MatrixLab

1.2

Generated by Doxygen 1.8.1.1

Mon Apr 27 2015 23:45:49

Contents

Chapter 1

Matrixlab

1.1 Introduction

Matrixlab is a generic C library for matrix routines. It contains over 250 functions for matrix operations. Many of the functions are multi-threaded.

Chapter 2

matrixlab

C Matrix Library

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Data Structure Documentation

5.1 mat_bayes_model Struct Reference

Bayes Classifier Model Structure.

```
#include <matrix.h>
```

Data Fields

- int num_of_classes
- int num_of_features
- INT_VECTOR class_labels
- MATRIX class_priors
- MATSTACK class_means
- MATSTACK class_covars

5.1.1 Detailed Description

Bayes Classifier Model Structure.

5.1.2 Field Documentation

5.1.2.1 MATSTACK mat_bayes_model::class_covars

Training data class covariances

5.1.2.2 INT_VECTOR mat_bayes_model::class_labels

Training data class label vector

5.1.2.3 MATSTACK mat_bayes_model::class_means

Training data class means

5.1.2.4 MATRIX mat_bayes_model::class_priors

Training data prior information

5.1.2.5 int mat_bayes_model::num_of_classes

Number of training class

5.1.2.6 int mat_bayes_model::num_of_features

Number of training features

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

- matrix.h

5.2 mat_gnode Struct Reference

Graph Node Structure.

```
#include <matrix.h>
```

Data Fields

- int v
- double weight
- struct mat_gnode ∗ next

5.2.1 Detailed Description

Graph Node Structure.

5.2.2 Field Documentation

5.2.2.1 struct mat_gnode∗ mat_gnode::next

Pointer to next node

5.2.2.2 int mat_gnode::v

Value

5.2.2.3 double mat_gnode::weight

Node weight

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

- matrix.h

5.3 mat_graph Struct Reference

Graph Structure.

```
#include <matrix.h>
```

Data Fields

- int nvertices
- int nedges
- int ∗ val
- int ∗ vseq
- int id
- MAT_GNODE ∗ adj
- MAT_GNODE z
- int ∗ dad
- int weighted
- MAT_INT_PRIORITYQUEUE pq

5.3.1 Detailed Description

Graph Structure.

5.3.2 Field Documentation

5.3.2.1 MAT_GNODE∗ mat_graph::adj

5.3.2.2 int∗ mat_graph::dad

5.3.2.3 int mat_graph::id

5.3.2.4 int mat_graph::nedges

Number of edges

5.3.2.5 int mat_graph::nvertices

Number of vertices

5.3.2.6 MAT_INT_PRIORITYQUEUE mat_graph::pq

5.3.2.7 int∗ mat_graph::val

5.3.2.8 int∗ mat_graph::vseq

5.3.2.9 int mat_graph::weighted

5.3.2.10 MAT_GNODE mat_graph::z

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

- matrix.h

5.4 mat_int_priorityqueue Struct Reference

Integer Priority Queue Structure.

```
#include <matrix.h>
```

Data Fields

- int p
- int type
- int length
- MAT_INTPQNODE element

5.4.1 Detailed Description

Integer Priority Queue Structure.

5.4.2 Field Documentation

5.4.2.1 MAT_INTPQNODE mat_int_priorityqueue::element

Pointer to priority queue data

5.4.2.2 int mat_int_priorityqueue::length

Total allocated priority queue length

5.4.2.3 int mat_int_priorityqueue::p

Current priority queue position

5.4.2.4 int mat_int_priorityqueue::type

Priority type

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

- matrix.h

5.5 mat_int_queue Struct Reference

Integer Queue Structure.

```
#include <matrix.h>
```

Data Fields

- int p
- MAT_QINTNODE head
- MAT_QINTNODE tail

5.5.1 Detailed Description

Integer Queue Structure.

5.5.2 Field Documentation

5.5.2.1 MAT_QINTNODE mat_int_queue::head

Queue head node

5.5.2.2 int mat_int_queue::p

Current queue position

5.5.2.3 MAT_QINTNODE mat_int_queue::tail

Queue tail node

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

- matrix.h

5.6 mat_int_stack Struct Reference

Integer Stack Structure.

```
#include <matrix.h>
```

Data Fields

- int p
- int length
- int ∗ stack

5.6.1 Detailed Description

Integer Stack Structure.

5.6.2 Field Documentation

5.6.2.1 int mat_int_stack::length

Total allocated stack length

5.6.2.2 int mat_int_stack::p

Current stack position

5.6.2.3 int∗ mat_int_stack::stack

Pointer to stack data

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

- matrix.h

5.7 mat_intpqnode Struct Reference

Integer Priority Queue Node Structure.

```
#include <matrix.h>
```

Data Fields

- int data
- int priority

5.7.1 Detailed Description

Integer Priority Queue Node Structure.

5.7.2 Field Documentation

5.7.2.1 int mat_intpqnode::data

Integer node data

5.7.2.2 int mat_intpqnode::priority

Node priority

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

- matrix.h

5.8 mat_kdnode Struct Reference

```
#include <matrix.h>
```

Data Fields

- mtype x [MAT_KDTREE_MAX_DIMS]
- int idx
- struct mat_kdnode ∗ left
- struct mat_kdnode ∗ right

5.8.1 Field Documentation

5.8.1.1 int mat_kdnode::idx

5.8.1.2 struct mat_kdnode∗ mat_kdnode::left

5.8.1.3 struct mat_kdnode ∗ mat_kdnode::right

5.8.1.4 mtype mat_kdnode::x[MAT_KDTREE_MAX_DIMS]

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

- matrix.h

5.9 mat_kdtree Struct Reference

```
#include <matrix.h>
```

Data Fields

- int ndims
- int length
- int _is_allocated
- MAT_KDNODE data
- MAT_KDNODE kdtree

5.9.1 Field Documentation

5.9.1.1 int mat_kdtree::_is_allocated

5.9.1.2 MAT_KDNODE mat_kdtree::data

5.9.1.3 MAT_KDNODE mat_kdtree::kdtree

5.9.1.4 int mat_kdtree::length

5.9.1.5 int mat_kdtree::ndims

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

- matrix.h

5.10 mat_mtype_priorityqueue Struct Reference

Mtype Priority Queue Structure.

```
#include <matrix.h>
```

Data Fields

- int p
- int type
- int length
- MAT_MTYPEPQNODE element

5.10.1 Detailed Description

Mtype Priority Queue Structure.

5.10.2 Field Documentation

5.10.2.1 MAT_MTYPEPQNODE mat_mtype_priorityqueue::element

Pointer to priority queue data

5.10.2.2 int mat_mtype_priorityqueue::length

Total allocated priority queue length

5.10.2.3 int mat_mtype_priorityqueue::p

Current priority queue position

5.10.2.4 int mat_mtype_priorityqueue::type

Priority type

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

- matrix.h

5.11 mat_mtype_queue Struct Reference

Mtype Queue Structure.

```
#include <matrix.h>
```

Data Fields

- int p
- MAT_QMTYPENODE head
- MAT_QMTYPENODE tail

5.11.1 Detailed Description

Mtype Queue Structure.

5.11.2 Field Documentation

5.11.2.1 MAT_QMTYPENODE mat_mtype_queue::head

Queue head node

5.11.2.2 int mat_mtype_queue::p

Current queue position

5.11.2.3 MAT_QMTYPENODE mat_mtype_queue::tail

Queue tail node

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

- matrix.h

5.12 mat_mtype_stack Struct Reference

Mtype Stack Structure.

```
#include <matrix.h>
```

Data Fields

- int p
- int length
- mtype * stack

5.12.1 Detailed Description

Mtype Stack Structure.

5.12.2 Field Documentation

5.12.2.1 int mat_mtype_stack::length

Total allocated stack length

5.12.2.2 int mat_mtype_stack::p

Current stack position

5.12.2.3 mtype* mat_mtype_stack::stack

Pointer to stack data

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

- matrix.h

5.13 mat_mtypepqnode Struct Reference

Mtype Priority Queue Node Structure.

```
#include <matrix.h>
```

Data Fields

- mtype data
- mtype priority

5.13.1 Detailed Description

Mtype Priority Queue Node Structure.

5.13.2 Field Documentation

5.13.2.1 mtype mat_mtypepqnode::data

Mtype node data

5.13.2.2 mtype mat_mtypepqnode::priority

Node priority

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

- matrix.h

5.14 mat_perceptron Struct Reference

Perceptron Classifier Model Structure.

```
#include <matrix.h>
```

Data Fields

- int num_of_classes
- int num_of_features
- INT_VECTOR class_labels
- MATRIX class_weights
- int istrained
- int num_of_iterations

5.14.1 Detailed Description

Perceptron Classifier Model Structure.

5.14.2 Field Documentation

5.14.2.1 INT_VECTOR mat_perceptron::class_labels

Training data class label vector

5.14.2.2 MATRIX mat_perceptron::class_weights

Trained Classifier Weights

5.14.2.3 int mat_perceptron::istrained

Is trained

5.14.2.4 int mat_perceptron::num_of_classes

Number of training classes

5.14.2.5 int mat_perceptron::num_of_features

Number of training features

5.14.2.6 int mat_perceptron::num_of_iterations

Number of training iterations

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

- matrix.h

5.15 mat_qintnode Struct Reference

Integer Queue Node Structure.

```
#include <matrix.h>
```

Data Fields

- int data
- struct mat_qintnode * next

5.15.1 Detailed Description

Integer Queue Node Structure.

5.15.2 Field Documentation

5.15.2.1 int mat_qintnode::data

Integer node data

5.15.2.2 struct mat_qintnode* mat_qintnode::next

Pointer to next node

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

- matrix.h

5.16 mat_qmtypenode Struct Reference

Mtype Queue Node Structure.

```
#include <matrix.h>
```

Data Fields

- mtype data
- struct mat_qmtypenode * next

5.16.1 Detailed Description

Mtype Queue Node Structure.

5.16.2 Field Documentation

5.16.2.1 mtype mat_qmtypenode::data

Mtype node data

5.16.2.2 struct mat_qmtypenode* mat_qmtypenode::next

Pointer to next node

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

- matrix.h

5.17 mat_tree_node Struct Reference

Search Tree Node Structure.

```
#include <matrix.h>
```

Data Fields

- mtype element
- struct mat_tree_node * left
- struct mat_tree_node * right

5.17.1 Detailed Description

Search Tree Node Structure.

5.17.2 Field Documentation

5.17.2.1 mtype mat_tree_node::element

Search tree node data

5.17.2.2 struct mat_tree_node* mat_tree_node::left

Pointer to left child node

5.17.2.3 struct mat_tree_node* mat_tree_node::right

Pointer to right child node

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

- matrix.h

Chapter 6

File Documentation

6.1 matabs.c File Reference

Functions

- MATRIX mat_abs (MATRIX A, MATRIX result)

  Computes absolute value of matrix.

6.1.1 Function Documentation

6.1.1.1 MATRIX mat_abs (MATRIX *A,* MATRIX *result*)

Computes absolute value of matrix.

Parameters

in	*A*	Input matrix
in	*result*	Matrix to store the result

Returns

$abs(A)$

6.2 matadd.c File Reference

Functions

- MATRIX mat_add (MATRIX A, MATRIX B, MATRIX result)

  Adds two matrices.
- MATRIX mat_adds (MATRIX A, mtype s, MATRIX result)

  Adds a scalar to a matrix.
- INT_VECTOR int_vec_add (INT_VECTOR A, INT_VECTOR B, INT_VECTOR result)

  Adds two integer vectors.
- INT_VECTOR int_vec_adds (INT_VECTOR A, int s, INT_VECTOR result)

  Adds an integer to an integer vector.

6.2.1 Function Documentation

6.2.1.1 INT_VECTOR int_vec_add (INT_VECTOR *A*, INT_VECTOR *B*, INT_VECTOR *result*)

Adds two integer vectors.

Parameters

`in`	*A*	Input vector
`in`	*B*	Input vector
`in`	*result*	Vector to store the result

Returns

$\mathbf{A}+\mathbf{B}$

6.2.1.2 INT_VECTOR int_vec_adds (INT_VECTOR *A*, int *s*, INT_VECTOR *result*)

Adds an integer to an integer vector.

Parameters

`in`	*A*	Input vector
`in`	*s*	Input scalar
`in`	*result*	Vector to store the result

Returns

$\mathbf{A}+s\mathbf{1}$

6.2.1.3 MATRIX mat_add (MATRIX *A*, MATRIX *B*, MATRIX *result*)

Adds two matrices.

Parameters

`in`	*A*	Input matrix
`in`	*B*	Input matrix
`in`	*result*	Matrix to store the result

Returns

$\mathbf{A}+\mathbf{B}$

6.2.1.4 MATRIX mat_adds (MATRIX *A*, mtype *s*, MATRIX *result*)

Adds a scalar to a matrix.

Parameters

`in`	*A*	Input matrix
`in`	*s*	Input scalar
`in`	*result*	Matrix to store the result

Returns

$\mathbf{A} + s\mathbf{1}\mathbf{1}^T$

6.3 matcompress.c File Reference

6.4 matconcat.c File Reference

Functions

- MATRIX mat_concat (MATRIX A, MATRIX B, int dim)

  Concatenates two matrices.
- INT_VECTOR int_vec_concat (INT_VECTOR a, INT_VECTOR b, INT_VECTOR result)

  Concatenates two integer vectors.

6.4.1 Function Documentation

6.4.1.1 INT_VECTOR int_vec_concat (INT_VECTOR *a*, INT_VECTOR *b*, INT_VECTOR *result*)

Concatenates two integer vectors.

Parameters

in	*a*	Input first vector
in	*b*	Input second vector
in	*dim*	Concatenation direction (ROWS/COLS)

Returns

$\begin{bmatrix} a & b \end{bmatrix}$ or $\begin{bmatrix} a \\ b \end{bmatrix}$

6.4.1.2 MATRIX mat_concat (MATRIX *A*, MATRIX *B*, int *dim*)

Concatenates two matrices.

Parameters

in	*A*	Input first matrix
in	*B*	Input second matrix
in	*dim*	Concatenation direction (ROWS/COLS)

Returns

$\begin{bmatrix} A & B \end{bmatrix}$ or $\begin{bmatrix} A \\ B \end{bmatrix}$

6.5 matconv.c File Reference

Functions

- INT_VECTOR mat_2int_vec (MATRIX A)

  Converts a matrix to an integer vector.

- MATRIX int_vec2_mat (INT_VECTOR a, int dir)

  Converts an integer vector to a matrix.
- MATRIX mat_vectorize (MATRIX A, MATRIX result)

  Reshapes a matrix to a vector.
- MATRIX mat_vectorize_tr (MATRIX A, MATRIX result)

  Reshapes transpose of a matrix to a vector.

6.5.1 Function Documentation

6.5.1.1 MATRIX int_vec2_mat (INT_VECTOR *a*, int *dir*)

Converts an integer vector to a matrix.

Parameters

in	*a*	Input vector
in	*dir*	Conversion direction

Returns

Output matrix

6.5.1.2 INT_VECTOR mat_2int_vec (MATRIX *A*)

Converts a matrix to an integer vector.

Parameters

in	*A*	Input matrix
out	*v*	Output vector

Returns

Output vector

6.5.1.3 MATRIX mat_vectorize (MATRIX *A*, MATRIX *result*)

Reshapes a matrix to a vector.

Parameters

in	*A*	Input matrix
in	*result*	Matrix to store the result

Returns

$vec(\mathbf{A})$

6.5.1.4 MATRIX mat_vectorize_tr (MATRIX *A*, MATRIX *result*)

Reshapes transpose of a matrix to a vector.

Parameters

in	*A*	Input matrix
in	*result*	Matrix to store the result

Returns

$vec(\mathbf{A}^T)$

6.6 matcreat.c File Reference

Functions

- MATRIX mat_creat (int row, int col, int type)

  Creates a matrix.
- MATSTACK matstack_creat (int len)

  Creates a matrix stack.
- MATSTACK matstack_append (MATSTACK s, MATRIX A)

  Appends a matrix to a matrix stack.
- int matstack_free (MATSTACK A)

  Frees a matrix stack.
- MATRIX mat_fill (MATRIX A, mtype val)

  Fills a matrix with a value.
- MATRIX mat_fill_type (MATRIX A, int type)

  Fills a matrix to a type.
- int mat_free (MATRIX A)

  Frees a matrix.
- INT_VECTOR int_vec_creat (int len, int type)

  Creates an integer vector.
- INT_VECTOR int_vec_fill (INT_VECTOR A, int val)

  Fills an integer vector with a value.
- INT_VECTOR int_vec_fill_type (INT_VECTOR A, int type)

  Fills an integer vector to a type.
- int int_vec_free (INT_VECTOR A)

  Frees an integer vector.
- INT_VECSTACK int_vecstack_creat (int len)

  Creates an integer vector stack.
- int int_vecstack_free (INT_VECSTACK A)

  Frees an integer vector stack.
- MAT_BAYES_MODEL mat_bayes_model_creat (void)

  Creates a Bayes model.
- int mat_bayes_model_free (MAT_BAYES_MODEL a)

  Frees a Bayes model.
- MAT_PERCEPTRON mat_perceptron_creat (void)

  Creates a perceptron.
- int mat_perceptron_free (MAT_PERCEPTRON a)

  Frees a perceptron.
- MATVEC_DPOINTER matvec_creat (void)

  Creates a matrix-vector pair.
- int matvec_free (MATVEC_DPOINTER a)

  Frees a matrix-vector pair.

- INT_VECTOR int_vec_append (INT_VECTOR a, int i)

  Appends an integer to an integer vector.
- INT_VECTOR int_vec_copy (INT_VECTOR a, INT_VECTOR result)

  Copies an integer vector.
- MATRIX mat_copy (MATRIX A, MATRIX result)

  Copies a matrix.
- MATRIX mat_xcopy (MATRIX A, int si, int ei, int sj, int ej, MATRIX result)

  Copies a sub-matrix.
- MATRIX mat_xjoin (MATRIX A11, MATRIX A12, MATRIX A21, MATRIX A22, MATRIX result)

  Copies a sub-matrix.
- MATRIX mat_rowcopy (MATRIX A, int rowa, int rowb, MATRIX result)

  Copies a row from a matrix.
- MATRIX mat_colcopy (MATRIX A, int cola, int colb, MATRIX result)

  Copies a column from a matrix.
- int mat_fgetmat (MATRIX A, MAT_FILEPOINTER fp)

  Gets matrix data from opened file.
- MATRIX mat_creat_diag (MATRIX diag_vals, MATRIX result)

  Creates a diagonal matrix from a 1-d matrix.

6.6.1 Function Documentation

6.6.1.1 INT_VECTOR int_vec_append (INT_VECTOR *a*, int *i*)

Appends an integer to an integer vector.

Parameters

in	*a*	Input vector
in	*i*	Integer to append

Returns

Appended vector

6.6.1.2 INT_VECTOR int_vec_copy (INT_VECTOR *a*, INT_VECTOR *result*)

Copies an integer vector.

Parameters

in	*a*	Input vector
in	*result*	Vector to store the result

Returns

Output vector

6.6.1.3 INT_VECTOR int_vec_creat (int *len*, int *type*)

Creates an integer vector.

Parameters

in	*len*	Length of the vector
in	*type*	Definition type (UNDEFINED/ZERO_INT_VECTOR/ONES_INT_VECTOR/SERIES_INT_VECTOR)

Returns

Output vector

6.6.1.4 INT_VECTOR int_vec_fill (INT_VECTOR *A*, int *val*)

Fills an integer vector with a value.

Parameters

in	*A*	Input vector
in	*val*	Value to fill with

Returns

Filled vector

6.6.1.5 INT_VECTOR int_vec_fill_type (INT_VECTOR *A*, int *type*)

Fills an integer vector to a type.

Parameters

in	*A*	Input vector
in	*type*	Definition type (UNDEFINED/ZERO_INT_VECTOR/ONES_INT_VECTOR/SERIES_INT_VECTOR)

Returns

Filled vector

6.6.1.6 int int_vec_free (INT_VECTOR *A*)

Frees an integer vector.

Parameters

in	*A*	Input vector

Returns

Success

6.6.1.7 INT_VECSTACK int_vecstack_creat (int *len*)

Creates an integer vector stack.

Parameters

in	*len*	Length of the stack

Returns

Output vector stack

6.6.1.8 int int_vecstack_free (INT_VECSTACK *A*)

Frees an integer vector stack.

Parameters

in	*A*	Input vector stack

Returns

Success

6.6.1.9 MAT_BAYES_MODEL mat_bayes_model_creat (void)

Creates a Bayes model.

Returns

Output Bayes model

6.6.1.10 int mat_bayes_model_free (MAT_BAYES_MODEL *a*)

Frees a Bayes model.

Parameters

in	*a*	Input Bayes model

Returns

Success

6.6.1.11 MATRIX mat_colcopy (MATRIX *A*, int *cola*, int *colb*, MATRIX *result*)

Copies a column from a matrix.

Parameters

in	*A*	Input matrix
in	*cola*	Source column
in	*colb*	Destination column
in	*result*	Matrix to store the result

Returns

Copied matrix

6.6.1.12 MATRIX mat_copy (MATRIX *A*, MATRIX *result*)

Copies a matrix.

Parameters

in	*A*	Input matrix
in	*result*	Matrix to store the result

Returns

Output matrix

6.6.1.13 MATRIX mat_creat (int *row*, int *col*, int *type*)

Creates a matrix.

Parameters

in	*row*	Number of rows
in	*col*	Number of columns
in	*type*	Definition type (UNDEFINED/ZERO_MATRIX/UNIT_MATRIX/ONES_MATRIX)

Returns

Output matrix

6.6.1.14 MATRIX mat_creat_diag (MATRIX *diag_vals*, MATRIX *result*)

Creates a diagonal matrix from a 1-d matrix.

Parameters

in	*diag_vals*	Input 1-d diagonal value matrix
in	*result*	Matrix to store the result

Returns

Diagonal matrix

6.6.1.15 int mat_fgetmat (MATRIX *A*, MAT_FILEPOINTER *fp*)

Gets matrix data from opened file.

Parameters

in	*A*	Matrix to store the data
in	*fp*	Pointer to opened file

Returns

Number of elements copied

6.6.1.16 MATRIX mat_fill (MATRIX *A*, mtype *val*)

Fills a matrix with a value.

Parameters

in	*A*	Input matrix
in	*val*	Value to fill with

Returns

Filled matrix

6.6.1.17 MATRIX mat_fill_type (MATRIX *A*, int *type*)

Fills a matrix to a type.

Parameters

in	*A*	Input matrix
in	*type*	Fill type (UNDEFINED/ZERO_MATRIX/UNIT_MATRIX/ONES_MATRIX)

Returns

Filled matrix

6.6.1.18 int mat_free (MATRIX *A*)

Frees a matrix.

Parameters

in	*A*	Input matrix

Returns

Success

6.6.1.19 MAT_PERCEPTRON mat_perceptron_creat (void)

Creates a perceptron.

Returns

Output perceptron

6.6.1.20 int mat_perceptron_free (MAT_PERCEPTRON *a*)

Frees a perceptron.

Parameters

in	*a*	Input perceptron

Returns

Success

6.6.1.21 MATRIX mat_rowcopy (MATRIX *A*, int *rowa*, int *rowb*, MATRIX *result*)

Copies a row from a matrix.

Parameters

in	*A*	Input matrix
in	*rowa*	Source row
in	*rowb*	Destination row
in	*result*	Matrix to store the result

Returns

Copied matrix

6.6.1.22 MATRIX mat_xcopy (MATRIX *A*, int *si*, int *ei*, int *sj*, int *ej*, MATRIX *result*)

Copies a sub-matrix.

Parameters

in	*A*	Input matrix
in	*si*	Start of first index, s_i
in	*ei*	End of first index, e_i
in	*sj*	Start of second index, s_j
in	*ej*	End of second index, e_j
in	*result*	Matrix to store the result

Returns

Extracted matrix $A_{s_i:e_i,s_j:e_j}$

6.6.1.23 MATRIX mat_xjoin (MATRIX *A11*, MATRIX *A12*, MATRIX *A21*, MATRIX *A22*, MATRIX *result*)

Copies a sub-matrix.

Parameters

in	*A11*	Input matrix, A_{11}
in	*A12*	Input matrix, A_{12}
in	*A21*	Input matrix, A_{21}
in	*A22*	Input matrix, A_{22}
in	*result*	Matrix to store the result

Returns

Block matrix $\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$

6.6.1.24 MATSTACK matstack_append (MATSTACK *s*, MATRIX *A*)

Appends a matrix to a matrix stack.

Parameters

`in`	*s*	Input matrix stack
`in`	*A*	Input matrix to append

Returns

Output matrix stack

6.6.1.25 MATSTACK matstack_creat (int *len*)

Creates a matrix stack.

Parameters

`in`	*len*	Length of the stack

Returns

Output matrix stack

6.6.1.26 int matstack_free (MATSTACK *A*)

Frees a matrix stack.

Parameters

`in`	*A*	Input matrix stack

Returns

Success

6.6.1.27 MATVEC_DPOINTER matvec_creat (void)

Creates a matrix-vector pair.

Returns

Output matrix-vector pair

6.6.1.28 int matvec_free (MATVEC_DPOINTER *a*)

Frees a matrix-vector pair.

Parameters

in	*a*	Input matrix-vector pair

Returns

Success

6.7 matdatastruct.c File Reference

Functions

- MAT_TREE mat_bs_make_null (void)
- MAT_TREE mat_bs_free (MAT_TREE T)
- MAT_TREE mat_bs_find (mtype x, MAT_TREE T)
- MAT_TREE mat_bs_find_min (MAT_TREE T)
- MAT_TREE mat_bs_find_max (MAT_TREE T)
- MAT_TREE mat_bs_insert (mtype x, MAT_TREE T)
- MAT_TREE mat_bs_delete (mtype x, MAT_TREE T)
- int mat_bs_inorder (MAT_TREE T, int index, mtype $**$p_ordered)
- MAT_INT_STACK mat_int_stack_creat (void)
- int mat_int_stack_free (MAT_INT_STACK s)
- void mat_int_stack_push (MAT_INT_STACK s, int value)
- int mat_int_stack_pop (MAT_INT_STACK s)
- int mat_int_stack_is_empty (MAT_INT_STACK s)
- MAT_MTYPE_STACK mat_mtype_stack_creat (void)
- int mat_mtype_stack_free (MAT_MTYPE_STACK s)
- void mat_mtype_stack_push (MAT_MTYPE_STACK s, mtype value)
- mtype mat_mtype_stack_pop (MAT_MTYPE_STACK s)
- int mat_mtype_stack_is_empty (MAT_MTYPE_STACK s)
- MAT_INT_QUEUE mat_int_queue_creat (void)
- int mat_int_queue_free (MAT_INT_QUEUE s)
- void mat_int_queue_enqueue (MAT_INT_QUEUE s, int value)
- int mat_int_queue_dequeue (MAT_INT_QUEUE s)
- int mat_int_queue_is_empty (MAT_INT_QUEUE s)
- MAT_MTYPE_QUEUE mat_mtype_queue_creat (void)
- int mat_mtype_queue_free (MAT_MTYPE_QUEUE s)
- void mat_mtype_queue_enqueue (MAT_MTYPE_QUEUE s, mtype value)
- mtype mat_mtype_queue_dequeue (MAT_MTYPE_QUEUE s)
- int mat_mtype_queue_is_empty (MAT_MTYPE_QUEUE s)
- MAT_INT_PRIORITYQUEUE mat_int_priorityqueue_creat (int type)
- void mat_int_priorityqueue_enqueue (MAT_INT_PRIORITYQUEUE H, int data, int priority)
- mat_intpqnode mat_int_priorityqueue_dequeue (MAT_INT_PRIORITYQUEUE H)
- int mat_int_priorityqueue_free (MAT_INT_PRIORITYQUEUE H)
- int mat_int_priorityqueue_update (MAT_INT_PRIORITYQUEUE H, int data, int priority, int type)
- int mat_int_priorityqueue_is_empty (MAT_INT_PRIORITYQUEUE H)
- MAT_MTYPE_PRIORITYQUEUE mat_mtype_priorityqueue_creat (int type)
- void mat_mtype_priorityqueue_enqueue (MAT_MTYPE_PRIORITYQUEUE H, mtype data, mtype priority)
- mat_mtypepqnode mat_mtype_priorityqueue_dequeue (MAT_MTYPE_PRIORITYQUEUE H)
- int mat_mtype_priorityqueue_free (MAT_MTYPE_PRIORITYQUEUE H)
- int mat_mtype_priorityqueue_update (MAT_MTYPE_PRIORITYQUEUE H, mtype data, mtype priority, int type)
- int mat_mtype_priorityqueue_is_empty (MAT_MTYPE_PRIORITYQUEUE H)

6.7.1 Function Documentation

6.7.1.1 MAT_TREE mat_bs_delete (mtype *x,* MAT_TREE *T*)

6.7.1.2 MAT_TREE mat_bs_find (mtype *x,* MAT_TREE *T*)

6.7.1.3 MAT_TREE mat_bs_find_max (MAT_TREE *T*)

6.7.1.4 MAT_TREE mat_bs_find_min (MAT_TREE *T*)

6.7.1.5 MAT_TREE mat_bs_free (MAT_TREE *T*)

6.7.1.6 int mat_bs_inorder (MAT_TREE *T,* int *index,* mtype ** *p_ordered*)

6.7.1.7 MAT_TREE mat_bs_insert (mtype *x,* MAT_TREE *T*)

6.7.1.8 MAT_TREE mat_bs_make_null (void)

6.7.1.9 MAT_INT_PRIORITYQUEUE mat_int_priorityqueue_creat (int *type*)

6.7.1.10 mat_intpqnode mat_int_priorityqueue_dequeue (MAT_INT_PRIORITYQUEUE *H*)

6.7.1.11 void mat_int_priorityqueue_enqueue (MAT_INT_PRIORITYQUEUE *H,* int *data,* int *priority*)

6.7.1.12 int mat_int_priorityqueue_free (MAT_INT_PRIORITYQUEUE *H*)

6.7.1.13 int mat_int_priorityqueue_is_empty (MAT_INT_PRIORITYQUEUE *H*)

6.7.1.14 int mat_int_priorityqueue_update (MAT_INT_PRIORITYQUEUE *H,* int *data,* int *priority,* int *type*)

6.7.1.15 MAT_INT_QUEUE mat_int_queue_creat (void)

6.7.1.16 int mat_int_queue_dequeue (MAT_INT_QUEUE *s*)

6.7.1.17 void mat_int_queue_enqueue (MAT_INT_QUEUE *s,* int *value*)

6.7.1.18 int mat_int_queue_free (MAT_INT_QUEUE *s*)

6.7.1.19 int mat_int_queue_is_empty (MAT_INT_QUEUE *s*)

6.7.1.20 MAT_INT_STACK mat_int_stack_creat (void)

6.7.1.21 int mat_int_stack_free (MAT_INT_STACK *s*)

6.7.1.22 int mat_int_stack_is_empty (MAT_INT_STACK *s*)

6.7.1.23 int mat_int_stack_pop (MAT_INT_STACK *s*)

6.7.1.24 void mat_int_stack_push (MAT_INT_STACK *s,* int *value*)

6.7.1.25 MAT_MTYPE_PRIORITYQUEUE mat_mtype_priorityqueue_creat (int *type*)

6.7.1.26 mat_mtypepqnode mat_mtype_priorityqueue_dequeue (MAT_MTYPE_PRIORITYQUEUE *H*)

6.7.1.27 void mat_mtype_priorityqueue_enqueue (MAT_MTYPE_PRIORITYQUEUE *H,* mtype *data,* mtype *priority*)

6.7.1.28 int mat_mtype_priorityqueue_free (MAT_MTYPE_PRIORITYQUEUE *H*)

6.7.1.29 int mat_mtype_priorityqueue_is_empty (MAT_MTYPE_PRIORITYQUEUE *H*)

6.7.1.30 int mat_mtype_priorityqueue_update (MAT_MTYPE_PRIORITYQUEUE *H*, mtype *data*, mtype *priority*, int *type*)

6.7.1.31 MAT_MTYPE_QUEUE mat_mtype_queue_creat (void)

6.7.1.32 mtype mat_mtype_queue_dequeue (MAT_MTYPE_QUEUE *s*)

6.7.1.33 void mat_mtype_queue_enqueue (MAT_MTYPE_QUEUE *s*, mtype *value*)

6.7.1.34 int mat_mtype_queue_free (MAT_MTYPE_QUEUE *s*)

6.7.1.35 int mat_mtype_queue_is_empty (MAT_MTYPE_QUEUE *s*)

6.7.1.36 MAT_MTYPE_STACK mat_mtype_stack_creat (void)

6.7.1.37 int mat_mtype_stack_free (MAT_MTYPE_STACK *s*)

6.7.1.38 int mat_mtype_stack_is_empty (MAT_MTYPE_STACK *s*)

6.7.1.39 mtype mat_mtype_stack_pop (MAT_MTYPE_STACK *s*)

6.7.1.40 void mat_mtype_stack_push (MAT_MTYPE_STACK *s*, mtype *value*)

6.8 matdet.c File Reference

Functions

- mtype mat_minor (MATRIX A, int i, int j)

  Computes a minor of a matrix.
- mtype mat_cofact (MATRIX A, int i, int j)

  Computes a cofactor of a matrix.
- mtype mat_det (MATRIX A)

  Computes the determinant of a matrix.

6.8.1 Function Documentation

6.8.1.1 mtype mat_cofact (MATRIX *A*, int *i*, int *j*)

Computes a cofactor of a matrix.

Parameters

in	*A*	Input matrix
in	*i*	Row index
in	*j*	Column index

Returns

Cofactor C_{ij}

6.8.1.2 mtype mat_det (MATRIX *A*)

Computes the determinant of a matrix.

Parameters

in	*A*	Input matrix

Returns

$\det(A)$

6.8.1.3 mtype mat_minor (MATRIX *A*, int *i*, int *j*)

Computes a minor of a matrix.

Parameters

in	*A*	Input matrix
in	*i*	Row index
in	*j*	Column index

Returns

Minor M_{ij}

6.9 matdiv.c File Reference

Functions

- MATRIX mat_div_dot (MATRIX A, MATRIX B, MATRIX result)

  Computes element-wise matrix division.
- MATRIX mat_divs (MATRIX A, mtype s, MATRIX result)

  Divides a matrix by a scalar.
- INT_VECTOR int_vec_div (INT_VECTOR A, INT_VECTOR B, INT_VECTOR result)

  Computes element-wise integer vector division.
- INT_VECTOR int_vec_divs (INT_VECTOR A, int x, INT_VECTOR result)

  Divides an integer vector by a scalar.

6.9.1 Function Documentation

6.9.1.1 INT_VECTOR int_vec_div (INT_VECTOR *A*, INT_VECTOR *B*, INT_VECTOR *result*)

Computes element-wise integer vector division.

Parameters

in	*A*	First input vector
in	*B*	Second input vector
in	*result*	Vector to store the result

Returns

$A./B$

6.9.1.2 INT_VECTOR int_vec_divs (INT_VECTOR *A*, int *x*, INT_VECTOR *result*)

Divides an integer vector by a scalar.

Parameters

in	*A*	Input vector
in	*s*	Scalar
in	*result*	Vector to store the result

Returns

$\frac{A}{s}$

6.9.1.3 MATRIX mat_div_dot (MATRIX *A*, MATRIX *B*, MATRIX *result*)

Computes element-wise matrix division.

Parameters

in	*A*	First input matrix
in	*B*	Second input matrix
in	*result*	Matrix to store the result

Returns

$A./B$

6.9.1.4 MATRIX mat_divs (MATRIX *A*, mtype *s*, MATRIX *result*)

Divides a matrix by a scalar.

Parameters

in	*A*	Input matrix
in	*s*	Scalar
in	*result*	Matrix to store the result

Returns

$\frac{A}{s}$

6.10 matdump.c File Reference

Functions

- void mat_dump (MATRIX A)

  Dumps a matrix in the stdout.

- void mat_dumpf (MATRIX A, const char ∗s)

  Dumps a matrix using a given format specifier in the stdout.
- void mat_fdump (MATRIX A, MAT_FILEPOINTER fp)

  Dumps a matrix in an opened file.
- void mat_fdumpf (MATRIX A, const char ∗s, MAT_FILEPOINTER fp)

  Dumps a matrix using a given format specifier in an opened file.
- void int_vec_dump (INT_VECTOR A)

  Dumps an integer vector in the stdout.
- void int_vec_dumpf (INT_VECTOR A, const char ∗s)

  Dumps an integer vector using a given format specifier in the stdout.
- void int_vec_fdump (INT_VECTOR A, MAT_FILEPOINTER fp)

  Dumps an integer vector in an opened file.
- void int_vec_fdumpf (INT_VECTOR A, const char ∗s, MAT_FILEPOINTER fp)

  Dumps an integer vector using a given format specifier in an opened file.

6.10.1 Function Documentation

6.10.1.1 void int_vec_dump (INT_VECTOR *A*)

Dumps an integer vector in the stdout.

Parameters

in	*A*	Input vector

6.10.1.2 void int_vec_dumpf (INT_VECTOR *A,* const char ∗ *s*)

Dumps an integer vector using a given format specifier in the stdout.

Parameters

in	*A*	Input vector
in	*s*	Format specifier

6.10.1.3 void int_vec_fdump (INT_VECTOR *A,* MAT_FILEPOINTER *fp*)

Dumps an integer vector in an opened file.

Parameters

in	*A*	Input vector
in	*fp*	Pointer to an opened file

6.10.1.4 void int_vec_fdumpf (INT_VECTOR *A,* const char ∗ *s,* MAT_FILEPOINTER *fp*)

Dumps an integer vector using a given format specifier in an opened file.

Parameters

in	*A*	Input vector
in	*s*	Format specifier
in	*fp*	Pointer to an opened file

6.10.1.5 void mat_dump (MATRIX *A*)

Dumps a matrix in the stdout.

Parameters

in	*A*	Input matrix

6.10.1.6 void mat_dumpf (MATRIX *A*, const char ∗ *s*)

Dumps a matrix using a given format specifier in the stdout.

Parameters

in	*A*	Input matrix
in	*s*	Format specifier

6.10.1.7 void mat_fdump (MATRIX *A*, MAT_FILEPOINTER *fp*)

Dumps a matrix in an opened file.

Parameters

in	*A*	Input matrix
in	*fp*	Pointer to an opened file

6.10.1.8 void mat_fdumpf (MATRIX *A*, const char ∗ *s*, MAT_FILEPOINTER *fp*)

Dumps a matrix using a given format specifier in an opened file.

Parameters

in	*A*	Input matrix
in	*s*	Format specifier
in	*fp*	Pointer to an opened file

6.11 matdurbn.c File Reference

Functions

- MATRIX mat_durbin (MATRIX R, MATRIX result)

  Runs Levinson-Durbin algorithm.
- MATRIX mat_lsolve_durbin (MATRIX A, MATRIX B, MATRIX result)

  Runs Levinson-Durbin algorithm.
- MATSTACK mat_qr (MATRIX A, MATSTACK qr)

  Computes QR decomposition.

6.11.1 Function Documentation

6.11.1.1 MATRIX mat_durbin (MATRIX *R*, MATRIX *result*)

Runs Levinson-Durbin algorithm.

Parameters

in	*R*	Input n^th correlation matrix $(n+1)\times 1$
in	*result*	Matrix to store the result

Returns

X where $\tilde{R}X = B$, $\tilde{R} = \begin{bmatrix} R[0][0] & R[1][0] & \cdots & R[n-1][0] \\ R[1][0] & R[0][0] & \cdots & R[n-2][0] \\ \vdots & \vdots & \ddots & \vdots \\ R[n-1][0] & R[n-2][0] & \cdots & R[0][0] \end{bmatrix}$ and $B = \begin{bmatrix} R[1][0] & R[2][0] & \cdots & R[n][0] \end{bmatrix}$

6.11.1.2 MATRIX mat_lsolve_durbin (MATRIX *A*, MATRIX *B*, MATRIX *result*)

Runs Levinson-Durbin algorithm.

Parameters

in	*A*	Input correlation matrix $A = \begin{bmatrix} r_0 & r_1 & \cdots & r_{n-1} \\ r_1 & r_0 & \cdots & r_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n-1} & r_{n-2} & \cdots & r_0 \end{bmatrix}$
in	*B*	Input correlation matrix $B = \begin{bmatrix} r_1 \\ r_2 \\ \cdots \\ r_n \end{bmatrix}$
in	*result*	Matrix to store the result

Returns

X where $RX = B$

6.11.1.3 MATSTACK mat_qr (MATRIX *A*, MATSTACK *qr*)

Computes QR decomposition.

Parameters

in	*A*	Input matrix
in	*qr*	Matrix stack to store result

Returns

Output QR Matrix stack

6.12 materr.c File Reference

Functions

- int gen_error (int err_)

  Generates error message for general errors and exits.
- MATRIX mat_error (int err_)

  Generates error message for matrix errors and exits.
- MATSTACK matstack_error (int err_)

Generates error message for matrix stack errors and exits.

- INT_VECTOR int_vec_error (int err_)

  Generates error message for integer vector errors and exits.
- INT_VECSTACK int_vecstack_error (int err_)

  Generates error message for integer vector stack errors and exits.
- int stack_error (int err_)

  Generates error message for stack errors and exits.
- int queue_error (int err_)

  Generates error message for queue errors and exits.
- int pq_error (int err_)

  Generates error message for priority queue errors and exits.
- int graph_error (int err_)

  Generates error message for graph errors and exits.

6.12.1 Function Documentation

6.12.1.1 int gen_error (int *err_*)

Generates error message for general errors and exits.

Parameters

in	*err*	Error type (GEN_NOT_CONVERGED/GEN_FNOTOPEN/ GEN_FNOTGETMAT/GEN_SIZEMISMATCH/GEN_MATH_ERROR/GEN_MALLOC/GEN_NOT_FOUND/GEN_SIZE_ERROR/GEN_BAD_TYPE)

6.12.1.2 int graph_error (int *err_*)

Generates error message for graph errors and exits.

Parameters

in	*err*	Error type (GRAPH_MALLOC/GRAPH_READ/GRAPH_ELSE)

6.12.1.3 INT_VECTOR int_vec_error (int *err_*)

Generates error message for integer vector errors and exits.

Parameters

in	*err*	Error type (INT_VEC_MALLOC/INT_VEC_FNOTOPEN/INT_VEC_FNOTGETINT_VEC/INT_VEC_SIZEMISMATCH)

6.12.1.4 INT_VECSTACK int_vecstack_error (int *err_*)

Generates error message for integer vector stack errors and exits.

Parameters

in	*err*	Error type (INT_VECSTACK_MALLOC/INT_VECSTACK_FNOTOPEN/INT_VECSTACK_FNOTGETINT_VEC/INT_VECSTACK_SIZEMISMATCH)

6.12.1.5 MATRIX mat_error (int *err_*)

Generates error message for matrix errors and exits.

Parameters

in	*err*	Error type (MAT_MALLOC/MAT_FNOTOPEN/MAT_FNOTGETMAT/MAT_SIZEMISMATCH/ MAT_INVERSE_ILL_COND/MAT_INVERSE_NOT_SQUARE/MAT_CHOLESKY_FAILED)

6.12.1.6 MATSTACK matstack_error (int *err_*)

Generates error message for matrix stack errors and exits.

Parameters

in	*err*	Error type (MATSTACK_MALLOC/MATSTACK_FNOTOPEN/MATSTACK_FNOTGETMAT/MATSTACK_SIZEMISMATCH/ MATSTACK_INVERSE_ERROR)

6.12.1.7 int pq_error (int *err_*)

Generates error message for priority queue errors and exits.

Parameters

in	*err*	Error type (PQ_MALLOC/PQ_EMPTY)

6.12.1.8 int queue_error (int *err_*)

Generates error message for queue errors and exits.

Parameters

in	*err*	Error type (QUEUE_MALLOC/QUEUE_EMPTY)

6.12.1.9 int stack_error (int *err_*)

Generates error message for stack errors and exits.

Parameters

in	*err*	Error type (STACK_MALLOC/STACK_EMPTY)

6.13 matfft.c File Reference

Functions

- MATSTACK mat_fft2 (MATSTACK c, int dir, MATSTACK result)

  Computes fast Fourier transform.

6.13.1 Function Documentation

6.13.1.1 MATSTACK mat_fft2 (MATSTACK *c*, int *dir*, MATSTACK *result*)

Computes fast Fourier transform.

Parameters

in	*C*	Complex data matrix stack
in	*dir*	FFT direction (ROWS/COLS)
in	*result*	Matrix stack to store the result

Returns

Transformed matrix stack

6.14 matfilter.c File Reference

Functions

- MATRIX mat_conv2 (MATRIX A, MATRIX mask, MATRIX scratch, MATRIX result)

  Computes 2-D convolution.

6.14.1 Function Documentation

6.14.1.1 MATRIX mat_conv2 (MATRIX *A*, MATRIX *mask*, MATRIX *scratch*, MATRIX *result*)

Computes 2-D convolution.

Parameters

in	*A*	Input matrix
in	*mask*	Input kernel/mask
in	*scratch*	Scratch matrix for temporary calculations
in	*result*	Matrix to store the result

Returns

Convolved output matrix

6.15 matfit.c File Reference

Functions

- MATRIX mat_linear_ls_fit (MATRIX A, MATRIX Y, int deg, MATRIX result)

  Polynomial model using least squares.
- MATRIX mat_least_squares (MATRIX A, MATRIX Y, MATRIX result)
- MATRIX mat_w_least_squares (MATRIX A, MATRIX Y, MATRIX w, MATRIX result)
- MATRIX mat_rob_least_squares (MATRIX A, MATRIX Y, int lossfunc, MATRIX result)
- MATRIX mat_robust_fit (MATRIX A, MATRIX Y, int deg, int lossfunc, MATRIX result)

6.15.1 Function Documentation

6.15.1.1 MATRIX mat_least_squares (MATRIX *A*, MATRIX *Y*, MATRIX *result*)

6.15.1.2 MATRIX mat_linear_ls_fit (MATRIX *A*, MATRIX *Y*, int *deg*, MATRIX *result*)

Polynomial model using least squares.

Parameters

in	*A*	Data matrix $N \times 1$
in	*Y*	Observation matrix $N \times 1$

Returns

6.15.1.3 MATRIX mat_rob_least_squares (MATRIX *A*, MATRIX *Y*, int *lossfunc*, MATRIX *result*)

6.15.1.4 MATRIX mat_robust_fit (MATRIX *A*, MATRIX *Y*, int *deg*, int *lossfunc*, MATRIX *result*)

6.15.1.5 MATRIX mat_w_least_squares (MATRIX *A*, MATRIX *Y*, MATRIX *w*, MATRIX *result*)

6.16 matflip.c File Reference

Functions

- MATRIX mat_fliplr (MATRIX A, MATRIX result)
- MATRIX mat_flipud (MATRIX A, MATRIX result)

6.16.1 Function Documentation

6.16.1.1 MATRIX mat_fliplr (MATRIX *A*, MATRIX *result*)

6.16.1.2 MATRIX mat_flipud (MATRIX *A*, MATRIX *result*)

6.17 matfuncs.c File Reference

Functions

- mtype __mat_addfunc (mtype x, mtype y)

  Computes addition function.
- mtype __mat_subfunc (mtype x, mtype y)

  Computes subtraction function.
- mtype __mat_mulfunc (mtype x, mtype y)

  Computes multiplication function.
- mtype __mat_divfunc (mtype x, mtype y)

  Computes division function.
- mtype __mat_sqrfunc (mtype x)

  Computes square function.
- mtype __mat_sqrtfunc (mtype x)

  Computes square root function.
- mtype __mat_huber_wt (mtype x, mtype k)

  Computes Huber weight function.

- mtype __mat_bisquare_wt (mtype x, mtype k)

  Computes bisquare weight function.
- mtype __mat_arcsinh (mtype x)

  Computes inverse hyperbolic sine function.
- mtype __mat_arccosh (mtype x)

  Computes inverse hyperbolic cosine function.
- mtype __mat_arctanh (mtype x)

  Computes inverse hyperbolic tangent function.
- mtype __mat_logplusone (mtype x)

  Computes logarithm plus one function.
- MATRIX mat_huber_wt (MATRIX A, mtype k, mtype sigma, MATRIX result)

  Computes Huber weight function element-wise on a matrix.
- MATRIX mat_bisquare_wt (MATRIX A, mtype k, mtype sigma, MATRIX result)

  Computes bisquare weight function element-wise on a matrix.
- MATRIX mat_gfunc (MATRIX A, mtype(∗pt2func)(mtype), MATRIX result)

  Computes a given function element-wise on a matrix.

6.17.1 Function Documentation

6.17.1.1 mtype __mat_addfunc (mtype *x*, mtype *y*)

Computes addition function.

Parameters

in	*x*	
in	*y*	

Returns

$x+y$

6.17.1.2 mtype __mat_arccosh (mtype *x*)

Computes inverse hyperbolic cosine function.

Parameters

in	*x*	

Returns

$\cosh^{-1}(x)$

6.17.1.3 mtype __mat_arcsinh (mtype *x*)

Computes inverse hyperbolic sine function.

Parameters

in	*x*	

Returns

$\sinh^{-1}(x)$

6.17.1.4 mtype __mat_arctanh (mtype *x*)

Computes inverse hyperbolic tangent function.

Parameters

in	*x*	

Returns

$\tanh^{-1}(x)$

6.17.1.5 mtype __mat_bisquare_wt (mtype *x,* mtype *k*)

Computes bisquare weight function.

Parameters

in	*x*	
in	*k*	

Returns

$$\begin{cases} \left(1-\left(\frac{x}{k}\right)^2\right)^2, & \text{for } |x| \le k, \\ 0, & \text{otherwise.} \end{cases}$$

6.17.1.6 mtype __mat_divfunc (mtype *x,* mtype *y*)

Computes division function.

Parameters

in	*x*	
in	*y*	

Returns

$\frac{x}{y}$

6.17.1.7 mtype __mat_huber_wt (mtype *x,* mtype *k*)

Computes Huber weight function.

Parameters

in	*x*	
in	*k*	

Returns

$$\begin{cases} 1, & \text{for } |x| \le k, \\ \frac{k}{|x|}, & \text{otherwise.} \end{cases}$$

6.17.1.8 mtype __mat_logplusone (mtype *x*)

Computes logarithm plus one function.

Parameters

in	*x*	

Returns

$\log(1+x)$

6.17.1.9 mtype __mat_mulfunc (mtype *x,* mtype *y*)

Computes multiplication function.

Parameters

in	*x*	
in	*y*	

Returns

xy

6.17.1.10 mtype __mat_sqrfunc (mtype *x*)

Computes square function.

Parameters

in	*x*	

Returns

x^2

6.17.1.11 mtype __mat_sqrtfunc (mtype *x*)

Computes square root function.

Parameters

in	*x*	

Returns

$\sqrt{x}$

6.17.1.12 mtype __mat_subfunc (mtype *x*, mtype *y*)

Computes subtraction function.

Parameters

in	*x*	
in	*y*	

Returns

$x - y$

6.17.1.13 MATRIX mat_bisquare_wt (MATRIX *A*, mtype *k*, mtype *sigma*, MATRIX *result*)

Computes bisquare weight function element-wise on a matrix.

Parameters

in	*A*	Input matrix
in	*k*	Bisquare parameter

Returns

$\mathbf{B}, b_{ij} = f_k(a_{ij})$ where f_k is the biquare weight function

6.17.1.14 MATRIX mat_gfunc (MATRIX *A*, mtype(*)(mtype) *pt2func*, MATRIX *result*)

Computes a given function element-wise on a matrix.

Parameters

in	*A*	Input matrix
in	*f*	Given function

Returns

$\mathbf{B}, b_{ij} = f(a_{ij})$

6.17.1.15 MATRIX mat_huber_wt (MATRIX *A*, mtype *k*, mtype *sigma*, MATRIX *result*)

Computes Huber weight function element-wise on a matrix.

Parameters

in	*A*	Input matrix
in	*k*	Huber parameter

Returns

$\mathbf{B}$, $b_{ij} = f_k(a_{ij})$ where f_k is the Huber weight function

6.18 matgraph.c File Reference

Functions

- MAT_GRAPH mat_graph_creat (void)
- void mat_graph_adjlist (MAT_GRAPH g, int directed, int weighted, MAT_FILEPOINTER fp)
- MAT_GRAPH mat_graph_reverse (MAT_GRAPH g, MAT_GRAPH r)
- void mat_graph_adjm_to_adjl (MAT_GRAPH g, MATRIX a)
- MAT_INT_QUEUE mat_graph_search (MAT_GRAPH g, int connected, int mst)
- void mat_graph_visit (MAT_GRAPH g, int k, int connected, int mst, MAT_INT_PRIORITYQUEUE pq, MAT_INT_QUEUE q)
- void mat_graph_dumpf (MAT_GRAPH g, int mst, MAT_FILEPOINTER fp)
- void mat_graph_dump (MAT_GRAPH g, int mst)

6.18.1 Function Documentation

6.18.1.1 void mat_graph_adjlist (**MAT_GRAPH** *g,* int *directed,* int *weighted,* MAT_FILEPOINTER *fp*)

6.18.1.2 void mat_graph_adjm_to_adjl (**MAT_GRAPH** *g,* **MATRIX** *a*)

6.18.1.3 **MAT_GRAPH** mat_graph_creat (void)

6.18.1.4 void mat_graph_dump (**MAT_GRAPH** *g,* int *mst*)

6.18.1.5 void mat_graph_dumpf (**MAT_GRAPH** *g,* int *mst,* MAT_FILEPOINTER *fp*)

6.18.1.6 **MAT_GRAPH** mat_graph_reverse (**MAT_GRAPH** *g,* **MAT_GRAPH** *r*)

6.18.1.7 **MAT_INT_QUEUE** mat_graph_search (**MAT_GRAPH** *g,* int *connected,* int *mst*)

6.18.1.8 void mat_graph_visit (**MAT_GRAPH** *g,* int *k,* int *connected,* int *mst,* **MAT_INT_PRIORITYQUEUE** *pq,* **MAT_INT_QUEUE** *q*)

6.19 matinnerprod.c File Reference

Functions

- mtype mat_innerprod (MATRIX A, MATRIX B)
- mtype mat_norm_inf (MATRIX A)
- mtype mat_norm_one (MATRIX A)
- mtype mat_norm_p (MATRIX A, mtype p)

6.19.1 Function Documentation

6.19.1.1 mtype mat_innerprod (**MATRIX** *A,* **MATRIX** *B*)

6.19.1.2 mtype mat_norm_inf (**MATRIX** *A*)

6.19.1.3 mtype mat_norm_one (**MATRIX** *A*)

6.19.1.4 mtype mat_norm_p (**MATRIX** *A*, mtype *p*)

6.20 matintegrate.c File Reference

Functions

- mtype mat_int_trapezoid (mtype(∗func)(mtype), int n, mtype lower, mtype upper)

  Computes trapezoid integration.
- mtype mat_int_simpson (mtype(∗func)(mtype), int n, mtype lower, mtype upper)

  Computes Simpson's integration.
- mtype mat_int_qadrat (mtype(∗func)(mtype), mtype lower, mtype upper)

  Computes Gauss quadrature integration.

6.20.1 Function Documentation

6.20.1.1 mtype mat_int_qadrat (mtype(∗)(mtype) *func*, mtype *lower*, mtype *upper*)

Computes Gauss quadrature integration.

Parameters

`in`	*func*	Function $f(\cdot)$ to integrate
`in`	*n*	Number of subdivisions
`in`	*lower*	Lower Limit
`in`	*upper*	Upper Limit

Returns

$\int_a^b f(x)\,dx$

6.20.1.2 mtype mat_int_simpson (mtype(∗)(mtype) *func*, int *n*, mtype *lower*, mtype *upper*)

Computes Simpson's integration.

Parameters

`in`	*func*	Function $f(\cdot)$ to integrate
`in`	*n*	Number of subdivisions
`in`	*lower*	Lower Limit
`in`	*upper*	Upper Limit

Returns

$\int_a^b f(x)\,dx$

6.20.1.3 mtype mat_int_trapezoid (mtype(∗)(mtype) *func*, int *n*, mtype *lower*, mtype *upper*)

Computes trapezoid integration.

Parameters

`in`	*func*	Function $f(\cdot)$ to integrate
`in`	*n*	Number of subdivisions

in	*lower*	Lower Limit
in	*upper*	Upper Limit

Returns

$\int_a^b f(x)\,dx$

6.21 matinv.c File Reference

Functions

- MATRIX mat_inv (MATRIX A, MATRIX result)

  Computes the inverse of a matrix.
- MATRIX mat_reg_inv (MATRIX A, mtype r, MATRIX result)

  Computes the regularized inverse of a matrix.

6.21.1 Function Documentation

6.21.1.1 MATRIX mat_inv (MATRIX *A,* MATRIX *result*)

Computes the inverse of a matrix.

Parameters

in	*A*	Input matrix
in	*result*	Matrix to store the result

Returns

A^{-1}

6.21.1.2 MATRIX mat_reg_inv (MATRIX *A,* mtype *r,* MATRIX *result*)

Computes the regularized inverse of a matrix.

Parameters

in	*A*	Input matrix
in	*r*	Regularizing constant
in	*result*	Matrix to store the result

Returns

$(A+rI)^{-1}$

6.22 matkdtree.c File Reference

Functions

- MAT_KDTREE mat_kdtree_make_tree (MATRIX A, MAT_KDTREE result)

  Creates a k-d tree from a data matrix.

- int mat_kdtree_free (MAT_KDTREE t)

  Frees a k-d tree.
- MATRIX mat_kdtree_nearest (MAT_KDTREE t, MATRIX A, MATRIX result)

  Computes nearest neighbors.
- MATRIX mat_kdtree_k_nearest (MAT_KDTREE t, MATRIX A, int k, MATRIX result)

  Computes k nearest neighbors.

6.22.1 Function Documentation

6.22.1.1 int mat_kdtree_free (MAT_KDTREE *t*)

Frees a k-d tree.

Parameters

in	*t*	Input k-d tree

Returns

Success

6.22.1.2 MATRIX mat_kdtree_k_nearest (MAT_KDTREE *t*, MATRIX *A*, int *k*, MATRIX *result*)

Computes k nearest neighbors.

Parameters

in	*t*	Input k-d tree
in	*A*	Input data matrix of size $d \times N$
in	*k*	Number of neighbors
in	*result*	Matrix to store the result

Returns

Output matrix B with index B[0][j] and squared distance B[1][j] for $j = 1, 2, \cdots, N$

6.22.1.3 MAT_KDTREE mat_kdtree_make_tree (MATRIX *A*, MAT_KDTREE *result*)

Creates a k-d tree from a data matrix.

Parameters

in	*A*	Input data matrix of size $d \times N$
in	*result*	K-d tree to store the result

Returns

Output k-d tree

6.22.1.4 MATRIX mat_kdtree_nearest (MAT_KDTREE *t*, MATRIX *A*, MATRIX *result*)

Computes nearest neighbors.

Parameters

in	*t*	Input k-d tree
in	*A*	Input data matrix of size $d \times N$
in	*result*	Matrix to store the result

Returns

Output matrix B with index B[0][j] and squared distance B[1][j] for $j = 1, 2, \cdots, N$

6.23 matmaxmin.c File Reference

Functions

- MATVEC_DPOINTER mat_max (MATRIX A, int dim)
- MATVEC_DPOINTER mat_min (MATRIX A, int dim)

6.23.1 Function Documentation

6.23.1.1 MATVEC_DPOINTER mat_max (MATRIX *A*, int *dim*)

6.23.1.2 MATVEC_DPOINTER mat_min (MATRIX *A*, int *dim*)

6.24 matmds.c File Reference

Functions

- MATRIX mat_mds (MATRIX d, int dims, int type, MATRIX result)
- MATRIX __mat_mds_metric (MATRIX d, int dims, MATRIX result)
- MATRIX __mat_mds_nonmetric (MATRIX d, int dims, MATRIX result)

6.24.1 Function Documentation

6.24.1.1 MATRIX __mat_mds_metric (MATRIX *d*, int *dims*, MATRIX *result*)

6.24.1.2 MATRIX __mat_mds_nonmetric (MATRIX *d*, int *dims*, MATRIX *result*)

6.24.1.3 MATRIX mat_mds (MATRIX *d*, int *dims*, int *type*, MATRIX *result*)

6.25 matmean.c File Reference

Functions

- mtype mat_mean (MATRIX A)
- MATRIX mat_mean_row (MATRIX A, MATRIX result)
- MATRIX mat_mean_col (MATRIX A, MATRIX result)

6.25.1 Function Documentation

6.25.1.1 mtype mat_mean (MATRIX *A*)

6.25.1.2 MATRIX mat_mean_col (MATRIX *A*, MATRIX *result*)

6.25.1.3 MATRIX mat_mean_row (MATRIX *A*, MATRIX *result*)

6.26 matmisc.c File Reference

Functions

- int mats_isnan (mtype x)

  Checks if scalar is NaN.
- int mats_isinf (mtype x)

  Checks if scalar is infinite.
- void mat_nextline (void)

  Prints nextline to stdout.
- void mat_fnextline (MAT_FILEPOINTER fp)

  Prints nextline to file.
- MATRIX mat_bsxfun (MATRIX A, MATRIX B, mtype(∗func)(mtype, mtype), MATRIX result)

  Computes element-wise binary function for two matrices.
- INT_VECTOR int_vec_permute_vect (int n, int k, INT_VECTOR result)

  Computes a randomly permutation of first k positive integers.
- INT_VECTOR mat_get_sub_vector (INT_VECTOR a, INT_VECTOR indices)

  Extracts sub-vector from an integer vector.
- MATRIX mat_get_sub_matrix_from_rows (MATRIX A, INT_VECTOR indices, MATRIX result)

  Extracts sub-matrix from rows of a matrix.
- MATRIX mat_get_sub_matrix_from_cols (MATRIX A, INT_VECTOR indices, MATRIX result)

  Extracts sub-matrix from columns of a matrix.
- MATRIX mat_calc_dist_sq (MATRIX A, MATRIX d, MATRIX result)

  Computes the Euclidean distances of points from a given point.
- INT_VECTOR mat_find_within_dist (MATRIX A, MATRIX d, mtype range)

  Finds points within a neighborhood.
- MATRIX mat_pick_row (MATRIX A, int r, MATRIX result)

  Picks a row from a matrix.
- MATRIX mat_pick_col (MATRIX A, int c, MATRIX result)

  Picks a column from a matrix.
- void __mat_cart2pol (mtype x, mtype y, mtype ∗rho, mtype ∗th)
- void __mat_pol2cart (mtype rho, mtype th, mtype ∗x, mtype ∗y)
- MATRIX mat_cart2pol (MATRIX A, int dim, MATRIX result)

  Converts Cartesian co-ordinates to polar co-ordinates.
- MATRIX mat_pol2cart (MATRIX A, int dim, MATRIX result)

  Converts polar co-ordinates to Cartesian co-ordinates.
- INT_VECTOR int_vec_unique (INT_VECTOR a)

  Extract only the unique integers from an integer vector.

6.26.1 Function Documentation

6.26.1.1 void __mat_cart2pol (mtype *x*, mtype *y*, mtype ∗ *rho*, mtype ∗ *th*)

6.26.1.2 void __mat_pol2cart (mtype *rho*, mtype *th*, mtype ∗ *x*, mtype ∗ *y*)

6.26.1.3 INT_VECTOR int_vec_permute_vect (int *n*, int *k*, INT_VECTOR *result*)

Computes a randomly permutation of first k positive integers.

Parameters

in	*n*	Number of random permutations to make
in	*k*	Integer upto which it will consider
in	*result*	Vector to store the result

Returns

Permuted vector

6.26.1.4 INT_VECTOR int_vec_unique (INT_VECTOR *a*)

Extract only the unique integers from an integer vector.

Parameters

in	*a*	Input vector

Returns

Unique vector

6.26.1.5 MATRIX mat_bsxfun (MATRIX *A,* MATRIX *B,* mtype(∗)(mtype, mtype) *func,* MATRIX *result*)

Computes element-wise binary function for two matrices.

Parameters

in	*A*	First matrix
in	*B*	Second matrix
in	*func*	Pointer to the function
in	*result*	Matrix to store the result

Returns

Output matrix

6.26.1.6 MATRIX mat_calc_dist_sq (MATRIX *A,* MATRIX *d,* MATRIX *result*)

Computes the Euclidean distances of points from a given point.

Parameters

in	*A*	Points matrix (d x N)
in	*d*	Matrix point from which the distance to be computed (d x 1)
in	*result*	Matrix to store the result

Returns

Euclidean distance matrix

6.26.1.7 MATRIX mat_cart2pol (MATRIX *A,* int *dim,* MATRIX *result*)

Converts Cartesian co-ordinates to polar co-ordinates.

Parameters

in	*A*	Input matrix
in	*dim*	Data order ROWS/COLS

Returns

Polar co-ordinate matrix

6.26.1.8 INT_VECTOR mat_find_within_dist (MATRIX *A*, MATRIX *d*, mtype *range*)

Finds points within a neighborhood.

Parameters

in	*A*	Points matrix (d x N)
in	*d*	Matrix point from which the distance to be computed (d x 1)
in	*range*	Radius to search within

Returns

Indices Vector

6.26.1.9 void mat_fnextline (MAT_FILEPOINTER *fp*)

Prints nextline to file.

Parameters

in	*fp*	Pointer to opened file

6.26.1.10 MATRIX mat_get_sub_matrix_from_cols (MATRIX *A*, INT_VECTOR *indices*, MATRIX *result*)

Extracts sub-matrix from columns of a matrix.

Parameters

in	*A*	Input matrix
in	*indices*	Columns to extract
in	*result*	Matrix to store the result

Returns

Extracted matrix

6.26.1.11 MATRIX mat_get_sub_matrix_from_rows (MATRIX *A*, INT_VECTOR *indices*, MATRIX *result*)

Extracts sub-matrix from rows of a matrix.

Parameters

in	*A*	Input matrix
in	*indices*	Rows to extract
in	*result*	Matrix to store the result

Returns

Extracted matrix

6.26.1.12 INT_VECTOR mat_get_sub_vector (INT_VECTOR *a*, INT_VECTOR *indices*)

Extracts sub-vector from an integer vector.

Parameters

in	*a*	Input vector
in	*indices*	Indices to extracted

Returns

Extracted vector

6.26.1.13 void mat_nextline (void)

Prints nextline to stdout.

6.26.1.14 MATRIX mat_pick_col (MATRIX *A*, int *c*, MATRIX *result*)

Picks a column from a matrix.

Parameters

in	*A*	Input matrix
in	*r*	Column index
in	*result*	Matrix to store the result

Returns

Column matrix

6.26.1.15 MATRIX mat_pick_row (MATRIX *A*, int *r*, MATRIX *result*)

Picks a row from a matrix.

Parameters

in	*A*	Input matrix
in	*r*	Row index
in	*result*	Matrix to store the result

Returns

Row matrix

6.26.1.16 MATRIX mat_pol2cart (MATRIX *A*, int *dim*, MATRIX *result*)

Converts polar co-ordinates to Cartesian co-ordinates.

Parameters

in	*A*	Input matrix
in	*dim*	Data order ROWS/COLS

Returns

Cartesian co-ordinate matrix

6.26.1.17 int mats_isinf (mtype *x*)

Checks if scalar is infinite.

Parameters

in	*x*	Input scalar

Returns

Zero/non-zero

6.26.1.18 int mats_isnan (mtype *x*)

Checks if scalar is NaN.

Parameters

in	*x*	Input scalar

Returns

Zero/non-zero

6.27 matmul.c File Reference

Functions

- MATRIX mat_mul (MATRIX A, MATRIX B, MATRIX result)
- MATRIX mat_mul_fast (MATRIX A, MATRIX B, MATRIX result)
- MATRIX mat_muls (MATRIX A, mtype s, MATRIX B)
- MATRIX mat_mul_dot (MATRIX A, MATRIX B, MATRIX C)
- mtype mat_diagmul (MATRIX A)
- INT_VECTOR int_vec_mul (INT_VECTOR A, INT_VECTOR B, INT_VECTOR result)
- INT_VECTOR int_vec_muls (INT_VECTOR A, int x, INT_VECTOR result)

6.27.1 Function Documentation

6.27.1.1 INT_VECTOR int_vec_mul (INT_VECTOR *A,* INT_VECTOR *B,* INT_VECTOR *result*)

6.27.1.2 INT_VECTOR int_vec_muls (INT_VECTOR *A,* int *x,* INT_VECTOR *result*)

6.27.1.3 mtype mat_diagmul (MATRIX *A*)

6.27.1.4 MATRIX mat_mul (MATRIX *A*, MATRIX *B*, MATRIX *result*)

6.27.1.5 MATRIX mat_mul_dot (MATRIX *A*, MATRIX *B*, MATRIX *C*)

6.27.1.6 MATRIX mat_mul_fast (MATRIX *A*, MATRIX *B*, MATRIX *result*)

6.27.1.7 MATRIX mat_muls (MATRIX *A*, mtype *s*, MATRIX *B*)

6.28 matpca.c File Reference

Functions

- MATSTACK mat_pca (MATRIX data, int pca_type)
- MATSTACK mat_eig_sym (MATRIX symmat, MATSTACK result)
- MATSTACK mat_corcol (MATRIX data)
- MATSTACK mat_covcol (MATRIX data)
- MATRIX mat_scpcol (MATRIX data)
- void mat_tred2 (MATRIX a, MATRIX d, MATRIX e)
- void mat_tqli (MATRIX d, MATRIX e, MATRIX z)

6.28.1 Function Documentation

6.28.1.1 MATSTACK mat_corcol (MATRIX *data*)

6.28.1.2 MATSTACK mat_covcol (MATRIX *data*)

6.28.1.3 MATSTACK mat_eig_sym (MATRIX *symmat*, MATSTACK *result*)

6.28.1.4 MATSTACK mat_pca (MATRIX *data*, int *pca_type*)

6.28.1.5 MATRIX mat_scpcol (MATRIX *data*)

6.28.1.6 void mat_tqli (MATRIX *d*, MATRIX *e*, MATRIX *z*)

6.28.1.7 void mat_tred2 (MATRIX *a*, MATRIX *d*, MATRIX *e*)

6.29 matpinv.c File Reference

Functions

- MATRIX mat_pinv (MATRIX A, MATRIX result)

  Computes pseudo-inverse of a matrix.
- MATRIX mat_wpinv (MATRIX A, MATRIX w, MATRIX result)

  Computes weighted pseudo-inverse of a matrix.

6.29.1 Function Documentation

6.29.1.1 MATRIX mat_pinv (MATRIX *A*, MATRIX *result*)

Computes pseudo-inverse of a matrix.

Parameters

in	*A*	Input matrix
in	*result*	Matrix to store the result

Returns

$(A^T A)^{-1} A^T$

6.29.1.2 MATRIX mat_wpinv (MATRIX *A*, MATRIX *w*, MATRIX *result*)

Computes weighted pseudo-inverse of a matrix.

Parameters

in	*A*	Input matrix
in	*w*	Weight matrix
in	*result*	Matrix to store the result

Returns

$(A^T W A)^{-1} A^T W$

6.30 matpoly.c File Reference

Functions

- MATRIX mat_poly_eval (MATRIX A, mtype x, int dir, MATRIX result)

  Evaluates polynomial at a point.
- MATRIX mat_poly_diff (MATRIX A, int dir, MATRIX result)

  Computes derivative polynomial of a polynomial.
- MATRIX mat_poly_diff_eval (MATRIX A, mtype x, int dir, MATRIX result)

  Evaluates derivative polynomial at a point.
- MATRIX mat_poly_add (MATRIX A, MATRIX B, MATRIX result)

  Adds two polynomials.
- MATRIX mat_poly_mul (MATRIX A, MATRIX B, MATRIX result)

  Multiplies two polynomials.
- MATSTACK mat_poly_div (MATRIX A, MATRIX B, MATSTACK result)

  Divides two polynomials.
- MATRIX mat_poly_scale (MATRIX A, mtype s, MATRIX result)

  Multiplies a polynomial with a scalar.
- MATRIX mat_poly_shift (MATRIX A, int s, MATRIX result)

  Shifts a polynomial.
- void mat_cheby_init ()

  Initializes the Chebyshev polynomial series.
- void mat_legendre_init ()

  Initializes the Legendre polynomial series.
- void mat_binom_init ()

  Initializes the binomial series.
- MATRIX mat_cheby (int n)

  Computes the n^{th} Chebyshev polynomial.
- MATRIX mat_legendre (int n)

Computes the n^{th} Legendre polynomial.

- mtype mat_binom (int n, int k)

  Computes a binomial co-efficient.
- MATRIX mat_cheby_coeffs_to_poly (MATRIX coeffs, MATRIX result)

  Converts Chebyshev co-efficients to a single polynomial.
- MATRIX mat_cheby_approx (mtype($*$f)(mtype), mtype a, mtype b, int n, MATRIX result)

  Approximates a function using Chebyshev polynomials.

Variables

- MATSTACK mat_cheby_series_table
- MATSTACK mat_legendre_series_table
- MATSTACK mat_binom_series_table

6.30.1 Function Documentation

6.30.1.1 mtype mat_binom (int *n*, int *k*)

Computes a binomial co-efficient.

Parameters

in	*n*	1^{st} argument
in	*k*	2^{nd} argument

Returns

$\binom{n}{k}$

6.30.1.2 void mat_binom_init ( )

Initializes the binomial series.

6.30.1.3 MATRIX mat_cheby (int *n*)

Computes the n^{th} Chebyshev polynomial.

Parameters

in	*n*	Polynomial series index

Returns

Output polynomial matrix

6.30.1.4 MATRIX mat_cheby_approx (mtype($*$)(mtype) *f*, mtype *a*, mtype *b*, int *n*, MATRIX *result*)

Approximates a function using Chebyshev polynomials.

Parameters

in	*f*	Function to approximate
in	*a*	Lower limit of domain of the function
in	*b*	Upper limit of domain of the function
in	*n*	Degree of the approximate polynomial
in	*result*	Matrix to store the result

Returns

Approximate polynomial matrix

6.30.1.5 MATRIX mat_cheby_coeffs_to_poly (MATRIX *coeffs*, MATRIX *result*)

Converts Chebyshev co-efficients to a single polynomial.

Parameters

in	*coeffs*	Chebyshev polynomial co-efficient matrix
in	*result*	Matrix to store the result

Returns

Polynomial matrix

6.30.1.6 void mat_cheby_init ( )

Initializes the Chebyshev polynomial series.

6.30.1.7 MATRIX mat_legendre (int *n*)

Computes the n^{th} Legendre polynomial.

Parameters

in	*n*	Polynomial series index

Returns

Output polynomial matrix

6.30.1.8 void mat_legendre_init ( )

Initializes the Legendre polynomial series.

6.30.1.9 MATRIX mat_poly_add (MATRIX *A*, MATRIX *B*, MATRIX *result*)

Adds two polynomials.

Parameters

in	*A*	First input polynomial matrix
in	*B*	Second input polynomial matrix
in	*result*	Matrix to store the result

Returns

Output matrix

6.30.1.10 MATRIX mat_poly_diff (MATRIX *A,* int *dir,* MATRIX *result*)

Computes derivative polynomial of a polynomial.

Parameters

in	*A*	Input polynomial matrix
in	*dir*	Direction (ROWS/COLS)
in	*result*	Matrix to store the result

Returns

Output matrix

6.30.1.11 MATRIX mat_poly_diff_eval (MATRIX *A,* mtype *x,* int *dir,* MATRIX *result*)

Evaluates derivative polynomial at a point.

Parameters

in	*A*	Input polynomial matrix
in	*x*	Value at which to evaluate the derivative
in	*dir*	Direction (ROWS/COLS)
in	*result*	Matrix to store the result

Returns

Output matrix

6.30.1.12 MATSTACK mat_poly_div (MATRIX *A,* MATRIX *B,* MATSTACK *result*)

Divides two polynomials.

Parameters

in	*A*	First input polynomial matrix
in	*B*	Second input polynomial matrix
in	*result*	Matrix to store the result

Returns

Output matrix

6.30.1.13 MATRIX mat_poly_eval (MATRIX *A,* mtype *x,* int *dir,* MATRIX *result*)

Evaluates polynomial at a point.

Parameters

in	*A*	Input polynomial matrix
in	*x*	Value at which to evaluate
in	*dir*	Direction (ROWS/COLS)
in	*result*	Matrix to store the result

Returns

Output matrix

6.30.1.14 MATRIX mat_poly_mul (MATRIX *A*, MATRIX *B*, MATRIX *result*)

Multiplies two polynomials.

Parameters

in	*a*	First input polynomial matrix
in	*b*	Second input polynomial matrix
in	*result*	Matrix to store the result

Returns

Output matrix

6.30.1.15 MATRIX mat_poly_scale (MATRIX *A*, mtype *s*, MATRIX *result*)

Multiplies a polynomial with a scalar.

Parameters

in	*A*	Input polynomial matrix
in	*s*	Scalar
in	*result*	Matrix to store the result

Returns

Output matrix

6.30.1.16 MATRIX mat_poly_shift (MATRIX *A*, int *s*, MATRIX *result*)

Shifts a polynomial.

Parameters

in	*A*	Input polynomial matrix
in	*s*	Scalar shift
in	*result*	Matrix to store the result

Returns

Output matrix

6.30.2 Variable Documentation

6.30.2.1 MATSTACK mat_binom_series_table

6.30.2.2 MATSTACK mat_cheby_series_table

6.30.2.3 MATSTACK mat_legendre_series_table

6.31 matprec.c File Reference

Functions

- MAT_BAYES_MODEL mat_bayes_classifier_train (MATRIX data, INT_VECTOR labels)
- INT_VECTOR mat_bayes_classifier_test (MATRIX data, MAT_BAYES_MODEL b_model)
- MAT_PERCEPTRON mat_perceptron_train (MATRIX data, INT_VECTOR labels, int num_of_iterations)
- MAT_PERCEPTRON mat_perceptron_train_ (MATRIX data1, MATRIX data2, MAT_PERCEPTRON p_model, int class_num)
- INT_VECTOR mat_perceptron_test (MATRIX data, MAT_PERCEPTRON p_model)
- MATVEC_DPOINTER mat_kmeans (MATRIX data, int k, int iters, MATVEC_DPOINTER result)

6.31.1 Function Documentation

6.31.1.1 INT_VECTOR mat_bayes_classifier_test (MATRIX *data,* MAT_BAYES_MODEL *b_model*)

6.31.1.2 MAT_BAYES_MODEL mat_bayes_classifier_train (MATRIX *data,* INT_VECTOR *labels*)

6.31.1.3 MATVEC_DPOINTER mat_kmeans (MATRIX *data,* int *k,* int *iters,* MATVEC_DPOINTER *result*)

6.31.1.4 INT_VECTOR mat_perceptron_test (MATRIX *data,* MAT_PERCEPTRON *p_model*)

6.31.1.5 MAT_PERCEPTRON mat_perceptron_train (MATRIX *data,* INT_VECTOR *labels,* int *num_of_iterations*)

6.31.1.6 MAT_PERCEPTRON mat_perceptron_train_ (MATRIX *data1,* MATRIX *data2,* MAT_PERCEPTRON *p_model,* int *class_num*)

6.32 matpursuit.c File Reference

Functions

- MATSTACK mat_omp (MATRIX A, MATRIX b, int k, mtype tol, MATSTACK result)

6.32.1 Function Documentation

6.32.1.1 MATSTACK mat_omp (MATRIX *A,* MATRIX *b,* int *k,* mtype *tol,* MATSTACK *result*)

6.33 matrand.c File Reference

Functions

- MATRIX mat_rand (int n, int m, MATRIX result)
- MATRIX mat_randn (int n, int m, MATRIX result)
- MATRIX mat_randexp (int n, int m, mtype mu, MATRIX result)

- MATRIX mat_randfun (int n, int m, mtype(∗fun)(mtype), mtype xmin, mtype xmax, MATRIX result)
- void mat_set_seed (int seed)
- mtype __mat_randfun (mtype(∗fun)(mtype), mtype xmin, mtype xmax)
- mtype __mat_rand (void)
- mtype __mat_randn (void)
- mtype __mat_randexp (mtype mu)
- MATRIX mat_randperm (int m, int n, MATRIX result)
- MATRIX mat_randperm_n (int n, MATRIX result)
- INT_VECTOR int_vec_randperm (int n, INT_VECTOR result)

Variables

- unsigned int MAT_SEED = 0
- int MAT_SET_SEED = 0

6.33.1 Function Documentation

6.33.1.1 mtype __mat_rand (void)

6.33.1.2 mtype __mat_randexp (mtype *mu*)

6.33.1.3 mtype __mat_randfun (mtype(∗)(mtype) *fun,* mtype *xmin,* mtype *xmax*)

6.33.1.4 mtype __mat_randn (void)

6.33.1.5 INT_VECTOR int_vec_randperm (int *n,* INT_VECTOR *result*)

6.33.1.6 MATRIX mat_rand (int *n,* int *m,* MATRIX *result*)

6.33.1.7 MATRIX mat_randexp (int *n,* int *m,* mtype *mu,* MATRIX *result*)

6.33.1.8 MATRIX mat_randfun (int *n,* int *m,* mtype(∗)(mtype) *fun,* mtype *xmin,* mtype *xmax,* MATRIX *result*)

6.33.1.9 MATRIX mat_randn (int *n,* int *m,* MATRIX *result*)

6.33.1.10 MATRIX mat_randperm (int *m,* int *n,* MATRIX *result*)

6.33.1.11 MATRIX mat_randperm_n (int *n,* MATRIX *result*)

6.33.1.12 void mat_set_seed (int *seed*)

6.33.2 Variable Documentation

6.33.2.1 unsigned int MAT_SEED = 0

6.33.2.2 int MAT_SET_SEED = 0

6.34 matrix.c File Reference

6.35 matrix.h File Reference

Data Structures

- struct mat_int_stack

Integer Stack Structure.

- struct mat_mtype_stack

  Mtype Stack Structure.
- struct mat_qintnode

  Integer Queue Node Structure.
- struct mat_int_queue

  Integer Queue Structure.
- struct mat_qmtypenode

  Mtype Queue Node Structure.
- struct mat_mtype_queue

  Mtype Queue Structure.
- struct mat_intpqnode

  Integer Priority Queue Node Structure.
- struct mat_int_priorityqueue

  Integer Priority Queue Structure.
- struct mat_mtypepqnode

  Mtype Priority Queue Node Structure.
- struct mat_mtype_priorityqueue

  Mtype Priority Queue Structure.
- struct mat_tree_node

  Search Tree Node Structure.
- struct mat_bayes_model

  Bayes Classifier Model Structure.
- struct mat_perceptron

  Perceptron Classifier Model Structure.
- struct mat_gnode

  Graph Node Structure.
- struct mat_graph

  Graph Structure.
- struct mat_kdnode
- struct mat_kdtree

Typedefs

- typedef struct mat_int_stack mat_int_stack

  Integer Stack Structure.
- typedef mat_int_stack ∗ MAT_INT_STACK
- typedef struct mat_mtype_stack mat_mtype_stack

  Mtype Stack Structure.
- typedef mat_mtype_stack ∗ MAT_MTYPE_STACK
- typedef struct mat_qintnode mat_qintnode

  Integer Queue Node Structure.
- typedef mat_qintnode ∗ MAT_QINTNODE
- typedef struct mat_int_queue mat_int_queue

  Integer Queue Structure.
- typedef mat_int_queue ∗ MAT_INT_QUEUE
- typedef struct mat_qmtypenode mat_qmtypenode

  Mtype Queue Node Structure.
- typedef mat_qmtypenode ∗ MAT_QMTYPENODE
- typedef struct mat_mtype_queue mat_mtype_queue

  Mtype Queue Structure.

- typedef mat_mtype_queue ∗ MAT_MTYPE_QUEUE
- typedef struct mat_intpqnode mat_intpqnode

  Integer Priority Queue Node Structure.
- typedef mat_intpqnode ∗ MAT_INTPQNODE
- typedef struct
  mat_int_priorityqueue mat_int_priorityqueue

  Integer Priority Queue Structure.
- typedef mat_int_priorityqueue ∗ MAT_INT_PRIORITYQUEUE
- typedef struct mat_mtypepqnode mat_mtypepqnode

  Mtype Priority Queue Node Structure.
- typedef mat_mtypepqnode ∗ MAT_MTYPEPQNODE
- typedef struct
  mat_mtype_priorityqueue mat_mtype_priorityqueue

  Mtype Priority Queue Structure.
- typedef mat_mtype_priorityqueue ∗ MAT_MTYPE_PRIORITYQUEUE
- typedef struct mat_tree_node mat_tree_node

  Search Tree Node Structure.
- typedef mat_tree_node ∗ MAT_TREE_NODE
- typedef mat_tree_node ∗ MAT_TREE
- typedef int ∗ INT_VECTOR
- typedef mtype ∗∗ MATRIX
- typedef INT_VECTOR ∗ INT_VECSTACK
- typedef MATRIX ∗ MATSTACK
- typedef void ∗∗ MATVEC_DPOINTER
- typedef struct mat_bayes_model mat_bayes_model

  Bayes Classifier Model Structure.
- typedef mat_bayes_model ∗ MAT_BAYES_MODEL
- typedef struct mat_perceptron mat_perceptron

  Perceptron Classifier Model Structure.
- typedef mat_perceptron ∗ MAT_PERCEPTRON
- typedef struct mat_gnode mat_gnode

  Graph Node Structure.
- typedef mat_gnode ∗ MAT_GNODE
- typedef struct mat_graph mat_graph

  Graph Structure.
- typedef mat_graph ∗ MAT_GRAPH
- typedef struct mat_kdnode mat_kdnode
- typedef mat_kdnode ∗ MAT_KDNODE
- typedef struct mat_kdtree mat_kdtree
- typedef mat_kdtree ∗ MAT_KDTREE

Functions

- int mats_isnan (mtype x)

  Checks if scalar is NaN.
- int mats_isinf (mtype x)

  Checks if scalar is infinite.
- INT_VECTOR __int_vec_creat (int len)
- INT_VECTOR int_vec_creat (int len, int type)

  Creates an integer vector.
- INT_VECTOR int_vec_fill (INT_VECTOR A, int val)

  Fills an integer vector with a value.

- INT_VECTOR int_vec_fill_type (INT_VECTOR A, int type)

    Fills an integer vector to a type.
- int int_vec_free (INT_VECTOR A)

    Frees an integer vector.
- INT_VECSTACK __int_vecstack_creat (int len)
- INT_VECSTACK int_vecstack_creat (int len)

    Creates an integer vector stack.
- int int_vecstack_free (INT_VECSTACK A)

    Frees an integer vector stack.
- MATRIX __mat_creat (int r, int c)
- MATRIX mat_creat (int r, int c, int type)

    Creates a matrix.
- MATRIX mat_creat_diag (MATRIX diag_vals, MATRIX result)

    Creates a diagonal matrix from a 1-d matrix.
- MATRIX mat_fill (MATRIX A, mtype val)

    Fills a matrix with a value.
- MATRIX mat_fill_type (MATRIX A, int type)

    Fills a matrix to a type.
- int mat_free (MATRIX A)

    Frees a matrix.
- MATSTACK matstack_creat (int len)

    Creates a matrix stack.
- MATSTACK __matstack_creat (int len)
- int matstack_free (MATSTACK A)

    Frees a matrix stack.
- MATSTACK matstack_append (MATSTACK s, MATRIX a)

    Appends a matrix to a matrix stack.
- MATVEC_DPOINTER matvec_creat (void)

    Creates a matrix-vector pair.
- int matvec_free (MATVEC_DPOINTER a)

    Frees a matrix-vector pair.
- MATRIX mat_copy (MATRIX A, MATRIX result)

    Copies a matrix.
- MATRIX mat_xcopy (MATRIX A, int si, int ei, int sj, int ej, MATRIX result)

    Copies a sub-matrix.
- MATRIX mat_xjoin (MATRIX A11, MATRIX A12, MATRIX A21, MATRIX A22, MATRIX result)

    Copies a sub-matrix.
- MATRIX mat_rowcopy (MATRIX A, int rowa, int rowb, MATRIX result)

    Copies a row from a matrix.
- MATRIX mat_colcopy (MATRIX A, int cola, int colb, MATRIX result)

    Copies a column from a matrix.
- int mat_fgetmat (MATRIX A, MAT_FILEPOINTER fp)

    Gets matrix data from opened file.
- void mat_dump (MATRIX A)

    Dumps a matrix in the stdout.
- void mat_dumpf (MATRIX A, const char ∗s)

    Dumps a matrix using a given format specifier in the stdout.
- void mat_fdump (MATRIX A, MAT_FILEPOINTER fp)

    Dumps a matrix in an opened file.
- void mat_fdumpf (MATRIX A, const char ∗s, MAT_FILEPOINTER fp)

    Dumps a matrix using a given format specifier in an opened file.

- void int_vec_dump (INT_VECTOR a)

  Dumps an integer vector in the stdout.
- void int_vec_dumpf (INT_VECTOR a, const char ∗s)

  Dumps an integer vector using a given format specifier in the stdout.
- void int_vec_fdump (INT_VECTOR a, MAT_FILEPOINTER fp)

  Dumps an integer vector in an opened file.
- void int_vec_fdumpf (INT_VECTOR a, const char ∗s, MAT_FILEPOINTER fp)

  Dumps an integer vector using a given format specifier in an opened file.
- INT_VECTOR int_vec_copy (INT_VECTOR a, INT_VECTOR result)

  Copies an integer vector.
- INT_VECTOR int_vec_unique (INT_VECTOR a)

  Extract only the unique integers from an integer vector.
- INT_VECTOR int_vec_append (INT_VECTOR a, int i)

  Appends an integer to an integer vector.
- INT_VECTOR int_vec_find (INT_VECTOR a, int rel_type, int n)
- INT_VECTOR int_vec_concat (INT_VECTOR a, INT_VECTOR b, INT_VECTOR result)

  Concatenates two integer vectors.
- INT_VECTOR mat_get_sub_vector (INT_VECTOR a, INT_VECTOR indices)

  Extracts sub-vector from an integer vector.
- int gen_error (int err_)

  Generates error message for general errors and exits.
- INT_VECTOR int_vec_error (int err_)

  Generates error message for integer vector errors and exits.
- INT_VECSTACK int_vecstack_error (int err_)

  Generates error message for integer vector stack errors and exits.
- MATRIX mat_error (int err_)

  Generates error message for matrix errors and exits.
- MATSTACK matstack_error (int err_)

  Generates error message for matrix stack errors and exits.
- int stack_error (int err_)

  Generates error message for stack errors and exits.
- int queue_error (int err_)

  Generates error message for queue errors and exits.
- int pq_error (int err_)

  Generates error message for priority queue errors and exits.
- int graph_error (int err_)

  Generates error message for graph errors and exits.
- mtype mat_mean (MATRIX A)
- MATRIX mat_mean_row (MATRIX A, MATRIX result)
- MATRIX mat_mean_col (MATRIX A, MATRIX result)
- mtype mat_sum (MATRIX A)
- MATRIX mat_sum_row (MATRIX A, MATRIX result)
- MATRIX mat_sum_col (MATRIX A, MATRIX result)
- MATRIX mat_abs (MATRIX A, MATRIX result)

  Computes absolute value of matrix.
- INT_VECTOR int_vec_add (INT_VECTOR A, INT_VECTOR B, INT_VECTOR result)

  Adds two integer vectors.
- INT_VECTOR int_vec_adds (INT_VECTOR A, int s, INT_VECTOR result)

  Adds an integer to an integer vector.
- INT_VECTOR int_vec_sub (INT_VECTOR A, INT_VECTOR B, INT_VECTOR result)
- INT_VECTOR int_vec_subs (INT_VECTOR A, int s, INT_VECTOR result)

- INT_VECTOR int_vec_mul (INT_VECTOR A, INT_VECTOR B, INT_VECTOR result)
- INT_VECTOR int_vec_muls (INT_VECTOR A, int s, INT_VECTOR result)
- INT_VECTOR int_vec_div (INT_VECTOR A, INT_VECTOR B, INT_VECTOR result)

  Computes element-wise integer vector division.
- INT_VECTOR int_vec_divs (INT_VECTOR A, int s, INT_VECTOR result)

  Divides an integer vector by a scalar.
- MATRIX mat_add (MATRIX A, MATRIX B, MATRIX result)

  Adds two matrices.
- MATRIX mat_adds (MATRIX A, mtype s, MATRIX result)

  Adds a scalar to a matrix.
- MATRIX mat_sub (MATRIX A, MATRIX B, MATRIX result)
- MATRIX mat_subs (MATRIX A, mtype s, MATRIX result)
- MATRIX mat_mul (MATRIX A, MATRIX B, MATRIX result)
- MATRIX mat_mul_fast (MATRIX A, MATRIX B, MATRIX result)
- MATRIX mat_mul_dot (MATRIX A, MATRIX B, MATRIX result)
- MATRIX mat_muls (MATRIX A, mtype s, MATRIX result)
- MATRIX mat_div_dot (MATRIX A, MATRIX B, MATRIX result)

  Computes element-wise matrix division.
- MATRIX mat_divs (MATRIX A, mtype s, MATRIX result)

  Divides a matrix by a scalar.
- mtype mat_innerprod (MATRIX A, MATRIX B)
- mtype mat_norm_inf (MATRIX A)
- mtype mat_norm_one (MATRIX A)
- mtype mat_norm_p (MATRIX A, mtype p)
- mtype mat_diagmul (MATRIX A)
- MATRIX mat_tran (MATRIX A, MATRIX result)

  Computes the transpose of a matrix.
- MATRIX mat_inv (MATRIX A, MATRIX result)

  Computes the inverse of a matrix.
- MATRIX mat_pinv (MATRIX A, MATRIX result)

  Computes pseudo-inverse of a matrix.
- MATRIX mat_wpinv (MATRIX A, MATRIX w, MATRIX result)

  Computes weighted pseudo-inverse of a matrix.
- MATRIX mat_reg_inv (MATRIX A, mtype r, MATRIX result)

  Computes the regularized inverse of a matrix.
- MATRIX mat_symtoeplz (MATRIX R, MATRIX result)
- int mat_lu (MATRIX A, MATRIX P)
- void mat_backsubs1 (MATRIX A, MATRIX B, MATRIX C, MATRIX P, int xcol)
- MATRIX mat_lsolve (MATRIX A, MATRIX b, MATRIX result)
- MATRIX mat_cholesky (MATRIX A, MATRIX result)

  Computes Cholesky factor of a matrix.
- MATRIX mat_conjgrad (MATRIX A, MATRIX b, MATRIX x0, mtype tol, int miters, MATRIX result)

  Solves a linear system with conjugate gradients method.
- MATRIX mat_submat (MATRIX A, int i, int j, MATRIX result)

  Deletes a row and a column of a matrix.
- mtype mat_cofact (MATRIX A, int i, int j)

  Computes a cofactor of a matrix.
- mtype mat_det (MATRIX A)

  Computes the determinant of a matrix.
- mtype mat_minor (MATRIX A, int i, int j)

  Computes a minor of a matrix.
- MATSTACK mat_qr (MATRIX A, MATSTACK qr)

Computes QR decomposition.
- MATRIX mat_durbin (MATRIX R, MATRIX result)

  Runs Levinson-Durbin algorithm.
- MATRIX mat_lsolve_durbin (MATRIX A, MATRIX B, MATRIX result)

  Runs Levinson-Durbin algorithm.
- mtype mat_median (MATRIX A)

  Computes the median of elements of a given matrix.
- mtype mat_order_statistic (MATRIX A, int k)

  Computes the k^{th} order statistic of elements of a given matrix.
- void __mat_quicksort (MATRIX A, int l, int r, int offset, MATRIX ind)
- MATSTACK mat_qsort (MATRIX A, int dim, MATSTACK result)

  Sorts elements of a given matrix.
- MATVEC_DPOINTER mat_max (MATRIX A, int dim)
- MATVEC_DPOINTER mat_min (MATRIX A, int dim)
- MATRIX mat_rand (int r, int c, MATRIX result)
- MATRIX mat_randn (int r, int c, MATRIX result)
- MATRIX mat_randexp (int r, int c, mtype mu, MATRIX result)
- INT_VECTOR int_vec_permute_vect (int n, int k, INT_VECTOR result)

  Computes a randomly permutation of first k positive integers.
- MATRIX mat_randfun (int r, int c, mtype(∗fun)(mtype), mtype xmin, mtype xmax, MATRIX result)
- void mat_set_seed (int seed)
- mtype __mat_randfun (mtype(∗fun)(mtype), mtype xmin, mtype xmax)
- mtype __mat_rand (void)
- mtype __mat_randn (void)
- mtype __mat_randexp (mtype mu)
- MATRIX mat_randperm (int m, int n, MATRIX result)
- MATRIX mat_randperm_n (int n, MATRIX result)
- INT_VECTOR int_vec_randperm (int n, INT_VECTOR result)
- MATRIX mat_least_squares (MATRIX A, MATRIX Y, MATRIX result)
- MATRIX mat_w_least_squares (MATRIX A, MATRIX Y, MATRIX w, MATRIX result)
- MATRIX mat_rob_least_squares (MATRIX A, MATRIX Y, int lossfunc, MATRIX result)
- MATRIX mat_linear_ls_fit (MATRIX A, MATRIX Y, int deg, MATRIX result)

  Polynomial model using least squares.
- MATRIX mat_robust_fit (MATRIX A, MATRIX Y, int deg, int lossfunc, MATRIX result)
- MATRIX mat_concat (MATRIX A, MATRIX B, int dim)

  Concatenates two matrices.
- MATRIX mat_get_sub_matrix_from_rows (MATRIX A, INT_VECTOR indices, MATRIX result)

  Extracts sub-matrix from rows of a matrix.
- MATRIX mat_get_sub_matrix_from_cols (MATRIX A, INT_VECTOR indices, MATRIX result)

  Extracts sub-matrix from columns of a matrix.
- MATRIX mat_pick_row (MATRIX A, int r, MATRIX result)

  Picks a row from a matrix.
- MATRIX mat_pick_col (MATRIX A, int c, MATRIX result)

  Picks a column from a matrix.
- INT_VECSTACK mat_find (MATRIX A, int rel_type, mtype x)
- MATRIX mat_fliplr (MATRIX A, MATRIX result)
- MATRIX mat_flipud (MATRIX A, MATRIX result)
- MATRIX mat_calc_dist_sq (MATRIX A, MATRIX d, MATRIX result)

  Computes the Euclidean distances of points from a given point.
- INT_VECTOR mat_find_within_dist (MATRIX A, MATRIX d, mtype range)

  Finds points within a neighborhood.
- void __mat_cart2pol (mtype x, mtype y, mtype ∗rho, mtype ∗th)

- void __mat_pol2cart (mtype rho, mtype th, mtype ∗x, mtype ∗y)
- MATRIX mat_cart2pol (MATRIX A, int dim, MATRIX result)

  Converts Cartesian co-ordinates to polar co-ordinates.
- MATRIX mat_pol2cart (MATRIX A, int dim, MATRIX result)

  Converts polar co-ordinates to Cartesian co-ordinates.
- mtype __mat_addfunc (mtype x, mtype y)

  Computes addition function.
- mtype __mat_subfunc (mtype x, mtype y)

  Computes subtraction function.
- mtype __mat_mulfunc (mtype x, mtype y)

  Computes multiplication function.
- mtype __mat_divfunc (mtype x, mtype y)

  Computes division function.
- mtype __mat_sqrfunc (mtype x)

  Computes square function.
- mtype __mat_sqrtfunc (mtype x)

  Computes square root function.
- mtype __mat_huber_wt (mtype x, mtype k)

  Computes Huber weight function.
- mtype __mat_bisquare_wt (mtype x, mtype k)

  Computes bisquare weight function.
- mtype __mat_logplusone (mtype x)

  Computes logarithm plus one function.
- mtype __mat_arcsinh (mtype x)

  Computes inverse hyperbolic sine function.
- mtype __mat_arccosh (mtype x)

  Computes inverse hyperbolic cosine function.
- mtype __mat_arctanh (mtype x)

  Computes inverse hyperbolic tangent function.
- MATRIX mat_bisquare_wt (MATRIX A, mtype k, mtype sigma, MATRIX result)

  Computes bisquare weight function element-wise on a matrix.
- MATRIX mat_huber_wt (MATRIX A, mtype k, mtype sigma, MATRIX result)

  Computes Huber weight function element-wise on a matrix.
- MATRIX mat_gfunc (MATRIX A, mtype(∗pt2func)(mtype), MATRIX result)

  Computes a given function element-wise on a matrix.
- MATRIX mat_bsxfun (MATRIX A, MATRIX B, mtype(∗func)(mtype, mtype), MATRIX result)

  Computes element-wise binary function for two matrices.
- MATSTACK mat_corcol (MATRIX data)
- MATSTACK mat_covcol (MATRIX data)
- MATRIX mat_scpcol (MATRIX data)
- void mat_tred2 (MATRIX a, MATRIX d, MATRIX e)
- void mat_tqli (MATRIX d, MATRIX e, MATRIX z)
- MATSTACK mat_pca (MATRIX data, int pca_type)
- MATSTACK mat_eig_sym (MATRIX symmat, MATSTACK result)
- void mat_nextline (void)

  Prints nextline to stdout.
- void mat_fnextline (MAT_FILEPOINTER fp)

  Prints nextline to file.
- int __mat_powerof2 (int width, int ∗m, int ∗twopm)
- MATSTACK mat_fft2 (MATSTACK c, int dir, MATSTACK result)

  Computes fast Fourier transform.

- int __mat_fft (int dir, int m, mtype ∗x, mtype ∗y)
- MATRIX mat_conv2 (MATRIX A, MATRIX mask, MATRIX scratch, MATRIX result)

  Computes 2-D convolution.
- INT_VECTOR mat_2int_vec (MATRIX a)

  Converts a matrix to an integer vector.
- MATRIX int_vec2_mat (INT_VECTOR a, int dir)

  Converts an integer vector to a matrix.
- MATRIX mat_vectorize (MATRIX a, MATRIX result)

  Reshapes a matrix to a vector.
- MATRIX mat_vectorize_tr (MATRIX a, MATRIX result)

  Reshapes transpose of a matrix to a vector.
- mtype mat_int_trapezoid (mtype(∗func)(mtype), int n, mtype lower, mtype upper)

  Computes trapezoid integration.
- mtype mat_int_simpson (mtype(∗func)(mtype), int n, mtype lower, mtype upper)

  Computes Simpson's integration.
- mtype __mat_lint (mtype ∗x, mtype(∗func)(mtype), mtype x0, mtype xn, mtype f0, mtype f2, mtype f3, mtype f5, mtype f6, mtype f7, mtype f9, mtype fl4, mtype hmin, mtype hmax, mtype re, mtype ae)
- mtype mat_int_qadrat (mtype(∗func)(mtype), mtype lower, mtype upper)

  Computes Gauss quadrature integration.
- MATRIX mat_poly_eval (MATRIX A, mtype x, int dir, MATRIX result)

  Evaluates polynomial at a point.
- MATRIX mat_poly_diff (MATRIX A, int dir, MATRIX result)

  Computes derivative polynomial of a polynomial.
- MATRIX mat_poly_diff_eval (MATRIX A, mtype x, int dir, MATRIX result)

  Evaluates derivative polynomial at a point.
- MATRIX mat_poly_add (MATRIX A, MATRIX B, MATRIX result)

  Adds two polynomials.
- MATRIX mat_poly_mul (MATRIX A, MATRIX B, MATRIX result)

  Multiplies two polynomials.
- MATSTACK mat_poly_div (MATRIX A, MATRIX B, MATSTACK result)

  Divides two polynomials.
- MATRIX mat_poly_scale (MATRIX A, mtype s, MATRIX result)

  Multiplies a polynomial with a scalar.
- MATRIX mat_poly_shift (MATRIX A, int s, MATRIX result)

  Shifts a polynomial.
- void mat_cheby_init ()

  Initializes the Chebyshev polynomial series.
- void mat_legendre_init ()

  Initializes the Legendre polynomial series.
- void mat_binom_init ()

  Initializes the binomial series.
- MATRIX mat_cheby (int n)

  Computes the n^{th} Chebyshev polynomial.
- MATRIX mat_legendre (int n)

  Computes the n^{th} Legendre polynomial.
- mtype mat_binom (int n, int k)

  Computes a binomial co-efficient.
- MATRIX mat_cheby_coeffs_to_poly (MATRIX coeffs, MATRIX result)

  Converts Chebyshev co-efficients to a single polynomial.
- MATRIX mat_cheby_approx (mtype(∗f)(mtype), mtype a, mtype b, int n, MATRIX result)

  Approximates a function using Chebyshev polynomials.

- MAT_BAYES_MODEL mat_bayes_model_creat (void)

  Creates a Bayes model.
- int mat_bayes_model_free (MAT_BAYES_MODEL a)

  Frees a Bayes model.
- MAT_PERCEPTRON mat_perceptron_creat (void)

  Creates a perceptron.
- int mat_perceptron_free (MAT_PERCEPTRON a)

  Frees a perceptron.
- MAT_BAYES_MODEL mat_bayes_classifier_train (MATRIX data, INT_VECTOR labels)
- INT_VECTOR mat_bayes_classifier_test (MATRIX data, MAT_BAYES_MODEL b_model)
- MAT_PERCEPTRON mat_perceptron_train (MATRIX data, INT_VECTOR labels, int num_of_iterations)
- INT_VECTOR mat_perceptron_test (MATRIX data, MAT_PERCEPTRON p_model)
- MAT_PERCEPTRON mat_perceptron_train_ (MATRIX data1, MATRIX data2, MAT_PERCEPTRON p_model, int class_num)
- MATVEC_DPOINTER mat_kmeans (MATRIX data, int k, int iters, MATVEC_DPOINTER result)
- MAT_TREE mat_bs_make_null (void)
- MAT_TREE mat_bs_free (MAT_TREE T)
- MAT_TREE mat_bs_find (mtype x, MAT_TREE T)
- MAT_TREE mat_bs_find_min (MAT_TREE T)
- MAT_TREE mat_bs_find_max (MAT_TREE T)
- MAT_TREE mat_bs_insert (mtype x, MAT_TREE T)
- MAT_TREE mat_bs_delete (mtype x, MAT_TREE T)
- int mat_bs_inorder (MAT_TREE T, int index, mtype ∗∗ordered)
- int gen_gt (mtype a)
- int gen_lt (mtype a)
- int gen_eq (mtype a)
- mtype gen_abs_ceil (mtype a)
- int mat_isnumeric (MAT_FILEPOINTER fp)

  Checks if current word in an opened file is numeric or not.
- int mat_go_next_word (MAT_FILEPOINTER fp)

  Moves to next word in an opened file.
- int mat_count_words_in_line (MAT_FILEPOINTER fp, int ∗count)

  Count words in current line in an opened file.
- int mat_read_word (MAT_FILEPOINTER fp, char ∗c_word)

  Reads current word from an opened file.
- MATRIX mat_dlmread (const char ∗fname)

  Reads a matrix from a file.
- void mat_dlmwrite (const char ∗fname, MATRIX A)

  Writes a matrix to a file.
- void mat_tic (void)

  Starts stopwatch timer.
- double mat_toc (void)

  Computes elapsed time from last start of timer.
- void mat_toc_print (void)

  Computes and prints elapsed time from last start of timer on the stdout.
- MAT_INT_STACK mat_int_stack_creat (void)
- int mat_int_stack_free (MAT_INT_STACK s)
- void mat_int_stack_push (MAT_INT_STACK s, int value)
- int mat_int_stack_pop (MAT_INT_STACK s)
- int mat_int_stack_is_empty (MAT_INT_STACK s)
- MAT_MTYPE_STACK mat_mtype_stack_creat (void)
- int mat_mtype_stack_free (MAT_MTYPE_STACK s)
- void mat_mtype_stack_push (MAT_MTYPE_STACK s, mtype value)

- mtype mat_mtype_stack_pop (MAT_MTYPE_STACK s)
- int mat_mtype_stack_is_empty (MAT_MTYPE_STACK s)
- MAT_INT_QUEUE mat_int_queue_creat (void)
- int mat_int_queue_free (MAT_INT_QUEUE s)
- void mat_int_queue_enqueue (MAT_INT_QUEUE s, int value)
- int mat_int_queue_dequeue (MAT_INT_QUEUE s)
- int mat_int_queue_is_empty (MAT_INT_QUEUE s)
- MAT_MTYPE_QUEUE mat_mtype_queue_creat (void)
- int mat_mtype_queue_free (MAT_MTYPE_QUEUE s)
- void mat_mtype_queue_enqueue (MAT_MTYPE_QUEUE s, mtype value)
- mtype mat_mtype_queue_dequeue (MAT_MTYPE_QUEUE s)
- int mat_mtype_queue_is_empty (MAT_MTYPE_QUEUE s)
- MAT_INT_PRIORITYQUEUE mat_int_priorityqueue_creat (int type)
- void mat_int_priorityqueue_enqueue (MAT_INT_PRIORITYQUEUE H, int data, int priority)
- mat_intpqnode mat_int_priorityqueue_dequeue (MAT_INT_PRIORITYQUEUE H)
- int mat_int_priorityqueue_free (MAT_INT_PRIORITYQUEUE H)
- int mat_int_priorityqueue_update (MAT_INT_PRIORITYQUEUE H, int data, int priority, int type)
- int mat_int_priorityqueue_is_empty (MAT_INT_PRIORITYQUEUE H)
- MAT_MTYPE_PRIORITYQUEUE mat_mtype_priorityqueue_creat (int type)
- void mat_mtype_priorityqueue_enqueue (MAT_MTYPE_PRIORITYQUEUE H, mtype data, mtype priority)
- mat_mtypepqnode mat_mtype_priorityqueue_dequeue (MAT_MTYPE_PRIORITYQUEUE H)
- int mat_mtype_priorityqueue_free (MAT_MTYPE_PRIORITYQUEUE H)
- int mat_mtype_priorityqueue_update (MAT_MTYPE_PRIORITYQUEUE H, mtype data, mtype priority, int type)
- int mat_mtype_priorityqueue_is_empty (MAT_MTYPE_PRIORITYQUEUE H)
- MATRIX mat_mds (MATRIX d, int dims, int type, MATRIX result)
- MATRIX __mat_mds_metric (MATRIX d, int dims, MATRIX result)
- MATRIX __mat_mds_nonmetric (MATRIX d, int dims, MATRIX result)
- MAT_GRAPH mat_graph_creat (void)
- void mat_graph_adjlist (MAT_GRAPH g, int directed, int weighted, MAT_FILEPOINTER fp)
- MAT_INT_QUEUE mat_graph_search (MAT_GRAPH g, int connected, int mst)
- void mat_graph_visit (MAT_GRAPH g, int k, int connected, int mst, MAT_INT_PRIORITYQUEUE pq, MAT_INT_QUEUE q)
- void mat_graph_dumpf (MAT_GRAPH g, int mst, MAT_FILEPOINTER fp)
- void mat_graph_dump (MAT_GRAPH g, int mst)
- void mat_graph_adjm_to_adjl (MAT_GRAPH g, MATRIX a)
- MAT_GRAPH mat_graph_reverse (MAT_GRAPH g, MAT_GRAPH r)
- MAT_KDTREE mat_kdtree_make_tree (MATRIX A, MAT_KDTREE result)

  Creates a k-d tree from a data matrix.
- int mat_kdtree_free (MAT_KDTREE t)

  Frees a k-d tree.
- MATRIX mat_kdtree_nearest (MAT_KDTREE t, MATRIX A, MATRIX result)

  Computes nearest neighbors.
- MATRIX mat_kdtree_k_nearest (MAT_KDTREE t, MATRIX A, int k, MATRIX result)

  Computes k nearest neighbors.
- MAT_KDNODE __mat_kdtree_make_tree (MAT_KDNODE t, int len, int i, int dim)
- MAT_KDNODE __mat_kd_find_median (MAT_KDNODE kd_start, MAT_KDNODE kd_end, int idx)
- void __mat_kdtree_nearest (MAT_KDNODE root, MAT_KDNODE nd, int i, int dim, MAT_KDNODE ∗best, mtype ∗best_dist)
- void __mat_kdtree_k_nearest (MAT_KDNODE root, MAT_KDNODE nd, int i, int dim, MAT_MTYPE_PRIORITYQUEUE pq, MATRIX bmax, MATRIX bmin)
- MATSTACK mat_omp (MATRIX A, MATRIX b, int k, mtype tol, MATSTACK result)

Variables

- clock_t MAT_CLOCK_TIME
- unsigned int MAT_SEED
- int MAT_SET_SEED
- MATSTACK mat_cheby_series_table
- MATSTACK mat_legendre_series_table
- MATSTACK mat_binom_series_table

6.35.1 Typedef Documentation

6.35.1.1 typedef INT_VECTOR∗ INT_VECSTACK

Integer Vector Stack

6.35.1.2 typedef int∗ INT_VECTOR

Integer Vector

6.35.1.3 typedef struct mat_bayes_model mat_bayes_model

Bayes Classifier Model Structure.

Bayes Classifier Model

6.35.1.4 typedef mat_bayes_model∗ MAT_BAYES_MODEL

Bayes Classifier Model Pointer

6.35.1.5 typedef struct mat_gnode mat_gnode

Graph Node Structure.

Graph Node

6.35.1.6 typedef mat_gnode∗ MAT_GNODE

Graph Node Pointer

6.35.1.7 typedef struct mat_graph mat_graph

Graph Structure.

6.35.1.8 typedef mat_graph∗ MAT_GRAPH

6.35.1.9 typedef struct mat_int_priorityqueue mat_int_priorityqueue

Integer Priority Queue Structure.

Integer Priority Queue

6.35.1.10 typedef mat_int_priorityqueue∗ MAT_INT_PRIORITYQUEUE

Integer Priority Queue Pointer

6.35.1.11 typedef struct mat_int_queue mat_int_queue

Integer Queue Structure.

Integer Queue

6.35.1.12 typedef mat_int_queue∗ MAT_INT_QUEUE

Integer Queue Pointer

6.35.1.13 typedef struct mat_int_stack mat_int_stack

Integer Stack Structure.

Integer Stack

6.35.1.14 typedef mat_int_stack∗ MAT_INT_STACK

Integer Stack Pointer

6.35.1.15 typedef struct mat_intpqnode mat_intpqnode

Integer Priority Queue Node Structure.

Integer Priority Queue Node

6.35.1.16 typedef mat_intpqnode∗ MAT_INTPQNODE

Integer Priority Queue Node Pointer

6.35.1.17 typedef struct mat_kdnode mat_kdnode

6.35.1.18 typedef mat_kdnode∗ MAT_KDNODE

6.35.1.19 typedef struct mat_kdtree mat_kdtree

6.35.1.20 typedef mat_kdtree∗ MAT_KDTREE

6.35.1.21 typedef struct mat_mtype_priorityqueue mat_mtype_priorityqueue

Mtype Priority Queue Structure.

Mtype Priority Queue

6.35.1.22 typedef mat_mtype_priorityqueue∗ MAT_MTYPE_PRIORITYQUEUE

Mtype Priority Queue Pointer

6.35.1.23 typedef struct mat_mtype_queue mat_mtype_queue

Mtype Queue Structure.

Mtype Queue

6.35.1.24 typedef mat_mtype_queue∗ MAT_MTYPE_QUEUE

Mtype Queue Pointer

6.35.1.25 typedef struct mat_mtype_stack mat_mtype_stack

Mtype Stack Structure.

Mtype Stack

6.35.1.26 typedef mat_mtype_stack∗ MAT_MTYPE_STACK

Mtype Stack Pointer

6.35.1.27 typedef struct mat_mtypepqnode mat_mtypepqnode

Mtype Priority Queue Node Structure.

Mtype Priority Queue Node

6.35.1.28 typedef mat_mtypepqnode∗ MAT_MTYPEPQNODE

Mtype Priority Queue Node Pointer

6.35.1.29 typedef struct mat_perceptron mat_perceptron

Perceptron Classifier Model Structure.

Perceptron Classifier Model

6.35.1.30 typedef mat_perceptron∗ MAT_PERCEPTRON

Perceptron Classifier Model Pointer

6.35.1.31 typedef struct mat_qintnode mat_qintnode

Integer Queue Node Structure.

Integer Queue Node

6.35.1.32 typedef mat_qintnode∗ MAT_QINTNODE

Integer Queue Node Pointer

6.35.1.33 typedef struct mat_qmtypenode mat_qmtypenode

Mtype Queue Node Structure.

Mtype Queue Node

6.35.1.34 typedef mat_qmtypenode* MAT_QMTYPENODE

Mtype Queue Node Pointer

6.35.1.35 typedef mat_tree_node* MAT_TREE

Search Tree Pointer

6.35.1.36 typedef struct mat_tree_node mat_tree_node

Search Tree Node Structure.

Search Tree Node

6.35.1.37 typedef mat_tree_node* MAT_TREE_NODE

Search Tree Node Pointer

6.35.1.38 typedef mtype** MATRIX

Mtype Matrix

6.35.1.39 typedef MATRIX* MATSTACK

Mtype Matrix Stack

6.35.1.40 typedef void** MATVEC_DPOINTER

Mtype Matrix - Integer Vector Pair

6.35.2 Function Documentation

6.35.2.1 INT_VECTOR __int_vec_creat (int *len*)

6.35.2.2 INT_VECSTACK __int_vecstack_creat (int *len*)

6.35.2.3 mtype __mat_addfunc (mtype *x*, mtype *y*)

Computes addition function.

Parameters

in	*x*	
in	*y*	

Returns

$x+y$

6.35.2.4 mtype __mat_arccosh (mtype x)

Computes inverse hyperbolic cosine function.

Parameters

in	*x*	

Returns

$\cosh^{-1}(x)$

6.35.2.5 mtype __mat_arcsinh (mtype x)

Computes inverse hyperbolic sine function.

Parameters

in	*x*	

Returns

$\sinh^{-1}(x)$

6.35.2.6 mtype __mat_arctanh (mtype x)

Computes inverse hyperbolic tangent function.

Parameters

in	*x*	

Returns

$\tanh^{-1}(x)$

6.35.2.7 mtype __mat_bisquare_wt (mtype x, mtype k)

Computes bisquare weight function.

Parameters

in	*x*	
in	*k*	

Returns

$$\begin{cases} \left(1-\left(\frac{x}{k}\right)^2\right)^2, & \text{for } |x| \leq k, \\ 0, & \text{otherwise.} \end{cases}$$

6.35.2.8 void __mat_cart2pol (mtype *x*, mtype *y*, mtype ∗ *rho*, mtype ∗ *th*)

6.35.2.9 MATRIX __mat_creat (int *r*, int *c*)

6.35.2.10 mtype __mat_divfunc (mtype *x*, mtype *y*)

Computes division function.

Parameters

in	*x*	
in	*y*	

Returns

$\frac{x}{y}$

6.35.2.11 int __mat_fft (int *dir*, int *m*, mtype ∗ *x*, mtype ∗ *y*)

6.35.2.12 mtype __mat_huber_wt (mtype *x*, mtype *k*)

Computes Huber weight function.

Parameters

in	*x*	
in	*k*	

Returns

$$\begin{cases} 1, & \text{for } |x| \leq k, \\ \frac{k}{|x|}, & \text{otherwise.} \end{cases}$$

6.35.2.13 MAT_KDNODE __mat_kd_find_median (MAT_KDNODE *kd_start*, MAT_KDNODE *kd_end*, int *idx*)

6.35.2.14 void __mat_kdtree_k_nearest (MAT_KDNODE *root*, MAT_KDNODE *nd*, int *i*, int *dim*, MAT_MTYPE_PRIORITYQUEUE *pq*, MATRIX *bmax*, MATRIX *bmin*)

6.35.2.15 MAT_KDNODE __mat_kdtree_make_tree (MAT_KDNODE *t*, int *len*, int *i*, int *dim*)

6.35.2.16 void __mat_kdtree_nearest (MAT_KDNODE *root*, MAT_KDNODE *nd*, int *i*, int *dim*, MAT_KDNODE ∗ *best*, mtype ∗ *best_dist*)

6.35.2.17 mtype __mat_lint (mtype ∗ *x*, mtype(∗)(mtype) *func*, mtype *x0*, mtype *xn*, mtype *f0*, mtype *f2*, mtype *f3*, mtype *f5*, mtype *f6*, mtype *f7*, mtype *f9*, mtype *fl4*, mtype *hmin*, mtype *hmax*, mtype *re*, mtype *ae*)

6.35.2.18 mtype __mat_logplusone (mtype *x*)

Computes logarithm plus one function.

Parameters

in	*x*	

Returns

$\log(1+x)$

6.35.2.19 MATRIX __mat_mds_metric (MATRIX *d*, int *dims*, MATRIX *result*)

6.35.2.20 MATRIX __mat_mds_nonmetric (MATRIX *d*, int *dims*, MATRIX *result*)

6.35.2.21 mtype __mat_mulfunc (mtype *x*, mtype *y*)

Computes multiplication function.

Parameters

in	*x*	
in	*y*	

Returns

xy

6.35.2.22 void __mat_pol2cart (mtype *rho*, mtype *th*, mtype * *x*, mtype * *y*)

6.35.2.23 int __mat_powerof2 (int *width*, int * *m*, int * *twopm*)

6.35.2.24 void __mat_quicksort (MATRIX *A*, int *l*, int *r*, int *offset*, MATRIX *ind*)

6.35.2.25 mtype __mat_rand (void)

6.35.2.26 mtype __mat_randexp (mtype *mu*)

6.35.2.27 mtype __mat_randfun (mtype(*)(mtype) *fun*, mtype *xmin*, mtype *xmax*)

6.35.2.28 mtype __mat_randn (void)

6.35.2.29 mtype __mat_sqrfunc (mtype *x*)

Computes square function.

Parameters

in	*x*	

Returns

x^2

6.35.2.30 mtype __mat_sqrtfunc (mtype *x*)

Computes square root function.

Parameters

in	*x*	

Returns

$\sqrt{x}$

6.35.2.31 mtype __mat_subfunc (mtype *x*, mtype *y*)

Computes subtraction function.

Parameters

in	*x*	
in	*y*	

Returns

$x-y$

6.35.2.32 MATSTACK __matstack_creat (int *len*)

6.35.2.33 mtype gen_abs_ceil (mtype *a*)

6.35.2.34 int gen_eq (mtype *a*)

6.35.2.35 int gen_error (int *err_*)

Generates error message for general errors and exits.

Parameters

in	*err*	Error type (GEN_NOT_CONVERGED/GEN_FNOTOPEN/ GEN_FNOTGETMAT/GEN_SIZEMISMATCH/GEN_MATH_ERROR/GEN_MALLOC/GEN_NOT_FOUND/GEN_SIZE_ERROR/GEN_BAD_TYPE)

6.35.2.36 int gen_gt (mtype *a*)

6.35.2.37 int gen_lt (mtype *a*)

6.35.2.38 int graph_error (int *err_*)

Generates error message for graph errors and exits.

Parameters

in	*err*	Error type (GRAPH_MALLOC/GRAPH_READ/GRAPH_ELSE)

6.35.2.39 MATRIX int_vec2_mat (INT_VECTOR *a*, int *dir*)

Converts an integer vector to a matrix.

Parameters

in	*a*	Input vector
in	*dir*	Conversion direction

Returns

Output matrix

6.35.2.40 INT_VECTOR int_vec_add (INT_VECTOR *A*, INT_VECTOR *B*, INT_VECTOR *result*)

Adds two integer vectors.

Parameters

in	*A*	Input vector
in	*B*	Input vector
in	*result*	Vector to store the result

Returns

$\mathbf{A}+\mathbf{B}$

6.35.2.41 INT_VECTOR int_vec_adds (INT_VECTOR *A*, int *s*, INT_VECTOR *result*)

Adds an integer to an integer vector.

Parameters

in	*A*	Input vector
in	*s*	Input scalar
in	*result*	Vector to store the result

Returns

$\mathbf{A}+s\mathbf{1}$

6.35.2.42 INT_VECTOR int_vec_append (INT_VECTOR *a*, int *i*)

Appends an integer to an integer vector.

Parameters

in	*a*	Input vector
in	*i*	Integer to append

Returns

Appended vector

6.35.2.43 INT_VECTOR int_vec_concat (INT_VECTOR *a*, INT_VECTOR *b*, INT_VECTOR *result*)

Concatenates two integer vectors.

Parameters

in	*a*	Input first vector
in	*b*	Input second vector
in	*dim*	Concatenation direction (ROWS/COLS)

Returns

$\begin{bmatrix} a & b \end{bmatrix}$ or $\begin{bmatrix} a \\ b \end{bmatrix}$

6.35.2.44 INT_VECTOR int_vec_copy (INT_VECTOR *a*, INT_VECTOR *result*)

Copies an integer vector.

Parameters

in	*a*	Input vector
in	*result*	Vector to store the result

Returns

Output vector

6.35.2.45 INT_VECTOR int_vec_creat (int *len*, int *type*)

Creates an integer vector.

Parameters

in	*len*	Length of the vector
in	*type*	Definition type (UNDEFINED/ZERO_INT_VECTOR/ONES_INT_VECTOR/SERIES_INT_VECTOR)

Returns

Output vector

6.35.2.46 INT_VECTOR int_vec_div (INT_VECTOR *A*, INT_VECTOR *B*, INT_VECTOR *result*)

Computes element-wise integer vector division.

Parameters

in	*A*	First input vector
in	*B*	Second input vector
in	*result*	Vector to store the result

Returns

$A./B$

6.35.2.47 INT_VECTOR int_vec_divs (INT_VECTOR *A*, int *x*, INT_VECTOR *result*)

Divides an integer vector by a scalar.

Parameters

in	*A*	Input vector
in	*s*	Scalar
in	*result*	Vector to store the result

Returns

$\frac{A}{s}$

6.35.2.48 void int_vec_dump (INT_VECTOR *A*)

Dumps an integer vector in the stdout.

Parameters

in	*A*	Input vector

6.35.2.49 void int_vec_dumpf (INT_VECTOR *A*, const char ∗ *s*)

Dumps an integer vector using a given format specifier in the stdout.

Parameters

in	*A*	Input vector
in	*s*	Format specifier

6.35.2.50 INT_VECTOR int_vec_error (int *err_*)

Generates error message for integer vector errors and exits.

Parameters

in	*err*	Error type (INT_VEC_MALLOC/INT_VEC_FNOTOPEN/INT_VEC_FNOTGET-INT_VEC/INT_VEC_SIZEMISMATCH)

6.35.2.51 void int_vec_fdump (INT_VECTOR *A*, MAT_FILEPOINTER *fp*)

Dumps an integer vector in an opened file.

Parameters

in	*A*	Input vector
in	*fp*	Pointer to an opened file

6.35.2.52 void int_vec_fdumpf (INT_VECTOR *A*, const char ∗ *s*, MAT_FILEPOINTER *fp*)

Dumps an integer vector using a given format specifier in an opened file.

Parameters

in	*A*	Input vector
in	*s*	Format specifier
in	*fp*	Pointer to an opened file

6.35.2.53 INT_VECTOR int_vec_fill (INT_VECTOR *A*, int *val*)

Fills an integer vector with a value.

Parameters

in	*A*	Input vector
in	*val*	Value to fill with

Returns

Filled vector

6.35.2.54 INT_VECTOR int_vec_fill_type (INT_VECTOR *A*, int *type*)

Fills an integer vector to a type.

Parameters

in	*A*	Input vector
in	*type*	Definition type (UNDEFINED/ZERO_INT_VECTOR/ONES_INT_VECTOR/SERIES_INT_VECTOR)

Returns

Filled vector

6.35.2.55 INT_VECTOR int_vec_find (INT_VECTOR *a*, int *rel_type*, int *n*)

6.35.2.56 int int_vec_free (INT_VECTOR *A*)

Frees an integer vector.

Parameters

in	*A*	Input vector

Returns

Success

6.35.2.57 INT_VECTOR int_vec_mul (INT_VECTOR *A*, INT_VECTOR *B*, INT_VECTOR *result*)

6.35.2.58 INT_VECTOR int_vec_muls (INT_VECTOR *A*, int *s*, INT_VECTOR *result*)

6.35.2.59 INT_VECTOR int_vec_permute_vect (int *n*, int *k*, INT_VECTOR *result*)

Computes a randomly permutation of first k positive integers.

Parameters

in	*n*	Number of random permutations to make
in	*k*	Integer upto which it will consider
in	*result*	Vector to store the result

Returns

Permuted vector

6.35.2.60 INT_VECTOR int_vec_randperm (int *n,* INT_VECTOR *result*)

6.35.2.61 INT_VECTOR int_vec_sub (INT_VECTOR *A,* INT_VECTOR *B,* INT_VECTOR *result*)

6.35.2.62 INT_VECTOR int_vec_subs (INT_VECTOR *A,* int *s,* INT_VECTOR *result*)

6.35.2.63 INT_VECTOR int_vec_unique (INT_VECTOR *a*)

Extract only the unique integers from an integer vector.

Parameters

in	*a*	Input vector

Returns

Unique vector

6.35.2.64 INT_VECSTACK int_vecstack_creat (int *len*)

Creates an integer vector stack.

Parameters

in	*len*	Length of the stack

Returns

Output vector stack

6.35.2.65 INT_VECSTACK int_vecstack_error (int *err_*)

Generates error message for integer vector stack errors and exits.

Parameters

in	*err*	Error type (INT_VECSTACK_MALLOC/INT_VECSTACK_FNOTOPEN/INT_VECSTACK_FNOTGETINT_VEC/INT_VECSTACK_SIZEMISMATCH)

6.35.2.66 int int_vecstack_free (INT_VECSTACK *A*)

Frees an integer vector stack.

Parameters

in	*A*	Input vector stack

Returns

Success

6.35.2.67 INT_VECTOR mat_2int_vec (MATRIX *A*)

Converts a matrix to an integer vector.

Parameters

in	*A*	Input matrix
out	*v*	Output vector

Returns

Output vector

6.35.2.68 MATRIX mat_abs (MATRIX *A*, MATRIX *result*)

Computes absolute value of matrix.

Parameters

in	*A*	Input matrix
in	*result*	Matrix to store the result

Returns

$abs(A)$

6.35.2.69 MATRIX mat_add (MATRIX *A*, MATRIX *B*, MATRIX *result*)

Adds two matrices.

Parameters

in	*A*	Input matrix
in	*B*	Input matrix
in	*result*	Matrix to store the result

Returns

$\mathbf{A}+\mathbf{B}$

6.35.2.70 MATRIX mat_adds (MATRIX *A*, mtype *s*, MATRIX *result*)

Adds a scalar to a matrix.

Parameters

in	*A*	Input matrix
in	*s*	Input scalar
in	*result*	Matrix to store the result

Returns

$\mathbf{A}+s\mathbf{1}\mathbf{1}^T$

6.35.2.71 void mat_backsubs1 (MATRIX *A*, MATRIX *B*, MATRIX *C*, MATRIX *P*, int *xcol*)

6.35.2.72 INT_VECTOR mat_bayes_classifier_test (MATRIX *data*, MAT_BAYES_MODEL *b_model*)

6.35.2.73 MAT_BAYES_MODEL mat_bayes_classifier_train (MATRIX *data*, INT_VECTOR *labels*)

6.35.2.74 MAT_BAYES_MODEL mat_bayes_model_creat (void)

Creates a Bayes model.

Returns

Output Bayes model

6.35.2.75 int mat_bayes_model_free (MAT_BAYES_MODEL *a*)

Frees a Bayes model.

Parameters

in	*a*	Input Bayes model

Returns

Success

6.35.2.76 mtype mat_binom (int *n*, int *k*)

Computes a binomial co-efficient.

Parameters

in	*n*	1^{st} argument
in	*k*	2^{nd} argument

Returns

$\binom{n}{k}$

6.35.2.77 void mat_binom_init ( )

Initializes the binomial series.

6.35.2.78 MATRIX mat_bisquare_wt (MATRIX *A*, mtype *k*, mtype *sigma*, MATRIX *result*)

Computes bisquare weight function element-wise on a matrix.

Parameters

in	*A*	Input matrix
in	*k*	Bisquare parameter

Returns

$\mathbf{B}, b_{ij} = f_k(a_{ij})$ where f_k is the biquare weight function

6.35.2.79 MAT_TREE mat_bs_delete (mtype *x*, MAT_TREE *T*)

6.35.2.80 MAT_TREE mat_bs_find (mtype *x*, MAT_TREE *T*)

6.35.2.81 MAT_TREE mat_bs_find_max (MAT_TREE *T*)

6.35.2.82 MAT_TREE mat_bs_find_min (MAT_TREE *T*)

6.35.2.83 MAT_TREE mat_bs_free (MAT_TREE *T*)

6.35.2.84 int mat_bs_inorder (MAT_TREE *T*, int *index*, mtype $**$ *ordered*)

6.35.2.85 MAT_TREE mat_bs_insert (mtype *x*, MAT_TREE *T*)

6.35.2.86 MAT_TREE mat_bs_make_null (void)

6.35.2.87 MATRIX mat_bsxfun (MATRIX *A*, MATRIX *B*, mtype($*$)(mtype, mtype) *func*, MATRIX *result*)

Computes element-wise binary function for two matrices.

Parameters

in	*A*	First matrix
in	*B*	Second matrix
in	*func*	Pointer to the function
in	*result*	Matrix to store the result

Returns

Output matrix

6.35.2.88 MATRIX mat_calc_dist_sq (MATRIX *A*, MATRIX *d*, MATRIX *result*)

Computes the Euclidean distances of points from a given point.

Parameters

in	*A*	Points matrix (d x N)
in	*d*	Matrix point from which the distance to be computed (d x 1)
in	*result*	Matrix to store the result

Returns

Euclidean distance matrix

6.35.2.89 MATRIX mat_cart2pol (MATRIX *A*, int *dim*, MATRIX *result*)

Converts Cartesian co-ordinates to polar co-ordinates.

Parameters

in	*A*	Input matrix
in	*dim*	Data order ROWS/COLS

Returns

Polar co-ordinate matrix

6.35.2.90 MATRIX mat_cheby (int *n*)

Computes the n^{th} Chebyshev polynomial.

Parameters

in	*n*	Polynomial series index

Returns

Output polynomial matrix

6.35.2.91 MATRIX mat_cheby_approx (mtype(*)(mtype) *f*, mtype *a*, mtype *b*, int *n*, MATRIX *result*)

Approximates a function using Chebyshev polynomials.

Parameters

in	*f*	Function to approximate
in	*a*	Lower limit of domain of the function
in	*b*	Upper limit of domain of the function
in	*n*	Degree of the approximate polynomial
in	*result*	Matrix to store the result

Returns

Approximate polynomial matrix

6.35.2.92 MATRIX mat_cheby_coeffs_to_poly (MATRIX *coeffs*, MATRIX *result*)

Converts Chebyshev co-efficients to a single polynomial.

Parameters

in	*coeffs*	Chebyshev polynomial co-efficient matrix
in	*result*	Matrix to store the result

Returns

Polynomial matrix

6.35.2.93 void mat_cheby_init ( )

Initializes the Chebyshev polynomial series.

6.35.2.94 MATRIX mat_cholesky (MATRIX *A*, MATRIX *result*)

Computes Cholesky factor of a matrix.

Parameters

in	*A*	Input matrix
in	*result*	Matrix to store the result

Returns

Cholesky factor

6.35.2.95 mtype mat_cofact (MATRIX *A*, int *i*, int *j*)

Computes a cofactor of a matrix.

Parameters

in	*A*	Input matrix
in	*i*	Row index
in	*j*	Column index

Returns

Cofactor C_{ij}

6.35.2.96 MATRIX mat_colcopy (MATRIX *A*, int *cola*, int *colb*, MATRIX *result*)

Copies a column from a matrix.

Parameters

in	*A*	Input matrix
in	*cola*	Source column
in	*colb*	Destination column
in	*result*	Matrix to store the result

Returns

Copied matrix

6.35.2.97 MATRIX mat_concat (MATRIX *A*, MATRIX *B*, int *dim*)

Concatenates two matrices.

Parameters

in	*A*	Input first matrix
in	*B*	Input second matrix
in	*dim*	Concatenation direction (ROWS/COLS)

Returns

$\begin{bmatrix} A & B \end{bmatrix}$ or $\begin{bmatrix} A \\ B \end{bmatrix}$

6.35.2.98 MATRIX mat_conjgrad (MATRIX *A*, MATRIX *b*, MATRIX *x0*, mtype *tol*, int *miters*, MATRIX *result*)

Solves a linear system with conjugate gradients method.

Parameters

in	*A*	Input matrix
in	*b*	Observed matrix
in	*result*	Matrix to store the result

Returns

x

6.35.2.99 MATRIX mat_conv2 (MATRIX *A*, MATRIX *mask*, MATRIX *scratch*, MATRIX *result*)

Computes 2-D convolution.

Parameters

in	*A*	Input matrix
in	*mask*	Input kernel/mask
in	*scratch*	Scratch matrix for temporary calculations
in	*result*	Matrix to store the result

Returns

Convolved output matrix

6.35.2.100 MATRIX mat_copy (MATRIX *A*, MATRIX *result*)

Copies a matrix.

Parameters

in	*A*	Input matrix
in	*result*	Matrix to store the result

Returns

Output matrix

6.35.2.101 MATSTACK mat_corcol (MATRIX *data*)

6.35.2.102 int mat_count_words_in_line (MAT_FILEPOINTER *fp*, int ∗ *count*)

Count words in current line in an opened file.

Parameters

in	*fp*	Pointer to an opened file
out	*count*	Pointer to output count

Returns

EOF reached

6.35.2.103 MATSTACK mat_covcol (MATRIX *data*)

6.35.2.104 MATRIX mat_creat (int *row*, int *col*, int *type*)

Creates a matrix.

Parameters

in	*row*	Number of rows
in	*col*	Number of columns
in	*type*	Definition type (UNDEFINED/ZERO_MATRIX/UNIT_MATRIX/ONES_MATRIX)

Returns

Output matrix

6.35.2.105 MATRIX mat_creat_diag (MATRIX *diag_vals*, MATRIX *result*)

Creates a diagonal matrix from a 1-d matrix.

Parameters

in	*diag_vals*	Input 1-d diagonal value matrix
in	*result*	Matrix to store the result

Returns

Diagonal matrix

6.35.2.106 mtype mat_det (MATRIX *A*)

Computes the determinant of a matrix.

Parameters

in	*A*	Input matrix

Returns

$\det(A)$

6.35.2.107 mtype mat_diagmul (MATRIX *A*)

6.35.2.108 MATRIX mat_div_dot (MATRIX *A*, MATRIX *B*, MATRIX *result*)

Computes element-wise matrix division.

Parameters

in	*A*	First input matrix
in	*B*	Second input matrix
in	*result*	Matrix to store the result

Returns

$A./B$

6.35.2.109 MATRIX mat_divs (MATRIX *A*, mtype *s*, MATRIX *result*)

Divides a matrix by a scalar.

Parameters

in	*A*	Input matrix
in	*s*	Scalar
in	*result*	Matrix to store the result

Returns

$\frac{A}{s}$

6.35.2.110 MATRIX mat_dlmread (const char ∗ *fname*)

Reads a matrix from a file.

Parameters

in	*fname*	Filename to read from

Returns

Output matrix

6.35.2.111 void mat_dlmwrite (const char ∗ *fname*, MATRIX *A*)

Writes a matrix to a file.

Parameters

in	*fname*	Filename to write into
in	*A*	Input matrix

6.35.2.112 void mat_dump (MATRIX *A*)

Dumps a matrix in the stdout.

Parameters

in	*A*	Input matrix

6.35.2.113 void mat_dumpf (MATRIX *A*, const char ∗ *s*)

Dumps a matrix using a given format specifier in the stdout.

Parameters

in	*A*	Input matrix
in	*s*	Format specifier

6.35.2.114 MATRIX mat_durbin (MATRIX *R*, MATRIX *result*)

Runs Levinson-Durbin algorithm.

Parameters

in	*R*	Input n^th correlation matrix $(n+1)\times 1$
in	*result*	Matrix to store the result

Returns

X where $\tilde{R}X = B$, $\tilde{R} = \begin{bmatrix} R[0][0] & R[1][0] & \cdots & R[n-1][0] \\ R[1][0] & R[0][0] & \cdots & R[n-2][0] \\ \vdots & \vdots & \ddots & \vdots \\ R[n-1][0] & R[n-2][0] & \cdots & R[0][0] \end{bmatrix}$ and $B = \begin{bmatrix} R[1][0] & R[2][0] & \cdots & R[n][0] \end{bmatrix}$

6.35.2.115 MATSTACK mat_eig_sym (MATRIX *symmat*, MATSTACK *result*)

6.35.2.116 MATRIX mat_error (int *err_*)

Generates error message for matrix errors and exits.

Parameters

in	*err*	Error type (MAT_MALLOC/MAT_FNOTOPEN/MAT_FNOTGETMAT/MAT_SIZEMISMATCH/ MAT_INVERSE_ILL_COND/MAT_INVERSE_NOT_SQUARE/MAT_CHOLESKY_FAILED)

6.35.2.117 void mat_fdump (MATRIX *A*, MAT_FILEPOINTER *fp*)

Dumps a matrix in an opened file.

Parameters

in	*A*	Input matrix
in	*fp*	Pointer to an opened file

6.35.2.118 void mat_fdumpf (MATRIX *A*, const char ∗ *s*, MAT_FILEPOINTER *fp*)

Dumps a matrix using a given format specifier in an opened file.

Parameters

in	*A*	Input matrix
in	*s*	Format specifier
in	*fp*	Pointer to an opened file

6.35.2.119 MATSTACK mat_fft2 (MATSTACK *c,* int *dir,* MATSTACK *result*)

Computes fast Fourier transform.

Parameters

in	*C*	Complex data matrix stack
in	*dir*	FFT direction (ROWS/COLS)
in	*result*	Matrix stack to store the result

Returns

Transformed matrix stack

6.35.2.120 int mat_fgetmat (MATRIX *A,* MAT_FILEPOINTER *fp*)

Gets matrix data from opened file.

Parameters

in	*A*	Matrix to store the data
in	*fp*	Pointer to opened file

Returns

Number of elements copied

6.35.2.121 MATRIX mat_fill (MATRIX *A,* mtype *val*)

Fills a matrix with a value.

Parameters

in	*A*	Input matrix
in	*val*	Value to fill with

Returns

Filled matrix

6.35.2.122 MATRIX mat_fill_type (MATRIX *A,* int *type*)

Fills a matrix to a type.

Parameters

in	*A*	Input matrix
in	*type*	Fill type (UNDEFINED/ZERO_MATRIX/UNIT_MATRIX/ONES_MATRIX)

Returns

Filled matrix

6.35.2.123 INT_VECSTACK mat_find (MATRIX *A*, int *rel_type*, mtype *x*)

6.35.2.124 INT_VECTOR mat_find_within_dist (MATRIX *A*, MATRIX *d*, mtype *range*)

Finds points within a neighborhood.

Parameters

in	*A*	Points matrix (d x N)
in	*d*	Matrix point from which the distance to be computed (d x 1)
in	*range*	Radius to search within

Returns

Indices Vector

6.35.2.125 MATRIX mat_fliplr (MATRIX *A*, MATRIX *result*)

6.35.2.126 MATRIX mat_flipud (MATRIX *A*, MATRIX *result*)

6.35.2.127 void mat_fnextline (MAT_FILEPOINTER *fp*)

Prints nextline to file.

Parameters

in	*fp*	Pointer to opened file

6.35.2.128 int mat_free (MATRIX *A*)

Frees a matrix.

Parameters

in	*A*	Input matrix

Returns

Success

6.35.2.129 MATRIX mat_get_sub_matrix_from_cols (MATRIX *A*, INT_VECTOR *indices*, MATRIX *result*)

Extracts sub-matrix from columns of a matrix.

Parameters

in	*A*	Input matrix
in	*indices*	Columns to extract
in	*result*	Matrix to store the result

Returns

Extracted matrix

6.35.2.130 MATRIX mat_get_sub_matrix_from_rows (MATRIX *A*, INT_VECTOR *indices*, MATRIX *result*)

Extracts sub-matrix from rows of a matrix.

Parameters

in	*A*	Input matrix
in	*indices*	Rows to extract
in	*result*	Matrix to store the result

Returns

Extracted matrix

6.35.2.131 INT_VECTOR mat_get_sub_vector (INT_VECTOR *a*, INT_VECTOR *indices*)

Extracts sub-vector from an integer vector.

Parameters

in	*a*	Input vector
in	*indices*	Indices to extracted

Returns

Extracted vector

6.35.2.132 MATRIX mat_gfunc (MATRIX *A*, mtype(∗)(mtype) *pt2func*, MATRIX *result*)

Computes a given function element-wise on a matrix.

Parameters

in	*A*	Input matrix
in	*f*	Given function

Returns

$\mathbf{B}, b_{ij} = f(a_{ij})$

6.35.2.133 int mat_go_next_word (MAT_FILEPOINTER *fp*)

Moves to next word in an opened file.

Parameters

in	*fp*	Pointer to an opened file

Returns

EOF reached

6.35.2.134 void mat_graph_adjlist (**MAT_GRAPH** *g,* int *directed,* int *weighted,* MAT_FILEPOINTER *fp*)

6.35.2.135 void mat_graph_adjm_to_adjl (**MAT_GRAPH** *g,* **MATRIX** *a*)

6.35.2.136 **MAT_GRAPH** mat_graph_creat (void)

6.35.2.137 void mat_graph_dump (**MAT_GRAPH** *g,* int *mst*)

6.35.2.138 void mat_graph_dumpf (**MAT_GRAPH** *g,* int *mst,* MAT_FILEPOINTER *fp*)

6.35.2.139 **MAT_GRAPH** mat_graph_reverse (**MAT_GRAPH** *g,* **MAT_GRAPH** *r*)

6.35.2.140 **MAT_INT_QUEUE** mat_graph_search (**MAT_GRAPH** *g,* int *connected,* int *mst*)

6.35.2.141 void mat_graph_visit (**MAT_GRAPH** *g,* int *k,* int *connected,* int *mst,* **MAT_INT_PRIORITYQUEUE** *pq,* **MAT_INT_QUEUE** *q*)

6.35.2.142 **MATRIX** mat_huber_wt (**MATRIX** *A,* mtype *k,* mtype *sigma,* **MATRIX** *result*)

Computes Huber weight function element-wise on a matrix.

Parameters

in	*A*	Input matrix
in	*k*	Huber parameter

Returns

$\mathbf{B}, b_{ij} = f_k(a_{ij})$ where f_k is the Huber weight function

6.35.2.143 mtype mat_innerprod (**MATRIX** *A,* **MATRIX** *B*)

6.35.2.144 **MAT_INT_PRIORITYQUEUE** mat_int_priorityqueue_creat (int *type*)

6.35.2.145 **mat_intpqnode** mat_int_priorityqueue_dequeue (**MAT_INT_PRIORITYQUEUE** *H*)

6.35.2.146 void mat_int_priorityqueue_enqueue (**MAT_INT_PRIORITYQUEUE** *H,* int *data,* int *priority*)

6.35.2.147 int mat_int_priorityqueue_free (**MAT_INT_PRIORITYQUEUE** *H*)

6.35.2.148 int mat_int_priorityqueue_is_empty (**MAT_INT_PRIORITYQUEUE** *H*)

6.35.2.149 int mat_int_priorityqueue_update (**MAT_INT_PRIORITYQUEUE** *H,* int *data,* int *priority,* int *type*)

6.35.2.150 mtype mat_int_qadrat (mtype(∗)(mtype) *func,* mtype *lower,* mtype *upper*)

Computes Gauss quadrature integration.

Parameters

in	*func*	Function $f(\cdot)$ to integrate
in	*n*	Number of subdivisions
in	*lower*	Lower Limit
in	*upper*	Upper Limit

Returns

$\int_a^b f(x)\,dx$

6.35.2.151 MAT_INT_QUEUE mat_int_queue_creat (void)

6.35.2.152 int mat_int_queue_dequeue (MAT_INT_QUEUE *s*)

6.35.2.153 void mat_int_queue_enqueue (MAT_INT_QUEUE *s,* int *value*)

6.35.2.154 int mat_int_queue_free (MAT_INT_QUEUE *s*)

6.35.2.155 int mat_int_queue_is_empty (MAT_INT_QUEUE *s*)

6.35.2.156 mtype mat_int_simpson (mtype(∗)(mtype) *func,* int *n,* mtype *lower,* mtype *upper*)

Computes Simpson's integration.

Parameters

in	*func*	Function $f(\cdot)$ to integrate
in	*n*	Number of subdivisions
in	*lower*	Lower Limit
in	*upper*	Upper Limit

Returns

$\int_a^b f(x)\,dx$

6.35.2.157 MAT_INT_STACK mat_int_stack_creat (void)

6.35.2.158 int mat_int_stack_free (MAT_INT_STACK *s*)

6.35.2.159 int mat_int_stack_is_empty (MAT_INT_STACK *s*)

6.35.2.160 int mat_int_stack_pop (MAT_INT_STACK *s*)

6.35.2.161 void mat_int_stack_push (MAT_INT_STACK *s,* int *value*)

6.35.2.162 mtype mat_int_trapezoid (mtype(∗)(mtype) *func,* int *n,* mtype *lower,* mtype *upper*)

Computes trapezoid integration.

Parameters

in	*func*	Function $f(\cdot)$ to integrate
in	*n*	Number of subdivisions
in	*lower*	Lower Limit
in	*upper*	Upper Limit

Returns

$\int_a^b f(x)\,dx$

6.35.2.163 MATRIX mat_inv (MATRIX *A*, MATRIX *result*)

Computes the inverse of a matrix.

Parameters

in	*A*	Input matrix
in	*result*	Matrix to store the result

Returns

A^{-1}

6.35.2.164 int mat_isnumeric (MAT_FILEPOINTER *fp*)

Checks if current word in an opened file is numeric or not.

Parameters

in	*fp*	Pointer to an opened file

Returns

Zero/non-zero

6.35.2.165 int mat_kdtree_free (MAT_KDTREE *t*)

Frees a k-d tree.

Parameters

in	*t*	Input k-d tree

Returns

Success

6.35.2.166 MATRIX mat_kdtree_k_nearest (MAT_KDTREE *t*, MATRIX *A*, int *k*, MATRIX *result*)

Computes k nearest neighbors.

Parameters

in	*t*	Input k-d tree
in	*A*	Input data matrix of size $d \times N$
in	*k*	Number of neighbors
in	*result*	Matrix to store the result

Returns

Output matrix B with index B[0][j] and squared distance B[1][j] for $j = 1, 2, \cdots, N$

6.35.2.167 MAT_KDTREE mat_kdtree_make_tree (MATRIX *A*, MAT_KDTREE *result*)

Creates a k-d tree from a data matrix.

Parameters

in	*A*	Input data matrix of size $d \times N$
in	*result*	K-d tree to store the result

Returns

Output k-d tree

6.35.2.168 MATRIX mat_kdtree_nearest (MAT_KDTREE *t*, MATRIX *A*, MATRIX *result*)

Computes nearest neighbors.

Parameters

in	*t*	Input k-d tree
in	*A*	Input data matrix of size $d \times N$
in	*result*	Matrix to store the result

Returns

Output matrix B with index B[0][j] and squared distance B[1][j] for $j = 1, 2, \cdots, N$

6.35.2.169 MATVEC_DPOINTER mat_kmeans (MATRIX *data*, int *k*, int *iters*, MATVEC_DPOINTER *result*)

6.35.2.170 MATRIX mat_least_squares (MATRIX *A*, MATRIX *Y*, MATRIX *result*)

6.35.2.171 MATRIX mat_legendre (int *n*)

Computes the n^{th} Legendre polynomial.

Parameters

in	*n*	Polynomial series index

Returns

Output polynomial matrix

6.35.2.172 void mat_legendre_init ( )

Initializes the Legendre polynomial series.

6.35.2.173 MATRIX mat_linear_ls_fit (MATRIX *A*, MATRIX *Y*, int *deg*, MATRIX *result*)

Polynomial model using least squares.

Parameters

in	*A*	Data matrix $N \times 1$
in	*Y*	Observation matrix $N \times 1$

Returns

6.35.2.174 MATRIX mat_lsolve (MATRIX *A*, MATRIX *b*, MATRIX *result*)

6.35.2.175 MATRIX mat_lsolve_durbin (MATRIX *A*, MATRIX *B*, MATRIX *result*)

Runs Levinson-Durbin algorithm.

Parameters

in	*A*	Input correlation matrix $A = \begin{bmatrix} r_0 & r_1 & \cdots & r_{n-1} \\ r_1 & r_0 & \cdots & r_{n-2} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n-1} & r_{n-2} & \cdots & r_0 \end{bmatrix}$
in	*B*	Input correlation matrix $B = \begin{bmatrix} r_1 \\ r_2 \\ \cdots \\ r_n \end{bmatrix}$
in	*result*	Matrix to store the result

Returns

X where $RX = B$

6.35.2.176 int mat_lu (MATRIX *A*, MATRIX *P*)

6.35.2.177 MATVEC_DPOINTER mat_max (MATRIX *A*, int *dim*)

6.35.2.178 MATRIX mat_mds (MATRIX *d*, int *dims*, int *type*, MATRIX *result*)

6.35.2.179 mtype mat_mean (MATRIX *A*)

6.35.2.180 MATRIX mat_mean_col (MATRIX *A*, MATRIX *result*)

6.35.2.181 MATRIX mat_mean_row (MATRIX *A*, MATRIX *result*)

6.35.2.182 mtype mat_median (MATRIX *A*)

Computes the median of elements of a given matrix.

Parameters

in	*A*	Input matrix

Returns

median $\left(\{a_{ij}\}\right)$

6.35.2.183 MATVEC_DPOINTER mat_min (MATRIX *A*, int *dim*)

6.35.2.184 mtype mat_minor (MATRIX *A*, int *i*, int *j*)

Computes a minor of a matrix.

Parameters

in	*A*	Input matrix
in	*i*	Row index
in	*j*	Column index

Returns

Minor M_{ij}

6.35.2.185 MAT_MTYPE_PRIORITYQUEUE mat_mtype_priorityqueue_creat (int *type*)

6.35.2.186 mat_mtypepqnode mat_mtype_priorityqueue_dequeue (MAT_MTYPE_PRIORITYQUEUE *H*)

6.35.2.187 void mat_mtype_priorityqueue_enqueue (MAT_MTYPE_PRIORITYQUEUE *H*, mtype *data*, mtype *priority*)

6.35.2.188 int mat_mtype_priorityqueue_free (MAT_MTYPE_PRIORITYQUEUE *H*)

6.35.2.189 int mat_mtype_priorityqueue_is_empty (MAT_MTYPE_PRIORITYQUEUE *H*)

6.35.2.190 int mat_mtype_priorityqueue_update (MAT_MTYPE_PRIORITYQUEUE *H*, mtype *data*, mtype *priority*, int *type*)

6.35.2.191 MAT_MTYPE_QUEUE mat_mtype_queue_creat (void)

6.35.2.192 mtype mat_mtype_queue_dequeue (MAT_MTYPE_QUEUE *s*)

6.35.2.193 void mat_mtype_queue_enqueue (MAT_MTYPE_QUEUE *s*, mtype *value*)

6.35.2.194 int mat_mtype_queue_free (MAT_MTYPE_QUEUE *s*)

6.35.2.195 int mat_mtype_queue_is_empty (MAT_MTYPE_QUEUE *s*)

6.35.2.196 MAT_MTYPE_STACK mat_mtype_stack_creat (void)

6.35.2.197 int mat_mtype_stack_free (MAT_MTYPE_STACK *s*)

6.35.2.198 int mat_mtype_stack_is_empty (MAT_MTYPE_STACK *s*)

6.35.2.199 mtype mat_mtype_stack_pop (MAT_MTYPE_STACK *s*)

6.35.2.200 void mat_mtype_stack_push (MAT_MTYPE_STACK *s*, mtype *value*)

6.35.2.201 MATRIX mat_mul (MATRIX *A*, MATRIX *B*, MATRIX *result*)

6.35.2.202 MATRIX mat_mul_dot (MATRIX *A*, MATRIX *B*, MATRIX *result*)

6.35.2.203 MATRIX mat_mul_fast (MATRIX *A*, MATRIX *B*, MATRIX *result*)

6.35.2.204 MATRIX mat_muls (MATRIX *A*, mtype *s*, MATRIX *result*)

6.35.2.205 void mat_nextline (void)

Prints nextline to stdout.

6.35.2.206 mtype mat_norm_inf (MATRIX *A*)

6.35.2.207 mtype mat_norm_one (MATRIX *A*)

6.35.2.208 mtype mat_norm_p (MATRIX *A*, mtype *p*)

6.35.2.209 MATSTACK mat_omp (MATRIX *A*, MATRIX *b*, int *k*, mtype *tol*, MATSTACK *result*)

6.35.2.210 mtype mat_order_statistic (MATRIX *A*, int *k*)

Computes the k^{th} order statistic of elements of a given matrix.

Parameters

in	*A*	Input matrix
in	*k*	Order

Returns

$\mathrm{O}_k\left(\{a_{ij}\}\right)$

6.35.2.211 MATSTACK mat_pca (MATRIX *data*, int *pca_type*)

6.35.2.212 MAT_PERCEPTRON mat_perceptron_creat (void)

Creates a perceptron.

Returns

Output perceptron

6.35.2.213 int mat_perceptron_free (MAT_PERCEPTRON *a*)

Frees a perceptron.

Parameters

in	*a*	Input perceptron

Returns

Success

6.35.2.214 **INT_VECTOR** mat_perceptron_test (**MATRIX** *data,* **MAT_PERCEPTRON** *p_model*)

6.35.2.215 **MAT_PERCEPTRON** mat_perceptron_train (**MATRIX** *data,* **INT_VECTOR** *labels,* int *num_of_iterations*)

6.35.2.216 **MAT_PERCEPTRON** mat_perceptron_train_ (**MATRIX** *data1,* **MATRIX** *data2,* **MAT_PERCEPTRON** *p_model,* int *class_num*)

6.35.2.217 **MATRIX** mat_pick_col (**MATRIX** *A,* int *c,* **MATRIX** *result*)

Picks a column from a matrix.

Parameters

in	*A*	Input matrix
in	*r*	Column index
in	*result*	Matrix to store the result

Returns

Column matrix

6.35.2.218 **MATRIX** mat_pick_row (**MATRIX** *A,* int *r,* **MATRIX** *result*)

Picks a row from a matrix.

Parameters

in	*A*	Input matrix
in	*r*	Row index
in	*result*	Matrix to store the result

Returns

Row matrix

6.35.2.219 **MATRIX** mat_pinv (**MATRIX** *A,* **MATRIX** *result*)

Computes pseudo-inverse of a matrix.

Parameters

in	*A*	Input matrix
in	*result*	Matrix to store the result

Returns

$\left(A^T A\right)^{-1} A^T$

6.35.2.220 **MATRIX** mat_pol2cart (**MATRIX** *A,* int *dim,* **MATRIX** *result*)

Converts polar co-ordinates to Cartesian co-ordinates.

Parameters

in	*A*	Input matrix
in	*dim*	Data order ROWS/COLS

Returns

Cartesian co-ordinate matrix

6.35.2.221 MATRIX mat_poly_add (MATRIX *A*, MATRIX *B*, MATRIX *result*)

Adds two polynomials.

Parameters

`in`	*A*	First input polynomial matrix
`in`	*B*	Second input polynomial matrix
`in`	*result*	Matrix to store the result

Returns

Output matrix

6.35.2.222 MATRIX mat_poly_diff (MATRIX *A*, int *dir*, MATRIX *result*)

Computes derivative polynomial of a polynomial.

Parameters

`in`	*A*	Input polynomial matrix
`in`	*dir*	Direction (ROWS/COLS)
`in`	*result*	Matrix to store the result

Returns

Output matrix

6.35.2.223 MATRIX mat_poly_diff_eval (MATRIX *A*, mtype *x*, int *dir*, MATRIX *result*)

Evaluates derivative polynomial at a point.

Parameters

`in`	*A*	Input polynomial matrix
`in`	*x*	Value at which to evaluate the derivative
`in`	*dir*	Direction (ROWS/COLS)
`in`	*result*	Matrix to store the result

Returns

Output matrix

6.35.2.224 MATSTACK mat_poly_div (MATRIX *A*, MATRIX *B*, MATSTACK *result*)

Divides two polynomials.

Parameters

in	*A*	First input polynomial matrix
in	*B*	Second input polynomial matrix
in	*result*	Matrix to store the result

Returns

Output matrix

6.35.2.225 MATRIX mat_poly_eval (MATRIX *A*, mtype *x*, int *dir*, MATRIX *result*)

Evaluates polynomial at a point.

Parameters

in	*A*	Input polynomial matrix
in	*x*	Value at which to evaluate
in	*dir*	Direction (ROWS/COLS)
in	*result*	Matrix to store the result

Returns

Output matrix

6.35.2.226 MATRIX mat_poly_mul (MATRIX *A*, MATRIX *B*, MATRIX *result*)

Multiplies two polynomials.

Parameters

in	*a*	First input polynomial matrix
in	*b*	Second input polynomial matrix
in	*result*	Matrix to store the result

Returns

Output matrix

6.35.2.227 MATRIX mat_poly_scale (MATRIX *A*, mtype *s*, MATRIX *result*)

Multiplies a polynomial with a scalar.

Parameters

in	*A*	Input polynomial matrix
in	*s*	Scalar
in	*result*	Matrix to store the result

Returns

Output matrix

6.35.2.228 MATRIX mat_poly_shift (MATRIX *A*, int *s*, MATRIX *result*)

Shifts a polynomial.

Parameters

in	*A*	Input polynomial matrix
in	*s*	Scalar shift
in	*result*	Matrix to store the result

Returns

Output matrix

6.35.2.229 MATSTACK mat_qr (MATRIX *A*, MATSTACK *qr*)

Computes QR decomposition.

Parameters

in	*A*	Input matrix
in	*qr*	Matrix stack to store result

Returns

Output QR Matrix stack

6.35.2.230 MATSTACK mat_qsort (MATRIX *A*, int *dim*, MATSTACK *result*)

Sorts elements of a given matrix.

Parameters

in	*A*	Input matrix
in	*dim*	Direction of sort (ROWS/COLS)
out	*result*	Output matrix stack

Returns

Output matrix stack of sorted A and their positions

6.35.2.231 MATRIX mat_rand (int *r*, int *c*, MATRIX *result*)

6.35.2.232 MATRIX mat_randexp (int *r*, int *c*, mtype *mu*, MATRIX *result*)

6.35.2.233 MATRIX mat_randfun (int *r*, int *c*, mtype(*)(mtype) *fun*, mtype *xmin*, mtype *xmax*, MATRIX *result*)

6.35.2.234 MATRIX mat_randn (int *r*, int *c*, MATRIX *result*)

6.35.2.235 MATRIX mat_randperm (int *m*, int *n*, MATRIX *result*)

6.35.2.236 MATRIX mat_randperm_n (int *n*, MATRIX *result*)

6.35.2.237 int mat_read_word (MAT_FILEPOINTER *fp*, char ∗ *c_word*)

Reads current word from an opened file.

Parameters

in	*fp*	Pointer to an opened file
out	*c_word*	Pointer to word read

Returns

EOF reached

6.35.2.238 MATRIX mat_reg_inv (MATRIX *A*, mtype *r*, MATRIX *result*)

Computes the regularized inverse of a matrix.

Parameters

in	*A*	Input matrix
in	*r*	Regularizing constant
in	*result*	Matrix to store the result

Returns

$(A+rI)^{-1}$

6.35.2.239 MATRIX mat_rob_least_squares (MATRIX *A*, MATRIX *Y*, int *lossfunc*, MATRIX *result*)

6.35.2.240 MATRIX mat_robust_fit (MATRIX *A*, MATRIX *Y*, int *deg*, int *lossfunc*, MATRIX *result*)

6.35.2.241 MATRIX mat_rowcopy (MATRIX *A*, int *rowa*, int *rowb*, MATRIX *result*)

Copies a row from a matrix.

Parameters

in	*A*	Input matrix
in	*rowa*	Source row
in	*rowb*	Destination row
in	*result*	Matrix to store the result

Returns

Copied matrix

6.35.2.242 MATRIX mat_scpcol (MATRIX *data*)

6.35.2.243 void mat_set_seed (int *seed*)

6.35.2.244 MATRIX mat_sub (MATRIX *A*, MATRIX *B*, MATRIX *result*)

6.35.2.245 MATRIX mat_submat (MATRIX *A*, int *i*, int *j*, MATRIX *result*)

Deletes a row and a column of a matrix.

Parameters

in	*A*	Input matrix
in	*i*	Row index
in	*j*	Column index
in	*result*	Matrix to store the result

Returns

Extracted matrix

6.35.2.246 MATRIX mat_subs (MATRIX *A*, mtype *s*, MATRIX *result*)

6.35.2.247 mtype mat_sum (MATRIX *A*)

6.35.2.248 MATRIX mat_sum_col (MATRIX *A*, MATRIX *result*)

6.35.2.249 MATRIX mat_sum_row (MATRIX *A*, MATRIX *result*)

6.35.2.250 MATRIX mat_symtoeplz (MATRIX *R*, MATRIX *result*)

6.35.2.251 void mat_tic (void)

Starts stopwatch timer.

6.35.2.252 double mat_toc (void)

Computes elapsed time from last start of timer.

Returns

Elapsed time

6.35.2.253 void mat_toc_print (void)

Computes and prints elapsed time from last start of timer on the stdout.

6.35.2.254 void mat_tqli (MATRIX *d*, MATRIX *e*, MATRIX *z*)

6.35.2.255 MATRIX mat_tran (MATRIX *A*, MATRIX *result*)

Computes the transpose of a matrix.

Parameters

in	*A*	Input matrix

Returns

A^T

6.35.2.256 void mat_tred2 (MATRIX *a*, MATRIX *d*, MATRIX *e*)

6.35.2.257 MATRIX mat_vectorize (MATRIX *A*, MATRIX *result*)

Reshapes a matrix to a vector.

Parameters

in	*A*	Input matrix
in	*result*	Matrix to store the result

Returns

$vec(\mathbf{A})$

6.35.2.258 MATRIX mat_vectorize_tr (MATRIX *A*, MATRIX *result*)

Reshapes transpose of a matrix to a vector.

Parameters

in	*A*	Input matrix
in	*result*	Matrix to store the result

Returns

$vec(\mathbf{A}^T)$

6.35.2.259 MATRIX mat_w_least_squares (MATRIX *A*, MATRIX *Y*, MATRIX *w*, MATRIX *result*)

6.35.2.260 MATRIX mat_wpinv (MATRIX *A*, MATRIX *w*, MATRIX *result*)

Computes weighted pseudo-inverse of a matrix.

Parameters

in	*A*	Input matrix
in	*w*	Weight matrix
in	*result*	Matrix to store the result

Returns

$\left(A^T W A\right)^{-1} A^T W$

6.35.2.261 MATRIX mat_xcopy (MATRIX *A*, int *si*, int *ei*, int *sj*, int *ej*, MATRIX *result*)

Copies a sub-matrix.

Parameters

in	*A*	Input matrix
in	*si*	Start of first index, s_i
in	*ei*	End of first index, e_i
in	*sj*	Start of second index, s_j
in	*ej*	End of second index, e_j
in	*result*	Matrix to store the result

Returns

Extracted matrix $A_{s_i:e_i,s_j:e_j}$

6.35.2.262 MATRIX mat_xjoin (MATRIX *A11*, MATRIX *A12*, MATRIX *A21*, MATRIX *A22*, MATRIX *result*)

Copies a sub-matrix.

Parameters

in	*A11*	Input matrix, A_{11}
in	*A12*	Input matrix, A_{12}
in	*A21*	Input matrix, A_{21}
in	*A22*	Input matrix, A_{22}
in	*result*	Matrix to store the result

Returns

Block matrix $\left[\begin{array}{cc} A_{11} & A_{12} \\ A_{21} & A_{22} \end{array}\right]$

6.35.2.263 int mats_isinf (mtype *x*)

Checks if scalar is infinite.

Parameters

in	*x*	Input scalar

Returns

Zero/non-zero

6.35.2.264 int mats_isnan (mtype *x*)

Checks if scalar is NaN.

Parameters

in	*x*	Input scalar

Returns

Zero/non-zero

6.35.2.265 MATSTACK matstack_append (MATSTACK *s*, MATRIX *A*)

Appends a matrix to a matrix stack.

Parameters

in	*s*	Input matrix stack
in	*A*	Input matrix to append

Returns

Output matrix stack

6.35.2.266 MATSTACK matstack_creat (int *len*)

Creates a matrix stack.

Parameters

in	*len*	Length of the stack

Returns

Output matrix stack

6.35.2.267 MATSTACK matstack_error (int *err_*)

Generates error message for matrix stack errors and exits.

Parameters

in	*err*	Error type (MATSTACK_MALLOC/MATSTACK_FNOTOPEN/MATSTACK_-FNOTGETMAT/MATSTACK_SIZEMISMATCH/ MATSTACK_INVERSE_ERROR)

6.35.2.268 int matstack_free (MATSTACK *A*)

Frees a matrix stack.

Parameters

in	*A*	Input matrix stack

Returns

Success

6.35.2.269 MATVEC_DPOINTER matvec_creat (void)

Creates a matrix-vector pair.

Returns

Output matrix-vector pair

6.35.2.270 int matvec_free (MATVEC_DPOINTER *a*)

Frees a matrix-vector pair.

Parameters

in	*a*	Input matrix-vector pair

Returns

Success

6.35.2.271 int pq_error (int *err_*)

Generates error message for priority queue errors and exits.

Parameters

in	*err*	Error type (PQ_MALLOC/PQ_EMPTY)

6.35.2.272 int queue_error (int *err_*)

Generates error message for queue errors and exits.

Parameters

in	*err*	Error type (QUEUE_MALLOC/QUEUE_EMPTY)

6.35.2.273 int stack_error (int *err_*)

Generates error message for stack errors and exits.

Parameters

in	*err*	Error type (STACK_MALLOC/STACK_EMPTY)

6.35.3 Variable Documentation

6.35.3.1 MATSTACK mat_binom_series_table

6.35.3.2 MATSTACK mat_cheby_series_table

6.35.3.3 clock_t MAT_CLOCK_TIME

6.35.3.4 MATSTACK mat_legendre_series_table

6.35.3.5 unsigned int MAT_SEED

6.35.3.6 int MAT_SET_SEED

6.36 matsearch.c File Reference

Functions

- INT_VECTOR int_vec_find (INT_VECTOR a, int rel_type, int n)
- INT_VECSTACK mat_find (MATRIX A, int rel_type, mtype x)

6.36.1 Function Documentation

6.36.1.1 INT_VECTOR int_vec_find (INT_VECTOR *a*, int *rel_type*, int *n*)

6.36.1.2 INT_VECSTACK mat_find (MATRIX *A*, int *rel_type*, mtype *x*)

6.37 matsolve.c File Reference

Functions

- int mat_lu (MATRIX A, MATRIX P)
- void mat_backsubs1 (MATRIX A, MATRIX B, MATRIX X, MATRIX P, int xcol)
- MATRIX mat_lsolve (MATRIX A, MATRIX b, MATRIX result)
- MATRIX mat_cholesky (MATRIX A, MATRIX result)

  Computes Cholesky factor of a matrix.
- MATRIX mat_conjgrad (MATRIX A, MATRIX b, MATRIX x0, mtype tol, int miters, MATRIX result)

  Solves a linear system with conjugate gradients method.

6.37.1 Function Documentation

6.37.1.1 void mat_backsubs1 (MATRIX *A*, MATRIX *B*, MATRIX *X*, MATRIX *P*, int *xcol*)

6.37.1.2 MATRIX mat_cholesky (MATRIX *A*, MATRIX *result*)

Computes Cholesky factor of a matrix.

Parameters

in	*A*	Input matrix
in	*result*	Matrix to store the result

Returns

Cholesky factor

6.37.1.3 MATRIX mat_conjgrad (MATRIX *A*, MATRIX *b*, MATRIX *x0*, mtype *tol*, int *miters*, MATRIX *result*)

Solves a linear system with conjugate gradients method.

Parameters

in	*A*	Input matrix
in	*b*	Observed matrix
in	*result*	Matrix to store the result

Returns

x

6.37.1.4 MATRIX mat_lsolve (MATRIX *A*, MATRIX *b*, MATRIX *result*)

6.37.1.5 int mat_lu (MATRIX *A*, MATRIX *P*)

6.38 matsort.c File Reference

Functions

- mtype mat_median (MATRIX A)

    Computes the median of elements of a given matrix.
- mtype mat_order_statistic (MATRIX A, int k)

    Computes the k^{th} order statistic of elements of a given matrix.
- MATSTACK mat_qsort (MATRIX A, int dim, MATSTACK result)

    Sorts elements of a given matrix.

6.38.1 Function Documentation

6.38.1.1 mtype mat_median (MATRIX *A*)

Computes the median of elements of a given matrix.

Parameters

in	*A*	Input matrix

Returns

$\text{median}\left(\{a_{ij}\}\right)$

6.38.1.2 mtype mat_order_statistic (MATRIX *A*, int *k*)

Computes the k^{th} order statistic of elements of a given matrix.

Parameters

in	*A*	Input matrix
in	*k*	Order

Returns

$\mathrm{O}_k\left(\{a_{ij}\}\right)$

6.38.1.3 MATSTACK mat_qsort (MATRIX *A*, int *dim*, MATSTACK *result*)

Sorts elements of a given matrix.

Parameters

in	*A*	Input matrix
in	*dim*	Direction of sort (ROWS/COLS)
out	*result*	Output matrix stack

Returns

Output matrix stack of sorted A and their positions

6.39 matstdrels.c File Reference

Functions

- int gen_gt (mtype a)
- int gen_lt (mtype a)
- int gen_eq (mtype a)
- mtype gen_abs_ceil (mtype a)

6.39.1 Function Documentation

6.39.1.1 mtype gen_abs_ceil (mtype *a*)

6.39.1.2 int gen_eq (mtype *a*)

6.39.1.3 int gen_gt (mtype *a*)

6.39.1.4 int gen_lt (mtype *a*)

6.40 matsub.c File Reference

Functions

- MATRIX mat_sub (MATRIX A, MATRIX B, MATRIX result)
- MATRIX mat_subs (MATRIX A, mtype s, MATRIX result)
- INT_VECTOR int_vec_sub (INT_VECTOR A, INT_VECTOR B, INT_VECTOR result)
- INT_VECTOR int_vec_subs (INT_VECTOR A, int x, INT_VECTOR result)

6.40.1 Function Documentation

6.40.1.1 INT_VECTOR int_vec_sub (INT_VECTOR *A,* INT_VECTOR *B,* INT_VECTOR *result*)

6.40.1.2 INT_VECTOR int_vec_subs (INT_VECTOR *A,* int *x,* INT_VECTOR *result*)

6.40.1.3 MATRIX mat_sub (MATRIX *A,* MATRIX *B,* MATRIX *result*)

6.40.1.4 MATRIX mat_subs (MATRIX *A,* mtype *s,* MATRIX *result*)

6.41 matsubx.c File Reference

Functions

- MATRIX mat_submat (MATRIX A, int i, int j, MATRIX result)

  Deletes a row and a column of a matrix.

6.41.1 Function Documentation

6.41.1.1 MATRIX mat_submat (MATRIX *A,* int *i,* int *j,* MATRIX *result*)

Deletes a row and a column of a matrix.

Parameters

in	*A*	Input matrix
in	*i*	Row index
in	*j*	Column index
in	*result*	Matrix to store the result

Returns

Extracted matrix

6.42 matsum.c File Reference

Functions

- mtype mat_sum (MATRIX A)
- MATRIX mat_sum_row (MATRIX A, MATRIX result)
- MATRIX mat_sum_col (MATRIX A, MATRIX result)

6.42.1 Function Documentation

6.42.1.1 mtype mat_sum (MATRIX *A*)

6.42.1.2 MATRIX mat_sum_col (MATRIX *A*, MATRIX *result*)

6.42.1.3 MATRIX mat_sum_row (MATRIX *A*, MATRIX *result*)

6.43 mattext.c File Reference

Functions

- int mat_isnumeric (MAT_FILEPOINTER fp)

  Checks if current word in an opened file is numeric or not.
- int mat_go_next_word (MAT_FILEPOINTER fp)

  Moves to next word in an opened file.
- int mat_count_words_in_line (MAT_FILEPOINTER fp, int ∗count)

  Count words in current line in an opened file.
- MATRIX mat_dlmread (const char ∗fname)

  Reads a matrix from a file.
- int mat_read_word (MAT_FILEPOINTER fp, char ∗c_word)

  Reads current word from an opened file.
- void mat_dlmwrite (const char ∗fname, MATRIX A)

  Writes a matrix to a file.

6.43.1 Function Documentation

6.43.1.1 int mat_count_words_in_line (MAT_FILEPOINTER *fp*, int ∗ *count*)

Count words in current line in an opened file.

Parameters

in	*fp*	Pointer to an opened file
out	*count*	Pointer to output count

Returns

EOF reached

6.43.1.2 MATRIX mat_dlmread (const char * *fname*)

Reads a matrix from a file.

Parameters

in	*fname*	Filename to read from

Returns

Output matrix

6.43.1.3 void mat_dlmwrite (const char * *fname*, MATRIX *A*)

Writes a matrix to a file.

Parameters

in	*fname*	Filename to write into
in	*A*	Input matrix

6.43.1.4 int mat_go_next_word (MAT_FILEPOINTER *fp*)

Moves to next word in an opened file.

Parameters

in	*fp*	Pointer to an opened file

Returns

EOF reached

6.43.1.5 int mat_isnumeric (MAT_FILEPOINTER *fp*)

Checks if current word in an opened file is numeric or not.

Parameters

in	*fp*	Pointer to an opened file

Returns

Zero/non-zero

6.43.1.6 int mat_read_word (MAT_FILEPOINTER *fp*, char * *c_word*)

Reads current word from an opened file.

Parameters

in	*fp*	Pointer to an opened file
out	*c_word*	Pointer to word read

Returns

EOF reached

6.44 mattimers.c File Reference

Functions

- void mat_tic (void)

  Starts stopwatch timer.
- double mat_toc (void)

  Computes elapsed time from last start of timer.
- void mat_toc_print (void)

  Computes and prints elapsed time from last start of timer on the stdout.

Variables

- clock_t MAT_CLOCK_TIME

6.44.1 Function Documentation

6.44.1.1 void mat_tic (void)

Starts stopwatch timer.

6.44.1.2 double mat_toc (void)

Computes elapsed time from last start of timer.

Returns

Elapsed time

6.44.1.3 void mat_toc_print (void)

Computes and prints elapsed time from last start of timer on the stdout.

6.44.2 Variable Documentation

6.44.2.1 clock_t MAT_CLOCK_TIME

6.45 mattoepz.c File Reference

Functions

- MATRIX mat_symtoeplz (MATRIX R, MATRIX result)

6.45.1 Function Documentation

6.45.1.1 MATRIX mat_symtoeplz (MATRIX *R*, MATRIX *result*)

6.46 mattran.c File Reference

Functions

- MATRIX mat_tran (MATRIX A, MATRIX result)

  Computes the transpose of a matrix.

6.46.1 Function Documentation

6.46.1.1 MATRIX mat_tran (MATRIX *A*, MATRIX *result*)

Computes the transpose of a matrix.

Parameters

in	*A*	Input matrix

Returns

A^T

6.47 README.md File Reference

Index