LA library

0.6

Generated by Doxygen 1.5.8

Wed Sep 8 18:01:42 2010

Contents

Chapter 1

Class 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

Class Documentation

3.1 LA::gencmp< T, type > Struct Template Reference

!! gencmp's and genswap are critical for performance, make sure that compiler really inlines them

Static Public Member Functions

- static SPMatindexdiff **EXEC** (register const SPMatindex i, register const SPMatindex j)

3.1.1 Detailed Description

template<class T, int type> struct LA::gencmp< T, type >

!! gencmp's and genswap are critical for performance, make sure that compiler really inlines them

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

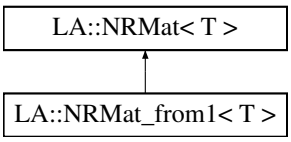
- sparsemat.cc

3.2 LA::NRMat< T > Class Template Reference

NRMat<T> class template implementing the matrix interface.

`#include <mat.h>`

Inheritance diagram for LA::NRMat< T >::



Public Member Functions

- ~NRMat ()

  standard destructor

- NRMat ()

  inlined constructor creating zero matrix of general type `T`

- NRMat (const int n, const int m, const GPUID loc=undefined)

  Inlined constructor creating matrix of given size and location. Because of performance reasons, no incialization is done.

- NRMat (const T &a, const int n, const int m, const GPUID loc)

  inlined constructor creating matrix of given size filled with prescribed value and stored at given location

- NRMat (const T &a, const int n, const int m)

  inlined constructor creating matrix of given size filled with prescribed value

- NRMat (const T *a, const int n, const int m)

  inlined constructor creating matrix of given size filled with data located at given memory location

- NRMat (const NRMat &rhs)

  inlined copy-constructor

- NRMat (const typename LA_traits_complex< T >::NRMat_Noncomplex_type &rhs, bool imagpart=false)

  complexifying constructor

- NRMat (const NRSMat< T > &rhs)

  explicit constructor converting symmetric matrix stored in packed form into a `NRMat<T>` *object*

- NRMat (const NRVec< T > &rhs, const int n, const int m, const int offset=0)

  explicit constructor converting vector into a `NRMat<T>` *object*

- const bool **operator!=** (const NRMat &rhs) const
- const bool **operator==** (const NRMat &rhs) const

- int getcount () const
  determine the count of references to this object
- void copyonwrite ()
  ensure that the data of this matrix are referenced exactly once
- GPUID getlocation () const
- void **moveto** (const GPUID dest)
- void randomize (const typename LA_traits< T >::normtype &x)
  fill the matrix with pseudorandom numbers (uniform distribution)
- NRMat & operator= (const NRMat &rhs)
  assigment operator performing shallow copy
- NRMat & operator|= (const NRMat &rhs)
  assigment operator performing deep copy
- NRMat & operator= (const T &a)
  assign scalar value to the diagonal elements
- NRMat & operator+= (const T &a)
  add scalar value to the diagonal elements
- NRMat & operator-= (const T &a)
  subtract scalar value to the diagonal elements
- NRMat & operator*= (const T &a)
  multiply by a scalar value
- NRMat & operator+= (const NRMat &rhs)
  add given matrix
- NRMat & operator-= (const NRMat &rhs)
  subtract given matrix
- NRMat & operator^= (const NRMat< T > &rhs)
  Hadamard element-wise product.
- NRMat & operator+= (const NRSMat< T > &rhs)
  add symmetric matrix stored in packed form
- NRMat & operator-= (const NRSMat< T > &rhs)
  subtract symmetric matrix stored in packed form
- const NRMat operator- () const
  unary minus
- const NRMat operator+ (const T &a) const
  add scalar value to all matrix elements and return the result by value

- const NRMat operator- (const T &a) const
  subtract scalar value from all matrix elements and return the result by value
- const NRMat operator∗ (const T &a) const
  multiply all matrix elements by a scalar value and return the result by value
- const NRMat operator+ (const NRMat &rhs) const
  add given matrix and return the result by value
- const NRMat operator+ (const NRSMat< T > &rhs) const
  add given symmetric matrix stored in packed form and return the result by value
- const NRMat operator- (const NRMat &rhs) const
  subtract given matrix and return the result by value
- const NRMat operator- (const NRSMat< T > &rhs) const
  subtract given symmetric matrix stored in packed form and return the result by value
- const NRMat operator∗ (const NRMat &rhs) const
  multiply by given matrix and return the result by value
- const NRMat operator∗ (const NRSMat< T > &rhs) const
  multiply by given symmetric matrix stored in packed form and return the result by value
- const NRMat operator& (const NRMat &rhs) const
  direct sum of two matrices
- const NRMat operator| (const NRMat< T > &rhs) const
  direct product of two matrices
- const NRVec< T > operator∗ (const NRVec< T > &rhs) const
  multiply by a vector
- const NRVec< complex< T > > operator∗ (const NRVec< complex< T > > &rhs) const
  multiply this matrix of general type `T` *by vector of type* `complex<T>`
- const T dot (const NRMat &rhs) const
  inner product of two matrices (taking conjugation into account in the complex case)
- const NRMat oplus (const NRMat &rhs) const
  direct sum
- const NRMat otimes (const NRMat &rhs, bool reversecolumns=false) const
  direct product
- void diagmultl (const NRVec< T > &rhs)
  multiply by diagonal matrix from left
- void diagmultr (const NRVec< T > &rhs)
  multiply by diagonal matrix from right

- const NRSMat< T > transposedtimes () const
  for this matrix A compute $A^{\mathrm{T}} \cdot A$
- const NRSMat< T > timestransposed () const
  for this matrix A compute $A \cdot A^{\mathrm{T}}$
- const NRVec< T > rsum () const
  sum the rows
- const NRVec< T > csum () const
  sum the columns
- void orthonormalize (const bool rowcol, const NRSMat< T > *metric=NULL)
  orthonormalize this matrix
- const NRVec< T > row (const int i, int l=-1) const
  get the i^{th} row
- const NRVec< T > column (const int j, int l=-1) const
  get the j^{th} column
- const T * diagonalof (NRVec< T > &, const bool divide=0, bool cache=false) const
  extract the digonal elements of this matrix and store them into a vector
- void diagonalset (const NRVec< T > &)
  set diagonal elements
- void gemv (const T beta, NRVec< T > &r, const char trans, const T alpha, const NRVec< T > &x) const
  perform the **gemv** *operation with vector of type* `T`
- void gemv (const T beta, NRVec< complex< T > > &r, const char trans, const T alpha, const NRVec< complex< T > > &x) const
  perform the **gemv** *operation with vector of type* `complex<T>`
- T * operator[ ] (const int i)
  determine the pointer to the i^{th} row
- const T * operator[ ] (const int i) const
  determine the const pointer to the i^{th} row
- T & operator() (const int i, const int j)
  get the reference to the element with indices (i,j)
- const T & operator() (const int i, const int j) const
  get the const reference to the element with indices (i,j)
- const T get_ij (const int i, const int j) const

get the copy of the element with indices (i,j)

- int nrows () const
  get the number of rows
- int ncols () const
  get the number of columns
- int size () const
  get the number of matrix elements
- void get (int fd, bool dimensions=1, bool transposed=false)
  unformatted input
- void put (int fd, bool dimensions=1, bool transposed=false) const
  unformatted output
- void fprintf (FILE *f, const char *format, const int modulo) const
  formatted output
- void fscanf (FILE *f, const char *format)
  formatted input
- void clear ()
  set all matrix elements equal to zero
- void resize (int n, int m)
  resize the matrix
- operator T * ()
  get the pointer to the data
- operator const T * () const
  get the const pointer to the data
- NRMat & transposeme (const int n=0)
  in case of square matrix, transpose the leading minor of order `n`
- NRMat & conjugateme ()
  conjugate a square matrix
- const NRMat transpose (bool conj=false) const
  transpose this matrix and return the result by value
- const NRMat conjugate () const
  conjugate this matrix and return the result by value
- const NRMat submatrix (const int fromrow, const int torow, const int fromcol, const int tocol) const
  extract specified submatrix

- void storesubmatrix (const int fromrow, const int fromcol, const NRMat &rhs)

  store given matrix at given position into the current matrix

- void gemm (const T &beta, const NRMat &a, const char transa, const NRMat &b, const char transb, const T &alpha)

  perform the **gemm** *operation*

- const LA_traits< T >::normtype norm (const T scalar=(T) 0) const

  compute the norm of this matrix

- void axpy (const T alpha, const NRMat &x)

  add up a scalar multiple of given matrix to the current matrix

- const T amax () const

  maximal element in the absolute value

- const T amin () const

  minimal element in the absolute value

- const T trace () const

  determine the sum of the diagonal elements

- NRMat & swap_rows ()

  swap the order of the rows of the current matrix

- NRMat & swap_cols ()

  swap the order of the columns of the current matrix

- NRMat & swap_rows_cols ()

  swap the order of the rows and columns of the current matrix

- SparseSMat< T > operator* (const SparseSMat< T > &rhs) const

  multiply by sparse matrix

- NRMat (const SparseMat< T > &rhs)

  explicit constructor converting sparse matrix into `NRMat<T>` *object*

- NRMat (const SparseSMat< T > &rhs)

  explicit constructor converting sparse symmetric matrix into `NRMat<T>` *object*

- NRMat & operator+= (const SparseMat< T > &rhs)

  add up given sparse matrix

- NRMat & operator-= (const SparseMat< T > &rhs)

  subtract given sparse matrix

- void gemm (const T &beta, const SparseMat< T > &a, const char transa, const NRMat &b, const char transb, const T &alpha)

  perform the **gemm** *operation*

- void **simplify** ()
- bool **issymmetric** () const
- void strassen (const T beta, const NRMat &a, const char transa, const NRMat &b, const char transb, const T alpha)

  Strassen's multiplication (better than $O(n^3)$, analogous syntax to.

- void **s_cutoff** (const int, const int, const int, const int) const
- template<>
  NRMat< double > & operator= (const double &a)
- template<>
  NRMat< complex< double > > & operator= (const complex< double > &a)
- template<>
  NRMat< double > & operator+= (const double &a)
- template<>
  NRMat< complex< double > > & operator+= (const complex< double > &a)
- template<>
  NRMat< double > & operator-= (const double &a)
- template<>
  NRMat< complex< double > > & operator-= (const complex< double > &a)
- template<>
  const NRMat< double > operator- () const
- template<>
  const NRMat< complex< double > > operator- () const
- template<>
  const NRVec< double > csum () const
- template<>
  const NRVec< complex< double > > csum () const
- template<>
  const NRVec< double > rsum () const
- template<>
  const NRVec< complex< double > > rsum () const
- template<>
  NRMat (const NRMat< double > &rhs, bool imagpart)
- template<>
  const NRSMat< double > transposedtimes () const
- template<>
  const NRSMat< complex< double > > transposedtimes () const
- template<>
  const NRSMat< double > timestransposed () const
- template<>
  const NRSMat< complex< double > > timestransposed () const
- template<>
  void randomize (const double &x)
- template<>
  void randomize (const double &x)
- template<>
  NRMat< double > & operator*= (const double &a)
- template<>
  NRMat< complex< double > > & operator*= (const complex< double > &a)
- template<>
  NRMat< double > & operator+= (const NRMat< double > &rhs)

- template<>
  NRMat< complex< double > > & operator+= (const NRMat< complex< double > > &rhs)
- template<>
  NRMat< double > & operator-= (const NRMat< double > &rhs)
- template<>
  NRMat< complex< double > > & operator-= (const NRMat< complex< double > > &rhs)
- template<>
  NRMat< double > & operator+= (const NRSMat< double > &rhs)
- template<>
  NRMat< complex< double > > & operator+= (const NRSMat< complex< double > > &rhs)
- template<>
  NRMat< double > & operator-= (const NRSMat< double > &rhs)
- template<>
  NRMat< complex< double > > & operator-= (const NRSMat< complex< double > > &rhs)
- template<>
  const double dot (const NRMat< double > &rhs) const
- template<>
  const complex< double > dot (const NRMat< complex< double > > &rhs) const
- template<>
  const NRMat< double > operator* (const NRMat< double > &rhs) const
- template<>
  const NRMat< complex< double > > operator* (const NRMat< complex< double > > &rhs) const
- template<>
  void diagmultl (const NRVec< double > &rhs)
- template<>
  void diagmultl (const NRVec< complex< double > > &rhs)
- template<>
  void diagmultr (const NRVec< double > &rhs)
- template<>
  void diagmultr (const NRVec< complex< double > > &rhs)
- template<>
  const NRMat< double > operator* (const NRSMat< double > &rhs) const
- template<>
  const NRMat< complex< double > > operator* (const NRSMat< complex< double > > &rhs) const
- template<>
  NRMat< double > & conjugateme ()
- template<>
  NRMat< complex< double > > & conjugateme ()
- template<>
  const NRMat< double > transpose (bool conj) const
- template<>
  const NRMat< complex< double > > transpose (bool conj) const
- template<>
  void gemm (const double &beta, const NRMat< double > &a, const char transa, const NRMat< double > &b, const char transb, const double &alpha)
- template<>
  void **gemm** (const complex< double > &beta, const NRMat< complex< double > > &a, const char transa, const NRMat< complex< double > > &b, const char transb, const complex< double > &alpha)

- template<>
  const double norm (const double scalar) const
- template<>
  const double norm (const complex< double > scalar) const
- template<>
  void axpy (const double alpha, const NRMat< double > &mat)
- template<>
  void axpy (const complex< double > alpha, const NRMat< complex< double > > &mat)
- template<>
  const double * diagonalof (NRVec< double > &r, const bool divide, bool cache) const
- template<>
  void diagonalset (const NRVec< double > &r)
- template<>
  void diagonalset (const NRVec< complex< double > > &r)
- template<>
  void orthonormalize (const bool rowcol, const NRSMat< double > *metric)
- template<>
  NRMat< double > & swap_rows ()
- template<>
  NRMat< complex< double > > & swap_rows ()
- template<>
  NRMat< double > & swap_cols ()
- template<>
  NRMat< complex< double > > & swap_cols ()
- template<>
  NRMat< double > & swap_rows_cols ()
- template<>
  NRMat< complex< double > > & swap_rows_cols ()
- template<>
  const double amax () const
- template<>
  const double amin () const
- template<>
  const complex< double > amax () const
- template<>
  const complex< double > amin () const
- template<>
  void **gemm** (const complex< double > &beta, const SparseMat< complex< double > > &a, const char transa, const NRMat< complex< double > > &b, const char transb, const complex< double > &alpha)
- template<>
  void **gemm** (const double &beta, const SparseMat< double > &a, const char transa, const NRMat< double > &b, const char transb, const double &alpha)
- template<>
  void **s_cutoff** (const int c, const int c1, const int c2, const int c3) const
- template<>
  void **strassen** (const double beta, const NRMat< double > &a, const char transa, const NRMat< double > &b, const char transb, const double alpha)

Protected Attributes

- int nn
  number of rows
- int mm
  number of columns
- T * v
  pointer to the data stored continuously in emmory
- int * count
  reference counter

Friends

- class **NRVec< T >**
- class **NRSMat< T >**

3.2.1 Detailed Description

template<typename T> class LA::NRMat< T >

NRMat<T> class template implementing the matrix interface.

See also:

NRVec<T>, NRSMat<T>

3.2.2 Constructor & Destructor Documentation

3.2.2.1 template<typename T > LA::NRMat< T >::~NRMat () [inline]

standard destructor

destructor for general type

See also:

NRMat<T>::count

3.2.2.2 template<typename T > LA::NRMat< T >::NRMat (const int *n*, const int *m*, const GPUID *loc* = undefined) [inline]

Inlined constructor creating matrix of given size and location. Because of performance reasons, no incialization is done.

Parameters:

← ***n*** vector size (count of elements)

← ***loc*** location of the underlying data (CPU/GPU)

matrix constructor

Parameters:

← ***n*** number of rows of the matrix being created

← ***m*** number of cols of the matrix being created

← ***loc*** location for storing the matrix

See also:

count, v, location

3.2.2.3 template<typename T> LA::NRMat< T >::NRMat (const T & *a*, const int *n*, const int *m*, const GPUID *loc*) [inline]

inlined constructor creating matrix of given size filled with prescribed value and stored at given location

matrix constructor

Parameters:

← ***a*** value of type T intended for matrix inicialization

← ***n*** number of rows of the matrix being created

← ***m*** number of cols of the matrix being created

See also:

count, v

3.2.2.4 template<typename T> LA::NRMat< T >::NRMat (const T & *a*, const int *n*, const int *m*) [inline]

inlined constructor creating matrix of given size filled with prescribed value

matrix constructor

Parameters:

← ***a*** value of type T intended for matrix inicialization

← ***n*** number of rows of the matrix being created

← ***m*** number of cols of the matrix being created

See also:

count, v

3.2.2.5 template<typename T> LA::NRMat< T >::NRMat (const T * *a*, const int *n*, const int *m*) [inline]

inlined constructor creating matrix of given size filled with data located at given memory location

matrix constructor

Parameters:

← ***a*** pointer to values of type T intended for matrix inicialization

← ***n*** number of rows of the matrix being created

← ***m*** number of cols of the matrix being created

See also:

count, v

3.2.2.6 template<typename T> LA::NRMat< T >::NRMat (const NRMat< T > & *rhs*) [inline]

inlined copy-constructor

copy constructor implementing shallow copy

Parameters:

← ***rhs*** reference object to be copied

See also:

count, v

3.2.2.7 template<typename T> LA::NRMat< T >::NRMat (const NRSMat< T > & *rhs*) [inline, explicit]

explicit constructor converting symmetric matrix stored in packed form into a NRMat<T> object

create matrix from a NRSMat object

Parameters:

← ***rhs*** NRSMat input object to be converted

See also:

count, v, vec.h, NRSMat<T>

3.2.2.8 template<typename T> LA::NRMat< T >::NRMat (const NRVec< T > & *rhs*, const int *n*, const int *m*, const int *offset* = 0) [inline, explicit]

explicit constructor converting vector into a NRMat<T> object

create matrix from a vector (shallow copy)

Parameters:

← ***rhs*** NRVec vector containing the data

← ***n*** number of rows of the matrix being created

← ***m*** number of cols of the matrix being created

See also:

count, v, vec.h

< make just shallow copy

< therefore increase the reference counter

3.2.2.9 template<> LA::NRMat< complex< double > >::NRMat (const NRMat< double > & *rhs*, bool *imagpart*) [inline]

icreate complex double-precision matrix from real double-precision matrix A

Parameters:

← ***rhs*** real double-precision matrix A

← ***imagpart*** flag indicating whether the matrix A should be considered as a real or imaginary part of the complex matrix being created

3.2.3 Member Function Documentation

3.2.3.1 template<> const complex< double > LA::NRMat< complex< double > >::amax () const [inline]

for this complex matrix A, determine the smallest index of the maximum magnitude element, i.e. maximal element in the 1-norm

Returns:

$A_{l,m}$ which maximizes $\{|\Re A_{i,j}| + |\Im A_{i,j}|$

3.2.3.2 template<> const double LA::NRMat< double >::amax () const [inline]

for this real matrix A, determine the first element with largest absolute value

Returns:

$A_{l,m}$ which maximizes $|A_{i,j}|$

3.2.3.3 template<> const complex< double > LA::NRMat< complex< double > >::amin () const [inline]

for this complex matrix A, determine the smallest index of the minimum magnitude element, i.e. minimal element in the 1-norm

Returns:

$A_{l,m}$ which minimizes $\{|\Re A_{i,j}| + |\Im A_{i,j}|$

3.2.3.4 template<> const double LA::NRMat< double >::amin () const [inline]

for this real matrix A, determine the first element with smallest absolute value

Returns:

$A_{l,m}$ which minimizes $|A_{i,j}|$

3.2.3.5 template<> void LA::NRMat< complex< double > >::axpy (const complex< double > *alpha*, const NRMat< complex< double > > & *mat*) [inline]

perform the **axpy** operation on the current complex matrix A, i.e.

$$A \leftarrow A + \alpha B$$

for real matrix B

Parameters:

← ***alpha*** complex parameter α

← ***mat*** complex matrix B

3.2.3.6 template<> void LA::NRMat< double >::axpy (const double *alpha*, const NRMat< double > & *mat*) [inline]

perform the **axpy** operation on the current real matrix A, i.e.

$$A \leftarrow A + \alpha B$$

for real matrix B

Parameters:

← ***alpha*** α parameter

← ***mat*** real matrix B

3.2.3.7 template<> NRMat< complex< double > > & LA::NRMat< complex< double > >::conjugateme () [inline]

conjugate this complex matrix A, i.e. do nothing :-)

Returns:

reference to the modified matrix

3.2.3.8 template<> NRMat< double > & LA::NRMat< double >::conjugateme () [inline]

conjugate this real matrix A, i.e. do nothing :-)

Returns:

reference to the (unmodified) matrix

3.2.3.9 template<typename T > void LA::NRMat< T >::copyonwrite () [inline]

ensure that the data of this matrix are referenced exactly once

create own deep copy

See also:

NRMat<T>::count, NRMat<T>::operator|=()

3.2.3.10 template<> const NRVec< complex< double > > LA::NRMat< complex< double > >::csum () const [inline]

sum up the columns of the current double-precision complex matrix

Returns:

summed columns in a form of a vector

3.2.3.11 template<> const NRVec< double > LA::NRMat< double >::csum () const [inline]

sum up the columns of the current double-precision real matrix

Returns:

summed columns in a form of a vector

3.2.3.12 template<typename T > const NRVec< T > LA::NRMat< T >::csum () const [inline]

sum the columns

sum up the columns of the current matrix of general type `T`

Returns:

summed columns in a form of a vector

3.2.3.13 template<> void LA::NRMat< complex< double > >::diagmultl (const NRVec< complex< double > > & *rhs*) [inline]

multiply this complex matrix A by diagonal complex matrix D from left because of cuBlas implementation, D is required to be placed in CPU memory

Parameters:

$\leftarrow$ ***rhs*** complex vector represeting the diagonal of matrix D

3.2.3.14 template<> void LA::NRMat< double >::diagmultl (const NRVec< double > & *rhs*) [inline]

multiply this real matrix A by diagonal real matrix D from left because of cuBlas implementation, D is required to be placed in CPU memory

Parameters:

← ***rhs*** real vector represeting the diagonal of matrix D

3.2.3.15 template<> void LA::NRMat< complex< double > >::diagmultr (const NRVec< complex< double > > & *rhs*) [inline]

multiply this complex matrix A by diagonal complex matrix D from left

Parameters:

← ***rhs*** complex vector represeting the diagonal of matrix D

3.2.3.16 template<> void LA::NRMat< double >::diagmultr (const NRVec< double > & *rhs*) [inline]

multiply this real matrix A by diagonal real matrix D from right because of cuBlas implementation, D is required to be placed in CPU memory

Parameters:

← ***rhs*** real vector represeting the diagonal of matrix D

3.2.3.17 template<> const double * LA::NRMat< double >::diagonalof (NRVec< double > & *r*, const bool *divide*, bool *cache*) const [inline]

get or divide by the diagonal of real double-precision matrix in case of nonsquare matrix A, use the diagonal of $A^{\mathrm{T}}A$

Parameters:

↔ ***r*** vector for storing the diagonal

← ***divide*** • `false` save the diagonal to vector r
• `true` divide the vector r by the diagonal elements element-wise

← ***cache*** reserved

Returns:

- `divide == true` NULL
- `divide == false` pointer to the first element of r

< temporary variable for storing the scaling factor

3.2.3.18 template<> void LA::NRMat< complex< double > >::diagonalset (const NRVec< complex< double > > & *r*) [inline]

sets the diagonal of complex matrix to a given complex vector

Parameters:

← ***r*** complex vector which is supposed to be assigned to the diagonal

Returns:

void

3.2.3.19 template<> void LA::NRMat< double >::diagonalset (const NRVec< double > & *r*) [inline]

sets the diagonal of real matrix to a given real vector

Parameters:

← ***r*** real vector which is supposed to be assigned to the diagonal

Returns:

void

3.2.3.20 template<> const complex< double > LA::NRMat< complex< double > >::dot (const NRMat< complex< double > > & *rhs*) const [inline]

compute scalar product of this matrix A with given complex matrix B i.e. determine $\sum_{i,j} A^*_{i,j} B_{i,j}$

Parameters:

← ***rhs*** matrix B

Returns:

computed scalar product

3.2.3.21 template<> const double LA::NRMat< double >::dot (const NRMat< double > & *rhs*) const [inline]

compute scalar product of this matrix A with given real matrix B i.e. determine $\sum_{i,j} A_{i,j} B_{i,j}$

Parameters:

← ***rhs*** matrix B

Returns:

computed scalar product

3.2.3.22 template<typename T > void LA::NRMat< T >::fprintf (FILE * *file*, const char * *format*, const int *modulo*) const [inline]

formatted output

output of a matrix of general type via lawritemat

3.2.3.23 template<typename T > void LA::NRMat< T >::fscanf (FILE * *f*, const char * *format*) [inline]

formatted input

input of a matrix of general type via lawritemat

3.2.3.24 template<> void LA::NRMat< double >::gemm (const double & *beta*, const NRMat< double > & *a*, const char *transa*, const NRMat< double > & *b*, const char *transb*, const double & *alpha*) [inline]

perform the gemm operation for this real matrix M, i.e. compute

$$M \leftarrow \alpha \times op(A) \times op(B) + \beta \times M$$

Parameters:

← ***beta*** β

← ***a*** real matrix A

← ***transa*** transposition flag of matrix A

← ***b*** real matrix B

← ***transb*** transposition flag of matrix B

← ***alpha*** α

3.2.3.25 template<typename T > void LA::NRMat< T >::get (int *fd*, bool *dim* = 1, bool *transp* = false) [inline]

unformatted input

routine for raw input

Parameters:

← ***fd*** file descriptor for input

← ***dim*** number of elements intended for input, for dim=0 perform copyonwrite

← ***transp*** reserved

See also:

NRVec<T>::get(), copyonwrite()

3.2.3.26 template<typename T > const T LA::NRMat< T >::get_ij (const int *i*, const int *j*) const [inline]

get the copy of the element with indices (i,j)

for a given matrix A, determine the element with indices (i,j)

Parameters:

← ***i*** row number

← ***j*** col number

Returns:

const reference to $A_{i,j}$

Reimplemented in LA::NRMat_from1< T >.

3.2.3.27 template<typename T> GPUID LA::NRMat< T >::getlocation () const [inline]

routines for CUDA related stuff

- `getlocation()` gets the protected data member location
- `moveto(const GPUID)` moves underlying data between CPU/GPU memory

3.2.3.28 template<typename T > int LA::NRMat< T >::ncols () const [inline]

get the number of columns

Returns:

number of columns

3.2.3.29 template<> const double LA::NRMat< complex< double > >::norm (const complex< double > *scalar*) const [inline]

compute the Frobenius norm of the current complex matrix A, i.e.

$$\sqrt{\sum_{i=1}^{N}\sum_{j=1}^{M}|A_{i,j}|^2}$$

where N and M is the number of rows and columns, respectively

Parameters:

← ***scalar*** complex value subtracted from the diagonal elements

Returns:

computed norm

3.2.3.30 template<> const double LA::NRMat< double >::norm (const double *scalar*) const [inline]

compute the Frobenius norm of the current real matrix A, i.e.

$$\sqrt{\sum_{i=1}^{N}\sum_{j=1}^{M}|A_{i,j}|^2}$$

where N and M is the number of rows and columns, respectively

Parameters:

← ***scalar*** real value subtracted from the diagonal elements

Returns:

computed norm

3.2.3.31 template<typename T > int LA::NRMat< T >::nrows () const [inline]

get the number of rows

Returns:

number of rows

3.2.3.32 template<typename T > LA::NRMat< T >::operator const T * () const [inline]

get the const pointer to the data

Returns:

const pointer of general type T to the underlying data

3.2.3.33 template<typename T > LA::NRMat< T >::operator T * () [inline]

get the pointer to the data

Returns:

pointer of general type T to the underlying data structure

3.2.3.34 template<typename T > const T & LA::NRMat< T >::operator() (const int *i*, const int *j*) const [inline]

get the const reference to the element with indices (i,j)

for a given matrix A, determine the element with indices (i,j)

Parameters:

← ***i*** row number

← ***j*** col number

Returns:

const reference to $A_{i,j}$

Reimplemented in LA::NRMat_from1< T >.

3.2.3.35 template<typename T > T & LA::NRMat< T >::operator() (const int *i*, const int *j*) [inline]

get the reference to the element with indices (i,j)

for a given matrix A, determine the element with indices (i,j)

Parameters:

← ***i*** row number

← ***j*** col number

Returns:

reference to $A_{i,j}$

See also:

NRMat<T>::count

Reimplemented in LA::NRMat_from1< T >.

3.2.3.36 template<> const NRMat< complex< double > > LA::NRMat< complex< double > >::operator∗ (const NRSMat< complex< double > > & *rhs*) const [inline]

multiply this complex matrix A by symmetric complex matrix S S is stored in packed form, therefore zhpmv routine is used

Parameters:

← ***rhs*** complex symmetric matrix S stored in packed form

Returns:

$A \times \S$ by value

3.2.3.37 template<> const NRMat< double > LA::NRMat< double >::operator∗ (const NRSMat< double > & *rhs*) const [inline]

multiply this real matrix A by symmetric matrix S S is stored in packed form, therefore dspmv routine is used

Parameters:

← ***rhs*** real symmetric matrix S stored in packed form

Returns:

$A \times \S$ by value

3.2.3.38 template<> const NRMat< complex< double > > LA::NRMat< complex< double > >::operator∗ (const NRMat< complex< double > > & *rhs*) const [inline]

compute product of this matrix A with given complex matrix B

Parameters:

← ***rhs*** matrix B

Returns:

computed product by value

3.2.3.39 template<> const NRMat< double > LA::NRMat< double >::operator∗ (const NRMat< double > & *rhs*) const [inline]

compute product of this matrix A with given real matrix B

Parameters:

← ***rhs*** matrix B

Returns:

computed product by value

3.2.3.40 template<> NRMat< complex< double > > & LA::NRMat< complex< double > >::operator∗= (const complex< double > & *a*) [inline]

scale complex matrix with a complex factor

Parameters:

← ***a*** scaling factor

Returns:

reference to the modified matrix

3.2.3.41 template<> NRMat< double > & LA::NRMat< double >::operator∗= (const double & *a*) [inline]

scale real matrix with a real factor

Parameters:

← ***a*** scaling factor

Returns:

reference to the modified matrix

3.2.3.42 template<typename T> NRMat< T > & LA::NRMat< T >::operator*= (const T & *a*) [inline]

multiply by a scalar value

scale matrix of type T with a factor

Parameters:

← ***a*** scaling factor

Returns:

reference to the modified matrix

3.2.3.43 template<typename T> const NRMat< T > LA::NRMat< T >::operator+ (const NRSMat< T > & *rhs*) const [inline]

add given symmetric matrix stored in packed form and return the result by value

NRMat + NRSmat via operator +=

Parameters:

← ***rhs*** NRSMat matrix to be subtracted from current matrix

Returns:

result of the subtraction

See also:

NRMat<T>::operator+=(const NRSMat<T> &)

3.2.3.44 template<> NRMat< complex< double > > & LA::NRMat< complex< double > >::operator+= (const NRSMat< complex< double > > & *rhs*) [inline]

add a given sparse complex matrix A stored in packed form to the current complex matrix

Parameters:

← ***rhs*** symmetric complex matrix A in packed form

Returns:

reference to the modified matrix

See also:

NRSMat<T>

3.2.3.45 template<> NRMat< double > & LA::NRMat< double >::operator+= (const NRSMat< double > & *rhs*) [inline]

add a given sparse real matrix A stored in packed form to the current real matrix

Parameters:

← ***rhs*** symmetric real matrix A in packed form

Returns:

reference to the modified matrix

See also:

NRSMat<T>

3.2.3.46 template<> NRMat< complex< double > > & LA::NRMat< complex< double > >::operator+= (const NRMat< complex< double > > & *rhs*) [inline]

add a given complex matrix A to the current complex matrix

Parameters:

← ***rhs*** complex matrix A

Returns:

reference to the modified matrix

3.2.3.47 template<> NRMat< double > & LA::NRMat< double >::operator+= (const NRMat< double > & *rhs*) [inline]

add a given real matrix A to the current real matrix

Parameters:

← ***rhs*** matrix A

Returns:

reference to the modified matrix

3.2.3.48 template<> NRMat< complex< double > > & LA::NRMat< complex< double > >::operator+= (const complex< double > & *a*) [inline]

adds a double-precision complex scalar value to the diagonal elements of this double-precision complex matrix

Parameters:

← ***a*** double-precision complex scalar value

Returns:

reference to the modified matrix

3.2.3.49 template<> NRMat< double > & LA::NRMat< double >::operator+= (const double & *a*) [inline]

adds a double-precision real scalar value to the diagonal elements of this double-precision real matrix

Parameters:

← ***a*** double-precision real scalar value

Returns:

reference to the modified matrix

3.2.3.50 template<typename T> NRMat< T > & LA::NRMat< T >::operator+= (const NRSMat< T > & *rhs*) [inline]

add symmetric matrix stored in packed form

add a given general sparse matrix A stored in packed form to the current general matrix (of type T)

Parameters:

← ***rhs*** symmetric general matrix A in packed form

Returns:

reference to the modified matrix

See also:

NRSMat<T>

3.2.3.51 template<typename T> NRMat< T > & LA::NRMat< T >::operator+= (const NRMat< T > & *rhs*) [inline]

add given matrix

add a given general matrix (type T) A to the current complex matrix

Parameters:

← ***rhs*** matrix A of type T

Returns:

reference to the modified matrix

3.2.3.52 template<typename T> NRMat< T > & LA::NRMat< T >::operator+= (const T & *a*) [inline]

add scalar value to the diagonal elements

add a scalar value of type `T` to the diagonal elements of this matrix of general type `T`

Returns:

reference to the modified matrix

3.2.3.53 template<> const NRMat< complex< double > > LA::NRMat< complex< double > >::operator- () const [inline]

implements unary minus operator for this double-precision complex matrix

Returns:

modified copy of this matrix

3.2.3.54 template<> const NRMat< double > LA::NRMat< double >::operator- () const [inline]

implements unary minus operator for this double-recision real matrix

Returns:

modified copy of this matrix

3.2.3.55 template<typename T> const NRMat< T > LA::NRMat< T >::operator- (const NRSMat< T > & *rhs*) const [inline]

subtract given symmetric matrix stored in packed form and return the result by value

NRMat - NRSmat via operator -=

Parameters:

← ***rhs*** NRSMat matrix to be subtracted from current matrix

Returns:

result of the subtraction

See also:

NRMat<T>::operator-=(const NRSMat<T> &)

3.2.3.56 template<typename T > const NRMat< T > LA::NRMat< T >::operator- () const [inline]

unary minus

implements unary minus operator for this matrix of general type T

Returns:

modified copy of this matrix

3.2.3.57 template<> NRMat< complex< double > > & LA::NRMat< complex< double > >::operator-= (const NRSMat< complex< double > > & *rhs*) [inline]

subtract a given sparse complex matrix A stored in packed form from the current complex matrix

Parameters:

← ***rhs*** symmetric complex matrix A in packed form

Returns:

reference to the modified matrix

See also:

NRSMat<T>

3.2.3.58 template<> NRMat< double > & LA::NRMat< double >::operator-= (const NRSMat< double > & *rhs*) [inline]

subtract a given sparse real matrix A stored in packed form from the current real matrix

Parameters:

← ***rhs*** symmetric real matrix A in packed form

Returns:

reference to the modified matrix

See also:

NRSMat<T>

3.2.3.59 template<> NRMat< complex< double > > & LA::NRMat< complex< double > >::operator-= (const NRMat< complex< double > > & *rhs*) [inline]

subtract a given complex matrix A from the current complex matrix

Parameters:

← ***rhs*** matrix A

Returns:

reference to the modified matrix

3.2.3.60 template<> NRMat< double > & LA::NRMat< double >::operator-= (const NRMat< double > & *rhs*) [inline]

subtract a given real matrix A from the current real matrix

Parameters:

← ***rhs*** matrix A

Returns:

reference to the modified matrix

3.2.3.61 template<> NRMat< complex< double > > & LA::NRMat< complex< double > >::operator-= (const complex< double > & *a*) [inline]

subtracts a double-precision complex scalar value from the diagonal elements of this double-precision complex matrix

Parameters:

← ***a*** double-precision complex scalar value

Returns:

reference to the modified matrix

3.2.3.62 template<> NRMat< double > & LA::NRMat< double >::operator-= (const double & *a*) [inline]

subtracts a double-precision real scalar value from the diagonal elements of this double-precision real matrix

Parameters:

← ***a*** double-precision real scalar value

Returns:

reference to the modified matrix

3.2.3.63 template<typename T> NRMat< T > & LA::NRMat< T >::operator-= (const NRSMat< T > & *rhs*) [inline]

subtract symmetric matrix stored in packed form

subtract a given general sparse matrix A stored in packed form from the current general matrix (of type T)

Parameters:

← ***rhs*** symmetric general matrix A in packed form

Returns:

reference to the modified matrix

See also:

NRSMat<T>

3.2.3.64 template<typename T> NRMat< T > & LA::NRMat< T >::operator-= (const NRMat< T > & *rhs*) [inline]

subtract given matrix

subtract a given general matrix (type T) A from the current matrix

Parameters:

← ***rhs*** matrix A of type T

Returns:

reference to the modified matrix

3.2.3.65 template<typename T> NRMat< T > & LA::NRMat< T >::operator-= (const T & *a*) [inline]

subtract scalar value to the diagonal elements

subtracts a scalar value of type `T` from the diagonal elements of this matrix of general type `T`

Returns:

reference to the modified matrix

3.2.3.66 template<> NRMat< complex< double > > & LA::NRMat< complex< double > >::operator= (const complex< double > & *a*) [inline]

assigns a double-precision complex scalar value to the diagonal elements of this double-precision complex matrix

Parameters:

← ***a*** double-precision complex scalar value

Returns:

reference to the modified matrix

3.2.3.67 template<> NRMat< double > & LA::NRMat< double >::operator= (const double & *a*) [inline]

assigns a double-precision real scalar value to the diagonal elements of this double-precision real matrix

Parameters:

← ***a*** double-precision real scalar value

Returns:

reference to the modified matrix

3.2.3.68 template<typename T> NRMat< T > & LA::NRMat< T >::operator= (const T & *a*) [inline]

assign scalar value to the diagonal elements

assigns a scalar value of general type `T` to the diagonal elements of this matrix of general type `T`

Parameters:

← ***a*** scalar value of type `T`

Returns:

reference to the modified matrix

3.2.3.69 template<typename T > NRMat< T > & LA::NRMat< T >::operator= (const NRMat< T > & *rhs*) [inline]

assigment operator performing shallow copy

assigment operator for general type between NRMat and NRMat

See also:

count

Returns:

reference to the newly assigned matrix

3.2.3.70 template<typename T > const T * LA::NRMat< T >::operator[ ] (const int *i*) const [inline]

determine the const pointer to the i^{th} row

Parameters:

← ***i*** row number

Returns:

const pointer to the first element in the i-th row

3.2.3.71 template<typename T > T * LA::NRMat< T >::operator[ ] (const int *i*) [inline]

determine the pointer to the i^{th} row

Parameters:

← ***i*** row number

Returns:

pointer to the first element in the i-th row

3.2.3.72 template<typename T> NRMat< T > & LA::NRMat< T >::operator^= (const NRMat< T > & *rhs*) [inline]

Hadamard element-wise product.

compute Hadamard (component-wise) product with a given matrix A

Parameters:

← ***rhs*** matrix A

See also:

count, operator∗

Returns:

reference to the multiplied matrix

3.2.3.73 template<typename T > NRMat< T > & LA::NRMat< T >::operator|= (const NRMat< T > & *rhs*) [inline]

assigment operator performing deep copy

perform an explicit deep copy of `NRMat` object

See also:

count

Returns:

reference to the newly copied matrix

3.2.3.74 template<typename T > const NRMat< T > LA::NRMat< T >::oplus (const NRMat< T > & *rhs*) const [inline]

direct sum

implements direct sum with a given matrix B via storesubmatrix()

Parameters:

← ***rhs*** input matrix B

Returns:

result of the computation (new instance of NRMat<T>)

See also:

submatrix()

3.2.3.75 template<> void LA::NRMat< double >::orthonormalize (const bool *rowcol*, const NRSMat< double > ∗ *metric*) [inline]

perform straightforward orthonormalization via modified Gram-Schmidt process

Parameters:

← ***rowcol*** flag regarding the interpretation of the current matrix

- `true` the vectors being orthonormalized are stored as rows

- `false` the vectors being orthonormalized are stored as columns

← ***metric*** pointer to real symmetric matrix stored in packed form which is used in computing the inner products in the process, the standard inner product is taken into account for `metric == NULL`

Returns:

void

3.2.3.76 template<typename T > const NRMat< T > LA::NRMat< T >::otimes (const NRMat< T > & *rhs*, bool *reversecolumns* = `false`) const [inline]

direct product

implements direct product with a given matrix B

Parameters:

← ***rhs*** input matrix B

Returns:

result of the computation (new instance of NRMat<T>)

3.2.3.77 template<typename T > void LA::NRMat< T >::put (int *fd*, bool *dim* = `1`, bool *transp* = `false`) const [inline]

unformatted output

routine for raw output

Parameters:

← ***fd*** file descriptor for output

← ***dim*** number of elements intended for output

← ***transp*** reserved

See also:

NRVec<T>::put()

3.2.3.78 template<> void LA::NRMat< complex< double > >::randomize (const double & *x*) [inline]

fill given complex matrix with random numbers real/imaginary components are generated independently

Parameters:

← ***x*** generate random numbers from the interval [0, x]

3.2.3.79 template<> void LA::NRMat< double >::randomize (const double & *x*) [inline]

fill given real matrix with random numbers

Parameters:

← ***x*** generate random numbers from the interval [0, x]

3.2.3.80 template<typename T > void LA::NRMat< T >::resize (int *n*, int *m*) [inline]

resize the matrix

resize given matrix

Parameters:

← ***n*** number of rows

← ***m*** number of cols

See also:

count, NRMat<T>::copyonwrite(), NRMat<T>::operator|=()

Returns:

reference to the newly copied matrix

3.2.3.81 template<typename T > const NRVec< T > LA::NRMat< T >::row (const int *i*, int *l* = -1) const [inline]

get the i^{th} row

extract given row of this matrix of general type `T`

Parameters:

← ***i*** row index starting from zero

← ***l*** consider this value as the count of columns

Returns:

extracted elements as a NRVec<T> object

3.2.3.82 template<> const NRVec< complex< double > > LA::NRMat< complex< double > >::rsum () const [inline]

sum up the rows of the current double-precision complex matrix

Returns:

summed rows in a form of a vector

3.2.3.83 template<> const NRVec< double > LA::NRMat< double >::rsum () const [inline]

sum up the rows of the current double-precision real matrix

Returns:

summed rows in a form of a vector

3.2.3.84 template<typename T > const NRVec< T > LA::NRMat< T >::rsum () const [inline]

sum the rows

sum up the rows of the current matrix of general type `T`

Returns:

summed rows in a form of a vector

3.2.3.85 template<typename T > int LA::NRMat< T >::size () const [inline]

get the number of matrix elements

Returns:

number of elements

3.2.3.86 template<typename T > void LA::NRMat< T >::storesubmatrix (const int *fromrow*, const int *fromcol*, const NRMat< T > & *rhs*) [inline]

store given matrix at given position into the current matrix

places given matrix as submatrix at given position

Parameters:

← ***fromrow*** row-coordinate of top left corner

← ***fromcol*** col-coordinate of top left corner

← ***rhs*** input matrix

3.2.3.87 template<typename T> void LA::NRMat< T >::strassen (const T *beta*, const NRMat< T > & *a*, const char *transa*, const NRMat< T > & *b*, const char *transb*, const T *alpha*)

Strassen's multiplication (better than $O(n^3)$, analogous syntax to.

See also:

NRMat<T>::gemm())

3.2.3.88 template<typename T > const NRMat< T > LA::NRMat< T >::submatrix (const int *fromrow*, const int *torow*, const int *fromcol*, const int *tocol*) const [inline]

extract specified submatrix

extract block submatrix

Parameters:

← ***fromrow*** starting row

← ***torow*** final row

← ***fromcol*** starting column

← ***tocol*** final column

Returns:

extracted block submatrix

3.2.3.89 template<> NRMat< complex< double > > & LA::NRMat< complex< double > >::swap_cols () [inline]

interchange the order of the columns of the current (complex) matrix

Returns:

reference to the modified matrix

3.2.3.90 template<> NRMat< double > & LA::NRMat< double >::swap_cols () [inline]

interchange the order of the columns of the current (real) matrix

Returns:

reference to the modified matrix

3.2.3.91 template<typename T > NRMat< T > & LA::NRMat< T >::swap_cols () [inline]

swap the order of the columns of the current matrix

interchange the order of the columns of the current general matrix of type T because of the cuBlas implementation, the GPU version requires that sizeof(T)sizeof(float)==0

Returns:

reference to the modified matrix

3.2.3.92 template<> NRMat< complex< double > > & LA::NRMat< complex< double > >::swap_rows () [inline]

interchange the order of the rows of the current (complex) matrix

Returns:

reference to the modified matrix

3.2.3.93 template<> NRMat< double > & LA::NRMat< double >::swap_rows () [inline]

interchange the order of the rows of the current (real) matrix

Returns:

reference to the modified matrix

3.2.3.94 template<typename T > NRMat< T > & LA::NRMat< T >::swap_rows () [inline]

swap the order of the rows of the current matrix

interchange the order of the rows of the current general matrix of type T for GPU computations, the condition sizeof(T)sizeof(float) is required

Returns:

reference to the modified matrix

3.2.3.95 template<> NRMat< complex< double > > & LA::NRMat< complex< double > >::swap_rows_cols () [inline]

interchange the order of the rows and columns of the current complex matrix A of type T, i.e. perform the operation

$$A_{i,j} \leftarrow A_{nn-1-i,mm-1-j}$$

where $0 \leq i \leq nn$ and $0 \leq j \leq mm$

Returns:

reference to the modified matrix

3.2.3.96 template<> NRMat< double > & LA::NRMat< double >::swap_rows_cols () [inline]

interchange the order of the rows and columns of the current real matrix A of type T, i.e. perform the operation

$$A_{i,j} \leftarrow A_{nn-1-i,mm-1-j}$$

where $0 \leq i \leq nn$ and $0 \leq j \leq mm$

Returns:

reference to the modified matrix

3.2.3.97 template<typename T > NRMat< T > & LA::NRMat< T >::swap_rows_cols () [inline]

swap the order of the rows and columns of the current matrix

interchange the order of the rows and columns of the current general matrix A of type T, i.e. perform the operation

$$A_{i,j} \leftarrow A_{nn-1-i,mm-1-j}$$

where $0 \leq i \leq nn$ and $0 \leq j \leq mm$

Returns:

reference to the modified matrix

3.2.3.98 template<> const NRSMat< complex< double > > LA::NRMat< complex< double > >::timestransposed () const [inline]

for a given complex matrix A compute $AA^\dagger$

Returns:

complex NRSMat object because of the hermiticity of $AA^\dagger$

3.2.3.99 template<> const NRSMat< double > LA::NRMat< double >::timestransposed () const [inline]

for a given real matrix A compute AA^{T}

Returns:

real NRSMat object because of the symmetry of AA^{T}

< resulting matrix has nn rows

3.2.3.100 template<typename T > const NRSMat< T > LA::NRMat< T >::timestransposed () const [inline]

for this matrix A compute $A \cdot A^{\mathrm{T}}$

for a given matrix A (general type) compute $A^{\mathrm{T}}A$

Returns:

NRSMat<T> object because of the symmetry of the result

3.2.3.101 template<typename T > const T LA::NRMat< T >::trace () const [inline]

determine the sum of the diagonal elements

compute the trace of current genenal square matrix A, i.e.

$$\sum_{i=1}^{N} A_{i,i}$$

where N is the order of the matrix

3.2.3.102 template<> const NRMat< complex< double > > LA::NRMat< complex< double > >::transpose (bool *conj*) const [inline]

compute transpose (optionally conjugated) of this real matrix A

Parameters:

← ***conj*** conjugation flag

Returns:

transposed (conjugated) matrix by value

3.2.3.103 template<> const NRMat< double > LA::NRMat< double >::transpose (bool *conj*) const [inline]

compute transpose (optionally conjugated) of this real matrix A

Parameters:

← ***conj*** conjugation flag, unused for real matrices

Returns:

transposed (conjugated) matrix by value

3.2.3.104 template<> const NRSMat< complex< double > > LA::NRMat< complex< double > >::transposedtimes () const [inline]

for a given complex matrix A compute $A^\dagger A$

Returns:

complex NRSMat object because of the hermiticity of $A^\dagger A$

3.2.3.105 template<> const NRSMat< double > LA::NRMat< double >::transposedtimes () const [inline]

for a given real matrix A compute $A^{\mathrm{T}} A$

Returns:

real NRSMat object because of the symmetry of $A^{\mathrm{T}} A$

< resulting matrix has mm rows

3.2.3.106 template<typename T > const NRSMat< T > LA::NRMat< T >::transposedtimes () const [inline]

for this matrix A compute $A^{\mathrm{T}} \cdot A$

for a given matrix A (general type) compute $A^{\mathrm{T}} A$

Returns:

NRSMat<T> object because of the symmetry of the result

3.2.3.107 template<typename T > NRMat< T > & LA::NRMat< T >::transposeme (const int _n = 0) [inline]

in case of square matrix, transpose the leading minor of order n

compute matrix transposition for a principal leading minor

Parameters:

← ***_n*** order of the leading minor

Returns:

reference to the modified matrix

< transpose the entire matrix

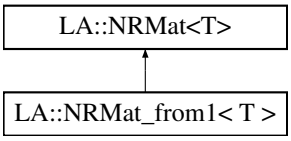
The documentation for this class was generated from the following files:

- mat.h
- mat.cc
- sparsemat.cc
- sparsemat.h
- sparsesmat.cc
- sparsesmat.h

3.3 LA::NRMat_from1< T > Class Template Reference

`#include <mat.h>`

Inheritance diagram for LA::NRMat_from1< T >::



Public Member Functions

- **NRMat_from1** (const int n)
- **NRMat_from1** (const NRMat< T > &rhs)
- NRMat_from1 (const int n, const int m)

  be able to convert the parent class transparently to this

- **NRMat_from1** (const T &a, const int n, const int m)
- **NRMat_from1** (const T *a, const int n, const int m)
- const T & operator() (const int i, const int j) const

  get the const reference to the element with indices (i,j)

- T & operator() (const int i, const int j)

  get the reference to the element with indices (i,j)

- const T get_ij (const int i, const int j) const

  get the copy of the element with indices (i,j)

3.3.1 Detailed Description

template<typename T> class LA::NRMat_from1< T >

implements `NRMat<T>` functionality with indexing from 1 all possible constructors have to be given explicitly, other stuff is inherited with exception of the operator() which differs

3.3.2 Member Function Documentation

3.3.2.1 template<typename T > const T LA::NRMat_from1< T >::get_ij (const int *i*, const int *j*) const [inline]

get the copy of the element with indices (i,j)

for a given matrix A, determine the element with indices (i,j)

Parameters:

← ***i*** row number

←*j* col number

Returns:

const reference to $A_{i,j}$

Reimplemented from LA::NRMat< T >.

3.3.2.2 template<typename T > T& LA::NRMat_from1< T >::operator() (const int *i*, const int *j*) [inline]

get the reference to the element with indices (i,j)

for a given matrix A, determine the element with indices (i,j)

Parameters:

←*i* row number

←*j* col number

Returns:

reference to $A_{i,j}$

See also:

NRMat<T>::count

Reimplemented from LA::NRMat< T >.

3.3.2.3 template<typename T > const T& LA::NRMat_from1< T >::operator() (const int *i*, const int *j*) const [inline]

get the const reference to the element with indices (i,j)

for a given matrix A, determine the element with indices (i,j)

Parameters:

←*i* row number

←*j* col number

Returns:

const reference to $A_{i,j}$

Reimplemented from LA::NRMat< T >.

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

- mat.h

3.4 LA::NRSMat< T > Class Template Reference

`#include <smat.h>`

Public Member Functions

- ~NRSMat ()
- NRSMat ()

  default constructor of null-matrix

- NRSMat (const int n, const GPUID loc=undefined)

  default constructor

- NRSMat (const T &a, const int n)

  constructor initializing the matrix being created by given scalar value

- NRSMat (const T *a, const int n)

  constructor initializing the matrix being created by data located at given memory position

- NRSMat (const NRSMat &rhs)

  copy constructor

- NRSMat (const typename LA_traits_complex< T >::NRSMat_Noncomplex_type &rhs, bool imagpart=false)

  constructor converting real matrix to its complex counterpart

- NRSMat (const NRMat< T > &rhs)

  constructor creating symmetric part of a general matrix

- NRSMat (const NRVec< T > &rhs, const int n)

  construct symmetric matrix by filling the lower triangle with data stored in a vector

- NRSMat & operator= (const NRSMat &rhs)

  assignment operator performing shallow copy

- NRSMat & operator|= (const NRSMat &rhs)

  assignment operator performing deep copy

- void randomize (const typename LA_traits< T >::normtype &x)

  fill the matrix with pseudorandom numbers (uniform distribution)

- NRSMat & operator= (const T &a)

  assign scalar value to diagonal elements

- int **getcount** () const
- GPUID **getlocation** () const
- void **moveto** (const GPUID dest)
- const bool operator!= (const NRSMat &rhs) const

  relational operator for testing nonequality

- const bool operator== (const NRSMat &rhs) const

  relational operator for testing equality

- NRSMat & operator*= (const T &a)
- NRSMat & operator+= (const T &a)
- NRSMat & operator-= (const T &a)
- NRSMat & operator+= (const NRSMat &rhs)
- NRSMat & operator-= (const NRSMat &rhs)
- const NRSMat operator- () const
- const NRSMat **operator*** (const T &a) const
- const NRSMat **operator+** (const T &a) const
- const NRSMat **operator-** (const T &a) const
- const NRSMat **operator+** (const NRSMat &rhs) const
- const NRSMat **operator-** (const NRSMat &rhs) const
- const NRMat< T > operator+ (const NRMat< T > &rhs) const
- const NRMat< T > operator- (const NRMat< T > &rhs) const
- const NRMat< T > **operator*** (const NRSMat &rhs) const
- const NRMat< T > **operator*** (const NRMat< T > &rhs) const
- const T **dot** (const NRSMat &rhs) const
- const T **dot** (const NRVec< T > &rhs) const
- const NRVec< T > **operator*** (const NRVec< T > &rhs) const
- const NRVec< complex< T > > **operator*** (const NRVec< complex< T > > &rhs) const
- const T * diagonalof (NRVec< T > &, const bool divide=0, bool cache=false) const
- void **gemv** (const T beta, NRVec< T > &r, const char trans, const T alpha, const NRVec< T > &x) const
- void **gemv** (const T beta, NRVec< complex< T > > &r, const char trans, const T alpha, const NRVec< complex< T > > &x) const
- const T & operator[ ] (const int ij) const
- T & operator[ ] (const int ij)
- const T & operator() (const int i, const int j) const
- T & operator() (const int i, const int j)
- int nrows () const
- int ncols () const
- int size () const
- bool **transp** (const int i, const int j) const
- const LA_traits< T >::normtype **norm** (const T scalar=(T) 0) const
- void **axpy** (const T alpha, const NRSMat &x)
- const T **amax** () const
- const T **amin** () const
- const T trace () const
- void get (int fd, bool dimensions=1, bool transp=0)
- void put (int fd, bool dimensions=1, bool transp=0) const
- void copyonwrite ()
- void **clear** ()
- void resize (const int n)
- operator T * ()
- operator const T * () const
- void fprintf (FILE *f, const char *format, const int modulo) const
- void fscanf (FILE *f, const char *format)

- **NRSMat** (const SparseMat< T > &rhs)
- **NRSMat** (const SparseSMat< T > &rhs)
- void **simplify** ()
- bool **issymmetric** () const
- template<>
  const NRSMat< double > operator- () const
- template<>
  const NRSMat< complex< double > > operator- () const
- template<>
  void randomize (const double &x)
- template<>
  void randomize (const double &x)
- template<>
  const NRMat< double > operator∗ (const NRMat< double > &rhs) const
- template<>
  const NRMat< complex< double > > operator∗ (const NRMat< complex< double > > &rhs) const
- template<>
  const NRMat< double > operator∗ (const NRSMat< double > &rhs) const
- template<>
  const NRMat< complex< double > > operator∗ (const NRSMat< complex< double > > &rhs) const
- template<>
  const double dot (const NRSMat< double > &rhs) const
- template<>
  const complex< double > dot (const NRSMat< complex< double > > &rhs) const
- template<>
  const double dot (const NRVec< double > &rhs) const
- template<>
  const complex< double > dot (const NRVec< complex< double > > &rhs) const
- template<>
  const double norm (const double scalar) const
- template<>
  const double norm (const complex< double > scalar) const
- template<>
  void axpy (const double alpha, const NRSMat< double > &x)
- template<>
  void axpy (const complex< double > alpha, const NRSMat< complex< double > > &x)
- template<>
  NRSMat (const NRSMat< double > &rhs, bool imagpart)
- template<>
  NRSMat< double > & operator∗= (const double &a)
- template<>
  NRSMat< complex< double > > & operator∗= (const complex< double > &a)
- template<>
  NRSMat< double > & operator+= (const NRSMat< double > &rhs)
- template<>
  NRSMat< complex< double > > & operator+= (const NRSMat< complex< double > > &rhs)
- template<>
  NRSMat< double > & operator-= (const NRSMat< double > &rhs)
- template<>
  NRSMat< complex< double > > & operator-= (const NRSMat< complex< double > > &rhs)

- template<>
  const double amax () const
- template<>
  const double amin () const
- template<>
  const complex< double > amax () const
- template<>
  const complex< double > amin () const

Protected Attributes

- int nn

  number of rows/columns of this symmetric matrix

- T * v

  internal pointer to the underlying data structure

- int * count

  pointer to the reference counter

Friends

- class **NRVec< T >**
- class **NRMat< T >**

3.4.1 Detailed Description

template<class T> class LA::NRSMat< T >

This class implements a general symmetric or hermitian matrix the elements of which are stored in packed form. Particularly the lower triangular part of a symmetric or hermitian matrix of order N is interpreted as a vector of length $N(N+1)/2$ in row-major storage scheme.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 template<typename T > LA::NRSMat< T >::~NRSMat () [inline]

destructor for general type `T`

See also:

NRSMat<T>::count, NRSMat<T>::v

3.4.2.2 template<typename T > LA::NRSMat< T >::NRSMat (const int *n*, const GPUID *loc* = undefined) [inline, explicit]

default constructor

constructor of a symmetric matrix stored in packed form

Parameters:

← ***n*** number of rows of the matrix being created

← ***loc*** location for storing the matrix

See also:

count, v, location

3.4.2.3 template<typename T> LA::NRSMat< T >::NRSMat (const T & *a*, const int *n*) [inline]

constructor initializing the matrix being created by given scalar value

constructor of a symmetric matrix stored in packed form (default location in used)

Parameters:

← ***a*** set all matrix elements equal to this value

← ***n*** number of rows of the matrix being created

See also:

count, v, location, NRSMat<T>::NRSMat(const int, const GPUID)

3.4.2.4 template<typename T> LA::NRSMat< T >::NRSMat (const T * *a*, const int *n*) [inline]

constructor initializing the matrix being created by data located at given memory position

constructor of a symmetric matrix stored in packed form (default location in used)

Parameters:

← ***a*** pointer to data of type T used for matrix inicialization

← ***n*** number of rows of the matrix being created

See also:

count, v, location, NRSMat<T>::NRSMat(const int, const GPUID), NRSMat<T>::NRSMat(const T&, const int)

3.4.2.5 template<typename T> LA::NRSMat< T >::NRSMat (const NRSMat< T > & *rhs*) [inline]

copy constructor

copy constructor implementing shallow copy

Parameters:

← ***rhs*** reference matrix being copied

See also:

count, v, location

3.4.2.6 template<typename T> LA::NRSMat< T >::NRSMat (const NRMat< T > & *rhs*) [inline, explicit]

constructor creating symmetric part of a general matrix

constructor symmetrizing given matrix A of general type `T` yielding $(A + A^{\mathrm{T}})/2$

Parameters:

← ***rhs*** matrix A

3.4.2.7 template<typename T> LA::NRSMat< T >::NRSMat (const NRVec< T > & *rhs*, const int *n*) [inline, explicit]

construct symmetric matrix by filling the lower triangle with data stored in a vector

constructor interpreting a vector of $n(n+1)/2$ elements as a symmetric matrix stored in packed form having n rows

Parameters:

← ***rhs*** reference matrix being copied

← ***n*** count of rows of the matrix being created

3.4.2.8 template<> LA::NRSMat< complex< double > >::NRSMat (const NRSMat< double > & *rhs*, bool *imagpart*) [inline]

create hermitian matrix H from given real double-precision symmetric matrix S

Parameters:

← ***rhs*** real double-precision symmetric matrix S

← ***imagpart*** flag determining whether S should correspond to the real or imaginary part of H

3.4.3 Member Function Documentation

3.4.3.1 template<> const complex< double > LA::NRSMat< complex< double > >::amax () const [inline]

for this complex symmetric matrix A, determine the first element with largest "absolute value"

Returns:

$A_{l,m}$ which maximizes $|\Re A_{i,j}| + |\Im A_{i,j}|$

3.4.3.2 template<> const double LA::NRSMat< double >::amax () const [inline]

for this real symmetric matrix A, determine the first element with largest absolute value

Returns:

$A_{l,m}$ which maximizes $|A_{i,j}|$

3.4.3.3 template<> const complex< double > LA::NRSMat< complex< double > >::amin () const [inline]

for this complex symmetric matrix A, determine the first element with smallest "absolute value"

Returns:

$A_{l,m}$ which minimizes $|\Re A_{i,j}| + |\Im A_{i,j}|$

3.4.3.4 template<> const double LA::NRSMat< double >::amin () const [inline]

for this real symmetric matrix A, determine the first element with smallest absolute value

Returns:

$A_{l,m}$ which minimizes $|A_{i,j}|$

3.4.3.5 template<> void LA::NRSMat< complex< double > >::axpy (const complex< double > *alpha*, const NRSMat< complex< double > > & *x*) [inline]

for this complex double-precision hermitian matrix H stored in packed form, complex scalar value α and complex double-precision hermitian matrix G, compute

$$H \leftarrow \alpha G + H$$

3.4.3.6 template<> void LA::NRSMat< double >::axpy (const double *alpha*, const NRSMat< double > & *x*) [inline]

for this real double-precision symmetric matrix S stored in packed form, real scalar value α and real double-precision symmetric matrix T, compute

$$S \leftarrow \alpha T + S$$

3.4.3.7 template<typename T > void LA::NRSMat< T >::copyonwrite () [inline]

detach this NRSMat<T> object and create own physical copy of the data

See also:

NRSMat<T>::operator|=, NRSMat<T>::copyonwrite()

3.4.3.8 template<typename T> const T * LA::NRSMat< T >::diagonalof (NRVec< T > & *r*, const bool *divide* = 0, bool *cache* = false) const [inline]

get or divide by the diagonal of real symmetric double-precision matrix

Parameters:

↔ ***r*** vector for storing the diagonal

← ***divide*** • `false` save the diagonal to vector r
• `true` divide the vector r by the diagonal elements element-wise

← ***cache*** reserved

Returns:

- `divide == true` NULL
- `divide == false` pointer to the first element of r

3.4.3.9 template<> const complex< double > LA::NRSMat< complex< double > >::dot (const NRVec< complex< double > > & *rhs*) const [inline]

compute inner product of this complex double-precision hermitian matrix H of order n with given complex double-precision vector $\vec{v}$ of length $n(n+1)/2$

Parameters:

← ***rhs*** complex double-precision vector $\vec{v}$

Returns:

computed inner product

3.4.3.10 template<> const double LA::NRSMat< double >::dot (const NRVec< double > & *rhs*) const [inline]

compute inner product of this real double-precision symmetric matrix S of order n with given real double-precision vector $\vec{v}$ of length $n(n+1)/2$

Parameters:

← ***rhs*** real double-precision vector $\vec{v}$

Returns:

computed inner product

3.4.3.11 template<> const complex< double > LA::NRSMat< complex< double > >::dot (const NRSMat< complex< double > > & *rhs*) const [inline]

compute inner product of this complex symmetric matrix A with given complex symmetric matrix B i.e. determine the value of

$$\sum_{i,j} A_{i,j} B_{i,j}$$

Parameters:

← ***rhs*** matrix B

Returns:

computed inner product

3.4.3.12 template<> const double LA::NRSMat< double >::dot (const NRSMat< double > & *rhs*) const [inline]

compute inner product of this real symmetric matrix A with given real symmetric matrix B i.e. determine the value of

$$\sum_{i,j} A_{i,j} B_{i,j}$$

Parameters:

← ***rhs*** matrix B

Returns:

computed inner product

3.4.3.13 template<typename T > void LA::NRSMat< T >::fprintf (FILE * *file*, const char * *format*, const int *modulo*) const [inline]

routine for formatted output via lawritemat

Parameters:

← ***file*** pointer to FILE structure representing the output file

← ***format*** format specification in standard printf-like form

← ***modulo***

See also:

lawritemat()

3.4.3.14 template<typename T > void LA::NRSMat< T >::fscanf (FILE * *f*, const char * *format*) [inline]

routine for formatted input via fscanf

Parameters:

← ***f*** pointer to FILE structure representing the input file

← ***format*** format specification in standard printf-like form

3.4.3.15 template<typename T > void LA::NRSMat< T >::get (int *fd*, bool *dim* = 1, bool *transp* = 0) [inline]

routine for raw input

Parameters:

← ***fd*** file descriptor for input

← ***dim*** number of elements intended for input

← ***transp*** reserved

See also:

NRSMat<T>::put(), NRSMat<T>::copyonwrite()

3.4.3.16 template<typename T > int LA::NRSMat< T >::ncols () const [inline]

Returns:

number of columns of this symmetric matrix of generalt type `T`

3.4.3.17 template<> const double LA::NRSMat< complex< double > >::norm (const complex< double > *scalar*) const [inline]

compute the Frobenius norm of this complex double-precision hermitian matrix

Parameters:

← ***scalar*** subtract this scalar value from the diagonal elements before the norm computation

3.4.3.18 template<> const double LA::NRSMat< double >::norm (const double *scalar*) const [inline]

compute the Frobenius norm of this real double-precision symmetric matrix

Parameters:

← ***scalar*** subtract this scalar value from the diagonal elements before the norm computation

3.4.3.19 template<typename T > int LA::NRSMat< T >::nrows () const [inline]

Returns:

number of rows of this symmetric matrix of generalt type `T`

3.4.3.20 template<typename T > LA::NRSMat< T >::operator const T * () const [inline]

Returns:

constant pointer of general type T to the underlying data structure

3.4.3.21 template<typename T > LA::NRSMat< T >::operator T * () [inline]

Returns:

pointer of general type T to the underlying data structure

3.4.3.22 template<typename T > T & LA::NRSMat< T >::operator() (const int *i*, const int *j*) [inline]

determine matrix element of this symmetric matrix of general type `T`

Parameters:

← ***i*** row index running from 0

←*j* column index running from 0

Returns:

reference to the corresponding matrix element

See also:

count, SMat_index, NRSMat<T>::operator[ ]

3.4.3.23 template<typename T > const T & LA::NRSMat< T >::operator() (const int *i*, const int *j*) const [inline]

determine matrix element of this symmetric matrix of general type `T`

Parameters:

←*i* row index running from 0

←*j* column index running from 0

Returns:

constant reference to the corresponding matrix element

See also:

count, SMat_index, NRSMat<T>::operator[ ]

3.4.3.24 template<> const NRMat< complex< double > > LA::NRSMat< complex< double > >::operator∗ (const NRSMat< complex< double > > & *rhs*) const [inline]

multiply this complex double-precision symmetric matrix G stored in packed form with complex double-precision symmetric matrix H

Returns:

matrix produt $G \times H$ (not necessarily symmetric)

3.4.3.25 template<> const NRMat< double > LA::NRSMat< double >::operator∗ (const NRSMat< double > & *rhs*) const [inline]

multiply this real double-precision symmetric matrix S stored in packed form with real double-precision symmetric matrix T

Returns:

matrix produt $S \times T$ (not necessarily symmetric)

3.4.3.26 template<> const NRMat< complex< double > > LA::NRSMat< complex< double > >::operator∗ (const NRMat< complex< double > > & *rhs*) const [inline]

multiply this real double-precision symmetric matrix S stored in packed form with real double-precision dense matrix A

Parameters:

← ***rhs*** real double-precision matrix A

Returns:

matrix produt $S \times A$

3.4.3.27 template<> const NRMat< double > LA::NRSMat< double >::operator∗ (const NRMat< double > & *rhs*) const [inline]

multiply this real double-precision symmetric matrix S stored in packed form with real double-precision dense matrix A

Parameters:

← ***rhs*** real double-precision matrix A

Returns:

matrix produt $S \times A$

3.4.3.28 template<> NRSMat< complex< double > > & LA::NRSMat< complex< double > >::operator∗= (const complex< double > & *a*) [inline]

multiply this complex symmetric matrix with complex scalar value

Parameters:

← ***a*** complex multiplicative factor

Returns:

reference to the modified matrix

3.4.3.29 template<> NRSMat< double > & LA::NRSMat< double >::operator∗= (const double & *a*) [inline]

multiply this real symmetric matrix with real scalar value

Parameters:

← ***a*** real multiplicative factor

Returns:

reference to the modified matrix

3.4.3.30 template<typename T> NRSMat< T > & LA::NRSMat< T >::operator*= (const T & *a*) [inline]

multiply this symmetric matrix of general type T stored in packed form with scalar value of type T

Parameters:

← ***a*** multiplicative factor of type T

Returns:

reference to the modified matrix

3.4.3.31 template<typename T> const NRMat< T > LA::NRSMat< T >::operator+ (const NRMat< T > & *rhs*) const [inline]

add up given dense matrix of general type T with this symmetric matrix of type T

Parameters:

← ***rhs*** dense matrix of type T to be added

Returns:

reference to the modified matrix

3.4.3.32 template<> NRSMat< complex< double > > & LA::NRSMat< complex< double > >::operator+= (const NRSMat< complex< double > > & *rhs*) [inline]

add up this complex symmetric matrix with given symmetric matrix

Parameters:

← ***rhs*** complex symmetric matrix to be added

Returns:

reference to the modified matrix

3.4.3.33 template<> NRSMat< double > & LA::NRSMat< double >::operator+= (const NRSMat< double > & *rhs*) [inline]

add up this real symmetric matrix with given symmetric matrix

Parameters:

← ***rhs*** real symmetric matrix to be added

Returns:

reference to the modified matrix

3.4.3.34 template<typename T> NRSMat< T > & LA::NRSMat< T >::operator+= (const NRSMat< T > & *rhs*) [inline]

add up this symmetric matrix of general type T with given symmetric matrix

Parameters:

← ***rhs*** complex matrix of general type T to be added

Returns:

reference to the modified matrix

3.4.3.35 template<typename T> NRSMat< T > & LA::NRSMat< T >::operator+= (const T & *a*) [inline]

add a scalar value α of general type T to the diagonal elements of this symmetric matrix of type T

Parameters:

← ***a*** scalar value α

Returns:

reference to the modified matrix

3.4.3.36 template<> const NRSMat< complex< double > > LA::NRSMat< complex< double > >::operator- () const [inline]

implements unary minus operator for this hermitian matrix

Returns:

modified copy of this matrix

3.4.3.37 template<> const NRSMat< double > LA::NRSMat< double >::operator- () const [inline]

implements unary minus operator for this real symmetric matrix

Returns:

modified copy of this matrix

3.4.3.38 template<typename T> const NRMat< T > LA::NRSMat< T >::operator- (const NRMat< T > & *rhs*) const [inline]

subtracts given dense matrix of general type T from this symmetric matrix of type T

Parameters:

← ***rhs*** dense matrix of type T to be added

Returns:

reference to the modified matrix

3.4.3.39 template<typename T > const NRSMat< T > LA::NRSMat< T >::operator- () const [inline]

implements unary minus operator for this symmetric matrix of general type T

Returns:

modified copy of this matrix

3.4.3.40 template<> NRSMat< complex< double > > & LA::NRSMat< complex< double > >::operator-= (const NRSMat< complex< double > > & *rhs*) [inline]

subtracts given complex symmetric matrix from this complex symmetric matrix

Parameters:

← ***rhs*** complex symmetric matrix to be subtracted

Returns:

reference to the modified matrix

3.4.3.41 template<> NRSMat< double > & LA::NRSMat< double >::operator-= (const NRSMat< double > & *rhs*) [inline]

subtracts given real symmetric matrix from this real symmetric matrix

Parameters:

← ***rhs*** real symmetric matrix to be subtracted

Returns:

reference to the modified matrix

3.4.3.42 template<typename T> NRSMat< T > & LA::NRSMat< T >::operator-= (const NRSMat< T > & *rhs*) [inline]

subtracts given symmetric matrix of general type T from this symmetric matrix of type T

Parameters:

← ***rhs*** symmetric matrix of general type T to be subtracted

Returns:

reference to the modified matrix

3.4.3.43 template<typename T> NRSMat< T > & LA::NRSMat< T >::operator-= (const T & *a*) [inline]

subtract a scalar value α of general type `T` from the diagonal elements of this symmetric matrix of type `T`

Parameters:

$\leftarrow$ ***a*** scalar value α

Returns:

reference to the modified matrix

3.4.3.44 template<typename T> NRSMat< T > & LA::NRSMat< T >::operator= (const T & *a*) [inline]

assign scalar value to diagonal elements

zero out this symmetric matrix of general type `T` and then set the diagonal elements to prescribed value

Parameters:

$\leftarrow$ ***a*** scalar value to be assigned to the diagonal

Returns:

reference to the modified matrix

3.4.3.45 template<typename T > NRSMat< T > & LA::NRSMat< T >::operator= (const NRSMat< T > & *rhs*) [inline]

assignment operator performing shallow copy

assignment operator implementing shallow copy of reference NRSMat<T> object

See also:

NRSMat<T>::operator|=, NRSMat<T>::copyonwrite()

3.4.3.46 template<typename T > T & LA::NRSMat< T >::operator[ ] (const int *ij*) [inline]

determine matrix element of this symmetric matrix of general type `T` using cumulative index increasing in a row-major way and corresponding to the lower triangular part of the respective dense matrix

Parameters:

$\leftarrow$ ***ij*** index of the requested element

Returns:

reference to the corresponding matrix element

3.4.3.47 template<typename T > const T & LA::NRSMat< T >::operator[ ] (const int *ij*) const [inline]

determine matrix element of this symmetric matrix of general type `T` using cumulative index increasing in a row-major way and corresponding to the lower triangular part of the respective dense matrix, i.e. $A_{i,j}$ for $N > i \geq j \geq 0$ corresponds to cumulative index $i(i+1)/2 + j$

Parameters:

← ***ij*** index of the requested element

Returns:

constant reference to the corresponding matrix element

3.4.3.48 template<typename T > NRSMat< T > & LA::NRSMat< T >::operator|= (const NRSMat< T > & *rhs*) [inline]

assignment operator performing deep copy

assigment operator implementing deep copy of the reference NRSMat<T> object

See also:

NRSMat<T>::operator=, NRSMat<T>::copyonwrite()

3.4.3.49 template<typename T > void LA::NRSMat< T >::put (int *fd*, bool *dim* = 1, bool *transp* = 0) const [inline]

routine for raw output

Parameters:

← ***fd*** file descriptor for output

← ***dim*** number of elements intended for output

← ***transp*** reserved

See also:

NRMat<T>::get(), NRSMat<T>::copyonwrite()

3.4.3.50 template<> void LA::NRSMat< complex< double > >::randomize (const double & *x*) [inline]

Fill this hermitian matrix with pseudorandom numbers generated from uniform distribution. The real and imaginary parts are generated independently.

3.4.3.51 template<> void LA::NRSMat< double >::randomize (const double & *x*) [inline]

fill this real symmetric matrix with pseudorandom numbers generated from uniform distribution

3.4.3.52 template<typename T > void LA::NRSMat< T >::resize (const int *n*) [inline]

resize this symmetric matrix of general type `T`

Parameters:

← ***n*** requested number of rows (columns)

3.4.3.53 template<typename T > int LA::NRSMat< T >::size () const [inline]

Returns:

number of elements of this symmetric matrix of generalt type `T`

3.4.3.54 template<typename T > const T LA::NRSMat< T >::trace () const [inline]

Returns:

the sum of the diagonal elements

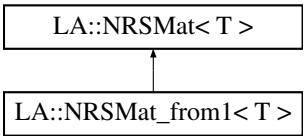
The documentation for this class was generated from the following files:

- smat.h
- smat.cc
- sparsemat.cc
- sparsesmat.h

3.5 LA::NRSMat_from1< T > Class Template Reference

`#include <smat.h>`

Inheritance diagram for LA::NRSMat_from1< T >::



Public Member Functions

- **NRSMat_from1** (const int n)
- **NRSMat_from1** (const NRSMat< T > &rhs)
- **NRSMat_from1** (const T &a, const int n)
- **NRSMat_from1** (const T *a, const int n)
- **NRSMat_from1** (const NRMat< T > &rhs)
- **NRSMat_from1** (const NRVec< T > &rhs, const int n)
- const T & **operator()** (const int i, const int j) const
- T & **operator()** (const int i, const int j)

3.5.1 Detailed Description

template<typename T> class LA::NRSMat_from1< T >

generate operators relating NRSMat<T> objects and scalars corresponding macro is defined in vec.h generate operators relating in general two NRSMat<T> objects corresponding macro is defined in vec.h class implementing NRSMat<T> funcitonality with indices running from 1 allmost all function members are inherited, only constructors are given explicitly

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

- smat.h

3.6 LA::NRVec< T > Class Template Reference

NRVec<T> class template implementing the vector interface.

`#include <vec.h>`

Public Types

- typedef T **ROWTYPE**

Public Member Functions

- ~NRVec ()

  standard destructor

- NRVec ()
- NRVec (const int n, const GPUID loc=undefined)
- NRVec (const T &a, const int n)

  inlined constructor creating vector of given size filled with prescribed value

- NRVec (const T *a, const int n)

  inlined constructor creating vector of given size filled with data located at given memory location

- NRVec (T *a, const int n, bool skeleton)

  inlined constructor creating vector of given size filled with data located at given memory location

- NRVec (const NRVec &rhs)

  inlined copy constructor

- NRVec (const typename LA_traits_complex< T >::NRVec_Noncomplex_type &rhs, bool imagpart=false)

  complexifying constructor

- NRVec (const NRSMat< T > &S)

  explicit inlined constructor converting symmetric matrix into a vector

- NRVec (const NRMat< T > &rhs)
- GPUID getlocation () const
- void **moveto** (const GPUID dest)
- void copyonwrite ()

  create separate copy of the data corresponding to this vector

- void clear ()

  purge this vector

- NRVec & operator= (const NRVec &rhs)

  assignment operator assigns given vector

- NRVec & operator= (const T &a)

assigment operator assigns given scalar to each element of this vector

- void randomize (const typename LA_traits< T >::normtype &x)

  fills in this vector with pseudo-random numbers generated using uniform distribution

- NRVec & operator|= (const NRVec &rhs)

  perform deep-copy of given vector

- const bool operator!= (const NRVec &rhs) const

  relational operators

- const bool **operator==** (const NRVec &rhs) const
- const bool operator> (const NRVec &rhs) const
- const bool operator< (const NRVec &rhs) const
- const bool **operator>=** (const NRVec &rhs) const
- const bool **operator<=** (const NRVec &rhs) const
- const NRVec operator- () const

  unary minus

- NRVec & operator+= (const NRVec &rhs)

  bunch of vector-vector arithmetic operators defined element-wise

- NRVec & operator-= (const NRVec &rhs)
- NRVec & operator*= (const NRVec &rhs)
- NRVec & operator/= (const NRVec &rhs)
- const NRVec **operator+** (const NRVec &rhs) const
- const NRVec **operator-** (const NRVec &rhs) const
- NRVec & operator+= (const T &a)

  bunch of scalar-vector arithmetic operators defined element-wise

- NRVec & operator-= (const T &a)
- NRVec & operator*= (const T &a)
- const NRVec **operator+** (const T &a) const
- const NRVec **operator-** (const T &a) const
- const NRVec **operator*** (const T &a) const
- int getcount () const

  determine the actual value of the reference counter

- const T operator* (const NRVec &rhs) const

  compute the Euclidean inner product (with conjugation in complex case)

- const T **dot** (const NRVec &rhs) const
- const T dot (const T *a, const int stride=1) const

  compute the Euclidean inner product (with conjugation in complex case) with a stride-vector

- void **gemv** (const T beta, const NRMat< T > &a, const char trans, const T alpha, const NRVec &x)
- void gemv (const T beta, const NRSMat< T > &a, const char trans, const T alpha, const NRVec &x)
- void **gemv** (const T beta, const SparseMat< T > &a, const char trans, const T alpha, const NRVec &x, const bool treat_as_symmetric=false)

- void **gemv** (const typename LA_traits_complex< T >::Component_type beta, const typename LA_traits_complex< T >::NRMat_Noncomplex_type &a, const char trans, const typename LA_traits_complex< T >::Component_type alpha, const NRVec &x)
- void **gemv** (const typename LA_traits_complex< T >::Component_type beta, const typename LA_traits_complex< T >::NRSMat_Noncomplex_type &a, const char trans, const typename LA_traits_complex< T >::Component_type alpha, const NRVec &x)
- const NRVec operator* (const NRMat< T > &mat) const

  multiply given matrix with this vector from left

- const NRVec operator* (const NRSMat< T > &mat) const

  multiply given symmetric matrix in packed form with this vector from left

- const NRVec operator* (const SparseMat< T > &mat) const

  multiply given sparse matrix with this vector from left

- const NRMat< T > otimes (const NRVec< T > &rhs, const bool conjugate=false, const T &scale=1) const

  compute the outer product of two vectors

- const NRMat< T > operator| (const NRVec< T > &rhs) const

  opeartor for outer product computation

- const T sum () const

  compute the sum of the vector elements

- const LA_traits< T >::normtype asum () const

  compute the sum of the absolute values of the elements of this vector

- T & operator[ ] (const int i)

  indexing operator - index running from zero

- const T & operator[ ] (const int i) const
- void setcoldim (int i)

  dummy routine

- operator T * ()

  get the pointer to the underlying data structure

- operator const T * () const

  get the constant pointer to the underlying data structure

- void axpy (const T alpha, const NRVec &x)

  add up a scalar multiple of a given vector

- void axpy (const T alpha, const T *x, const int stride=1)

  add up a scalar multiple of a given vector with given stride

- int size () const

  determine the number of elements

- void resize (const int n)
  resize the current vector
- const LA_traits< T >::normtype norm () const
  determine the norm of this vector
- NRVec & normalize (typename LA_traits< T >::normtype *norm=0)
  normalize this vector and optionally save the norm
- const NRVec unitvector () const
  get normalized copy of this vector
- const T amax () const
  determine the maximal element (in the absolute value) of this vector
- const T amin () const
  determine the minimal element (in the absolute value) of this vector
- void fprintf (FILE *f, const char *format, const int modulo) const
  routine for formatted output
- void put (int fd, bool dimensions=1, bool transp=0) const
  routine for unformatted output
- void fscanf (FILE *f, const char *format)
  routine for formatted input
- void get (int fd, bool dimensions=1, bool transp=0)
  routine for unformatted input
- NRVec (const SparseMat< T > &rhs)
  constructor creating vector from sparse matrix
- void simplify ()
  routine for compatibility with sparse types
- bool bigger (int i, int j) const
  determine whether the i^{th} element is bigger than the j^{th} element
- bool smaller (int i, int j) const
  determine whether the i^{th} element is bigger than the j^{th} element
- void swap (int i, int j)
  swap the i^{th} and j^{th} element
- int sort (int direction=0, int from=0, int to=-1, int *perm=NULL)
  sort by default in ascending order and return the parity of corresponding permutation resulting to this order
- NRVec & call_on_me (T(*_F)(const T &))

apply given function to each element

- template<>
  const NRVec< double > operator- () const
- template<>
  const NRVec< complex< double > > operator- () const
- template<>
  void randomize (const double &x)
- template<>
  void randomize (const double &x)
- template<>
  NRVec (const NRVec< double > &rhs, bool imagpart)
- template<>
  void axpy (const double alpha, const NRVec< double > &x)
- template<>
  void axpy (const complex< double > alpha, const NRVec< complex< double > > &x)
- template<>
  void axpy (const double alpha, const double *x, const int stride)
- template<>
  void axpy (const complex< double > alpha, const complex< double > *x, const int stride)
- template<>
  NRVec< double > & operator= (const double &a)
- template<>
  NRVec< complex< double > > & operator= (const complex< double > &a)
- template<>
  NRVec< double > & normalize (double *norm)
- template<>
  NRVec< complex< double > > & normalize (double *norm)
- template<>
  void gemv (const double beta, const NRMat< double > &A, const char trans, const double alpha, const NRVec &x)
- template<>
  void gemv (const double beta, const NRMat< double > &A, const char trans, const double alpha, const NRVec< complex< double > > &x)
- template<>
  void gemv (const complex< double > beta, const NRMat< complex< double > > &A, const char trans, const complex< double > alpha, const NRVec< complex< double > > &x)
- template<>
  void gemv (const double beta, const NRSMat< double > &A, const char trans, const double alpha, const NRVec &x)
- template<>
  void gemv (const double beta, const NRSMat< double > &A, const char trans, const double alpha, const NRVec< complex< double > > &x)
- template<>
  void gemv (const complex< double > beta, const NRSMat< complex< double > > &A, const char trans, const complex< double > alpha, const NRVec< complex< double > > &x)
- template<>
  const NRMat< double > otimes (const NRVec< double > &b, const bool conj, const double &scale) const
- template<>
  const NRMat< complex< double > > otimes (const NRVec< complex< double > > &b, const bool conj, const complex< double > &scale) const

- template<>
  NRVec< double > & operator+= (const double &a)
- template<>
  NRVec< complex< double > > & operator+= (const complex< double > &a)
- template<>
  NRVec< double > & operator-= (const double &a)
- template<>
  NRVec< complex< double > > & operator-= (const complex< double > &a)
- template<>
  NRVec< double > & operator+= (const NRVec< double > &rhs)
- template<>
  NRVec< complex< double > > & operator+= (const NRVec< complex< double > > &rhs)
- template<>
  NRVec< double > & operator-= (const NRVec< double > &rhs)
- template<>
  NRVec< complex< double > > & operator-= (const NRVec< complex< double > > &rhs)
- template<>
  NRVec< double > & operator*= (const double &a)
- template<>
  NRVec< complex< double > > & operator*= (const complex< double > &a)
- template<>
  const double operator* (const NRVec< double > &rhs) const
- template<>
  const complex< double > operator* (const NRVec< complex< double > > &rhs) const
- template<>
  const double dot (const double *y, const int stride) const
- template<>
  const complex< double > dot (const complex< double > *y, const int stride) const
- template<>
  const double asum () const
- template<>
  const double asum () const
- template<>
  const double norm () const
- template<>
  const double norm () const
- template<>
  const double amax () const
- template<>
  const double amin () const
- template<>
  const complex< double > amax () const
- template<>
  const complex< double > amin () const

Protected Attributes

- int nn

  size of the vector

- T * v

pointer to the underlying data structure

- int * count

  pointer to the reference-counter

Friends

- class **NRSMat**< **T** >
- class **NRMat**< **T** >
- template<typename U >
  NRVec< complex< U > > **complexify** (const NRVec< U > &)

3.6.1 Detailed Description

template<typename T> class LA::NRVec< T >

NRVec<T> class template implementing the vector interface.

See also:

NRMat<T>, NRSMat<T>

3.6.2 Constructor & Destructor Documentation

3.6.2.1 template<typename T > LA::NRVec< T >::~NRVec () [inline]

standard destructor

generate operators involving vector and scalar generate operators involving vector and vector destructor for general vector decreases the reference count and performs deallocation if neccessary

3.6.2.2 template<typename T> LA::NRVec< T >::NRVec () [inline]

inlined constructor creating zero vector of general type `T`

3.6.2.3 template<typename T> LA::NRVec< T >::NRVec (const int *n*, const GPUID *loc* = undefined) [inline, explicit]

Explicit inlined constructor creating vector of given size and location. Because of performance reasons, no incialization is done.

Parameters:

← ***n*** vector size (count of elements)

← ***loc*** location of the underlying data (CPU/GPU)

3.6.2.4 template<typename T> LA::NRVec< T >::NRVec (const T & *a*, const int *n*) [inline]

inlined constructor creating vector of given size filled with prescribed value

inline constructor creating vector of given size filled with prescribed value

Parameters:

← ***a*** value to be assigned to all vector elements

← ***n*** required vector size

3.6.2.5 template<typename T> LA::NRVec< T >::NRVec (const T * *a*, const int *n*) [inline]

inlined constructor creating vector of given size filled with data located at given memory location

inline constructor creating vector of given size filled with given data

Parameters:

← ***a*** pointer to the data

← ***n*** required vector size

3.6.2.6 template<typename T> LA::NRVec< T >::NRVec (T * *a*, const int *n*, bool *skeleton*) [inline]

inlined constructor creating vector of given size filled with data located at given memory location

inline constructor creating vector of given size filled with given data

Parameters:

← ***a*** pointer to the data

← ***n*** required vector size

← ***skeleton*** if equal to true, only the internal data pointer is modified and reference counter is set to two, i.e. no data deallocation occurs in destructor

3.6.2.7 template<typename T> LA::NRVec< T >::NRVec (const NRVec< T > & *rhs*) [inline]

inlined copy constructor

inline copy constructor

Parameters:

← ***rhs*** reference vector being copied

3.6.2.8 template<typename T> LA::NRVec< T >::NRVec (const NRSMat< T > & *rhs*) [inline, explicit]

explicit inlined constructor converting symmetric matrix into a vector

inline constructor interpreting symmetric matrix of order n stored in packed form as a linear vector consisting of $n(n+1)/2$ elements

Parameters:

← ***rhs*** symmetric matrix of type NRSMat<T>

See also:

NRSMat<T>

using macro NN2 defined in smat.h

3.6.2.9 template<typename T> LA::NRVec< T >::NRVec (const NRMat< T > & *rhs*) [inline, explicit]

conversion constructor interpreting a given matrix with N rows and M columns of general type `T` as a vector of $N \times M$ elements

Parameters:

← ***rhs*** matrix being converted

See also:

NRMat<T>::NRMat()

3.6.2.10 template<> LA::NRVec< complex< double > >::NRVec (const NRVec< double > & *rhs*, bool *imagpart*) [inline]

constructor creating complex vector from a real one

Parameters:

← ***rhs*** the real vector being converted into the complex one

← ***imagpart*** • `true` vector `rhs` is interpreted as the imaginary part of the new complex vector
• `false` vector `rhs` is interpreted as the real part of the new complex vector

Returns:

- `false` current vector is bigger than vector `rhs`
- `true` current vector is smaller than vector `rhs`

3.6.3 Member Function Documentation

3.6.3.1 template<> const complex< double > LA::NRVec< complex< double > >::amax () const [inline]

for a given complex vector $\vec{v}$, determine the smallest index of the maximum magnitude element, i.e. maximal element in the 1-norm

Returns:

$\vec{v}_j$ which maximizes $\{|\Re\vec{v}_i| + |\Im\vec{v}_i|$

3.6.3.2 template<> const double LA::NRVec< double >::amax () const [inline]

for this real vector $\vec{x}$ determine the element with largest absolute value

Returns:

$\vec{x}_i$ where $\left|\vec{x]_i}\right| = \max_j |\vec{x}_j|$

3.6.3.3 template<> const complex< double > LA::NRVec< complex< double > >::amin () const [inline]

for a given complex vector $\vec{v}$, determine the smallest index of the minimum magnitude element, i.e. minimal element in the 1-norm

Returns:

$\vec{v}_j$ which minimizes $\{|\Re\vec{v}_i| + |\Im\vec{v}_i|$

3.6.3.4 template<> const double LA::NRVec< double >::amin () const [inline]

for this real vector $\vec{x}$ determine the element with smallest absolute value

Returns:

$\vec{x}_i$ where $\left|\vec{x]_i}\right| = \min_j |\vec{x}_j|$

3.6.3.5 template<> const double LA::NRVec< complex< double > >::asum () const [inline]

for this complex vector $\vec{x}$ compute the expression

$$\sum_{i=1}^{N} |\Re\vec{x}_i| + |\Im\vec{x}_i|$$

Returns:

the value of this sum

3.6.3.6 template<> const double LA::NRVec< double >::asum () const [inline]

computes the sum of the absolute values of the elements of this real vector $\vec{x}$

Returns:

$\sum_{i=1}^{N} |\vec{x}_i|$

3.6.3.7 template<> void LA::NRVec< complex< double > >::axpy (const complex< double > *alpha*, const complex< double > * *x*, const int *stride*) [inline]

perform the **axpy** operation on the current complex vector $\vec{v}$, i.e.

$$\vec{v} \leftarrow \vec{v} + \alpha\vec{x}$$

Parameters:

← ***alpha*** double-precision complex parameter α

← ***x*** pointer to double-precision complex data

← ***stride*** sets the stride

3.6.3.8 template<> void LA::NRVec< double >::axpy (const double *alpha*, const double * *x*, const int *stride*) [inline]

perform the **axpy** operation on the current real vector $\vec{v}$, i.e.

$$\vec{v} \leftarrow \vec{v} + \alpha\vec{x}$$

Parameters:

← ***alpha*** α parameter

← ***x*** pointer to double-precision real data

← ***stride*** sets the stride

3.6.3.9 template<> void LA::NRVec< complex< double > >::axpy (const complex< double > *alpha*, const NRVec< complex< double > > & *x*) [inline]

perform the **axpy** operation on the current complex vector $\vec{v}$, i.e.

$$\vec{v} \leftarrow \vec{v} + \alpha\vec{x}$$

Parameters:

← ***alpha*** α parameter

← ***x*** complex vector $\vec{x}$

3.6.3.10 template<> void LA::NRVec< double >::axpy (const double *alpha*, const NRVec< double > & *x*) [inline]

perform the **axpy** operation on the current real vector $\vec{v}$, i.e.

$$\vec{v} \leftarrow \vec{v} + \alpha\vec{x}$$

Parameters:

← ***alpha*** double-precision real parameter α

← ***x*** double-precision real vector $\vec{x}$

3.6.3.11 template<typename T > void LA::NRVec< T >::copyonwrite () [inline]

create separate copy of the data corresponding to this vector

make own copy of the underlying data connected with this vector

3.6.3.12 template<> const complex< double > LA::NRVec< complex< double > >::dot (const complex< double > * *y*, const int *stride*) const [inline]

computes the inner product of this complex vector $\vec{x}$ with given complex data

Parameters:

← ***y*** pointer to the double-precision complex array (sufficient length assumed)

← ***stride*** specifies the stride regarding the data pointe to by `y`

Returns:

$\sum_{i=1}^{N} \vec{x}_i \cdot y_{\text{stride}\cdot(i-1)+1}$

3.6.3.13 template<> const double LA::NRVec< double >::dot (const double * *y*, const int *stride*) const [inline]

computes the inner product of this real vector $\vec{x}$ with given real data

Parameters:

← ***y*** pointer to the double-precision real array (sufficient length assumed)

← ***stride*** specifies the stride regarding the data pointe to by `y`

Returns:

$\sum_{i=1}^{N} \vec{x}_i \cdot y_{\text{stride}\cdot(i-1)+1}$

3.6.3.14 template<typename T > void LA::NRVec< T >::fprintf (FILE * *file*, const char * *format*, const int *modulo*) const [inline]

routine for formatted output

routine for formatted output via lawritemat

Parameters:

← ***file*** pointer to `FILE` structure representing the output file

← ***format*** format specification in standard printf-like form

← ***modulo***

See also:

lawritemat()

3.6.3.15 template<typename T > void LA::NRVec< T >::fscanf (FILE * *f*, const char * *format*) [inline]

routine for formatted input

routine for formatted input via fscanf

Parameters:

← ***f*** pointer to `FILE` structure representing the input file

← ***format*** format specification in standard printf-like form

3.6.3.16 template<> void LA::NRVec< complex< double > >::gemv (const complex< double > *beta*, const NRSMat< complex< double > > & *A*, const char *trans*, const complex< double > *alpha*, const NRVec< complex< double > > & *x*) [inline]

perform the **gemv** operation on this complex vector $\vec{y}$, i.e.

$$\vec{y} \leftarrow \alpha op(A) \cdot \vec{x} + \beta \vec{y}$$

Parameters:

← ***beta*** complex parameter β

← ***A*** **complex Hermitian** matrix A stored in packed form

← ***trans*** not used

← ***alpha*** complex parameter α

← ***x*** complex vector $\vec{x}$

See also:

gemm, NRSMat<T>

3.6.3.17 template<> void LA::NRVec< complex< double > >::gemv (const double *beta*, const NRSMat< double > & *A*, const char *trans*, const double *alpha*, const NRVec< complex< double > > & *x*) [inline]

perform the `gemv` operation on this complex vector $\vec{y}$, i.e.

$$\vec{y} \leftarrow \alpha op(A) \cdot \vec{x} + \beta \vec{y}$$

Parameters:

← ***beta*** real parameter β

← ***A*** **real symmetric** matrix A stored in packed form

← ***trans*** if `trans == 'n'` use A directly, otherwise $op(A) \equiv A^{\mathrm{T}}$

← ***alpha*** real parameter α

← ***x*** complex vector $\vec{x}$

See also:

gemm, NRSMat<T>

3.6.3.18 template<> void LA::NRVec< double >::gemv (const double *beta*, const NRSMat< double > & *A*, const char *trans*, const double *alpha*, const NRVec< T > & *x*) [inline]

perform the **gemv** operation on this real vector $\vec{y}$, i.e.

$$\vec{y} \leftarrow \alpha op(A) \cdot \vec{x} + \beta \vec{y}$$

Parameters:

← ***beta*** real parameter β

← ***A*** real symmetric matrix A stored in packed form

← ***trans*** if `trans == 'n'` use A directly, otherwise $op(A) \equiv A^{\mathrm{T}}$

← ***alpha*** real parameter α

← ***x*** real vector $\vec{x}$

See also:

gemm, NRSMat<T>

3.6.3.19 template<> void LA::NRVec< complex< double > >::gemv (const complex< double > *beta*, const NRMat< complex< double > > & *A*, const char *trans*, const complex< double > *alpha*, const NRVec< complex< double > > & *x*) [inline]

perform the **gemv** operation on this complex vector $\vec{y}$, i.e.

$$\vec{y} \leftarrow \alpha op(A) \cdot \vec{x} + \beta \vec{y}$$

Parameters:

← ***beta*** complex parameter β

← ***A*** **complex** matrix A

← ***trans*** if `trans == 'n'` use A directly, otherwise $op(A) \equiv A^{\mathrm{T}}$

← ***alpha*** complex parameter α

← ***x*** real vector $\vec{x}$

See also:

gemm

3.6.3.20 template<> void LA::NRVec< complex< double > >::gemv (const double *beta*, const NRMat< double > & *A*, const char *trans*, const double *alpha*, const NRVec< complex< double > > & *x*) [inline]

perform the **gemv** operation on this complex vector $\vec{y}$, i.e.

$$\vec{y} \leftarrow \alpha op(A) \cdot \vec{x} + \beta \vec{y}$$

Parameters:

← ***beta*** real parameter β

← ***A*** **real** matrix A

← ***trans*** if `trans == 'n'` use A directly, otherwise $op(A) \equiv A^{\mathrm{T}}$

← ***alpha*** real parameter α

← ***x*** real vector $\vec{x}$

See also:

gemm

3.6.3.21 template<> void LA::NRVec< double >::gemv (const double *beta*, const NRMat< double > & *A*, const char *trans*, const double *alpha*, const NRVec< T > & *x*) [inline]

perform the **gemv** operation on this real vector $\vec{y}$, i.e.

$$\vec{y} \leftarrow \alpha op(A) \cdot \vec{x} + \beta \vec{y}$$

Parameters:

← ***beta*** real parameter β

← ***A*** real matrix A

← ***trans*** if `trans == 'n'` use A directly, otherwise $op(A) \equiv A^{\mathrm{T}}$

← ***alpha*** real parameter α

← ***x*** real vector $\vec{x}$

See also:

NRMat<T>::gemm

3.6.3.22 template<typename T> void LA::NRVec< T >::gemv (const T *beta*, const NRSMat< T > & *a*, const char *trans*, const T *alpha*, const NRVec< T > & *x*)

Parameters:

trans just for compatibility reasons

3.6.3.23 template<typename T > void LA::NRVec< T >::get (int *fd*, bool *dim* = 1, bool *transp* = 0) [inline]

routine for unformatted input

routine for raw input

Parameters:

← ***fd*** file descriptor for input

← ***dim*** number of elements intended for input, for dim=0 perform copyonwrite

← ***transp*** reserved

See also:

NRMat<T>::get(), copyonwrite()

3.6.3.24 template<typename T> GPUID LA::NRVec< T >::getlocation () const [inline]

routines for CUDA related stuff

- `getlocation()` gets the protected data member location
- `moveto(const GPUID)` moves underlying data between CPU/GPU memory

3.6.3.25 template<> const double LA::NRVec< complex< double > >::norm () const [inline]

for this complex vector $\vec{x}$ (of N elements) determine the Frobenius norm

Returns:

$\sum_{i=1}^{N} |\vec{x}_i|^2$

3.6.3.26 template<> const double LA::NRVec< double >::norm () const [inline]

for this real vector $\vec{x}$ (of N elements) determine the Frobenius norm

Returns:

$\sum_{i=1}^{N} |\vec{x}_i|^2$

3.6.3.27 template<> NRVec< complex< double > > & LA::NRVec< complex< double > >::normalize (double * *norm*) [inline]

normalize current complex vector (in the Euclidean norm)

Parameters:

← ***norm*** if not NULL, the norm of this vector is stored into *norm

Returns:

reference to the modified vector

3.6.3.28 template<> NRVec< double > & LA::NRVec< double >::normalize (double * *norm*) [inline]

normalize current real vector (in the Euclidean norm)

Parameters:

← ***norm*** if not NULL, the norm of this vector is stored into ∗norm

Returns:

reference to the modified vector

3.6.3.29 template<typename T > LA::NRVec< T >::operator const T * () const [inline]

get the constant pointer to the underlying data structure

get the constant pointer to the underlying data of this vector

Returns:

constant pointer to the first vector element

3.6.3.30 template<typename T > LA::NRVec< T >::operator T * () [inline]

get the pointer to the underlying data structure

get the pointer to the underlying data of this vector

Returns:

pointer to the first vector element

3.6.3.31 template<> const complex< double > LA::NRVec< complex< double > >::operator∗ (const NRVec< complex< double > > & *rhs*) const [inline]

computes the inner product of this complex vector $\vec{x}$ with given complex vector $\vec{y}$ taking conjugation of vector $\vec{x}$ into account

Parameters:

← ***rhs*** complex vector $\vec{y}$

Returns:

$\sum_{i=1}^{N} \vec{x}_i \cdot \vec{y}_i$

3.6.3.32 template<> const double LA::NRVec< double >::operator∗ (const NRVec< double > & *rhs*) const [inline]

computes the inner product of this real vector $\vec{x}$ with given real vector $\vec{y}$]

Parameters:

← ***rhs*** real vector $\vec{y}$

Returns:

$\sum_{i=1}^{N} \vec{x}_i \cdot \vec{y}_i$

3.6.3.33 template<typename T> const T LA::NRVec< T >::operator∗ (const NRVec< T > & *rhs*) const [inline]

compute the Euclidean inner product (with conjugation in complex case)

compute scalar product d of this vector $\vec{x}$ of general type `T` with given vector $\vec{y}$ of type `T` and order N

$$d = \sum_{i=1}^{N} \vec{x}_i \cdot \vec{y}_i$$

Parameters:

← ***rhs*** general vector $\vec{y}$

Returns:

reference to the modified vector

3.6.3.34 template<> NRVec< complex< double > > & LA::NRVec< complex< double > >::operator∗= (const complex< double > & *a*) [inline]

multiplies this complex vector $\vec{x}$ by a complex scalar value α

$$\vec{x}_i \leftarrow \alpha \vec{x}_i$$

Parameters:

← ***a*** complex scalar value α

Returns:

reference to the modified vector

3.6.3.35 template<> NRVec< double > & LA::NRVec< double >::operator∗= (const double & *a*) [inline]

multiplies this real vector $\vec{x}$ by a real scalar value α

$$\vec{x}_i \leftarrow \alpha \vec{x}_i$$

Parameters:

← ***a*** real scalar value α

Returns:

reference to the modified vector

3.6.3.36 template<typename T> NRVec< T > & LA::NRVec< T >::operator*= (const T & *a*) [inline]

multiply this general vector $\vec{x}$ by scalar value λ

$$\vec{x}_i \leftarrow \lambda \vec{x}_i$$

Parameters:

$\leftarrow$ ***a*** scalar value λ

Returns:

reference to the modified vector

3.6.3.37 template<typename T > NRVec< T > & LA::NRVec< T >::operator*= (const NRVec< T > & *rhs*) [inline]

multiplies this vector $\vec{y}$ componentwise by general vector $\vec{x}$

$$\vec{x}_i = \vec{x}_i \times \vec{y}_i$$

Parameters:

$\leftarrow$ ***rhs*** general vector $\vec{y}$

Returns:

reference to the modified vector

3.6.3.38 template<> NRVec< complex< double > > & LA::NRVec< complex< double > >::operator+= (const NRVec< complex< double > > & *rhs*) [inline]

adds a complex vector $\vec{y}$ to this complex vector $\vec{x}$

$$\vec{x} \leftarrow \vec{x} + \vec{y}$$

Parameters:

$\leftarrow$ ***rhs*** complex vector $\vec{y}$

Returns:

reference to the modified vector

3.6.3.39 template<> NRVec< double > & LA::NRVec< double >::operator+= (const NRVec< double > & *rhs*) [inline]

adds a real vector $\vec{y}$ to this real vector $\vec{x}$

$$\vec{x} \leftarrow \vec{x} + \vec{y}$$

Parameters:

$\leftarrow$ ***rhs*** real vector $\vec{y}$

Returns:

reference to the modified vector

3.6.3.40 template<> NRVec< complex< double > > & LA::NRVec< complex< double > >::operator+= (const complex< double > & *a*) [inline]

adds a complex scalar value α to all elements of this complex vector $\vec{x}$

$$\vec{x}_i \leftarrow \vec{x}_i + \alpha$$

Parameters:

$\leftarrow$ ***a*** complex scalar value α being added

Returns:

reference to the modified vector

3.6.3.41 template<> NRVec< double > & LA::NRVec< double >::operator+= (const double & *a*) [inline]

routine for moving vector data between CPU and GPU memory

Parameters:

$\leftarrow$ ***dest*** required location

See also:

NRVec<T>::location, NRVec<T>::getlocation() adds a real scalar value α to all elements of this real vector $\vec{x}$

$$\vec{x}_i \leftarrow \vec{x}_i + \alpha$$

Parameters:

$\leftarrow$ ***a*** real scalar value α being added

Returns:

reference to the modified vector

3.6.3.42 template<typename T> NRVec< T > & LA::NRVec< T >::operator+= (const T & *a*) [inline]

bunch of scalar-vector arithmetic operators defined element-wise

adds given scalar value of type T to all vector elements

Parameters:

← ***a*** scalar value being added

Returns:

reference to the modified vector

3.6.3.43 template<typename T > NRVec< T > & LA::NRVec< T >::operator+= (const NRVec< T > & *rhs*) [inline]

bunch of vector-vector arithmetic operators defined element-wise

adds a vector $\vec{y}$ of general type `T` to this vector $\vec{x}$

$$\vec{x} \leftarrow \vec{x} + \vec{y}$$

Parameters:

← ***rhs*** vector $\vec{y}$ of type `T`

Returns:

reference to the modified vector

3.6.3.44 template<> const NRVec< complex< double > > LA::NRVec< complex< double > >::operator- () const [inline]

unary minus operator in case of complex double-precision vector

Returns:

the modified vector by value

3.6.3.45 template<> const NRVec< double > LA::NRVec< double >::operator- () const [inline]

unary minus operator in case of real double-precision vector

Returns:

the modified vector by value

3.6.3.46 template<typename T > const NRVec< T > LA::NRVec< T >::operator- () const [inline]

unary minus

unary minus operator for vector of general type

Returns:

the modified vector

3.6.3.47 template<> NRVec< complex< double > > & LA::NRVec< complex< double > >::operator-= (const NRVec< complex< double > > & *rhs*) [inline]

subtracts a complex vector $\vec{y}$ from this complex vector $\vec{x}$

$$\vec{x} \leftarrow \vec{x} - \vec{y}$$

Parameters:

$\leftarrow$ ***rhs*** double-precision complex vector $\vec{y}$

Returns:

reference to the modified vector

3.6.3.48 template<> NRVec< double > & LA::NRVec< double >::operator-= (const NRVec< double > & *rhs*) [inline]

subtracts a real vector $\vec{y}$ from this real vector $\vec{x}$

$$\vec{x} \leftarrow \vec{x} - \vec{y}$$

Parameters:

$\leftarrow$ ***rhs*** real vector $\vec{y}$

Returns:

reference to the modified vector

3.6.3.49 template<> NRVec< complex< double > > & LA::NRVec< complex< double > >::operator-= (const complex< double > & *a*) [inline]

subtracts a complex scalar value α from all elements of this complex vector $\vec{x}$

$$\vec{x}_i \leftarrow \vec{x}_i - \alpha$$

Parameters:

$\leftarrow$ ***a*** complex scalar value α being subtracted

Returns:

reference to the modified vector

3.6.3.50 template<> NRVec< double > & LA::NRVec< double >::operator-= (const double & *a*) [inline]

subtracts a real scalar value α from all elements of this real vector $\vec{x}$

$$\vec{x}_i \leftarrow \vec{x}_i - \alpha$$

Parameters:

$\leftarrow$ ***a*** real scalar value α being subtracted

Returns:

reference to the modified vector

3.6.3.51 template<typename T> NRVec< T > & LA::NRVec< T >::operator-= (const T & *a*) [inline]

subtracts given scalar value of type T from all vector elements

Parameters:

$\leftarrow$ ***a*** scalar value being subtracted

Returns:

reference to the modified vector

3.6.3.52 template<typename T > NRVec< T > & LA::NRVec< T >::operator-= (const NRVec< T > & *rhs*) [inline]

subtracts given vector $\vec{y}$ from this vector $\vec{x}$

$$\vec{x}_i = \vec{x}_i - \vec{y}_i$$

Parameters:

$\leftarrow$ ***rhs*** vector $\vec{y}$

Returns:

reference to the modified vector

3.6.3.53 template<typename T > NRVec< T > & LA::NRVec< T >::operator/= (const NRVec< T > & *rhs*) [inline]

divides this vector $\vec{y}$ componentwise by general vector $\vec{x}$

$$\vec{x}_i = \vec{x}_i / \vec{y}_i$$

Parameters:

← ***rhs*** general vector $\vec{y}$

Returns:

reference to the modified vector

3.6.3.54 template<typename T > const bool LA::NRVec< T >::operator< (const NRVec< T > & *rhs*) const [inline]

comparison operator (lexicographical order)

Parameters:

← ***rhs*** vector intended for comparison

Returns:

- `false` current vector is bigger than vector `rhs`
- `true` current vector is smaller than vector `rhs`

3.6.3.55 template<> NRVec< complex< double > > & LA::NRVec< complex< double > >::operator= (const complex< double > & *a*) [inline]

assign complex scalar value to every element of the current vector

Parameters:

← ***a*** scalar value to be assigned

Returns:

reference to the modified vector

3.6.3.56 template<> NRVec< double > & LA::NRVec< double >::operator= (const double & *a*) [inline]

assign real scalar value to every element of the current vector

Parameters:

← ***a*** scalar value to be assigned

Returns:

reference to the modified vector

3.6.3.57 template<typename T> NRVec< T > & LA::NRVec< T >::operator= (const T & *a*) [inline]

assigment operator assigns given scalar to each element of this vector

assign scalar value to every element of the current vector of general type `T`

Parameters:

← ***a*** scalar value to be assigned

Returns:

reference to the modified vector

3.6.3.58 template<typename T > NRVec< T > & LA::NRVec< T >::operator= (const NRVec< T > & *rhs*) [inline]

assignment operator assigns given vector

assigns general vector $\vec{y}$ to this vector $\vec{x}$

- checks for self-assignment
- decreases the reference count and performs deallocation if neccesary
- links the internal data structures with corresponding properties of vector $\vec{y}$
- updates the reference count properly

3.6.3.59 template<typename T > const bool LA::NRVec< T >::operator> (const NRVec< T > & *rhs*) const [inline]

comparison operator (lexicographical order)

Parameters:

← ***rhs*** vector intended for comparison

Returns:

- `true` current vector is bigger than vector `rhs`
- `false` current vector is smaller than vector `rhs`

3.6.3.60 template<typename T > const T & LA::NRVec< T >::operator[ ] (const int *i*) const [inline]

indexing operator giving the element at given position with range checking in the DEBUG mode

Parameters:

← ***i*** position of the required vector element (starting from 0)

Returns:

constant reference to the requested element

3.6.3.61 template<typename T > T & LA::NRVec< T >::operator[ ] (const int *i*) [inline]

indexing operator - index running from zero

indexing operator giving the element at given position with range checking in the DEBUG mode

Parameters:

← ***i*** position of the required vector element (starting from 0)

Returns:

reference to the requested element

3.6.3.62 template<typename T > NRVec< T > & LA::NRVec< T >::operator|= (const NRVec< T > & *rhs*) [inline]

perform deep-copy of given vector

perfrom deep copy

Parameters:

← ***rhs*** vector being copied

See also:

NRVec<T>::copyonwrite()

3.6.3.63 template<> const NRMat< complex< double > > LA::NRVec< complex< double > >::otimes (const NRVec< complex< double > > & *b*, const bool *conj*, const complex< double > & *scale*) const [inline]

computes the outer product of this complex vector $\vec{a}$ with given complex vector $\vec{b}$ and scales the resulting matrix with factor α, i.e. the matrix elements of the final matrix A can be expressed as

$$A_{i,j} = \alpha \cdot \vec{a}_i \vec{b}_j$$

in case `conj = true`, the result is

$$A_{i,j} = \alpha \cdot \vec{a}_i \vec{b}_j^*$$

Parameters:

← ***b*** complex vector $\vec{b}$

← ***conj*** determines whther the vector $\vec{b}$ is conjugated

← ***scale*** complex scaling factor α

3.6.3.64 template<> const NRMat< double > LA::NRVec< double >::otimes (const NRVec< double > & *b*, const bool *conj*, const double & *scale*) const [inline]

computes the outer product of this real vector $\vec{a}$ with given real vector $\vec{b}$ and scales the resulting matrix with factor α, i.e. the matrix elements of the final matrix A can be expressed as

$$A_{i,j} = \alpha \cdot \vec{a}_i \vec{b}_j$$

Parameters:

← ***b*** real vector $\vec{b}$
← ***conj*** not used
← ***scale*** real factor α

3.6.3.65 template<typename T > void LA::NRVec< T >::put (int *fd*, bool *dim* = 1, bool *transp* = 0) const [inline]

routine for unformatted output

routine for raw output

Parameters:

← ***fd*** file descriptor for output
← ***dim*** number of elements intended for output
← ***transp*** reserved

See also:

NRMat<T>::put()

3.6.3.66 template<> void LA::NRVec< complex< double > >::randomize (const double & *x*) [inline]

fill the complex vector with pseudorandom numbers generated using uniform distribution the real and imaginary parts are generated independently

Parameters:

← ***x*** specification of the interval $[0, x]$ for the random number generator

Returns:

- `false` current vector is bigger than vector `rhs`
- `true` current vector is smaller than vector `rhs`

3.6.3.67 template<> void LA::NRVec< double >::randomize (const double & *x*) [inline]

fill the real vector with pseudorandom numbers generated using uniform distribution

Parameters:

← ***x*** specification of the interval $[0, x]$ for the random number generator

3.6.3.68 template<typename T > void LA::NRVec< T >::resize (const int *n*) [inline]

resize the current vector

resizes this vector

Parameters:

← ***n*** requested size

3.6.3.69 template<typename T > int LA::NRVec< T >::size () const [inline]

determine the number of elements

determine the number of elements of this vector

Returns:

length of this vector

3.6.3.70 template<typename T > const NRVec< T > LA::NRVec< T >::unitvector () const [inline]

get normalized copy of this vector

create normalized copy of this vector

Returns:

copy of this vector after normalization

See also:

NRVec<T>::normalize()

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

- vec.h
- sparsemat.cc
- vec.cc

Index