

DMatrix and SparseMatrix C++ classes

Generated by Doxygen 1.7.3

Thu Jul 28 2011 11:41:53

Contents

1	DMatrix and SparseMatrix classes	1
1.1	Introduction	1
1.2	License	1
1.3	Installing the library	2
1.4	Examples of use	2
2	Class Index	3
2.1	Class List	3
3	Class Documentation	5
3.1	alist Struct Reference	5
3.2	cilist Struct Reference	5
3.3	cclist Struct Reference	5
3.4	complex Struct Reference	6
3.5	DMatrix Class Reference	6
3.5.1	Detailed Description	28
3.5.2	Constructor & Destructor Documentation	28
3.5.2.1	DMatrix	28
3.5.2.2	DMatrix	28
3.5.2.3	DMatrix	29
3.5.2.4	DMatrix	29
3.5.2.5	DMatrix	29
3.5.3	Member Function Documentation	29
3.5.3.1	Allocate	29
3.5.3.2	AllocateAuxArr	29
3.5.3.3	assign	30
3.5.3.4	AssignmentToColonReference	30
3.5.3.5	AssignmentToColonReference	30
3.5.3.6	colMult	31
3.5.3.7	Column	31
3.5.3.8	compMat	31
3.5.3.9	DeAllocate	31
3.5.3.10	DeAllocateAuxArr	32
3.5.3.11	diag	32
3.5.3.12	elem	32
3.5.3.13	elem	32
3.5.3.14	element	33
3.5.3.15	FillWithZeros	33
3.5.3.16	find	33

3.5.3.17	find	33
3.5.3.18	Fprint	34
3.5.3.19	geta	34
3.5.3.20	getatype	34
3.5.3.21	GetAuxPr	34
3.5.3.22	GetConstPr	35
3.5.3.23	getm	35
3.5.3.24	getm	35
3.5.3.25	GetMType	35
3.5.3.26	getn	35
3.5.3.27	getn	35
3.5.3.28	GetNoCols	36
3.5.3.29	GetNoCols	36
3.5.3.30	GetNoRows	36
3.5.3.31	GetNoRows	36
3.5.3.32	GetPr	36
3.5.3.33	GetReferencedDMatrixPointer	36
3.5.3.34	input_matrix	37
3.5.3.35	isEmpty	37
3.5.3.36	isThereError	37
3.5.3.37	isVector	37
3.5.3.38	Load	37
3.5.3.39	MemCpyArray	38
3.5.3.40	mpow	38
3.5.3.41	operator!=	38
3.5.3.42	operator!=	38
3.5.3.43	operator%	39
3.5.3.44	operator&	39
3.5.3.45	operator&&	39
3.5.3.46	operator()	40
3.5.3.47	operator()	40
3.5.3.48	operator()	40
3.5.3.49	operator()	41
3.5.3.50	operator()	41
3.5.3.51	operator()	41
3.5.3.52	operator()	42
3.5.3.53	operator()	42
3.5.3.54	operator()	42
3.5.3.55	operator()	42
3.5.3.56	operator()	43
3.5.3.57	operator()	43
3.5.3.58	operator()	43
3.5.3.59	operator()	44
3.5.3.60	operator()	44
3.5.3.61	operator()	44
3.5.3.62	operator*	45
3.5.3.63	operator*	45
3.5.3.64	operator*==	45
3.5.3.65	operator*==	46
3.5.3.66	operator+	46

3.5.3.67	operator+	46
3.5.3.68	operator+=	46
3.5.3.69	operator-	47
3.5.3.70	operator-	47
3.5.3.71	operator-=	47
3.5.3.72	operator/	47
3.5.3.73	operator/	48
3.5.3.74	operator/=	48
3.5.3.75	operator<	48
3.5.3.76	operator<	48
3.5.3.77	operator<=	49
3.5.3.78	operator<=	49
3.5.3.79	operator=	49
3.5.3.80	operator=	50
3.5.3.81	operator=	50
3.5.3.82	operator==	50
3.5.3.83	operator==	51
3.5.3.84	operator>	51
3.5.3.85	operator>	51
3.5.3.86	operator>=	52
3.5.3.87	operator>=	52
3.5.3.88	operator^	52
3.5.3.89	operator	53
3.5.3.90	operator	53
3.5.3.91	Print	53
3.5.3.92	PrintInfo	54
3.5.3.93	random_gaussian	54
3.5.3.94	random_uniform	54
3.5.3.95	Read	54
3.5.3.96	Resize	55
3.5.3.97	Row	55
3.5.3.98	rowMult	55
3.5.3.99	Save	55
3.5.3.100	SetColIndexPointer	56
3.5.3.101	SetColumn	56
3.5.3.102	SetMType	56
3.5.3.103	SetPrintLevel	56
3.5.3.104	SetReferencedDMatrixPointer	57
3.5.3.105	SetRow	57
3.5.3.106	SetRowIndexPointer	57
3.5.3.107	SetSubMatrix	57
3.5.3.108	sub_matrix	58
3.5.3.109	SwapColumns	58
3.5.3.110	SwapRows	58
3.5.3.111	Transpose	59
3.5.4	Friends And Related Function Documentation	59
3.5.4.1	Abs	59
3.5.4.2	any	59
3.5.4.3	Chol	59
3.5.4.4	CholeskyDecomp	60

3.5.4.5	CholeskyRoot	60
3.5.4.6	CholeskySolution	60
3.5.4.7	CholFSolve	61
3.5.4.8	CholSolve	61
3.5.4.9	colon	62
3.5.4.10	colon	62
3.5.4.11	colon	62
3.5.4.12	colon	63
3.5.4.13	colon	63
3.5.4.14	cond	63
3.5.4.15	cos	63
3.5.4.16	cosh	64
3.5.4.17	cov	64
3.5.4.18	cov	64
3.5.4.19	cross	65
3.5.4.20	crossProduct	65
3.5.4.21	det	65
3.5.4.22	diag	66
3.5.4.23	dot	66
3.5.4.24	dotProduct	66
3.5.4.25	eig	66
3.5.4.26	elemDivision	67
3.5.4.27	elemProduct	67
3.5.4.28	enorm	67
3.5.4.29	error_message	68
3.5.4.30	exp	68
3.5.4.31	expm	68
3.5.4.32	eye	69
3.5.4.33	eye	69
3.5.4.34	find	69
3.5.4.35	Fnorm	69
3.5.4.36	identity	70
3.5.4.37	identity	70
3.5.4.38	InfNorm	70
3.5.4.39	inv	70
3.5.4.40	isSymmetric	71
3.5.4.41	kronProduct	71
3.5.4.42	length	71
3.5.4.43	linspace	72
3.5.4.44	log	72
3.5.4.45	LQ	72
3.5.4.46	LSMNSolve	72
3.5.4.47	LU	73
3.5.4.48	LUFSSolve	73
3.5.4.49	LUSolve	73
3.5.4.50	MatrixSign	74
3.5.4.51	Max	74
3.5.4.52	MaxAbs	75
3.5.4.53	mean	75
3.5.4.54	Min	75

3.5.4.55	MinAbs	76
3.5.4.56	mpow	76
3.5.4.57	norm	76
3.5.4.58	null	76
3.5.4.59	ones	77
3.5.4.60	operator*	77
3.5.4.61	operator-	77
3.5.4.62	orth	78
3.5.4.63	pinv	78
3.5.4.64	prod	78
3.5.4.65	Product	78
3.5.4.66	ProductT	79
3.5.4.67	QR	79
3.5.4.68	QRFSolve	79
3.5.4.69	QRSolve	80
3.5.4.70	randn	80
3.5.4.71	randu	80
3.5.4.72	rank	81
3.5.4.73	rcond	81
3.5.4.74	reshape	81
3.5.4.75	schur	82
3.5.4.76	sin	82
3.5.4.77	sinh	82
3.5.4.78	sort	82
3.5.4.79	sort	83
3.5.4.80	Sqrt	83
3.5.4.81	Std	83
3.5.4.82	sum	84
3.5.4.83	SVD	84
3.5.4.84	SVDSolve	84
3.5.4.85	tan	85
3.5.4.86	tanh	85
3.5.4.87	toc	85
3.5.4.88	TProduct	85
3.5.4.89	TProductT	86
3.5.4.90	tra	86
3.5.4.91	trace	86
3.5.4.92	triu	87
3.5.4.93	var	87
3.5.4.94	vec	87
3.5.4.95	zeros	87
3.5.5	Member Data Documentation	88
3.5.5.1	atype	88
3.5.5.2	mtype	88
3.6	doublecomplex Struct Reference	88
3.7	ErrorHandler Class Reference	88
3.7.1	Detailed Description	89
3.7.2	Constructor & Destructor Documentation	89
3.7.2.1	ErrorHandler	89
3.8	icilist Struct Reference	89

3.9	InitializeDMatrixClass Class Reference	90
3.9.1	Detailed Description	90
3.10	inlist Struct Reference	90
3.11	Multitype Union Reference	91
3.12	Namelist Struct Reference	91
3.13	olist Struct Reference	92
3.14	Vardesc Struct Reference	92

Chapter 1

DMatrix and SparseMatrix classes

1.1 Introduction

The author developed the main features of the [DMatrix](#) class between 1994 and 1999. The class makes extensive use of operator overloading in order to facilitate the implementation in C++ of complicated matrix expressions, and it has interfaces to a number of LAPACK routines. In 2008, the class has been tested with current compilers, its functionality was expanded, and the code was published under the GNU Lesser General Public License. The class at present is restricted to dense and real matrices. In 2008, the SparseMatrix class was added to the library to incorporate basic sparse matrix functionality. The SparseMatrix class offers interfaces to some functions available in the CXSparse and LUSOL libraries.

The library should compile without problems with the following C++ compilers: GNU C++ version 4.X and Microsoft Visual Studio 2005.

1.2 License

This work is copyright (c) Victor M. Becerra (2009)

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details. You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA, or visit <http://www.gnu.org/licenses/>

Author: Dr. Victor M. Becerra, University of Reading, School of Systems Engineering, P.O. Box 225, Reading RG6 6AY, United Kingdom, e-mail: v.m.becerra@ieee.org.

1.3 Installing the library

See the INSTALL file.

1.4 Examples of use

See the source code in the examples directory.

Chapter 2

Class Index

2.1 Class List

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

alist	5
cilst	5
cllist	5
complex	6
DMatrix (DMatrix class)	6
doublecomplex	88
ErrorHandler (ErrorHandler class)	88
icilst	89
InitializeDMatrixClass (InitializeDMatrixClass class)	90
inlist	90
Multitype	91
Namelist	91
olist	92
Vardesc	92

Chapter 3

Class Documentation

3.1 alist Struct Reference

Public Attributes

- flag **aerr**
- ftntint **aunit**

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

- f2c.h

3.2 cilist Struct Reference

Public Attributes

- flag **cierr**
- ftntint **ciunit**
- flag **ciend**
- char * **cifmt**
- ftntint **cirec**

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

- f2c.h

3.3 cllist Struct Reference

Public Attributes

- flag **cerr**

- `ftnint cunit`
- `char * csta`

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

- `f2c.h`

3.4 complex Struct Reference

Public Attributes

- `real r`
- `real i`

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

- `f2c.h`

3.5 DMatrix Class Reference

[DMatrix](#) class.

```
#include <dmatrixv.h>
```

Public Member Functions

- `void input_matrix ()`
Allows the user to enter the elements of a matrix using command line prompts.
- `void Print (const char *text) const`
Prints the elements of a [DMatrix](#) object.
- `void PrintInfo (const char *text) const`
Prints information about a [DMatrix](#) object.
- `void Read (FILE *filex)`
Reads the elements of a matrix from a file.
- `void Save (const char *FileName)`
Saves the elements of a matrix to a file.
- `void Load (const char *FileName)`
Reads the elements of a matrix from a file.

- void **Fprint** (FILE *filex)
Prints the elements of a matrix elements matrix to a file.
- void **FillWithZeros** (void)
Assigns a zero value to each element of a matrix.
- void **SwapRows** (int i, int j)
Swaps two rows of a matrix.
- void **SwapColumns** (int i, int j)
Swaps two columns of a matrix.
- void **Transpose** (void)
Transposes a matrix.
- void **diag** (const **DMatrix** &dd)
Assigns values to the diagonal elements of a matrix, while all off-diagonal elements are set to zero.
- void **SetColumn** (const **DMatrix** &Col, int icol)
Assigns values to a column of a matrix, while other columns are left untouched.
- void **SetRow** (const **DMatrix** &Row, int irow)
Assigns values to a row of a matrix, while other columns are left untouched.
- void **colMult** (long c, double x)
Multiplies the elements of a specified column of a matrix by a constant scalar value.
- void **rowMult** (long r, double x)
Multiplies the elements of a specified row of a matrix by a constant scalar value.
- **DMatrix** & **find** (**DMatrix** &I, **DMatrix** &J) const
Finds non-zero values of a matrix.
- **DMatrix** & **find** (int *I, int *J) const
Finds non-zero values of a matrix.
- **DMatrix** & **sub_matrix** (long r1, long r2, long c1, long c2) const
Extracts a specified sub-matrix from a matrix.
- void **SetSubMatrix** (long row, long col, const **DMatrix** &A)
Assigns the elements of a matrix object to a section of the calling object.
- long **getn** ()
Gets the number of rows from the calling object.

- long `getm ()`
Gets the number of columns from the calling object.
- long `getn () const`
Gets the number of rows from the calling object.
- long `getm () const`
Gets the number of columns from the calling object.
- long `GetNoRows ()`
Gets the number of rows from the calling object.
- long `GetNoRows () const`
Gets the number of rows from the calling object.
- long `GetNoCols ()`
Gets the number of columns from the calling object.
- long `GetNoCols () const`
Gets the number of columns from the calling object.
- double * `GetPr ()`
Gets the pointer to the array where the elements of the matrix are stored.
- double * `GetConstPr () const`
Gets a pointer to the array where the elements of the matrix are stored (for const objects)
- double * `geta ()`
Gets a pointer to the array where the elements of the matrix are stored.
- int `getatype ()`
Gets the type of element storage of a matrix.
- int `isEmpty ()`
Check if a matrix object is empty, if other words this method checks if the calling object has zero elements.
- int `isVector () const`
Check if a matrix object contains a row or column vector.
- double `element (long i, long j)`
Returns the value of a specified element of a matrix.
- double & `elem (long i, long j)`
Returns a reference to the specified element of a matrix.

- double [elem](#) (long i, long j) const
Returns the value of a specified element of a matrix.
- [DMatrix](#) & [Column](#) (long icol) const
Returns a [DMatrix](#) object containing a specified column of the calling object.
- [DMatrix](#) & [Row](#) (long irow) const
Returns a [DMatrix](#) object containing a specified row of the calling object.
- [DMatrix](#) & [mpow](#) (int p)
Computes and returns the integer power of a matrix.
- [DMatrix](#) (void)
Default constructor. Creates an empty matrix.
- [DMatrix](#) (long Initn, long Initm)
Constructor with dimensions. Creates a matrix with allocated storage given specified numbers of rows and columns.
- [DMatrix](#) (long vDim, double *v, long Initn, long Initm)
Constructor with dimensions using pre-allocated storage.
- [DMatrix](#) (long Initn)
Constructor with a single dimension, creates a column vector.
- [DMatrix](#) (long Initn, long Initm, double a1,...)
Constructor using a variable list of element values.
- [DMatrix](#) (const [DMatrix](#) &A)
Copy constructor. Creates a new [DMatrix](#) object with the same dimensions and element values as a given [DMatrix](#) object.
- [~DMatrix](#) ()
Destructor. Destroys a previously created [DMatrix](#) object and frees any allocated memory.
- void [Resize](#) (long nnrow, long nncol)
Changes the number of rows and columns of an existing matrix. Allocates new memory if necessary. If the calling object uses preallocated memory and the requested size would exceed that memory, an error is thrown.
- void [assign](#) (long rows, long columns, double a1,...)
Assigns values to the elements of an existing [DMatrix](#) object using a variable list of values. Resizes the matrix if necessary.
- void [MemCpyArray](#) (double *aptr)

Copies values to the elements of a [DMatrix](#) object from an existing array. The number of elements of the [DMatrix](#) object is assumed to be the number of values to be copied.

- [DMatrix](#) & [operator+](#) (const [DMatrix](#) &rval) const
Matrix addition operator. The sizes of the matrices being added should be the same, otherwise an error is thrown.
- [DMatrix](#) & [operator+=](#) (const [DMatrix](#) &rval)
Matrix addition and substitution operator. The sizes of the matrices being added should be the same, otherwise an error is thrown. The left hand side object elements are replaced with the result of the operation.
- [DMatrix](#) & [operator+](#) (double x) const
Adds a scalar real value to each element of the matrix. .
- [DMatrix](#) & [operator-=](#) (const [DMatrix](#) &rval)
Matrix subtraction and substitution operator. The sizes of the matrices being subtracted should be the same, otherwise an error is thrown. The left hand side object elements are replaced with the result of the operation.
- [DMatrix](#) & [operator-](#) (const [DMatrix](#) &rval) const
Matrix subtraction operator. The sizes of the matrices being subtracted should be the same, otherwise an error is thrown.
- [DMatrix](#) & [operator-](#) (double x) const
Subtracts a scalar real value from each element of the matrix. .
- [DMatrix](#) & [operator*](#) (const [DMatrix](#) &rval) const
Matrix product operator. Returns the result of the matrix product of the calling object (left hand side of the operator) and the right hand side object. The inner dimensions of the objects being multiplied should be consistent, otherwise an error will be thrown.
- [DMatrix](#) & [operator*=](#) (const [DMatrix](#) &rval)
Matrix product operator with substitution. Computes the matrix product of the calling object (left hand side of the operator) and the right hand side object. The inner dimensions of the objects being multiplied should be consistent, otherwise an error will be thrown. The calling object is modified to store the results of the operation.
- [DMatrix](#) & [operator*](#) (double Arg) const
Computes the product of a matrix (left hand side of the operator) times a real scalar (right hand side value) and replaces the left hand side object with the result of the operation.
- [DMatrix](#) & [operator*=](#) (double Arg)
Computes the product of a matrix (left hand side of the operator) times a real scalar (right hand side value), and modifies the calling object to store the result of the operation.
- [DMatrix](#) & [operator/](#) (double Arg) const

Computes the division of a matrix (left hand side of the operator) by a real scalar (right hand side value).

- **DMatrix & operator/** (const **DMatrix** &rval) const

Computes the right division of a matrix (left hand side of the operator) by another matrix (right hand side value). This is conceptually equivalent to multiplying the left object by the inverse of the right hand side object but it is computed in a more efficient way. The dimensions of the matrices must be consistent, otherwise an error is returned. The right hand side object must be a square matrix.

- **DMatrix & operator%** (const **DMatrix** &rval) const

Computes the left division of a matrix (left hand side of the operator) by another matrix (right hand side value). This is conceptually equivalent to multiplying the inverse of the left object by the right hand side object but it is computed in a more efficient way. The dimensions of the matrices must be consistent, otherwise an error is returned. The left hand side object must be a square matrix.

- **DMatrix & operator/=** (double Arg)

Computes the division of a matrix (left hand side of the operator) by a real scalar (right hand side value) and modifies the left hand side object with the result of the operation.

- **DMatrix & operator=** (const **DMatrix** &rval)

Matrix assignment. The size of the left hand side object is modified if necessary, and the values of all real elements of the right hand side object are copied to the left hand side object.

- **DMatrix & operator=** (double val)

Matrix assignment to a scalar. The size of the left hand side object is modified to one row by one column if necessary, and the value of the right hand side argument is copied to the single element of the matrix. If the calling object is a "colon reference" matrix, then the right hand side value is copied to each element of the referenced array elements.

- **DMatrix & operator=** (const char *str)

Matrix assignment to a constant matrix defined as a character string using the bracket notation used in Matlab and Octave. The size of the left hand side object is modified if necessary. For example, the identity matrix of size two by two would be entered as "[1.0 0.0;0.0 1.0]".

- **DMatrix & operator||** (const **DMatrix** &B) const

Concatenates two matrices side by side. The dimensions number of rows of the matrices involved must be the same, otherwise an error is thrown. The number of columns of the resulting matrix is the addition of the number of columns of both matrices involved.

- **DMatrix & operator&&** (const **DMatrix** &B) const

Stacks the right hand side matrix below the left hand side matrix. The dimensions number of columns of the matrices involved must be the same, otherwise an error is thrown. The number of rows of the resulting matrix is the addition of the number of rows of both matrices involved.

- [DMatrix](#) & [operator[^]](#) (double x)

*Elementwise power operator. Returns a [DMatrix](#) object with the same dimensions of the calling object and each of its elements is computed as the corresponding element of the calling object to the power of the right hand side argument. Care must be taken when using this operator as the associations do not work in the same way as with the * operator. It is highly recommended to use parenthesis every time this operator is used. For example use it as follows: (A[^]x).*

- [DMatrix](#) & [operator&](#) (const [DMatrix](#) B) const

*Elementwise product operator. Returns a [DMatrix](#) object with the same dimensions of the calling objects and each of its elements is computed as the product of the corresponding elements of the calling object and the right hand side object. The dimensions of the calling objects must be the same, otherwise an error is thrown. Care must be taken when using this operator as the associations do not work in the same way as with the * operator. It is highly recommended to use parenthesis every time this operator is used. For example use it as follows: (A&B).*

- [DMatrix](#) & [operator|](#) (const [DMatrix](#) B) const

Elementwise division operator. Returns a [DMatrix](#) object with the same dimensions of the calling objects and each of its elements is computed as the of the corresponding element of the calling object by the corresponding element of the right hand side object. The dimensions of the calling objects must be the same, otherwise an error is thrown. Care must be taken when using this operator as the associations do not work in the same way as with the / operator. It is highly recommended to use parenthesis every time this operator is used. For example use it as follows: (A|B).

- double & [operator\(\)](#) (long row, long col)

Matrix indexing. Returns a reference to the matrix element located at the position indicated by the row and column indices. Indices start from 1.

- double & [operator\(\)](#) (long row, const char *end)

Matrix indexing. Returns a reference to the matrix element located at the position indicated by the row index and the last column. Indices start from 1. An error is thrown in case of zero or negative indices. The matrix is resized if necessary.

- double & [operator\(\)](#) (const char *end, long col)

Matrix indexing. Returns a reference to the matrix element located at the position indicated by the column index and the last row. Indices start from 1. An error is thrown in case of a range violation.

- double [operator\(\)](#) (long row, long col) const

Matrix indexing. Returns the value of the matrix element located at the position indicated by the row and column indices. Indices start from 1. An error is thrown in case of a range violation.

- double [operator\(\)](#) (long row, const char *end) const

Matrix indexing. Returns the value of the matrix element located at the position indicated by the row index and the last column. Indices start from 1. An error is thrown in case of a range violation.

- double `operator()` (const char *end, long col) const
Matrix indexing. Returns the value of the matrix element located at the position indicated by the column index and the last row. Indices start from 1. An error is thrown in case of a range violation.
- double & `operator()` (long index)
Single index matrix indexing. Returns a reference to the matrix element located at the linear position indicated by the index, assuming column major storage. The index starts from 1. The matrix is resized if necessary. An error is thrown in case of zero or negative indices.
- double & `operator()` (const char *end)
Access to last linear element. Returns a reference to the last linear matrix element, assuming column major storage.
- double `operator()` (long k) const
Single index matrix indexing. Returns the value of the matrix element located at the linear position indicated by the index, assuming column major storage. The index starts from 1. An error is thrown in case of range error.
- double `operator()` (const char *end) const
Access to last linear element. Returns the value of the last linear matrix element, assuming column major storage.
- `DMatrix & operator>` (double val) const
Checks if each element of the `DMatrix` object on the left hand side is greater than the right hand side value. The result is a `DMatrix` object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise.
- `DMatrix & operator<` (double val) const
Checks if each element of the `DMatrix` object on the left hand side is lower than the right hand side value. The result is a `DMatrix` object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise.
- `DMatrix & operator>=` (double val) const
Checks if each element of the `DMatrix` object on the left hand side is greater or equal than the right hand side value. The result is a `DMatrix` object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise.
- `DMatrix & operator<=` (double val) const
Checks if each element of the `DMatrix` object on the left hand side is lower or equal than the right hand side value. The result is a `DMatrix` object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise.
- `DMatrix & operator==` (double val) const
Checks if each element of the `DMatrix` object on the left hand side is equal to the right hand side value. The result is a `DMatrix` object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise.

- **DMatrix & operator!=** (double val) const

*Checks if each element of the **DMatrix** object on the left hand side is different from the right hand side value. The result is a **DMatrix** object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise.*

- **DMatrix & operator>** (const **DMatrix** &val) const

*Elementwise matrix comparison. Checks if each element of the matrix on the left hand side is greater than the corresponding element of the right hand side matrix. The result is a **DMatrix** object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise. The dimensions of the two matrices involved must be the same, otherwise an error is thrown.*

- **DMatrix & operator<** (const **DMatrix** &val) const

*Elementwise matrix comparison. Checks if each element of the matrix on the left hand side is lower than the corresponding element of the right hand side matrix. The result is a **DMatrix** object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise. The dimensions of the two matrices involved must be the same, otherwise an error is thrown.*

- **DMatrix & operator>=** (const **DMatrix** &val) const

*Elementwise matrix comparison. Checks if each element of the matrix on the left hand side is greater or equal than the corresponding element of the right hand side matrix. The result is a **DMatrix** object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise. The dimensions of the two matrices involved must be the same, otherwise an error is thrown.*

- **DMatrix & operator<=** (const **DMatrix** &val) const

*Elementwise matrix comparison. Checks if each element of the matrix on the left hand side is lower or equal than the corresponding element of the right hand side matrix. The result is a **DMatrix** object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise. The dimensions of the two matrices involved must be the same, otherwise an error is thrown.*

- **DMatrix & operator==** (const **DMatrix** &val) const

*Elementwise matrix comparison. Checks if each element of the matrix on the left hand side is equal to the corresponding element of the right hand side matrix. The result is a **DMatrix** object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise. The dimensions of the two matrices involved must be the same, otherwise an error is thrown.*

- **DMatrix & operator!=** (const **DMatrix** &val) const

*Elementwise matrix comparison. Checks if each element of the matrix on the left hand side is different from the corresponding element of the right hand side matrix. The result is a **DMatrix** object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise. The dimensions of the two matrices involved must be the same, otherwise an error is thrown.*

- **DMatrix & operator()** (const **DMatrix** &RowIndex, const **DMatrix** &ColIndx)

Submatrix extraction and referencing using arrays of indices.

- [DMatrix](#) & [operator\(\)](#) (const [DMatrix](#) &RowIndex)
Linear sub-vector extraction and referencing using an array of indices assuming column-major storage.
- [DMatrix](#) & [operator\(\)](#) (const [DMatrix](#) &RowIndex, long col)
Sub-vector extraction and referencing using an array of row indices for a given column.
- [DMatrix](#) & [operator\(\)](#) (long row, const [DMatrix](#) &ColIndex)
Sub-vector extraction and referencing using an array of column indices for a given row.
- [DMatrix](#) & [operator\(\)](#) (const [DMatrix](#) &RowIndex, const char *end)
Sub-vector extraction and referencing using an array of row indices for the last column of a matrix.
- [DMatrix](#) & [operator\(\)](#) (const char *end, const [DMatrix](#) &ColIndex)
Sub-vector extraction and referencing using an array of column indices for the last column of a matrix.

Static Public Member Functions

- static void [AllocateAuxArr](#) (void)
Allocates the array of auxiliary (temporary) objects used by the class.
- static void [DeAllocateAuxArr](#) (void)
De-allocates the array of auxiliary (temporary) objects previously allocated by [AllocateAuxArr\(\)](#)
- static double [random_uniform](#) (void)
Returns a pseudo-random uniformly distributed number in the range [0,1].
- static double [random_gaussian](#) (void)
Returns a pseudo-random Gaussian distributed number with zero mean and unit variance.
- static [DMatrix](#) ** [GetAuxPr](#) (void)
Gets a pointer to the array of auxiliary objects.
- static int [isThereError](#) (void)
Checks if the error flag has been raised. If so, a 1 is returned, 0 otherwise.
- static void [SetPrintLevel](#) (int plevel)
Sets the print level.

- static int **PrintLevel** ()
- static double **GetEPS** ()

This function returns the machine numerical precision.

Protected Member Functions

- void **SetAuxFlag** (int arg)
Sets the value of auxFlag.
- int **GetAuxFlag** ()
Gets the value of auxFlag.
- void **Allocate** (long size)
Allocate memory to store matrix elements.
- void **DeAllocate** ()
*De-allocate memory previously allocated with **DMatrix::Allocate**().*
- **DMatrix** & **compMat** (const **DMatrix** &m2, char op) const
Elementwise comparison of matrix elements.
- void **SetReferencedDMatrixPointer** (**DMatrix** *arg)
Sets the value of the referenced matrix pointer.
- **DMatrix** * **GetReferencedDMatrixPointer** ()
Gets the value of the referenced matrix pointer.
- void **SetRowIndexPointer** (const **DMatrix** *arg)
Sets the row index pointer.
- void **SetColIndexPointer** (const **DMatrix** *arg)
Sets the column index pointer.
- void **SetMType** (int arg)
Sets the type of matrix.
- int **GetMType** ()
Gets the type of matrix.
- **DMatrix** & **AssignmentToColonReference** (const **DMatrix** &A)
Assigns a matrix to the values pointed to by a colon reference matrix.
- **DMatrix** & **AssignmentToColonReference** (double arg)
Assigns a double value to each value pointed to by a colon reference matrix.

Static Protected Member Functions

- static [DMatrix *](#) [GetTempPr](#) (int i)
returns pointer to the i-th temporary object
- static int [GetMemberFlag](#) ()
Gets the memberFlag value from the object.
- static void [SetMemberFlag](#) (int arg)
Sets the memberFlag value.
- static void [ChkAuxArrays](#) ()
Checks the auxiliary objects.
- static int [GetNoAuxArr](#) ()
Gets the number of temporary objects.
- static long [GetDimAux](#) ()
Gets the dimensions of the array of temporary objects.
- static int [GetInitFlag](#) ()
Gets the value of initFlag.
- static void [SetInitFlag](#) (int arg)
Sets the value of initFlag.
- static int [GetAuxIndx](#) ()
Gets the current index of temporary objects.
- static int [IncrementAuxIndx](#) ()
Increments the index of temporary objects.
- static int [DecrementAuxIndx](#) ()
Decrements the index of temporary objects.
- static void [SetAuxIndx](#) (int i)
Sets the index of temporary objects.
- static void [SetDimAux](#) (long dd)
Sets the dimension of each object in the array of temporary objects.
- static void [SetNoAuxArr](#) (int nn)
Sets the number of auxiliary objects.
- static void [RaiseErrorFlag](#) ()
Sets the errorFlag member to true.

- static clock_t [GetStartTicks](#) (void)
Gets the value of start_clock member.
- static void [SetStartTicks](#) (clock_t st)
Sets the value of start_clock member.

Protected Attributes

- double * [a](#)
Array of doubles to store matrix elements using column major storage.
- long [n](#)
Number of matrix rows.
- long [m](#)
Number of matrix columns.
- long [asize](#)
Number of allocated elements in a.
- int [atype](#)
- int [mtype](#)
- int [auxFlag](#)
Flag to indicate auxiliary (temporary) matrix flag = 1 if temporary matrix, 0 otherwise.
- int [allocated](#)
Flag to indicate that element storage has been allocated.
- [DMatrix](#) * [mt](#)
Referenced matrix pointer.
- const [DMatrix](#) * [rowIndx](#)
Row indices.
- const [DMatrix](#) * [colIndx](#)
Column indices.

Static Protected Attributes

- static [DMatrix](#) * [auxPr](#)
Array of Temporary matrices.

- static int [noAuxArr](#)
Number of auxiliary matrices.
- static long [dimAux](#)
Dimension of auxiliary arrays.
- static int [auxIndx](#)
Index of used auxiliary matrices.
- static int [memberFlag](#)
Member function flag to control resetting of auxIndx.
- static int [initFlag](#)
Flag to indicate aux arrays allocation.
- static const double [MACH_EPS](#)
Machine precision constant.
- static int [errorFlag](#)
Flag to indicate error condition.
- static int [print_level](#)
Print level flag, 1: output sent to stderr, 0: no output sent.
- static long [seed](#) []
current state of each stream
- static int [stream](#)
stream index for pseudo-random number generator
- static time_t [start_time](#)
variable to store start time after [tic\(\)](#) call.
- static clock_t [start_clock](#)
clock_t variable

Friends

- void [CholeskyDecomp](#) (DMatrix &A, int n, DMatrix &pM)
Cholesky decomposition of a matrix.
- void [CholeskySolution](#) (const DMatrix &A, int n, const DMatrix &pM, const DMatrix &bM, DMatrix &xM)
Cholesky solution using the Cholesky decomposition of a matrix.

- **DMatrix & operator-** (const **DMatrix** &A)
Matrix unary minus operator. Returns an object of the same dimensions as A but with changed element signs.
- **DMatrix & colon** (double i1, double increment, double i2)
*This function generates a **DMatrix** object with a vector starting from a given value, with given increments and ending in a given value.*
- **DMatrix & colon** (int i1, int increment, int i2)
*This function generates **DMatrix** object with a vector starting from a given value, with given increments and ending in a given value.*
- **DMatrix & colon** (int i1, int i2)
*This function generates **DMatrix** object with a vector starting from a given value, with unit increments, and ending in a given value.*
- **DMatrix & colon** (double i1, double i2)
*This function generates **DMatrix** object with a vector starting from a given value, with unit increments, and ending in a given value.*
- **DMatrix & colon** (void)
*This function generates a special **DMatrix** object with one row and one column which is understood by the indexing functions that take a **DMatrix** object as an argument to mean "all rows" or "all columns".*
- **int any** (const **DMatrix** &A)
*This function returns a 1 if any element of **DMatrix** object that is passed as argument is non-zero, otherwise it returns a zero.*
- **DMatrix & mpow** (**DMatrix** &A, int p)
This function calculates the integer matrix power.
- **DMatrix & operator*** (double r, const **DMatrix** &A)
This function multiplies a real number by a matrix.
- **DMatrix & tra** (const **DMatrix** &A)
This function returns the transpose of a given matrix.
- **DMatrix & inv** (const **DMatrix** &A)
This function returns the inverse of a given square matrix. If the argument is not a square matrix an error is thrown.
- **DMatrix & pinv** (const **DMatrix** &A)
This function returns the pseudo-inverse of a given rectangular matrix.
- **DMatrix & identity** (long n)

This function returns the identity matrix with a given number of rows and columns.

- [DMatrix & identity](#) (long [n](#), long [m](#))

This function returns a truncated identity matrix with specified numbers of rows and columns.

- [DMatrix & eye](#) (long [n](#))

This function returns the identity matrix with a given number of rows and columns.

- [DMatrix & eye](#) (long [n](#), long [m](#))

This function returns a truncated identity matrix with specified numbers of rows and columns.

- [DMatrix & zeros](#) (long [n](#), long [m](#))

This function returns a matrix full of zeros with specified numbers of rows and columns.

- [DMatrix & ones](#) (long [n](#), long [m](#))

This function returns a matrix full of ones with specified numbers of rows and columns.

- [DMatrix & expm](#) (const [DMatrix](#) &[A](#))

This function returns the exponential matrix of a given square matrix.

- [DMatrix & sin](#) (const [DMatrix](#) &[A](#))

This function returns a matrix with the sine of each element of the input matrix.

- [DMatrix & cos](#) (const [DMatrix](#) &[A](#))

This function returns a matrix with the cosine of each element of the input matrix.

- [DMatrix & tan](#) (const [DMatrix](#) &[A](#))

This function returns a matrix with the tangent of each element of the input matrix.

- [DMatrix & exp](#) (const [DMatrix](#) &[A](#))

This function returns a matrix with the natural exponential of each element of the input matrix.

- [DMatrix & sinh](#) (const [DMatrix](#) &[A](#))

This function returns a matrix with the hyperbolic sine of each element of the input matrix.

- [DMatrix & cosh](#) (const [DMatrix](#) &[A](#))

This function returns a matrix with the hyperbolic cosine of each element of the input matrix.

- [DMatrix & tanh](#) (const [DMatrix](#) &[A](#))

This function returns a matrix with the hyperbolic tangent of each element of the input matrix.

- [DMatrix](#) & [log](#) (const [DMatrix](#) &A)

This function returns a matrix with the natural logarithm of each element of the input matrix.

- [DMatrix](#) & [diag](#) (const [DMatrix](#) &A)

if A is a matrix this function extracts a column vector with the diagonal values of A. If A is a vector this function returns a matrix having the elements of A in the diagonal

- [DMatrix](#) & [TProduct](#) (const [DMatrix](#) &A, const [DMatrix](#) &B)

This function returns the product of the first matrix transposed times the second matrix. The number of rows of both matrices must be the same, otherwise an error is thrown.

- [DMatrix](#) & [ProductT](#) (const [DMatrix](#) &A, const [DMatrix](#) &B)

This function returns the product of the first matrix times the second matrix transposed. The number of columns of both matrices must be the same, otherwise an error is thrown.

- [DMatrix](#) & [TProductT](#) (const [DMatrix](#) &A, const [DMatrix](#) &B)

This function returns the product of the first matrix transposed times the second matrix transposed. The number of rows of the first matrix must be the same as the number of columns of the second matrix, otherwise an error is thrown.

- [DMatrix](#) & [Product](#) (const [DMatrix](#) &A, const [DMatrix](#) &B)

This function calculates the product of two matrices. The number of columns of the first matrix must be the same as the number of rows of the second matrix, otherwise an error is thrown.

- [DMatrix](#) & [LUSolve](#) (const [DMatrix](#) &A, const [DMatrix](#) &b)

Solves the system of equations $Ax = b$ using LU factorisation.

- [DMatrix](#) & [LUSolve](#) (const [DMatrix](#) &ALU, const [DMatrix](#) &b)

Solves the system of equations $Ax = b$ using LU factorisation using a previously found LU factors.

- [DMatrix](#) & [CholSolve](#) (const [DMatrix](#) &A, const [DMatrix](#) &b)

Solves the system of equations $Ax = b$ using Cholesky factorisation.

- [DMatrix](#) & [CholFSolve](#) (const [DMatrix](#) &Achol, const [DMatrix](#) &b)

Solves the system of equations $Ax = b$ using Cholesky factorisation. The function uses a previously found Cholesky factorisation.

- [DMatrix](#) & [Chol](#) (const [DMatrix](#) &A)

Returns the Cholesky factorisation of a matrix A, which must be a positive definite symmetric matrix.

- [DMatrix](#) & [CholeskyRoot](#) (const [DMatrix](#) &A)

Returns the Cholesky root R of a given matrix A , such that $A=R'R$. A must be a positive definite symmetric matrix.

- **DMatrix & QRSolve** (const **DMatrix** &A, const **DMatrix** &b)
Solves the system of equations $Ax = b$ using QR factorisation. The number of rows of matrix A must be greater or equal than the number of columns.
- **DMatrix & QRFSolve** (const **DMatrix** &A, const **DMatrix** &b)
Solves the system of equations $Ax = b$ using QR factorisation. The function uses a previously found QR factorisation.
- **DMatrix & QR** (const **DMatrix** &A)
Returns the QR factorisation of a matrix A . The number of rows of matrix A must be greater or equal than the number of columns.
- **DMatrix & LSMNSolve** (const **DMatrix** &A, const **DMatrix** &B)
This function solves overdetermined or underdetermined real linear systems $Ax = B$ using a QR or LQ factorization of A . It is assumed that matrix A has full rank. The function uses the LAPACK routine `dgels`.
- **DMatrix & LQ** (const **DMatrix** &A, **DMatrix** *Q)
Returns the LQ factorisation of a matrix A . The function uses the LAPACK routine `dgelqf_()`.
- **DMatrix & LU** (const **DMatrix** &A)
Returns the LU factorisation of a matrix A .
- **DMatrix & SVD** (const **DMatrix** &A, **DMatrix** *U=NULL, **DMatrix** *V=NULL)
Returns the singular value decomposition of a matrix $A = U' \text{diag}(s)V$, where vector s contains the singular values of A . The function uses the LAPACK routine `dgesvd_()`.
- **DMatrix & orth** (const **DMatrix** &A)
This function returns Q , the orthonormal basis for the range of a matrix A , such that $QQ' = I$. The number of columns of Q is the rank of A .
- **DMatrix & null** (const **DMatrix** &A)
This function returns Z , the orthonormal basis for the null space of a matrix A , such that $ZZ' = I$ and $AZ = 0$. The number of columns of Z is the nullity of A .
- **DMatrix & SVDSolve** (const **DMatrix** &A, const **DMatrix** &B)
This function uses the LAPACK routine `dgels_()` to compute the minimum norm solution to a real linear least squares problem: Minimize $\|B - Ax\|_2$ using the singular value decomposition (SVD) of A . A is a rectangular matrix which may be rank-deficient.
- **DMatrix & schur** (const **DMatrix** &A, **DMatrix** *U=NULL)

This function computes and returns the Schur decomposition of a matrix A , such that $A = Q^H U Q$, where U is an upper triangular matrix and Q is a unitary matrix. This function uses the LAPACK routine `dgees_()`

- **DMatrix & eig** (const **DMatrix** &A, **DMatrix** *V=NULL)

This function computes the eigenvalues and (optionally) the eigenvectors of a matrix A . This function uses the LAPACK routines `dsyev_()` and `dgeev_()`.

- double **enorm** (const **DMatrix** &A)

This function computes and return the Euclidean norm of a matrix A , which is the square root of the sum of its squared elements.

- double **norm** (const **DMatrix** &A)

This function computes 2-norm of matrix A , which is computed as the maximum singular value of A .

- double **InfNorm** (const **DMatrix** &A)

This function computes infinity norm of matrix A , which is computed as the maximum absolute value row sum.

- double **Fnorm** (const **DMatrix** &A)

This function computes Frobenius norm of matrix A .

- **DMatrix & Abs** (const **DMatrix** &A)

This function computes and returns the element-wise absolute value of matrix A .

- double **Max** (const **DMatrix** &A, int *rindx=NULL, int *cindx=NULL)

This function finds and returns the element of matrix A with maximum value. It also returns the indices of such element. If more than one element has the same maximum value, the indices of the first element found when searching column by column is returned.

- double **MaxAbs** (const **DMatrix** &A, int *rindx=NULL, int *cindx=NULL)

This function finds and returns the element of matrix A with maximum absolute value. It also returns the indices of such element. If more than one element has the same maximum absolute value, the indices of the first element found when searching column by column is returned.

- double **Min** (const **DMatrix** &A, int *rindx=NULL, int *cindx=NULL)

This function finds and returns the element of matrix A with minimum value. It also returns the indices of such element. If more than one element has the same minimum value, the indices of the first element found when searching column by column is returned.

- double **MinAbs** (const **DMatrix** &A, int *rindx=NULL, int *cindx=NULL)

This function finds and returns the element of matrix A with minimum absolute value. It also returns the indices of such element. If more than one element has the same minimum absolute value, the indices of the first element found when searching column by column is returned.

- void [sort](#) (DMatrix &x, int indx[]=NULL)
This function sorts the input vector x in ascending order. Optionally, it also returns an integer array of sorted indices. If the input object is not a vector, then an error is thrown.
- void [sort](#) (DMatrix &x, DMatrix &indx)
This function sorts the input vector x in ascending order. It also returns a DMatrix object with the sorted indices. If the input object is not a vector, then an error is thrown.
- double [dotProduct](#) (const DMatrix &x, const DMatrix &y)
This function computes the dot product of two vectors. If any of the input arguments does not contain a vector, or if the vector lengths are not equal, an error is thrown.
- double [dot](#) (const DMatrix &x, const DMatrix &y)
This function computes the dot product of two vectors. If any of the input arguments does not contain a vector, or if the vector lengths are not equal, an error is thrown.
- DMatrix & [crossProduct](#) (const DMatrix &x, const DMatrix &y)
This function computes the cross product of two vectors. If any of the input arguments does not contain a vector, or if the length of any of the vectors is not 3, an error is thrown.
- DMatrix & [cross](#) (const DMatrix &x, const DMatrix &y)
This function computes the cross product of two vectors. If any of the input arguments does not contain a vector, or if the length of any of the vectors is not 3, an error is thrown.
- int [isSymmetric](#) (const DMatrix &A)
This function checks if the input matrix is symmetric. If the input matrix is not square, an error is thrown.
- double [cond](#) (const DMatrix &A)
This function calculates the 2-norm condition number of a matrix, which is the ratio of the maximum singular value to the minimum singular value of the matrix. A large condition number indicates a nearly singular matrix. If the input matrix is not square, an error is thrown.
- double [rcond](#) (const DMatrix &A)
This function estimates the 1-norm reciprocal condition number of a matrix. The function uses the LAPACK function dgecon. If A is well conditioned, then rcond(A) is near 1. If A is badly conditioned, then rcond(A) is close to the machine numerical precision (very small). If the input matrix is not square, an error is thrown.
- int [rank](#) (const DMatrix &A)
This function returns an estimate of the rank of a matrix, which is the number of linearly independent rows or columns.

- **double det** (const **DMatrix** &A)

This function returns the determinant of a square matrix. If the input matrix is not square, an error is thrown.
- **double trace** (const **DMatrix** &A)

This function returns the trace of a square matrix. If the input matrix is not square, an error is thrown.
- **DMatrix & mean** (const **DMatrix** &A)

This function returns a row vector with the mean values of the columns of matrix A.
- **DMatrix & Std** (const **DMatrix** &A, int ntype=0)

This function returns a row vector with the standard deviation of each column of matrix A. If ntype is 0 (default) the result is normalised with (n-1), where n is the number of rows of A. Otherwise, the result is normalised with n.
- **DMatrix & cov** (const **DMatrix** &A, int ntype=0)

Computes the covariance matrix of a data matrix where the N rows correspond to samples and the M columns are variables. The result is returned as an M x M matrix. If ntype=0 (default) then the result is normalised with N-1. Otherwise, if ntype=1, the result is normalised with N.
- **DMatrix & cov** (**DMatrix** &X, **DMatrix** &Y, int ntype=0)

*Computes the covariance matrix of two vectors X and Y of dimension N. The result is returned as an 1 x 1 **DMatrix** object. If ntype=0 (default) then the result is normalised with N-1. Otherwise, if ntype=1, the result is normalised with N.*
- **DMatrix & var** (**DMatrix** &A, int ntype=0)

This function returns a row vector with the variance of each column of matrix A. If ntype is 0 (default) the result is normalised with (n-1), where n is the number of rows of A. Otherwise, the result is normalised with n.
- **DMatrix & sum** (const **DMatrix** &A)

This function returns a row vector with the sum of the elements of each column of matrix A.
- **DMatrix & prod** (const **DMatrix** &A)

This function returns a row vector with the product of the elements of each column of matrix A.
- **DMatrix & elemProduct** (const **DMatrix** &A, const **DMatrix** &B)

This function computes and returns the element-wise product of two matrices of the same dimensions. If the dimensions of the two input matrices are not the same, an error is thrown.
- **DMatrix & elemDivision** (const **DMatrix** &A, const **DMatrix** &B)

This function computes and returns the element-wise division of two matrices of the same dimensions. If the dimensions of the two input matrices are not the same, an error is thrown. The dimensions of the returned object are the same as the dimensions of the factors.

- [DMatrix](#) & [kronProduct](#) (const [DMatrix](#) &A, const [DMatrix](#) &B)

This function computes and returns the Kronecker product of two matrices. The row (column) dimension of the returned object is the product of the row (column) dimensions of both factors.

- [DMatrix](#) & [vec](#) (const [DMatrix](#) &A)

This function returns a column vector made by stacking the columns of a matrix one below the other from left to right.

- [DMatrix](#) & [MatrixSign](#) (const [DMatrix](#) &A)

This function returns a [DMatrix](#) object with the same dimensions as the input matrix such that each of its elements is 1 if the corresponding value of the input matrix is positive, -1 if the corresponding value of the input matrix is negative, and 0 if the corresponding value of the input matrix is 0.

- [DMatrix](#) & [find](#) (const [DMatrix](#) &A)

This function returns a column vector with the linear indices of the non-zero elements of the input matrix A. The linear index is 1 for element (1,1) of the input matrix A, and $\text{length}(A)$ for the (nrows,ncols) element of the input matrix A.

- [DMatrix](#) & [randu](#) (long n, long m)

This function returns an nxm matrix where each element is a uniform pseudo-random number in the range (0,1).

- [DMatrix](#) & [randn](#) (long n, long m)

This function returns an nxm matrix where each element is a Gaussian pseudo-random number in the range with zero mean and variance 1.

- [DMatrix](#) & [linspace](#) (double X1, double X2, long N)

This function returns a linearly spaced vector with N points between the values X1 and X2.

- void [error_message](#) (const char *input_text)

This function prints an error message and throws an exception to be handled by the [ErrorHandler](#) class.

- void [tic](#) (void)

This function, which is to be used in conjunction with function [toc\(\)](#), starts counting elapsed CPU time.

- double [toc](#) ()

This function, which is to be used in conjunction with function [tic\(\)](#), stops counting CPU time, and it prints and returns the elapsed time in seconds since the function [tic\(\)](#) was called.

- [DMatrix](#) & [Sqrt](#) (const [DMatrix](#) &A)

This function computes the square root of each element of the input matrix A, and it returns a [DMatrix](#) object with the same dimensions as the input matrix. If any element of the input matrix is negative, an error is thrown.

- [DMatrix](#) & [triu](#) (const [DMatrix](#) &A)

This function extracts and return the triangular upper part of the input matrix A. The returned object has the same dimensions as the input object.

- [DMatrix](#) & [reshape](#) ([DMatrix](#) &A, long N, long M)

*This function returns the N-by-M matrix whose elements are taken columnwise from the input matrix A. An error is thrown if A does not have N*M elements.*

- long [length](#) (const [DMatrix](#) &A)

This function returns the number of elements of a matrix A.

3.5.1 Detailed Description

[DMatrix](#) class. A C++ class for dense and real matrix and vector computations with interfaces to a number of LAPACK functions

3.5.2 Constructor & Destructor Documentation

3.5.2.1 [DMatrix::DMatrix](#) (long *Initn*, long *Initm*)

Constructor with dimensions. Creates a matrix with allocated storage given specified numbers of rows and columns.

Parameters

<i>Initn</i> ,:	number of rows
<i>Initm</i> ,:	number of columns

3.5.2.2 [DMatrix::DMatrix](#) (long *vDim*, double * *v*, long *Initn*, long *Initm*)

Constructor with dimensions using pre-allocated storage.

Parameters

<i>vDim</i> ,:	Allocated length of array v
<i>v</i> ,:	double array to be used as storage by the DMatrix object
<i>Initn</i> ,:	number of rows
<i>Initm</i> ,:	number of columns

3.5.2.3 DMatrix::DMatrix (long *Initn*)

Constructor with a single dimension, creates a column vector.

Parameters

<i>Initn</i> ,:	number of rows
-----------------	----------------

3.5.2.4 DMatrix::DMatrix (long *Initn*, long *Initm*, double *a11*, ...)

Constructor using a variable list of element values.

Parameters

<i>Initn</i> ,:	number of rows
<i>Initm</i> ,:	number of columns
<i>a11</i> ,:	first element of list doubles, values are entered column by column

3.5.2.5 DMatrix::DMatrix (const DMatrix & *A*)

Copy constructor. Creates a new [DMatrix](#) object with the same dimensions and element values as a given [DMatrix](#) object.

Parameters

<i>A</i> ,:	DMatrix object to be copied
-------------	---

3.5.3 Member Function Documentation**3.5.3.1 void DMatrix::Allocate (long *size*) [protected]**

Allocate memory to store matrix elements.

Parameters

<i>size</i>	number of double elements to allocate
-------------	---------------------------------------

Returns

void

See also

[DeAllocate\(\)](#)

3.5.3.2 static void DMatrix::AllocateAuxArr (void) [static]

Allocates the array of auxiliary (temporary) objects used by the class.

Allocates a [DMatrix](#) array of size N_TEMP_OBJECTS. Each element is allocated a storage of size D_TEMP_OBJECTS. These two macros are given default values but may be changed by the user at compilation time. The purpose of the array of temporary objects is to store the intermediate objects resulting from single lines of code that call various operators and functions returning [DMatrix](#) objects. A simple example is as follows. Consider the C++ statement $D = A * B + C$; where A, B, C and D are [DMatrix](#) objects. This statement involves two temporary objects: one to store the result of $A * B$, and another one to store the result of $(A * B) + C$;

Returns

void

3.5.3.3 void DMatrix::assign (long rows, long columns, double a11, ...)

Assigns values to the elements of an existing [DMatrix](#) object using a variable list of values. Resizes the matrix if necessary.

Parameters

<i>rows,:</i>	number of rows
<i>columns,:</i>	number of columns
<i>a11,:</i>	first element of the list of double arguments.

3.5.3.4 DMatrix& DMatrix::AssignmentToColonReference (const DMatrix & A) [protected]

Assigns a matrix to the values pointed to by a colon reference matrix.

Parameters

A	DMatrix object
---	--------------------------------

Returns

reference to [DMatrix](#) object

3.5.3.5 DMatrix& DMatrix::AssignmentToColonReference (double arg) [protected]

Assigns a double value to each value pointed to by a colon reference matrix.

Parameters

<i>arg</i>	is a double value to be assigned.
------------	-----------------------------------

Returns

reference to [DMatrix](#) object

3.5.3.6 void DMatrix::colMult (long *c*, double *x*)

Multiplies the elements of a specified column of a matrix by a constant scalar value.

Parameters

<i>c</i> ,:	index to the column that is to be changed
<i>x</i> ,:	scalar value

Returns

void

3.5.3.7 DMatrix& DMatrix::Column (long *icol*) const

Returns a [DMatrix](#) object containing a specified column of the calling object.

Parameters

<i>icol</i> ,:	column index (starting from 1)
----------------	--------------------------------

Returns

[DMatrix](#) object with the specified column

3.5.3.8 DMatrix& DMatrix::compMat (const DMatrix & *m2*, char *op*) const
[protected]

Elementwise comparison of matrix elements.

Parameters

<i>m2</i>	DMatrix object to be compared with the calling object.
<i>op</i>	(char) indicates type of operator. Use 1 for >, 2 for >=, 3 for <, 4 for <=, 5 for ==, 6 for != comparisons

Returns

A reference to a [DMatrix](#) object such that its elements are 1 if the comparison is true, 0 otherwise.

3.5.3.9 void DMatrix::DeAllocate () [protected]

De-allocate memory previously allocated with [DMatrix::Allocate\(\)](#).

Returns

void

See also

[Allocate\(\)](#)

3.5.3.10 static void DMatrix::DeAllocateAuxArr (void) [static]

De-allocates the array of auxiliary (temporary) objects previously allocated by [AllocateAuxArr\(\)](#)

Returns

void

3.5.3.11 void DMatrix::diag (const DMatrix & dd)

Assigns values to the diagonal elements of a matrix, while all off-diagonal elements are set to zero.

The dimensions of dd should be consistent with the dimensions of the calling object

Parameters

<i>dd,:</i>	reference to constant DMatrix object which should contain a vector with the desired diagonal elements
-------------	---

Returns

void

3.5.3.12 double DMatrix::elem (long i, long j) const [inline]

Returns the value of a specified element of a matrix.

Parameters

<i>i,:</i>	row index (starting from 1)
<i>j,:</i>	column index (starting from 1)

Returns

double value

3.5.3.13 double& DMatrix::elem (long i, long j) [inline]

Returns a reference to the specified element of a matrix.

Parameters

$i,:$	row index (starting from 1)
$j,:$	column index (starting from 1)

Returns

double reference

3.5.3.14 double DMatrix::element (long i , long j)

Returns the value of a specified element of a matrix.

Parameters

$i,:$	row index (starting from 1)
$j,:$	column index (starting from 1)

Returns

double value

3.5.3.15 void DMatrix::FillWithZeros (void)

Assigns a zero value to each element of a matrix.

Returns

void

3.5.3.16 DMatrix& DMatrix::find (DMatrix & I , DMatrix & J) const

Finds non-zero values of a matrix.

Parameters

$I,:$	DMatrix object with the row index of each non-zero element
$J,:$	DMatrix object with the column index of each non-zero element

Returns

[DMatrix](#) object with the same dimensions as the calling object, and with elements which are 0 if the corresponding element of the calling object is 0, 1 otherwise.

3.5.3.17 DMatrix& DMatrix::find (int * I , int * J) const

Finds non-zero values of a matrix.

Parameters

<i>I</i> ,:	C++ double array with the row index of each non-zero element
<i>J</i> ,:	C++ double array with the column index of each non-zero element

Returns

[DMatrix](#) object with the same dimensions as the calling object, and with elements which are 0 if the corresponding element of the calling object is 0, 1 otherwise.

3.5.3.18 void DMatrix::Fprint (FILE * *filex*)

Prints the elements of a matrix elements matrix to a file.

Parameters

<i>filex</i>	is a pointer to a file already opened using "fopen()".
--------------	--

Returns

void

3.5.3.19 double* DMatrix::geta () [inline]

Gets a pointer to the array where the elements of the matrix are stored.

Returns

double pointer

3.5.3.20 int DMatrix::getatype () [inline]

Gets the type of element storage of a matrix.

Returns

int value: 0 if the array is allocated, 1 if the storage is done using a previously declared array of doubles.

3.5.3.21 static DMatrix DMatrix::GetAuxPr (void) [inline, static]**

Gets a pointer to the array of auxiliary objects.

Returns

DMatrix** pointer

3.5.3.22 `double* DMatrix::GetConstPr () const [inline]`

Gets a pointer to the array where the elements of the matrix are stored (for const objects)

Returns

double pointer

3.5.3.23 `long DMatrix::getm () [inline]`

Gets the number of columns from the calling object.

Returns

long value with the number of columns

3.5.3.24 `long DMatrix::getm () const [inline]`

Gets the number of columns from the calling object.

Returns

long value with the number of columns

3.5.3.25 `int DMatrix::GetMType () [inline, protected]`

Gets the type of matrix.

Returns

int value

3.5.3.26 `long DMatrix::getn () [inline]`

Gets the number of rows from the calling object.

Returns

long value with the number of rows.

3.5.3.27 `long DMatrix::getn () const [inline]`

Gets the number of rows from the calling object.

Returns

long value with the number of rows.

3.5.3.28 `long DMatrix::GetNoCols () [inline]`

Gets the number of columns from the calling object.

Returns

long value with the number of columns

3.5.3.29 `long DMatrix::GetNoCols () const [inline]`

Gets the number of columns from the calling object.

Returns

long value with the number of columns

3.5.3.30 `long DMatrix::GetNoRows () const [inline]`

Gets the number of rows from the calling object.

Returns

long value with the number of rows.

3.5.3.31 `long DMatrix::GetNoRows () [inline]`

Gets the number of rows from the calling object.

Returns

long value with the number of rows.

3.5.3.32 `double* DMatrix::GetPr () [inline]`

Gets the pointer to the array where the elements of the matrix are stored.

Returns

double pointer

3.5.3.33 `DMatrix* DMatrix::GetReferencedDMatrixPointer () [inline,
protected]`

Gets the value of the referenced matrix pointer.

Returns

pointer to [DMatrix](#) object

3.5.3.34 void DMatrix::input_matrix ()

Allows the user to enter the elements of a matrix using command line prompts.

Returns

void

3.5.3.35 int DMatrix::isEmpty () [inline]

Check if a matrix object is empty, if other words this method checks if the calling object has zero elements.

Returns

int value: 0 if calling object is not empty, 1 otherwise

3.5.3.36 static int DMatrix::isThereError (void) [inline, static]

Checks if the error flag has been raised. If so, a 1 is returned, 0 otherwise.

Returns

int value

3.5.3.37 int DMatrix::isVector () const [inline]

Check if a matrix object contains a row or column vector.

Returns

int value: 1 if calling object is a vector, 0 otherwise

3.5.3.38 void DMatrix::Load (const char * *FileName*)

Reads the elements of a matrix from a file.

The calling object should have the appropriate number of rows and columns.

Parameters

<i>FileName</i>	is a string with the file name where the matrix elements are stored.
-----------------	--

Returns

void

3.5.3.39 void DMatrix::MemCpyArray (double * *aptr*)

Copies values to the elements of a [DMatrix](#) object from an existing array. The number of elements of the [DMatrix](#) object is assumed to be the number of values to be copied.

Parameters

<i>aptr</i> ,:	pointer to the start of the array of doubles to be copied.
----------------	--

3.5.3.40 DMatrix& DMatrix::mpow (int *p*)

Computes and returns the integer power of a matrix.

Parameters

<i>p</i> ,:	integer value, starting from 0
-------------	--------------------------------

Returns

[DMatrix](#) object with the result of the calculation

3.5.3.41 DMatrix& DMatrix::operator!= (double *val*) const

Checks if each element of the [DMatrix](#) object on the left hand side is different from the right hand side value. The result is a [DMatrix](#) object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise.

Parameters

<i>val</i> ,:	right hand side value
---------------	-----------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.42 DMatrix& DMatrix::operator!= (const DMatrix & *val*) const

Elementwise matrix comparison. Checks if each element of the matrix on the left hand side is different from the corresponding element of the right hand side matrix. The result is a [DMatrix](#) object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise. The dimensions of the two matrices involved must be the same, otherwise an error is thrown.

Parameters

<i>val</i> ,:	<i>val</i> : right hand side object
---------------	-------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.43 DMatrix& DMatrix::operator%(const DMatrix & rval) const

Computes the left division of a matrix (left hand side of the operator) by another matrix (right hand side value). This is conceptually equivalent to multiplying the inverse of the left object by the right hand side object but it is computed in a more efficient way. The dimensions of the matrices must be consistent, otherwise an error is returned. The left hand side object must be a square matrix.

Parameters

<i>rval</i> ,:	DMatrix object at the right hand side of the operator.
----------------	--

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.44 DMatrix& DMatrix::operator& (const DMatrix B) const

Elementwise product operator. Returns a [DMatrix](#) object with the same dimensions of the calling objects and each of its elements is computed as the product of the corresponding elements of the calling object and the right hand side object. The dimensions of the calling objects must be the same, otherwise an error is thrown. Care must be taken when using this operator as the associations do not work in the same way as with the `*` operator. It is highly recommended to use parenthesis every time this operator is used. For example use it as follows: `(A&B)`.

Parameters

<i>B</i> ,:	DMatrix object at the right hand side of the operator
-------------	---

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.45 DMatrix& DMatrix::operator&& (const DMatrix & B) const

Stacks the right hand side matrix below the left hand side matrix. The dimensions number of columns of the matrices involved must be the same, otherwise an error is thrown. The number of rows of the resulting matrix is the addition of the number of rows of both matrices involved.

Parameters

<i>B</i> ,:	DMatrix object at the right hand side of the operator.
-------------	--

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.46 DMatrix& DMatrix::operator() (const DMatrix & RowIndx, long col)

Sub-vector extraction and referencing using an array of row indices for a given column.

Parameters

<i>RowIndx</i>	is a DMatrix array that contains row index values usually generated using the colon() function.
<i>col</i>	is a column index

Returns

Reference to a [DMatrix](#) mtype 1 object that maps to the referenced elements of the calling object

3.5.3.47 double DMatrix::operator() (const char * end, long col) const

Matrix indexing. Returns the value of the matrix element located at the position indicated by the column index and the last row. Indices start from 1. An error is thrown in case of a range violation.

Parameters

<i>end,:</i>	Character string containing the word "end".
<i>col,:</i>	Column index starting from 1.

Returns

double value of the indexed matrix element.

3.5.3.48 double DMatrix::operator() (const char * end) const

Access to last linear element. Returns the value of the last linear matrix element, assuming column major storage.

Parameters

<i>end,:</i>	Character string containing the word "end".
--------------	---

Returns

value of the last matrix element.

3.5.3.49 DMatrix& DMatrix::operator() (const char * *end*, const DMatrix & *ColIdx*)

Sub-vector extraction and referencing using an array of column indices for the last column of a matrix.

Parameters

<i>end</i>	is a character string containing the word "end".
<i>ColIdx</i>	is a DMatrix array that contains column index values usually generated using the colon() function.

Returns

Reference to a [DMatrix](#) mtype 1 object that maps to the referenced elements of the calling object

3.5.3.50 double& DMatrix::operator() (const char * *end*)

Access to last linear element. Returns a reference to the last linear matrix element, assuming column major storage.

Parameters

<i>end</i> ,:	Character string containing the word "end".
---------------	---

Returns

reference to indexed matrix element.

3.5.3.51 DMatrix& DMatrix::operator() (const DMatrix & *RowIdx*, const DMatrix & *ColIdx*)

Submatrix extraction and referencing using arrays of indices.

Parameters

<i>RowIdx</i>	is a DMatrix array that contains row index values usually generated using the colon() function.
<i>ColIdx</i>	is a DMatrix array that contains column index values usually generated using the colon() function.

Returns

Reference to a [DMatrix](#) mtype 1 object that maps to the referenced elements of the calling object.

3.5.3.52 DMatrix& DMatrix::operator() (const DMatrix & RowIdx)

Linear sub-vector extraction and referencing using an array of indices assuming column-major storage.

Parameters

<i>RowIdx</i>	is a DMatrix array that contains row index values usually generated using the colon() function.
---------------	---

Returns

Reference to a [DMatrix](#) mtype 1 object that maps to the referenced elements of the calling object

3.5.3.53 double& DMatrix::operator() (long row, long col)

Matrix indexing. Returns a reference to the matrix element located at the position indicated by the row and column indices. Indices start from 1.

Parameters

<i>row,:</i>	Row index starting from 1.
<i>col,:</i>	Column index starting from 1.

Returns

Reference to the indexed matrix element.

3.5.3.54 double& DMatrix::operator() (long row, const char * end)

Matrix indexing. Returns a reference to the matrix element located at the position indicated by the row index and the last column. Indices start from 1. An error is thrown in case of zero or negative indices. The matrix is resized if necessary.

Parameters

<i>row,:</i>	Row index starting from 1.
<i>end,:</i>	Character string containing the word "end".

Returns

Reference to the indexed matrix element.

3.5.3.55 double& DMatrix::operator() (const char * end, long col)

Matrix indexing. Returns a reference to the matrix element located at the position indicated by the column index and the last row. Indices start from 1. An error is

thrown in case of a range violation.

Parameters

<i>end</i> ,:	Character string containing the word "end".
<i>col</i> ,:	Column index starting from 1.

Returns

Reference to the indexed matrix element.

3.5.3.56 DMatrix& DMatrix::operator() (long row, const DMatrix & ColIndx)

Sub-vector extraction and referencing using an array of column indices for a given row.

Parameters

<i>row</i>	is a row index.
<i>ColIndx</i>	is a DMatrix array that contains column index values usually generated using the colon() function.

Returns

Reference to a [DMatrix](#) mtype 1 object that maps to the referenced elements of the calling object

3.5.3.57 double DMatrix::operator() (long row, const char * end) const

Matrix indexing. Returns the value of the matrix element located at the position indicated by the row index and the last column. Indices start from 1. An error is thrown in case of a range violation.

Parameters

<i>row</i> ,:	Row index starting from 1.
<i>end</i> ,:	Character string containing the word "end".

Returns

double value of the indexed matrix element.

3.5.3.58 double& DMatrix::operator() (long index)

Single index matrix indexing. Returns a reference to the matrix element located at the linear position indicated by the index, assuming column major storage. The indexes start from 1. The matrix is resized if necessary. An error is thrown in case of zero or negative indices.

Parameters

<i>index,:</i>	index value
----------------	-------------

Returns

reference to indexed matrix element.

3.5.3.59 DMatrix& DMatrix::operator() (const DMatrix & RowIdx, const char * end)

Sub-vector extraction and referencing using an array of row indices for the last column of a matrix.

Parameters

<i>RowIdx</i>	is a DMatrix array that contains row index values usually generated using the colon() function.
<i>end</i>	is a character string containing the word "end".

Returns

Reference to a [DMatrix](#) mtype 1 object that maps to the referenced elements of the calling object

3.5.3.60 double DMatrix::operator() (long k) const

Single index matrix indexing. Returns the value of the matrix element located at the linear position indicated by the index, assuming column major storage. The index starts from 1. An error is thrown in case of range error.

Parameters

<i>k,:</i>	index value
------------	-------------

Returns

Value of indexed matrix element.

3.5.3.61 double DMatrix::operator() (long row, long col) const

Matrix indexing. Returns the value of the matrix element located at the position indicated by the row and column indices. Indices start from 1. An error is thrown in case of a range violation.

Parameters

<i>row,:</i>	Row index starting from 1.
<i>col,:</i>	Column index starting from 1.

Returns

double value of the indexed element.

3.5.3.62 DMatrix& DMatrix::operator* (const DMatrix & rval) const

Matrix product operator. Returns the result of the matrix product of the calling object (left hand side of the operator) and the right hand side object. The inner dimensions of the objects being multiplied should be consistent, otherwise an error will be thrown.

Parameters

<i>rval</i> ,:	matrix located at the right hand side of the operator.
----------------	--

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.63 DMatrix& DMatrix::operator* (double Arg) const

Computes the product of a matrix (left hand side of the operator) times a real scalar (right hand side value) and replaces the left hand side object with the result of the operation.

Parameters

<i>Arg</i> ,:	double value that will multiply each element of the matrix.
---------------	---

Returns

Reference the calling [DMatrix](#) object

3.5.3.64 DMatrix& DMatrix::operator*= (double Arg)

Computes the product of a matrix (left hand side of the operator) times a real scalar (right hand side value), and modifies the calling object to store the result of the operation.

Parameters

<i>Arg</i> ,:	double value that will multiply each element of the matrix.
---------------	---

Returns

Reference to the calling [DMatrix](#) object

3.5.3.65 DMatrix& DMatrix::operator*= (const DMatrix & rval)

Matrix product operator with substitution. Computes the matrix product of the calling object (left hand side of the operator) and the right hand side object. The inner dimensions of the objects being multiplied should be consistent, otherwise an error will be thrown. The calling object is modified to store the results of the operation.

Parameters

<i>rval,:</i>	matrix located at the right hand side of the operator.
---------------	--

Returns

Reference to the calling [DMatrix](#) object

3.5.3.66 DMatrix& DMatrix::operator+ (const DMatrix & rval) const

Matrix addition operator. The sizes of the matrices being added should be the same, otherwise an error is thrown.

Parameters

<i>rval,:</i>	matrix located at the right hand side of the operator.
---------------	--

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.67 DMatrix& DMatrix::operator+ (double x) const

Adds a scalar real value to each element of the matrix. .

Parameters

<i>x,:</i>	double value to be added
------------	--------------------------

3.5.3.68 DMatrix& DMatrix::operator+= (const DMatrix & rval)

Matrix addition and substitution operator. The sizes of the matrices being added should be the same, otherwise an error is thrown. The left hand side object elements are replaced with the result of the operation.

Parameters

<i>rval,:</i>	matrix located right hand side of the operator.
---------------	---

Returns

Reference to the calling object

3.5.3.69 DMatrix& DMatrix::operator- (double *x*) const

Subtracts a scalar real value from each element of the matrix. .

Parameters

<i>x</i> ,:	double value to be subtracted
-------------	-------------------------------

3.5.3.70 DMatrix& DMatrix::operator- (const DMatrix & *rval*) const

Matrix subtraction operator. The sizes of the matrices being subtracted should be the same, otherwise an error is thrown.

Parameters

<i>rval</i> ,:	matrix located at the right hand side of the operator.
----------------	--

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.71 DMatrix& DMatrix::operator-= (const DMatrix & *rval*)

Matrix subtraction and substitution operator. The sizes of the matrices being subtracted should be the same, otherwise an error is thrown. The left hand side object elements are replaced with the result of the operation.

Parameters

<i>rval</i> ,:	matrix located right hand side of the operator.
----------------	---

Returns

Reference to the calling object

3.5.3.72 DMatrix& DMatrix::operator/ (const DMatrix & *rval*) const

Computes the right division of a matrix (left hand side of the operator) by another matrix (right hand side value). This is conceptually equivalent to multiplying the left object by the inverse of the right hand side object but it is computed in a more efficient way. The dimensions of the matrices must be consistent, otherwise an error is returned. The right hand side object must be a square matrix.

Parameters

<i>rval</i> ,:	DMatrix object at the right hand side of the operator.
----------------	--

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.73 DMatrix& DMatrix::operator/ (double Arg) const

Computes the division of a matrix (left hand side of the operator) by a real scalar (right hand side value).

Parameters

<i>Arg,:</i>	double value that will divide each element of the matrix.
--------------	---

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.74 DMatrix& DMatrix::operator/= (double Arg)

Computes the division of a matrix (left hand side of the operator) by a real scalar (right hand side value) and modifies the left hand side object with the result of the operation.

Parameters

<i>Arg,:</i>	double value that will divide each element of the matrix.
--------------	---

Returns

Reference to the calling object

3.5.3.75 DMatrix& DMatrix::operator< (double val) const

Checks if each element of the [DMatrix](#) object on the left hand side is lower than the right hand side value. The result is a [DMatrix](#) object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise.

Parameters

<i>val,:</i>	right hand side value
--------------	-----------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.76 DMatrix& DMatrix::operator< (const DMatrix & val) const

Elementwise matrix comparison. Checks if each element of the matrix on the left hand side is lower than the corresponding element of the right hand side matrix. The

result is a [DMatrix](#) object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise. The dimensions of the two matrices involved must be the same, otherwise an error is thrown.

Parameters

<i>val</i> ,:	val: right hand side object
---------------	-----------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.77 DMatrix& DMatrix::operator<= (const DMatrix & *val*) const

Elementwise matrix comparison. Checks if each element of the matrix on the left hand side is lower or equal than the corresponding element of the right hand side matrix. The result is a [DMatrix](#) object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise. The dimensions of the two matrices involved must be the same, otherwise an error is thrown.

Parameters

<i>val</i> ,:	val: right hand side object
---------------	-----------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.78 DMatrix& DMatrix::operator<= (double *val*) const

Checks if each element of the [DMatrix](#) object on the left hand side is lower or equal than the right hand side value. The result is a [DMatrix](#) object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise.

Parameters

<i>val</i> ,:	right hand side value
---------------	-----------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.79 DMatrix& DMatrix::operator= (const DMatrix & *rval*)

Matrix assignment. The size of the left hand side object is modified if necessary, and the values of all real elements of the right hand side object are copied to the left hand side object.

Parameters

<i>rval</i> ,:	DMatrix object at the right hand side of the operator
----------------	---

Returns

Reference to the calling object

3.5.3.80 DMatrix& DMatrix::operator= (double *val*)

Matrix assignment to a scalar. The size of the left hand side object is modified to one row by one column if necessary, and the value of the right hand side argument is copied to the single element of the matrix. If the calling object is a "colon reference" matrix, then the right hand side value is copied to each element of the referenced array elements.

Parameters

<i>val</i> ,:	double value at the right hand side of the operator
---------------	---

Returns

Reference to the calling object

3.5.3.81 DMatrix& DMatrix::operator= (const char * *str*)

Matrix assignment to a constant matrix defined as a character string using the bracket notation used in Matlab and Octave. The size of the left hand side object is modified if necessary. For example, the identity matrix of size two by two would be entered as "[1.0 0.0;0.0 1.0]".

Parameters

<i>str</i> ,:	Character string containing the constant matrix defined using Matlab/Octave bracket notation.
---------------	---

Returns

Reference to the calling object

3.5.3.82 DMatrix& DMatrix::operator== (const DMatrix & *val*) const

Elementwise matrix comparison. Checks if each element of the matrix on the left hand side is equal to the corresponding element of the right hand side matrix. The result is a [DMatrix](#) object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise. The dimensions of the two matrices involved must be the same, otherwise an error is thrown.

Parameters

<i>val</i> ,:	val: right hand side object
---------------	-----------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.83 DMatrix& DMatrix::operator==(double *val*) const

Checks if each element of the [DMatrix](#) object on the left hand side is equal to the right hand side value. The result is a [DMatrix](#) object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise.

Parameters

<i>val</i> ,:	right hand side value
---------------	-----------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.84 DMatrix& DMatrix::operator> (const DMatrix & *val*) const

Elementwise matrix comparison. Checks if each element of the matrix on the left hand side is greater than the corresponding element of the right hand side matrix. The result is a [DMatrix](#) object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise. The dimensions of the two matrices involved must be the same, otherwise an error is thrown.

Parameters

<i>val</i> ,:	val: right hand side object
---------------	-----------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.85 DMatrix& DMatrix::operator> (double *val*) const

Checks if each element of the [DMatrix](#) object on the left hand side is greater than the right hand side value. The result is a [DMatrix](#) object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise.

Parameters

<i>val</i> ,:	right hand side value
---------------	-----------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.86 DMatrix& DMatrix::operator>= (double val) const

Checks if each element of the [DMatrix](#) object on the left hand side is greater or equal than the right hand side value. The result is a [DMatrix](#) object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise.

Parameters

<i>val</i> ,:	right hand side value
---------------	-----------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.87 DMatrix& DMatrix::operator>= (const DMatrix & val) const

Elementwise matrix comparison. Checks if each element of the matrix on the left hand side is greater or equal than the corresponding element of the right hand side matrix. The result is a [DMatrix](#) object where each element has the value of 1 if the corresponding comparison was true, 0 otherwise. The dimensions of the two matrices involved must be the same, otherwise an error is thrown.

Parameters

<i>val</i> ,:	val: right hand side object
---------------	-----------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.88 DMatrix& DMatrix::operator^ (double x)

Elementwise power operator. Returns a [DMatrix](#) object with the same dimensions of the calling object and each of its elements is computed as the corresponding element of the calling object to the power of the right hand side argument. Care must be taken when using this operator as the associations do not work in the same way as with the * operator. It is highly recommended to use parenthesis every time this operator is used. For example use it as follows: (A^x).

Parameters

<i>x</i> ,:	double argument at the right hand side of the operator.
-------------	---

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.89 DMatrix& DMatrix::operator| (const DMatrix *B*) const

Elementwise division operator. Returns a [DMatrix](#) object with the same dimensions of the calling objects and each of its elements is computed as the of the corresponding element of the calling object by the corresponding element of the right hand side object. The dimensions of the calling objects must be the same, otherwise an error is thrown. Care must be taken when using this operator as the associations do not work in the same way as with the / operator. It is highly recommended to use parenthesis every time this operator is used. For example use it as follows: (*A*|*B*).

Parameters

<i>B</i> ,:	DMatrix object at the right hand side of the operator
-------------	---

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.90 DMatrix& DMatrix::operator|| (const DMatrix & *B*) const

Concatenates two matrices side by side. The dimensions number of rows of the matrices involved must be the same, otherwise an error is thrown. The number of columns of the resulting matrix is the addition of the number of columns of both matrices involved.

Parameters

<i>B</i> ,:	DMatrix object at the right hand side of the operator.
-------------	--

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.3.91 void DMatrix::Print (const char * *text*) const

Prints the elements of a [DMatrix](#) object.

Parameters

<i>text</i>	is a string that serves as a prompt for the matrix being entered.
-------------	---

Returns

void

3.5.3.92 void DMatrix::PrintInfo (const char * *text*) const

Prints information about a [DMatrix](#) object.

Parameters

<i>text</i>	is a string that serves as a label for the matrix being printed.
-------------	--

Returns

void

3.5.3.93 static double DMatrix::random_gaussian (void) [static]

Returns a pseudo-random Gaussian distributed number with zero mean and unit variance.

Returns

double Gaussian pseudo-random value

3.5.3.94 static double DMatrix::random_uniform (void) [static]

Returns a pseudo-random uniformly distributed number in the range [0,1].

Returns

double pseudo-random value in the range [0,1]

3.5.3.95 void DMatrix::Read (FILE * *filex*)

Reads the elements of a matrix from a file.

The file in question should contain the elements of the matrix row by row. The elements should be separated by spaces and each row should be separated by a new line. The calling object should have the appropriate number of rows and columns.

Parameters

<i>filex</i>	pointer to a file already opened using "fopen()".
--------------	---

Returns

void

3.5.3.96 void DMatrix::Resize (long *nnrow*, long *nncol*)

Changes the number of rows and columns of an existing matrix. Allocates new memory if necessary. If the calling object uses preallocated memory and the requested size would exceed that memory, an error is thrown.

Parameters

<i>nnrow</i> ,:	new number of rows
<i>nncol</i> ,:	new number of columns

3.5.3.97 DMatrix& DMatrix::Row (long *irow*) const

Returns a [DMatrix](#) object containing a specified row of the calling object.

Parameters

<i>irow</i> ,:	row index (starting from 1)
----------------	-----------------------------

Returns

[DMatrix](#) object with the specified row

3.5.3.98 void DMatrix::rowMult (long *r*, double *x*)

Multiplies the elements of a specified row of a matrix by a constant scalar value.

Parameters

<i>r</i> ,:	index to the row that is to be changed
<i>x</i> ,:	scalar value

Returns

void

3.5.3.99 void DMatrix::Save (const char * *FileName*)

Saves the elements of a matrix to a file.

Parameters

<i>FileName</i>	is a string with the desired file name
-----------------	--

Returns

void

3.5.3.100 `void DMatrix::SetColIndexPointer (const DMatrix * arg)` [inline, protected]

Sets the column index pointer.

Parameters

<i>arg</i>	Pointer to DMatrix object
------------	---

Returns

void

3.5.3.101 `void DMatrix::SetColumn (const DMatrix & Col, int icol)`

Assigns values to a column of a matrix, while other columns are left untouched.

Parameters

<i>Col,:</i>	reference to constant DMatrix object which should contain a vector with the desired column values
<i>icol,:</i>	index to the column that is to be changed

Returns

void

3.5.3.102 `void DMatrix::SetMType (int arg)` [protected]

Sets the type of matrix.

Parameters

<i>arg</i>	(int) should be 0 for a normal matrix, or 1 for a colon reference matrix
------------	--

Returns

void

3.5.3.103 `static void DMatrix::SetPrintLevel (int plevel)` [static]

Sets the print level.

Parameters

<i>plevel</i>	desired print level
---------------	---------------------

Returns

void

3.5.3.104 void DMatrix::SetReferencedDMatrixPointer (DMatrix * *arg*) [inline, protected]

Sets the value of the referenced matrix pointer.

Parameters

<i>arg</i>	Pointer to DMatrix object
------------	---

Returns

void

3.5.3.105 void DMatrix::SetRow (const DMatrix & *Row*, int *irow*)

Assigns values to a row of a matrix, while other columns are left untouched.

Parameters

<i>Row</i> ,:	reference to constant DMatrix object which should contain a vector with the desired row values
<i>irow</i> ,:	index to the row that is to be changed

Returns

void

3.5.3.106 void DMatrix::SetRowIndexPointer (const DMatrix * *arg*) [inline, protected]

Sets the row index pointer.

Parameters

<i>arg</i>	Pointer to DMatrix object
------------	---

Returns

void

3.5.3.107 void DMatrix::SetSubMatrix (long *row*, long *col*, const DMatrix & *A*)

Assigns the elements of a matrix object to a section of the calling object.

Parameters

<i>row,:</i>	start of row range
<i>col,:</i>	start of column range
<i>A,:</i>	DMatrix object whose element values are to be copied into the calling object

Returns

void

3.5.3.108 DMatrix& DMatrix::sub_matrix (long *r1*, long *r2*, long *c1*, long *c2*) const

Extracts a specified sub-matrix from a matrix.

Parameters

<i>r1,:</i>	start of row range
<i>r2,:</i>	end of row range
<i>c1,:</i>	start of column range
<i>c2,:</i>	end of column range

Returns

[DMatrix](#) object with the specified sub-matrix

3.5.3.109 void DMatrix::SwapColumns (int *i*, int *j*)

Swaps two columns of a matrix.

Parameters

<i>i,:</i>	first column index
<i>j,:</i>	second column index.

Returns

void

3.5.3.110 void DMatrix::SwapRows (int *i*, int *j*)

Swaps two rows of a matrix.

Parameters

<i>i,:</i>	first row index
<i>j,:</i>	second row index.

Returns

void

3.5.3.111 void DMatrix::Transpose (void)

Transposes a matrix.

Returns

void

3.5.4 Friends And Related Function Documentation**3.5.4.1 DMatrix& Abs (const DMatrix & A) [friend]**

This function computes and returns the element-wise absolute value of matrix A.

Parameters

A	is a DMatrix object.
-----	--------------------------------------

Returns

a reference to a temporary [DMatrix](#) object with the result of the operation.

3.5.4.2 int any (const DMatrix & A) [friend]

This function returns a 1 if any element of [DMatrix](#) object that is passed as argument is non-zero, otherwise it returns a zero.

Parameters

A	is a DMatrix object.
-----	--------------------------------------

Returns

an integer value which is either 1 or 0.

3.5.4.3 DMatrix& Chol (const DMatrix & A) [friend]

Returns the Cholesky factorisation of a matrix A, which must be a positive definite symmetric matrix.

Parameters

A	is a DMatrix object.
-----	--------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

See also

[CholFSolve\(\)](#), [CholSolve\(\)](#)

3.5.4.4 void CholeskyDecomp (DMatrix & A, int n, DMatrix & pM) [friend]

Cholesky decomposition of a matrix.

Parameters

<i>A</i>	is a DMatrix object
<i>n</i>	is the number of columns of the input matrix A.
<i>pM</i>	(modified), is a DMatrix object which on output contains the Cholesky decomposition of input matrix a.

Returns

void

See also

[CholeskySolution\(\)](#)

3.5.4.5 DMatrix& CholeskyRoot (const DMatrix & A) [friend]

Returns the Cholesky root R of a given matrix A, such that $A=R'R$. A must be a positive definite symmetric matrix.

Parameters

<i>A</i>	is a DMatrix object.
----------	--------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.6 void CholeskySolution (const DMatrix & A, int n, const DMatrix & pM, const DMatrix & bM, DMatrix & xM) [friend]

Cholesky solution using the Cholesky decomposition of a matrix.

Parameters

<i>A</i>	DMatrix object
<i>n</i>	is the number of columns of the input matrix A.

pM	DMatrix object which on output contains the Cholesky decomposition of input matrix a .
bM	DMatrix object with a vector of right-hand-side values
xM	(modified) DMatrix object which on return contains the solution to the system of equations

Returns

void

See also[CholeskyDecomp\(\)](#)**3.5.4.7 DMatrix& CholFSolve (const DMatrix & Achol, const DMatrix & b)**

[friend]

Solves the system of equations $Ax = b$ using Cholesky factorisation. The function uses a previously found Cholesky factorisation.

Parameters

$Achol$	is a DMatrix object resulting from a previous call to Chol() .
b	is a DMatrix object.

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation (the resulting vector x)

See also[CholFSolve\(\)](#), [Chol\(\)](#)**3.5.4.8 DMatrix& CholSolve (const DMatrix & A, const DMatrix & b)**

[friend]

Solves the system of equations $Ax = b$ using Cholesky factorisation.

Parameters

A	is a DMatrix object
b	is a DMatrix object.

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation (the resulting vector x)

See also

[CholFSolve\(\)](#), [Chol\(\)](#)

3.5.4.9 DMatrix& colon (void) [friend]

This function generates a special [DMatrix](#) object with one row and one column which is understood by the indexing functions that take a [DMatrix](#) object as an argument to mean "all rows" or "all columns".

Returns

Reference to a temporary [DMatrix](#) object

3.5.4.10 DMatrix& colon (double *i1*, double *increment*, double *i2*) [friend]

This function generates a [DMatrix](#) object with a vector starting from a given value, with given increments and ending in a given value.

Parameters

<i>i1</i>	is the first value
<i>increment</i>	is the increment
<i>i2</i>	is the last value

Returns

Reference to a temporary [DMatrix](#) object

3.5.4.11 DMatrix& colon (int *i1*, int *increment*, int *i2*) [friend]

This function generates [DMatrix](#) object with a vector starting from a given value, with given increments and ending in a given value.

Parameters

<i>i1</i>	is the first value
<i>increment</i>	is the increment
<i>i2</i>	is the last value

Returns

Reference to a temporary [DMatrix](#) object

3.5.4.12 DMatrix& colon (int *i1*, int *i2*) [friend]

This function generates [DMatrix](#) object with a vector starting from a given value, with unit increments, and ending in a given value.

Parameters

<i>i1</i>	is the first value
<i>i2</i>	is the last value

Returns

Reference to a temporary [DMatrix](#) object

3.5.4.13 DMatrix& colon (double *i1*, double *i2*) [friend]

This function generates [DMatrix](#) object with a vector starting from a given value, with unit increments, and ending in a given value.

Parameters

<i>i1</i>	is the first value
<i>i2</i>	is the last value

Returns

Reference to a temporary [DMatrix](#) object

3.5.4.14 double cond (const DMatrix & *A*) [friend]

This function calculates the 2-norm condition number of a matrix, which is the ratio of the maximum singular value to the minimum singular value of the matrix. A large condition number indicates a nearly singular matrix. If the input matrix is not square, an error is thrown.

Parameters

<i>A</i>	is a DMatrix object.
----------	--------------------------------------

Returns

the 2-norm condition number

3.5.4.15 DMatrix& cos (const DMatrix & *A*) [friend]

This function returns a matrix with the cosine of each element of the input matrix.

Parameters

<i>A</i>	is a DMatrix object
----------	-------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.16 DMatrix& cosh (const DMatrix & A) [friend]

This function returns a matrix with the hyperbolic cosine of each element of the input matrix.

Parameters

<i>A</i>	is a DMatrix object
----------	-------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.17 DMatrix& cov (const DMatrix & A, int ntype = 0) [friend]

Computes the covariance matrix of a data matrix where the N rows correspond to samples and the M columns are variables. The result is returned as an M x M matrix. If ntype=0 (default) then the result is normalised with N-1. Otherwise, if ntype=1, the result is normalised with N.

Parameters

<i>A</i>	is a DMatrix object.
<i>ntype</i>	is the type of normalization, 0 (default) or 1.

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.18 DMatrix& cov (DMatrix & X, DMatrix & Y, int ntype = 0) [friend]

Computes the covariance matrix of two vectors X and Y of dimension N. The result is returned as an 1 x 1 [DMatrix](#) object. If ntype=0 (default) then the result is normalised with N-1. Otherwise, if ntype=1, the result is normalised with N.

Parameters

<i>X</i>	is a DMatrix object.
<i>Y</i>	is a DMatrix object
<i>ntype</i>	is the type of normalization, 0 (default) or 1.

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.19 DMatrix& cross (const DMatrix & x, const DMatrix & y) [friend]

This function computes the cross product of two vectors. If any of the input arguments does not contain a vector, or if the length of any of the vectors is not 3, an error is thrown.

Parameters

x	is a DMatrix object.
y	is a DMatrix object

Returns

the value of the cross product of x and y .

3.5.4.20 DMatrix& crossProduct (const DMatrix & x, const DMatrix & y) [friend]

This function computes the cross product of two vectors. If any of the input arguments does not contain a vector, or if the length of any of the vectors is not 3, an error is thrown.

Parameters

x	is a DMatrix object.
y	is a DMatrix object

Returns

the value of the cross product of x and y .

3.5.4.21 double det (const DMatrix & A) [friend]

This function returns the determinant of a square matrix. If the input matrix is not square, an error is thrown.

Parameters

A	is a DMatrix object.
-----	--------------------------------------

Returns

the determinant of the matrix.

3.5.4.22 DMatrix& diag (const DMatrix & A) [friend]

if A is a matrix this function extracts a column vector with the diagonal values of A. If A is a vector this function returns a matrix having the elements of A in the diagonal

Parameters

A	is a DMatrix object
---	-------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.23 double dot (const DMatrix & x, const DMatrix & y) [friend]

This function computes the dot product of two vectors. If any of the input arguments does not contain a vector, or if the vector lengths are not equal, an error is thrown.

Parameters

x	is a DMatrix object.
y	is a DMatrix object

Returns

the value of the dot product of x and y.

3.5.4.24 double dotProduct (const DMatrix & x, const DMatrix & y) [friend]

This function computes the dot product of two vectors. If any of the input arguments does not contain a vector, or if the vector lengths are not equal, an error is thrown.

Parameters

x	is a DMatrix object.
y	is a DMatrix object

Returns

the value of the dot product of x and y.

3.5.4.25 DMatrix& eig (const DMatrix & A, DMatrix * V=NULL) [friend]

This function computes the eigenvalues and (optionally) the eigenvectors of a matrix A. This function uses the LAPACK routines dsyev_() and dgeev_.

Parameters

A	is a DMatrix object.
V	is a pointer to a DMatrix object.

Returns

Reference to a temporary [DMatrix](#) object with the real part of the eigenvalues in the first column and the complex part of the eigenvalues in the second column.

3.5.4.26 DMatrix& elemDivision (const DMatrix & A , const DMatrix & B)

[friend]

This function computes and returns the element-wise division of two matrices of the same dimensions. If the dimensions of the two input matrices are not the same, an error is thrown. The dimensions of the returned object are the same as the dimensions of the factors.

Parameters

A	is a DMatrix object.
B	is a DMatrix object.

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.27 DMatrix& elemProduct (const DMatrix & A , const DMatrix & B)

[friend]

This function computes and returns the element-wise product of two matrices of the same dimensions. If the dimensions of the two input matrices are not the same, an error is thrown.

Parameters

A	is a DMatrix object.
B	is a DMatrix object.

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.28 double enorm (const DMatrix & A) [friend]

This function computes and return the Euclidean norm of a matrix A , which is the square root of the sum of its squared elements.

Parameters

<i>A</i>	is a DMatrix object.
----------	--------------------------------------

Returns

the value of the Euclidean norm.

3.5.4.29 void error_message (const char * *input_text*) [friend]

This function prints an error message and throws an exception to be handled by the [ErrorHandler](#) class.

Parameters

<i>input_text</i>	is the error message
-------------------	----------------------

Returns

void

3.5.4.30 DMatrix& exp (const DMatrix & *A*) [friend]

This function returns a matrix with the natural exponential of each element of the input matrix.

Parameters

<i>A</i>	is a DMatrix object
----------	-------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.31 DMatrix& expm (const DMatrix & *A*) [friend]

This function returns the exponential matrix of a given square matrix.

Parameters

<i>A</i>	is a DMatrix object
----------	-------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.32 DMatrix& eye (long n) [friend]

This function returns the identity matrix with a given number of rows and columns.

Parameters

n	is the desired number of rows and columns
-----	---

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.33 DMatrix& eye (long n , long m) [friend]

This function returns a truncated identity matrix with specified numbers of rows and columns.

Parameters

n	is the desired number of rows
m	is the desired number of columns

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.34 DMatrix& find (const DMatrix & A) [friend]

This function returns a column vector with the linear indices of the non-zero elements of the input matrix A . The linear index is 1 for element (1,1) of the input matrix A , and $\text{length}(A)$ for the (nrows,ncols) element of the input matrix A .

Parameters

A	is a DMatrix object.
-----	--------------------------------------

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.35 double Fnorm (const DMatrix & A) [friend]

This function computes Frobenius norm of matrix A .

Parameters

A	is a DMatrix object.
-----	--------------------------------------

Returns

the value of the Frobenius norm

3.5.4.36 DMatrix& identity (long n , long m) [friend]

This function returns a truncated identity matrix with specified numbers of rows and columns.

Parameters

n	is the desired number of rows
m	is the desired number of columns

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.37 DMatrix& identity (long n) [friend]

This function returns the identity matrix with a given number of rows and columns.

Parameters

n	is the desired number of rows and columns
-----	---

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.38 double InfNorm (const DMatrix & A) [friend]

This function computes infinity norm of matrix A , which is computed as the maximum absolute value row sum.

Parameters

A	is a DMatrix object.
-----	--------------------------------------

Returns

the value of the infinity norm

3.5.4.39 DMatrix& inv (const DMatrix & A) [friend]

This function returns the inverse of a given square matrix. If the argument is not a square matrix an error is thrown.

Parameters

A	is a DMatrix object.
-----	--------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.40 int isSymmetric (const DMatrix & A) [friend]

This function checks if the input matrix is symmetric. If the input matrix is not square, an error is thrown.

Parameters

A	is a DMatrix object.
-----	--------------------------------------

Returns

1 if the input matrix is symmetric, 0 otherwise.

3.5.4.41 DMatrix& kronProduct (const DMatrix & A, const DMatrix & B) [friend]

This function computes and returns the Kronecker product of two matrices. The row (column) dimension of the returned object is the product of the row (column) dimensions of both factors.

Parameters

A	is a DMatrix object.
B	is a DMatrix object.

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.42 long length (const DMatrix & A) [friend]

This function returns the number of elements of a matrix A.

Parameters

A	is a DMatrix object
-----	-------------------------------------

Returns

the number of elements of the input matrix.

3.5.4.43 DMatrix& linspace (double X1, double X2, long N) [friend]

This function returns a linearly spaced vector with N points between the values X1 and X2.

Parameters

X1	is a real number
X2	is a real number
N	is the desired length of the returned vector

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.44 DMatrix& log (const DMatrix & A) [friend]

This function returns a matrix with the natural logarithm of each element of the input matrix.

Parameters

A	is a DMatrix object
---	-------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.45 DMatrix& LQ (const DMatrix & A, DMatrix * Q) [friend]

Returns the LQ factorisation of a matrix A. The function uses the LAPACK routine dgelqf().

Parameters

A	is a DMatrix object.
Q	is a pointer to a DMatrix object, which is modified on output to contain the Q factor of the decomposition.

Returns

Reference to a temporary [DMatrix](#) object with the L factor of the decomposition.

3.5.4.46 DMatrix& LSMNSolve (const DMatrix & A, const DMatrix & B) [friend]

This function solves overdetermined or underdetermined real linear systems $Ax = B$ using a QR or LQ factorization of A. It is assumed that matrix A has full rank. The function uses the LAPACK routine dgels.

Parameters

A	is a DMatrix object
B	is a DMatrix object.

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation (the resulting vector x)

3.5.4.47 DMatrix& LU (const DMatrix & A) [friend]

Returns the LU factorisation of a matrix A .

Parameters

A	is a DMatrix object.
-----	--------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

See also

[LUSolve\(\)](#), [LUSolve\(\)](#)

3.5.4.48 DMatrix& LUSolve (const DMatrix & ALU, const DMatrix & b) [friend]

Solves the system of equations $Ax = b$ using LU factorisation using a previously found LU factors.

Parameters

ALU	is a DMatrix object resulting from a previous call to the LU() function
b	is a DMatrix object.

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation (the resulting vector x)

See also

[LU\(\)](#), [LUSolve\(\)](#)

3.5.4.49 DMatrix& LUSolve (const DMatrix & A, const DMatrix & b) [friend]

Solves the system of equations $Ax = b$ using LU factorisation.

Parameters

A	is a DMatrix object
b	is a DMatrix object.

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation (the resulting vector x)

See also

[LUSolve\(\)](#), [LU\(\)](#)

3.5.4.50 DMatrix& MatrixSign (const DMatrix & A) [friend]

This function returns a [DMatrix](#) object with the same dimensions as the input matrix such that each of its elements is 1 if the corresponding value of the input matrix is positive, -1 if the corresponding value of the input matrix is negative, and 0 if the corresponding value of the input matrix is 0.

Parameters

A	is a DMatrix object.
-----	--------------------------------------

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.51 double Max (const DMatrix & A, int * rindx = NULL, int * cindx = NULL) [friend]

This function finds and returns the element of matrix A with maximum value. It also returns the indices of such element. If more than one element has the same maximum value, the indices of the first element found when searching column by column is returned.

Parameters

A	is a DMatrix object.
$rindx$	is an optional pointer to an integer which is modified with the row index.
$cindx$	is an optional pointer to an integer which is modified with the column index.

Returns

the value of the element with maximum value.

3.5.4.52 `double MaxAbs (const DMatrix & A, int * rindx = NULL, int * cindx = NULL)`
 [friend]

This function finds and returns the element of matrix A with maximum absolute value. It also returns the indices of such element. If more than one element has the same maximum absolute value, the indices of the first element found when searching column by column is returned.

Parameters

<i>A</i>	is a DMatrix object.
<i>rindx</i>	is a pointer to an integer which is modified with the row index.
<i>cindx</i>	is a pointer to an integer which is modified with the column index.

Returns

the absolute value of the element with maximum absolute value.

3.5.4.53 `DMatrix& mean (const DMatrix & A)` [friend]

This function returns a row vector with the mean values of the columns of matrix A.

Parameters

<i>A</i>	is a DMatrix object.
----------	--------------------------------------

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.54 `double Min (const DMatrix & A, int * rindx = NULL, int * cindx = NULL)`
 [friend]

This function finds and returns the element of matrix A with minimum value. It also returns the indices of such element. If more than one element has the same minimum value, the indices of the first element found when searching column by column is returned.

Parameters

<i>A</i>	is a DMatrix object.
<i>rindx</i>	is a pointer to an integer which is modified with the row index.
<i>cindx</i>	is a pointer to an integer which is modified with the column index.

Returns

the absolute value of the element with minimum absolute value.

3.5.4.55 `double MinAbs (const DMatrix & A, int * rindx = NULL, int * cindx = NULL)` [friend]

This function finds and returns the element of matrix A with minimum absolute value. It also returns the indices of such element. If more than one element has the same minimum absolute value, the indices of the first element found when searching column by column is returned.

Parameters

<i>A</i>	is a DMatrix object.
<i>rindx</i>	is a pointer to an integer which is modified with the row index.
<i>cindx</i>	is a pointer to an integer which is modified with the column index.

Returns

the absolute value of the element with minimum absolute value.

3.5.4.56 `DMatrix& mpow (DMatrix & A, int p)` [friend]

This function calculates the integer matrix power.

Parameters

<i>A</i>	is a DMatrix object.
<i>p</i>	is an integer value which can be positive or negative

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.57 `double norm (const DMatrix & A)` [friend]

This function computes 2-norm of matrix A, which is computed as the maximum singular value of A.

Parameters

<i>A</i>	is a DMatrix object.
----------	--------------------------------------

Returns

the value of the 2-norm

3.5.4.58 `DMatrix& null (const DMatrix & A)` [friend]

This function returns Z, the orthonormal basis for the null space of a matrix A, such that $ZZ' = I$ and $AZ = 0$. The number of columns of Z is the nullity of A.

Parameters

A	is a DMatrix object
-----	-------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.59 DMatrix& ones (long n , long m) [friend]

This function returns a matrix full of ones with specified numbers of rows and columns.

Parameters

n	is the desired number of rows
m	is the desired number of columns

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.60 DMatrix& operator* (double r , const DMatrix & A) [friend]

This function multiplies a real number by a matrix.

Parameters

r	is a double value
A	is a DMatrix object.

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.61 DMatrix& operator- (const DMatrix & A) [friend]

Matrix unary minus operator. Returns an object of the same dimensions as A but with changed element signs.

Parameters

$A, :$	matrix located at the right hand side of the operator.
--------	--

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.62 DMatrix& orth (const DMatrix & A) [friend]

This function returns Q , the orthonormal basis for the range of a matrix A , such that $QQ' = I$. The number of columns of Q is the rank of A .

Parameters

A	is a DMatrix object
-----	-------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.63 DMatrix& pinv (const DMatrix & A) [friend]

This function returns the pseudo-inverse of a given rectangular matrix.

Parameters

A	is a DMatrix object.
-----	--------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.64 DMatrix& prod (const DMatrix & A) [friend]

This function returns a row vector with the product of the elements of each column of matrix A .

Parameters

A	is a DMatrix object.
-----	--------------------------------------

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.65 DMatrix& Product (const DMatrix & A, const DMatrix & B) [friend]

This function calculates the product of two matrices. The number of columns of the first matrix must be the same as the number of rows of the second matrix, otherwise an error is thrown.

Parameters

A	is a DMatrix object
B	is a DMatrix object.

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.66 DMatrix& ProductT (const DMatrix & A, const DMatrix & B) [friend]

This function returns the product of the first matrix times the second matrix transposed. The number of columns of both matrices must be the same, otherwise an error is thrown.

Parameters

<i>A</i>	is a DMatrix object
<i>B</i>	is a DMatrix object.

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.67 DMatrix& QR (const DMatrix & A) [friend]

Returns the QR factorisation of a matrix A. The number of rows of matrix A must be greater or equal than the number of columns.

Parameters

<i>A</i>	is a DMatrix object.
----------	--------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

See also

[QRFSolve\(\)](#), [QRSolve\(\)](#)

3.5.4.68 DMatrix& QRFSolve (const DMatrix & A, const DMatrix & b) [friend]

Solves the system of equations $Ax = b$ using QR factorisation. The function uses a previously found QR factorisation.

Parameters

<i>A</i>	is a DMatrix object.
<i>b</i>	is a DMatrix object.

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation (the re-

sulting vector x)

See also

[QRSolve\(\)](#), [QR\(\)](#)

3.5.4.69 **DMatrix& QRSolve (const DMatrix & A , const DMatrix & b)** [friend]

Solves the system of equations $Ax = b$ using QR factorisation. The number of rows of matrix A must be greater or equal than the number of columns.

Parameters

A	is a DMatrix object.
b	is a DMatrix object.

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation (the resulting vector x)

See also

[QRFSolve\(\)](#), [QR\(\)](#)

3.5.4.70 **DMatrix& randn (long n , long m)** [friend]

This function returns an $n \times m$ matrix where each element is a Gaussian pseudo-random number in the range with zero mean and variance 1.

Parameters

n	is the desired number of rows of the returned matrix
m	is the desired number of columns of the returned matrix

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.71 **DMatrix& randu (long n , long m)** [friend]

This function returns an $n \times m$ matrix where each element is a uniform pseudo-random number in the range (0,1).

Parameters

n	is the desired number of rows of the returned matrix
m	is the desired number of columns of the returned matrix

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.72 int rank (const DMatrix & A) [friend]

This function returns an estimate of the rank of a matrix, which is the number of linearly independent rows or columns.

Parameters

A	is a DMatrix object.
-----	--------------------------------------

Returns

the rank estimate.

3.5.4.73 double rcond (const DMatrix & A) [friend]

This function estimates the 1-norm reciprocal condition number of a matrix. The function uses the LAPACK function dgecon. if A is well conditioned, then rcond(A) is near 1. If A is badly conditioned, then rcond(A) is close to the machine numerical precision (very small). If the input matrix is not square, an error is thrown.

Parameters

A	is a DMatrix object.
-----	--------------------------------------

Returns

the reciprocal condition number estimate

3.5.4.74 DMatrix& reshape (DMatrix & A, long N, long M) [friend]

This function returns the N-by-M matrix whose elements are taken columnwise from the input matrix A. An error is thrown if A does not have N*M elements.

Parameters

A	is a DMatrix object
N	is the desired number of rows of the reshaped matrix
M	is the desired number of columns of the reshaped matrix

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.75 DMatrix& schur (const DMatrix & A, DMatrix * U=NULL) [friend]

This function computes and returns the Schur decomposition of a matrix A, such that $A = Q'UQ$, where U is an upper triangular matrix and Q is a unitary matrix. This function uses the LAPACK routine dgees_()

Parameters

A	is a DMatrix object.
U	is a pointer to a DMatrix object.

Returns

Reference to a temporary [DMatrix](#) object with the unitary matrix Q.

3.5.4.76 DMatrix& sin (const DMatrix & A) [friend]

This function returns a matrix with the sine of each element of the input matrix.

Parameters

A	is a DMatrix object
---	-------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.77 DMatrix& sinh (const DMatrix & A) [friend]

This function returns a matrix with the hyperbolic sine of each element of the input matrix.

Parameters

A	is a DMatrix object
---	-------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.78 void sort (DMatrix & x, DMatrix & indx) [friend]

This function sorts the input vector x in ascending order. It also returns a [DMatrix](#) object with the sorted indices. If the input object is not a vector, then an error is thrown.

Parameters

x	is a DMatrix object which upon input contains the unsorted vector and upon output contains the sorted vector.
---	---

<i>indx</i>	is a DMatrix object which upon output contains the values of the sorted indices.
-------------	--

Returns

void

3.5.4.79 void sort (DMatrix & x, int indx[] = NULL) [friend]

This function sorts the input vector x in ascending order. Optionally, it also returns an integer array of sorted indices. If the input object is not a vector, then an error is thrown.

Parameters

<i>x</i>	is a DMatrix object which upon input contains the unsorted vector and upon output contains the sorted vector.
<i>indx</i>	is a pointer to the first element of the array of sorted indices.

Returns

void

3.5.4.80 DMatrix& Sqrt (const DMatrix & A) [friend]

This function computes the square root of each element of the input matrix A, and it returns a [DMatrix](#) object with the same dimensions as the input matrix. If any element of the input matrix is negative, an error is thrown.

Parameters

<i>A</i>	is a DMatrix object.
----------	--------------------------------------

Returnsa temporary [DMatrix](#) object with the result of the operation**3.5.4.81 DMatrix& Std (const DMatrix & A, int ntype = 0) [friend]**

This function returns a row vector with the standard deviation of each column of matrix A. If ntype is 0 (default) the result is normalised with (n-1), where n is the number of rows of A. Otherwise, the result is normalised with n.

Parameters

<i>A</i>	is a DMatrix object.
<i>ntype</i>	is the type of normalization, 0 (default) or 1.

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.82 DMatrix& sum (const DMatrix & A) [friend]

This function returns a row vector with the sum of the elements of each column of matrix A.

Parameters

A	is a DMatrix object.
---	--------------------------------------

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.83 DMatrix& SVD (const DMatrix & A, DMatrix * U=NULL, DMatrix * V=NULL) [friend]

Returns the singular value decomposition of a matrix $A = U' \text{diag}(s)V$, where vector s contains the singular values of A. The function uses the LAPACK routine dgesvd().

Parameters

A	is a DMatrix object.
U	is a pointer to a DMatrix object, which is modified on output to contain the U factor of the decomposition.
V	is a pointer to a DMatrix object, which is modified on output to contain the V factor of the decomposition.

Returns

Reference to a temporary [DMatrix](#) object with a vector that contains the singular values of matrix A.

3.5.4.84 DMatrix& SVDSolve (const DMatrix & A, const DMatrix & B) [friend]

This function uses the LAPACK routine dgels_() to compute the minimum norm solution to a real linear least squares problem: Minimize $\|B - Ax\|_2$ using the singular value decomposition (SVD) of A. A is a rectangular matrix which may be rank-deficient.

Parameters

A	is a DMatrix object.
B	is a DMatrix object.

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation (the resulting vector x)

See also

[SVD\(\)](#)

3.5.4.85 DMatrix& tan (const DMatrix & A) [friend]

This function returns a matrix with the tangent of each element of the input matrix.

Parameters

A	is a DMatrix object
-----	-------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.86 DMatrix& tanh (const DMatrix & A) [friend]

This function returns a matrix with the hyperbolic tangent of each element of the input matrix.

Parameters

A	is a DMatrix object
-----	-------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.87 double toc () [friend]

This function, which is to be used in conjunction with function [tic\(\)](#), stops counting CPU time, and it prints and returns the elapsed time in seconds since the function [tic\(\)](#) was called.

Returns

the elapsed time in seconds.

3.5.4.88 DMatrix& TProduct (const DMatrix & A, const DMatrix & B) [friend]

This function returns the product of the first matrix transposed times the second matrix. The number of rows of both matrices must be the same, otherwise an error is thrown.

Parameters

<i>A</i>	is a DMatrix object
<i>B</i>	is a DMatrix object.

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.89 DMatrix& TProductT (const DMatrix & A, const DMatrix & B) [friend]

This function returns the product of the first matrix transposed times the second matrix transposed. The number of rows of the first matrix must be the same as the number of columns of the second matrix, otherwise an error is thrown.

Parameters

<i>A</i>	is a DMatrix object
<i>B</i>	is a DMatrix object.

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.90 DMatrix& tra (const DMatrix & A) [friend]

This function returns the transpose of a given matrix.

Parameters

<i>A</i>	is a DMatrix object.
----------	--------------------------------------

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.4.91 double trace (const DMatrix & A) [friend]

This function returns the trace of a square matrix. If the input matrix is not square, an error is thrown.

Parameters

<i>A</i>	is a DMatrix object.
----------	--------------------------------------

Returns

the trace of the matrix.

3.5.4.92 DMatrix& triu (const DMatrix & A) [friend]

This function extracts and return the triangular upper part of the input matrix A. The returned object has the same dimensions as the input object.

Parameters

<i>A</i>	is a DMatrix object
----------	-------------------------------------

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.93 DMatrix& var (DMatrix & A, int ntype = 0) [friend]

This function returns a row vector with the variance of each column of matrix A. If ntype is 0 (default) the result is normalised with (n-1), where n is the number of rows of A. Otherwise, the result is normalised with n.

Parameters

<i>A</i>	is a DMatrix object.
<i>ntype</i>	is the type of normalization, 0 (default) or 1.

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.94 DMatrix& vec (const DMatrix & A) [friend]

This function returns a column vector made by stacking the columns of a matrix one below the other from left to right.

Parameters

<i>A</i>	is a DMatrix object.
----------	--------------------------------------

Returns

a temporary [DMatrix](#) object with the result of the operation

3.5.4.95 DMatrix& zeros (long n, long m) [friend]

This function returns a matrix full of zeros with specified numbers of rows and columns.

Parameters

<i>n</i>	is the desired number of rows
<i>m</i>	is the desired number of columns

Returns

Reference to a temporary [DMatrix](#) object with the result of the operation

3.5.5 Member Data Documentation

3.5.5.1 `int DMatrix::atype` [protected]

Flag to indicate type of allocation. type = 0 : allocated matrix type = 1 : non-allocated matrix, uses predefined array for storage

3.5.5.2 `int DMatrix::mtype` [protected]

Flag to indicate type of matrix mtype = 0 : normal matrix mtype = 1 : colon - reference matrix

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

- `dmatrixv.h`

3.6 doublecomplex Struct Reference

Public Attributes

- `doublereal r`
- `doublereal i`

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

- `f2c.h`

3.7 ErrorHandler Class Reference

[ErrorHandler](#) class.

```
#include <dmatrixv.h>
```

Public Member Functions

- [ErrorHandler](#) (const string m)

A constructor which takes the error message as an argument and assigns it to error_message.

Public Attributes

- string [error_message](#)

A string of characters which contains the error message.

3.7.1 Detailed Description

[ErrorHandler](#) class. This is a C++ class intended to handle error conditions.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 ErrorHandler::ErrorHandler (const string *m*)

A constructor which takes the error message as an argument and assigns it to `error_message`.

Parameters

<i>m</i>	is the error message string.
----------	------------------------------

See also

function [error_message\(\)](#).

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

- `dmatrixv.h`

3.8 icilist Struct Reference

Public Attributes

- flag **icierr**
- char * **iciunit**
- flag **iciend**
- char * **icifmt**
- ftntint **icirlen**
- ftntint **icirnum**

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

- `f2c.h`

3.9 InitializeDMatrixClass Class Reference

[InitializeDMatrixClass](#) class.

```
#include <dmatrixv.h>
```

Public Member Functions

- [InitializeDMatrixClass](#) ()

This is the default constructor which calls the function [DMatrix::AllocateAuxArr\(\)](#).

- [~InitializeDMatrixClass](#) ()

This is the destructor which calls the function [DMatrix::DeAllocateAuxArr\(\)](#).

3.9.1 Detailed Description

[InitializeDMatrixClass](#) class. This is a dummy C++ class intended to initialise the temporary objects of the [DMatrix](#) class.

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

- [dmatrixv.h](#)

3.10 inlist Struct Reference

Public Attributes

- flag **inerr**
- ftntint **inunit**
- char * **infile**
- ftntlen **inflen**
- ftntint * **inex**
- ftntint * **inopen**
- ftntint * **innum**
- ftntint * **innamed**
- char * **inname**
- ftntlen **innamlen**
- char * **inacc**
- ftntlen **inacclen**
- char * **inseq**
- ftntlen **inseqlen**
- char * **indir**
- ftntlen **indirlen**
- char * **infmt**

- `ftnlen` **infmtlen**
- `char *` **inform**
- `ftnint` **informlen**
- `char *` **inunf**
- `ftnlen` **inunflen**
- `ftnint *` **inrecl**
- `ftnint *` **innrec**
- `char *` **inblank**
- `ftnlen` **inblanklen**

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

- `f2c.h`

3.11 Multitype Union Reference

Public Attributes

- integer1 **g**
- shortint **h**
- integer **i**
- real **r**
- doublereal **d**
- [complex](#) **c**
- [doublecomplex](#) **z**

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

- `f2c.h`

3.12 Namelist Struct Reference

Public Attributes

- `char *` **name**
- [Vardesc](#) **** vars**
- `int` **nvars**

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

- `f2c.h`

3.13 olist Struct Reference

Public Attributes

- flag **oerr**
- ftntint **ounit**
- char * **ofnm**
- ftntlen **ofnmlen**
- char * **osta**
- char * **oacc**
- char * **ofm**
- ftntint **orl**
- char * **oblnc**

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

- f2c.h

3.14 Vardesc Struct Reference

Public Attributes

- char * **name**
- char * **addr**
- ftntlen * **dims**
- int **type**

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

- f2c.h

Index

- Abs
 - DMatrix, [59](#)
- alist, [5](#)
- Allocate
 - DMatrix, [29](#)
- AllocateAuxArr
 - DMatrix, [29](#)
- any
 - DMatrix, [59](#)
- assign
 - DMatrix, [30](#)
- AssignmentToColonReference
 - DMatrix, [30](#)
- atype
 - DMatrix, [88](#)
- Chol
 - DMatrix, [59](#)
- CholeskyDecomp
 - DMatrix, [60](#)
- CholeskyRoot
 - DMatrix, [60](#)
- CholeskySolution
 - DMatrix, [60](#)
- CholFSolve
 - DMatrix, [61](#)
- CholSolve
 - DMatrix, [61](#)
- cilist, [5](#)
- cllist, [5](#)
- colMult
 - DMatrix, [31](#)
- colon
 - DMatrix, [62](#), [63](#)
- Column
 - DMatrix, [31](#)
- complex, [6](#)
- compMat
 - DMatrix, [31](#)
- cond
 - DMatrix, [63](#)
- cos
 - DMatrix, [63](#)
- cosh
 - DMatrix, [64](#)
- cov
 - DMatrix, [64](#)
- cross
 - DMatrix, [65](#)
- crossProduct
 - DMatrix, [65](#)
- DeAllocate
 - DMatrix, [31](#)
- DeAllocateAuxArr
 - DMatrix, [32](#)
- det
 - DMatrix, [65](#)
- diag
 - DMatrix, [32](#), [65](#)
- DMatrix, [6](#)
 - Abs, [59](#)
 - Allocate, [29](#)
 - AllocateAuxArr, [29](#)
 - any, [59](#)
 - assign, [30](#)
 - AssignmentToColonReference, [30](#)
 - atype, [88](#)
 - Chol, [59](#)
 - CholeskyDecomp, [60](#)
 - CholeskyRoot, [60](#)
 - CholeskySolution, [60](#)
 - CholFSolve, [61](#)
 - CholSolve, [61](#)
 - colMult, [31](#)
 - colon, [62](#), [63](#)
 - Column, [31](#)
 - compMat, [31](#)
 - cond, [63](#)
 - cos, [63](#)
 - cosh, [64](#)
 - cov, [64](#)