operator- (Vector< REAL > & *x*) const [inline]

vector subtraction x-y

Example:

```
hdnum::Vector<double> x(2);
x.scientific(false); // set fixed point display mode
x[0] = 12.0;
x[1] = 3.0;
std::cout << "x=" << x << std::endl;
hdnum::Vector<double> y(2);
y[0] = 4.0;
y[1] = -1.0;
std::cout << "y=" << y << std::endl;
std::cout << "x-y = " << x-y << std::endl;
```

Output:

```
x=
[ 0]      12.0000000
[ 1]       3.0000000

y=
[ 0]       4.0000000
[ 1]      -1.0000000

x-y =
[ 0]       8.0000000
[ 1]       4.0000000
```

4.20.2.5 template<typename REAL> Vector& hdnum::Vector< REAL >::operator= (const REAL *value*) [inline]

Assign all values of the Vector from one scalar value: x = value.

Example:

```
hdnum::Vector<double> x(4);
x = 1.23;
std::cout << "x=" << x << std::endl;
```

Output:

```
x=
[ 0]  1.2340000e+00
[ 1]  1.2340000e+00
[ 2]  1.2340000e+00
[ 3]  1.2340000e+00
```

4.20.2.6 template<typename REAL> void hdnum::Vector< REAL >::scientific (bool *b*) const [inline]

scientific(true) is the default, scientific(false) switches to the fixed point representation

Example:

```
hdnum::Vector<double> x(3);
x[0] = 2.0;
x[1] = 2.0;
x[2] = 1.0;
std::cout << "x=" << x << std::endl;
x.scientific(false); // set fixed point display mode
std::cout << "x=" << x << std::endl;
```

Output:

```
x=
[ 0]  2.0000000e+00
[ 1]  2.0000000e+00
[ 2]  1.0000000e+00

x=
[ 0]      2.0000000
[ 1]      2.0000000
[ 2]      1.0000000
```

4.20.2.7 template<typename REAL> Vector hdnum::Vector< REAL >::sub (size_type *i*, size_type *m*) [inline]

Subvector extraction.

Returns a new vector that is a subset of the components of the given vector.

Parameters

in	*i*	first index of the new vector
in	*m*	size of the new vector, i.e. it has components [i,i+m-1]

4.20.2.8 template<typename REAL> REAL hdnum::Vector< REAL >::two_norm ( ) const [inline]

Euclidean norm of a vector.

Example:

```
hdnum::Vector<double> x(3);
x.scientific(false); // set fixed point display mode
x[0] = 2.0;
x[1] = 2.0;
x[2] = 1.0;
std::cout << "x=" << x << std::endl;
std::cout << "euclidean norm of x = " << x.two_norm() << std::endl;
```

Output:

```
x=
[ 0]      2.0000000
[ 1]      2.0000000
[ 2]      1.0000000

euclidean norm of x = 3.0000000
```

4.20.3 Friends And Related Function Documentation

4.20.3.1 template<class REAL > void fill (Vector< REAL > & *x*, const REAL & *t*, const REAL & *dt*) [related]

Fill vector, with entries starting at t, consecutively shifted by dt.

Example:

```
hdnum::Vector<double> x(5);
fill(x,2.01,0.1);
x.scientific(false); // set fixed point display mode
std::cout << "x=" << x << std::endl;
```

Output:

```
x=
[ 0]       2.0100000
[ 1]       2.1100000
[ 2]       2.2100000
[ 3]       2.3100000
[ 4]       2.4100000
```

4.20.3.2 template<typename REAL > void gnuplot (const std::string & *fname*, const Vector< REAL > *x*) [related]

Output contents of a Vector x to a text file named fname.

Example:

```
hdnum::Vector<double> x(5);
unitvector(x,3);
x.scientific(false); // set fixed point display mode
gnuplot("test.dat",x);
```

Output:

```
Contents of 'test.dat':
0       0.0000000
1       0.0000000
2       0.0000000
3       1.0000000
4       0.0000000
```

4.20.3.3 template<typename REAL > std::ostream & operator<< (std::ostream & *os*, const Vector< REAL > & *x*) [related]

Output operator for Vector.

Example:

```
hdnum::Vector<double> x(3);
x[0] = 2.0;
x[1] = 2.0;
x[2] = 1.0;
std::cout << "x=" << x << std::endl;
```

Output:

```
x=
[ 0]  2.0000000e+00
[ 1]  2.0000000e+00
[ 2]  1.0000000e+00
```

4.20.3.4 template<typename REAL > void readVectorFromFile (const std::string & *filename*, Vector< REAL > & *vector*) [related]

Read vector from a text file.

Parameters

in	*filename*	name of the text file
in,out	*x*	reference to a Vector

Example:

```
hdnum::Vector<number> x;
readVectorFromFile("x.dat", x );
std::cout << "x=" << x << std::endl;
```

Output:

```
Contents of "x.dat":
1.0
2.0
3.0

would give:
x=
[ 0]  1.0000000e+00
[ 1]  2.0000000e+00
[ 2]  3.0000000e+00
```

4.20.3.5 template<class REAL > void unitvector (Vector< REAL > & *x*, std::size_t *j*) [related]

Defines j-th unitvector (j=0,...,n-1) where n = length of the vector.

Example:

```
hdnum::Vector<double> x(5);
unitvector(x,3);
x.scientific(false); // set fixed point display mode
std::cout << "x=" << x << std::endl;
```

Output:

```
x=
[ 0]       0.0000000
[ 1]       0.0000000
[ 2]       0.0000000
[ 3]       1.0000000
[ 4]       0.0000000
```

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

- src/vector.hh

Chapter 5

File Documentation

5.1 src/array.hh File Reference

This file implements a basic dynamic array class.

Classes

- class hdnum::Array< T >

  A basic dynamic array class.

5.1.1 Detailed Description

This file implements a basic dynamic array class.

5.2 src/countablearray.hh File Reference

This file implements a basic dynamic array class.

```
#include "countingptr.hh"
#include "array.hh"
```

Classes

- class hdnum::CountableArray< T >

  Dynamic array that can be used with the reference counting pointer.

5.2.1 Detailed Description

This file implements a basic dynamic array class.

5.3 src/countingptr.hh File Reference

This file implements a counting pointer with configurable memory management policy Adapted from dune-pdelab.

```
#include <iostream>
```

Classes

- class hdnum::NondeletingMemoryManagementPolicy

  Don't delete target if reference count reaches zero.
- class hdnum::DeletingMemoryManagementPolicy

  Delete target if reference count reaches zero.
- class hdnum::CP< T, P >

  Pointer with a reference count in the pointed-to object.
- class hdnum::CountableException
- class hdnum::Countable

  Base class for object pointed to by CP.

5.3.1 Detailed Description

This file implements a counting pointer with configurable memory management policy Adapted from dune-pdelab.

5.4 src/exceptions.hh File Reference

A few common exception classes.

```
#include <string>
#include <sstream>
```

Classes

- class hdnum::Exception

  Base class for Exceptions.
- class hdnum::IOError

  Default exception class for I/O errors.
- class hdnum::MathError

  Default exception class for mathematical errors.
- class hdnum::RangeError

  Default exception class for range errors.
- class hdnum::NotImplemented

  Default exception for dummy implementations.
- class hdnum::SystemError

  Default exception class for OS errors.
- class hdnum::OutOfMemoryError

  Default exception if memory allocation fails.
- class hdnum::InvalidStateException

  Default exception if a function was called while the object is not in a valid state for that function.
- class hdnum::ErrorException

  General Error.

Macros

- #define **THROWSPEC**(E) #E << ": "
- #define HDNUM_THROW(E, m)
- #define **HDNUM_ERROR**(m)

Functions

- std::ostream & **hdnum::operator**<< (std::ostream &stream, const Exception &e)

5.4.1 Detailed Description

A few common exception classes. This file defines a common framework for generating exception subclasses and to throw them in a simple manner. Taken from the DUNE project www.dune-project.org

5.4.2 Macro Definition Documentation

5.4.2.1 #define HDNUM_ERROR(*m*)

Value:

```
do { hdnum::ErrorException th__ex; std::ostringstream th__out;             \
        th__out << THROWSPEC(hdnum::ErrorException) << m; \
        th__ex.message(th__out.str()); \
        std::cout << th__ex.what() << std::endl; \
        throw th__ex;                                \
  } while (0)
```

5.4.2.2 #define HDNUM_THROW(*E, m*)

Value:

```
do { E th__ex; std::ostringstream th__out;              \
        th__out << THROWSPEC(E) << m; th__ex.message(th__out.str()); throw th__ex; \
  } while (0)
```

Macro to throw an exception

Parameters

E	exception class derived from Dune::Exception
m	reason for this exception in ostream-notation

Example:

```
    if (filehandle == 0)
DUNE_THROW(FileError, "Could not open " << filename << " for reading!")
```

DUNE_THROW automatically adds information about the exception thrown to the text. If DUNE_DEVEL_MODE is defined more detail about the function where the exception happened is included. This mode can be activated via the `-enable-dunedevel` switch of ./configure

5.5 src/precision.hh File Reference

find machine precision for given float type

Functions

- template<typename X >
  int **hdnum::precision** (X &eps)

5.5.1 Detailed Description

find machine precision for given float type

5.6 src/timer.hh File Reference

A simple timing class.

```
#include <sys/resource.h>
#include <ctime>
#include <cstring>
#include <cerrno>
#include "exceptions.hh"
```

Classes

- class hdnum::TimerError

  Exception thrown by the Timer class
- class hdnum::Timer

  A simple stop watch.

5.6.1 Detailed Description

A simple timing class.

Index