MatrixLab

1.3

Generated by Doxygen 1.8.13

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

![alt tag](https://raw.githubusercontent.com/mohammadul/matrixlab/master/matrixlab.ico) Matrixlab

A C Matrix Library Originally adapted from Small Matrix Toolbox for C programmers, ver. 0.4 by Patrick Ko Shu-pui

- Matrixlab is a generic C library for matrix routines.
- It contains over 250 functions for matrix operations.
- Many of the functions are multi-threaded.
  ** For more details and updates, visit http://mohammadulhaque.alotspace.com.

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 class_covars

```
MATSTACK mat_bayes_model::class_covars
```

Training data class covariances

5.1.2.2 class_labels

```
INT_VECTOR mat_bayes_model::class_labels
```

Training data class label vector

5.1.2.3 class_means

```
MATSTACK mat_bayes_model::class_means
```

Training data class means

5.1.2.4 class_priors

```
MATRIX mat_bayes_model::class_priors
```

Training data prior information

5.1.2.5 num_of_classes

```
int mat_bayes_model::num_of_classes
```

Number of training class

5.1.2.6 num_of_features

```
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>
```

Collaboration diagram for mat_gnode:

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 next

```
struct mat_gnode* mat_gnode::next
```

Pointer to next node

5.2.2.2 v

```
int mat_gnode::v
```

Value

5.2.2.3 weight

```
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>
```

Collaboration diagram for mat_graph:

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 adj

```
MAT_GNODE* mat_graph::adj
```

5.3.2.2 dad

```
int* mat_graph::dad
```

5.3.2.3 id

```
int mat_graph::id
```

5.3.2.4 nedges

```
int mat_graph::nedges
```

Number of edges

5.3.2.5 nvertices

```
int mat_graph::nvertices
```

Number of vertices

5.3.2.6 pq

```
MAT_INT_PRIORITYQUEUE mat_graph::pq
```

5.3.2.7 val

```
int* mat_graph::val
```

5.3.2.8 vseq

```
int* mat_graph::vseq
```

5.3.2.9 weighted

```
int mat_graph::weighted
```

5.3.2.10 z

```
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>
```

Collaboration diagram for mat_int_priorityqueue:

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 element

```
MAT_INTPQNODE mat_int_priorityqueue::element
```

Pointer to priority queue data

5.4.2.2 length

```
int mat_int_priorityqueue::length
```

Total allocated priority queue length

5.4.2.3 p

```
int mat_int_priorityqueue::p
```

Current priority queue position

5.4.2.4 type

```
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>
```

Collaboration diagram for mat_int_queue:

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 head

`MAT_QINTNODE mat_int_queue::head`

Queue head node

5.5.2.2 p

`int mat_int_queue::p`

Current queue position

5.5.2.3 tail

`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 length

```
int mat_int_stack::length
```

Total allocated stack length

5.6.2.2 p

```
int mat_int_stack::p
```

Current stack position

5.6.2.3 stack

```
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 data

```
int mat_intpqnode::data
```

Integer node data

5.7.2.2 priority

```
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>
```

Collaboration diagram for mat_kdnode:

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 idx

```
int mat_kdnode::idx
```

5.8.1.2 left

```
struct mat_kdnode* mat_kdnode::left
```

5.8.1.3 right

```
struct mat_kdnode * mat_kdnode::right
```

5.8.1.4 x

```
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>
```

Collaboration diagram for mat_kdtree:

Data Fields

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

5.9.1 Field Documentation

5.9.1.1 _is_allocated

```
int mat_kdtree::_is_allocated
```

5.9.1.2 data

```
MAT_KDNODE mat_kdtree::data
```

5.9.1.3 kdtree

```
MAT_KDNODE mat_kdtree::kdtree
```

5.9.1.4 length

```
int mat_kdtree::length
```

5.9.1.5 ndims

```
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>
```

Collaboration diagram for mat_mtype_priorityqueue:

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 element

```
MAT_MTYPEPQNODE mat_mtype_priorityqueue::element
```

Pointer to priority queue data

5.10.2.2 length

```
int mat_mtype_priorityqueue::length
```

Total allocated priority queue length

5.10.2.3 p

```
int mat_mtype_priorityqueue::p
```

Current priority queue position

5.10.2.4 type

```
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>
```

Collaboration diagram for mat_mtype_queue:

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 head

`MAT_QMTYPENODE mat_mtype_queue::head`

Queue head node

5.11.2.2 p

`int mat_mtype_queue::p`

Current queue position

5.11.2.3 tail

`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 length

```
int mat_mtype_stack::length
```

Total allocated stack length

5.12.2.2 p

```
int mat_mtype_stack::p
```

Current stack position

5.12.2.3 stack

```
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 data

```
mtype mat_mtypepqnode::data
```

Mtype node data

5.13.2.2 priority

```
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 class_labels

```
INT_VECTOR mat_perceptron::class_labels
```

Training data class label vector

5.14.2.2 class_weights

```
MATRIX mat_perceptron::class_weights
```

Trained Classifier Weights

5.14.2.3 istrained

```
int mat_perceptron::istrained
```

Is trained

5.14.2.4 num_of_classes

```
int mat_perceptron::num_of_classes
```

Number of training classes

5.14.2.5 num_of_features

```
int mat_perceptron::num_of_features
```

Number of training features

5.14.2.6 num_of_iterations

```
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>
```

Collaboration diagram for mat_qintnode:

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 data

```
int mat_qintnode::data
```

Integer node data

5.15.2.2 next

```
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>
```

Collaboration diagram for mat_qmtypenode:

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 data

```
mtype mat_qmtypenode::data
```

Mtype node data

5.16.2.2 next

```
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>
```

Collaboration diagram for mat_tree_node:

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 element

```
mtype mat_tree_node::element
```

Search tree node data

5.17.2.2 left

```
struct mat_tree_node* mat_tree_node::left
```

Pointer to left child node

5.17.2.3 right

```
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.
- INT_VECTOR int_vec_abs (INT_VECTOR A, INT_VECTOR result)

  Computes absolute value of an integer vector.

6.1.1 Function Documentation

6.1.1.1 int_vec_abs()

```
INT_VECTOR int_vec_abs (
            INT_VECTOR A,
            INT_VECTOR result )
```

Computes absolute value of an integer vector.

Parameters

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

Returns

$\text{abs}(A)$

Here is the call graph for this function:

6.1.1.2 mat_abs()

```
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

$\mathrm{abs}(\mathbf{A})$

Here is the call graph for this function: Here is the caller graph for this function:

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_vec_add()

```
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

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

Here is the call graph for this function:

6.2.1.2 int_vec_adds()

```
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

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

Here is the call graph for this function:

6.2.1.3 mat_add()

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

Adds two matrices.

Parameters

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

Returns

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

Here is the call graph for this function: Here is the caller graph for this function:

6.2.1.4 mat_adds()

```
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

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

Here is the call graph for this function: Here is the caller graph for this function:

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_vec_concat()

```
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	*result*	Vector to store the result

Returns

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

Here is the call graph for this function:

6.4.1.2 mat_concat()

```
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 $\begin{bmatrix} A \\ B \end{bmatrix}$

Here is the call graph for this function: Here is the caller graph for this function:

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 int_vec2_mat()

```
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

Here is the call graph for this function:

6.5.1.2 mat_2int_vec()

```
INT_VECTOR mat_2int_vec (
             MATRIX A )
```

Converts a matrix to an integer vector.

Parameters

in	*A*	Input matrix
in	*v*	Vector to store the result

Returns

Output vector

Here is the call graph for this function: Here is the caller graph for this function:

6.5.1.3 mat_vectorize()

```
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

$vec(\mathbf{A})$

Here is the call graph for this function:

6.5.1.4 mat_vectorize_tr()

```
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

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

Here is the call graph for this function: Here is the caller graph for this function:

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_vec_append()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.6.1.2 int_vec_copy()

```
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

Here is the call graph for this function:

6.6.1.3 int_vec_creat()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.6.1.4 int_vec_fill()

```
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_vec_fill_type()

```
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

Here is the caller graph for this function:

6.6.1.6 int_vec_free()

```
int int_vec_free (
             INT_VECTOR A )
```

Frees an integer vector.

Parameters

in	*A*	Input vector

Returns

Success

Here is the call graph for this function: Here is the caller graph for this function:

6.6.1.7 int_vecstack_creat()

```
INT_VECSTACK int_vecstack_creat (
             int len )
```

Creates an integer vector stack.

Parameters

in	*len*	Length of the stack

Returns

Output vector stack

Here is the call graph for this function: Here is the caller graph for this function:

6.6.1.8 int_vecstack_free()

```
int int_vecstack_free (
             INT_VECSTACK A )
```

Frees an integer vector stack.

Parameters

in	*A*	Input vector stack

Returns

Success

Here is the call graph for this function: Here is the caller graph for this function:

6.6.1.9 mat_bayes_model_creat()

```
MAT_BAYES_MODEL mat_bayes_model_creat (
             void )
```

Creates a Bayes model.

Returns

Output Bayes model

Here is the caller graph for this function:

6.6.1.10 mat_bayes_model_free()

```
int mat_bayes_model_free (
            MAT_BAYES_MODEL a )
```

Frees a Bayes model.

Parameters

in	*a*	Input Bayes model

Returns

Success

Here is the call graph for this function:

6.6.1.11 mat_colcopy()

```
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

Here is the caller graph for this function:

6.6.1.12 mat_copy()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.6.1.13 mat_creat()

```
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

Here is the call graph for this function:

6.6.1.14 mat_creat_diag()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.6.1.15 mat_fgetmat()

```
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 mat_fill()

```
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

Here is the caller graph for this function:

6.6.1.17 mat_fill_type()

```
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

Here is the caller graph for this function:

6.6.1.18 mat_free()

```
int mat_free (
            MATRIX A )
```

Frees a matrix.

Parameters

in	*A*	Input matrix

Returns

Success

Here is the call graph for this function: Here is the caller graph for this function:

6.6.1.19 mat_perceptron_creat()

```
MAT_PERCEPTRON mat_perceptron_creat (
            void )
```

Creates a perceptron.

Returns

Output perceptron

Here is the caller graph for this function:

6.6.1.20 mat_perceptron_free()

```
int mat_perceptron_free (
            MAT_PERCEPTRON a )
```

Frees a perceptron.

Parameters

in	*a*	Input perceptron

Returns

Success

Here is the call graph for this function:

6.6.1.21 mat_rowcopy()

```
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 mat_xcopy()

```
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 $A_{s_i:e_i,s_j:e_j}$

Here is the call graph for this function: Here is the caller graph for this function:

6.6.1.23 mat_xjoin()

```
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 $\left[\begin{array}{cc} A_{11} & A_{12} \\ A_{21} & A_{22} \end{array}\right]$

Here is the call graph for this function: Here is the caller graph for this function:

6.6.1.24 matstack_append()

```
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

Here is the call graph for this function:

6.6.1.25 matstack_creat()

```
MATSTACK matstack_creat (
             int len )
```

Creates a matrix stack.

Parameters

in	*len*	Length of the stack

Returns

Output matrix stack

Here is the call graph for this function: Here is the caller graph for this function:

6.6.1.26 matstack_free()

```
int matstack_free (
             MATSTACK A )
```

Frees a matrix stack.

Parameters

in	*A*	Input matrix stack

Returns

Success

Here is the call graph for this function: Here is the caller graph for this function:

6.6.1.27 matvec_creat()

```
MATVEC_DPOINTER matvec_creat (
             void  )
```

Creates a matrix-vector pair.

Returns

Output matrix-vector pair

Here is the caller graph for this function:

6.6.1.28 matvec_free()

```
int matvec_free (
             MATVEC_DPOINTER a )
```

Frees a matrix-vector pair.

Parameters

in	*a*	Input matrix-vector pair

Returns

Success

Here is the call graph for this function: Here is the caller graph for this function:

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_bs_delete()

```
MAT_TREE mat_bs_delete (
            mtype x,
            MAT_TREE T )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.7.1.2 mat_bs_find()

```
MAT_TREE mat_bs_find (
            mtype x,
            MAT_TREE T )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.7.1.3 mat_bs_find_max()

```
MAT_TREE mat_bs_find_max (
            MAT_TREE T )
```

6.7.1.4 mat_bs_find_min()

```
MAT_TREE mat_bs_find_min (
            MAT_TREE T )
```

Here is the caller graph for this function:

6.7.1.5 mat_bs_free()

```
MAT_TREE mat_bs_free (
            MAT_TREE T )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.7.1.6 mat_bs_inorder()

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

Here is the call graph for this function: Here is the caller graph for this function:

6.7.1.7 mat_bs_insert()

```
MAT_TREE mat_bs_insert (
            mtype x,
            MAT_TREE T )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.7.1.8 mat_bs_make_null()

```
MAT_TREE mat_bs_make_null (
            void )
```

Here is the caller graph for this function:

6.7.1.9 mat_int_priorityqueue_creat()

```
MAT_INT_PRIORITYQUEUE mat_int_priorityqueue_creat (
            int type )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.7.1.10 mat_int_priorityqueue_dequeue()

```
mat_intpqnode mat_int_priorityqueue_dequeue (
            MAT_INT_PRIORITYQUEUE H )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.7.1.11 mat_int_priorityqueue_enqueue()

```
void mat_int_priorityqueue_enqueue (
            MAT_INT_PRIORITYQUEUE H,
            int data,
            int priority )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.7.1.12 mat_int_priorityqueue_free()

```
int mat_int_priorityqueue_free (
            MAT_INT_PRIORITYQUEUE H )
```

Here is the caller graph for this function:

6.7.1.13 mat_int_priorityqueue_is_empty()

```
int mat_int_priorityqueue_is_empty (
            MAT_INT_PRIORITYQUEUE H )
```

6.7.1.14 mat_int_priorityqueue_update()

```
int mat_int_priorityqueue_update (
            MAT_INT_PRIORITYQUEUE H,
            int data,
            int priority,
            int type )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.7.1.15 mat_int_queue_creat()

```
MAT_INT_QUEUE mat_int_queue_creat (
            void )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.7.1.16 mat_int_queue_dequeue()

```
int mat_int_queue_dequeue (
            MAT_INT_QUEUE s )
```

Here is the call graph for this function:

6.7.1.17 mat_int_queue_enqueue()

```
void mat_int_queue_enqueue (
            MAT_INT_QUEUE s,
            int value )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.7.1.18 mat_int_queue_free()

```
int mat_int_queue_free (
            MAT_INT_QUEUE s )
```

6.7.1.19 mat_int_queue_is_empty()

```
int mat_int_queue_is_empty (
            MAT_INT_QUEUE s )
```

6.7.1.20 mat_int_stack_creat()

```
MAT_INT_STACK mat_int_stack_creat (
            void )
```

Here is the call graph for this function:

6.7.1.21 mat_int_stack_free()

```
int mat_int_stack_free (
            MAT_INT_STACK s )
```

6.7.1.22 mat_int_stack_is_empty()

```
int mat_int_stack_is_empty (
            MAT_INT_STACK s )
```

6.7.1.23 mat_int_stack_pop()

```
int mat_int_stack_pop (
            MAT_INT_STACK s )
```

Here is the call graph for this function:

6.7.1.24 mat_int_stack_push()

```
void mat_int_stack_push (
            MAT_INT_STACK s,
            int value )
```

Here is the call graph for this function:

6.7.1.25 mat_mtype_priorityqueue_creat()

```
MAT_MTYPE_PRIORITYQUEUE mat_mtype_priorityqueue_creat (
            int type )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.7.1.26 mat_mtype_priorityqueue_dequeue()

```
mat_mtypepqnode mat_mtype_priorityqueue_dequeue (
             MAT_MTYPE_PRIORITYQUEUE H )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.7.1.27 mat_mtype_priorityqueue_enqueue()

```
void mat_mtype_priorityqueue_enqueue (
             MAT_MTYPE_PRIORITYQUEUE H,
             mtype data,
             mtype priority )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.7.1.28 mat_mtype_priorityqueue_free()

```
int mat_mtype_priorityqueue_free (
             MAT_MTYPE_PRIORITYQUEUE H )
```

Here is the caller graph for this function:

6.7.1.29 mat_mtype_priorityqueue_is_empty()

```
int mat_mtype_priorityqueue_is_empty (
             MAT_MTYPE_PRIORITYQUEUE H )
```

6.7.1.30 mat_mtype_priorityqueue_update()

```
int mat_mtype_priorityqueue_update (
             MAT_MTYPE_PRIORITYQUEUE H,
             mtype data,
             mtype priority,
             int type )
```

Here is the call graph for this function:

6.7.1.31 mat_mtype_queue_creat()

```
MAT_MTYPE_QUEUE mat_mtype_queue_creat (
             void )
```

Here is the call graph for this function:

6.7.1.32 mat_mtype_queue_dequeue()

```
mtype mat_mtype_queue_dequeue (
            MAT_MTYPE_QUEUE s )
```

Here is the call graph for this function:

6.7.1.33 mat_mtype_queue_enqueue()

```
void mat_mtype_queue_enqueue (
            MAT_MTYPE_QUEUE s,
            mtype value )
```

Here is the call graph for this function:

6.7.1.34 mat_mtype_queue_free()

```
int mat_mtype_queue_free (
            MAT_MTYPE_QUEUE s )
```

6.7.1.35 mat_mtype_queue_is_empty()

```
int mat_mtype_queue_is_empty (
            MAT_MTYPE_QUEUE s )
```

6.7.1.36 mat_mtype_stack_creat()

```
MAT_MTYPE_STACK mat_mtype_stack_creat (
            void  )
```

Here is the call graph for this function:

6.7.1.37 mat_mtype_stack_free()

```
int mat_mtype_stack_free (
            MAT_MTYPE_STACK s )
```

6.7.1.38 mat_mtype_stack_is_empty()

```
int mat_mtype_stack_is_empty (
            MAT_MTYPE_STACK s )
```

6.7.1.39 mat_mtype_stack_pop()

```
mtype mat_mtype_stack_pop (
            MAT_MTYPE_STACK s )
```

Here is the call graph for this function:

6.7.1.40 mat_mtype_stack_push()

```
void mat_mtype_stack_push (
            MAT_MTYPE_STACK s,
            mtype value )
```

Here is the call graph for this function:

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 mat_cofact()

```
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 mat_det()

```
mtype mat_det (
            MATRIX A )
```

Computes the determinant of a matrix.

Parameters

in	*A*	Input matrix

Returns

$\det\left(A\right)$

Here is the call graph for this function: Here is the caller graph for this function:

6.8.1.3 mat_minor()

```
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}

Here is the call graph for this function:

6.9 matdiv.c File Reference

Functions

- MATRIX mat_inv_dot (MATRIX A, MATRIX result)

Computes element-wise matrix inverse.

- 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.

- MATRIX mat_divs_inv (MATRIX A, mtype s, MATRIX result)

  Divides a scalar by a matrix.

- 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.

- INT_VECTOR int_vec_inv (INT_VECTOR A, INT_VECTOR result)

  Computes element-wise integer vector inverse.

- INT_VECTOR int_vec_divs_inv (INT_VECTOR A, int s, INT_VECTOR result)

  Divides a scalar by an integer vector.

6.9.1 Function Documentation

6.9.1.1 int_vec_div()

```
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

$A./B$

Here is the call graph for this function:

6.9.1.2 int_vec_divs()

```
INT_VECTOR int_vec_divs (
             INT_VECTOR A,
             int s,
             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

$\frac{A}{s}$

Here is the call graph for this function:

6.9.1.3 int_vec_divs_inv()

```
INT_VECTOR int_vec_divs_inv (
             INT_VECTOR A,
             int s,
             INT_VECTOR result )
```

Divides a scalar by an integer vector.

Parameters

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

Returns

$s1./A$

Here is the call graph for this function:

6.9.1.4 int_vec_inv()

```
INT_VECTOR int_vec_inv (
             INT_VECTOR A,
             INT_VECTOR result )
```

Computes element-wise integer vector inverse.

Parameters

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

Returns

$$1./A$$

Here is the call graph for this function:

6.9.1.5 mat_div_dot()

```
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

$$\mathbf{A}./\mathbf{B}$$

Here is the call graph for this function:

6.9.1.6 mat_divs()

```
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

$$\frac{\mathbf{A}}{s}$$

Here is the call graph for this function: Here is the caller graph for this function:

6.9.1.7 mat_divs_inv()

```
MATRIX mat_divs_inv (
            MATRIX A,
            mtype s,
            MATRIX result )
```

Divides a scalar by a matrix.

Parameters

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

Returns

$s\mathbf{1}\mathbf{1}^T./\mathbf{A}$

Here is the call graph for this function:

6.9.1.8 mat_inv_dot()

```
MATRIX mat_inv_dot (
            MATRIX A,
            MATRIX result )
```

Computes element-wise matrix inverse.

Parameters

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

Returns

$\mathbf{1}\mathbf{1}^T./\mathbf{A}$

Here is the call graph for this function:

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 int_vec_dump()

```
void int_vec_dump (
            INT_VECTOR A )
```

Dumps an integer vector in the stdout.

Parameters

in	*A*	Input vector

Here is the call graph for this function:

6.10.1.2 int_vec_dumpf()

```
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

Here is the call graph for this function:

6.10.1.3 int_vec_fdump()

```
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

Here is the call graph for this function:

6.10.1.4 int_vec_fdumpf()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.10.1.5 mat_dump()

```
void mat_dump (
             MATRIX A )
```

Dumps a matrix in the stdout.

Parameters

in	*A*	Input matrix

Here is the call graph for this function:

6.10.1.6 mat_dumpf()

```
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

Here is the call graph for this function:

6.10.1.7 mat_fdump()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.10.1.8 mat_fdumpf()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

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 mat_durbin()

```
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 $B = \begin{bmatrix} R[1][0] & R[2][0] & \cdots & R[n][0] \end{bmatrix}$

Here is the call graph for this function: Here is the caller graph for this function:

6.11.1.2 mat_lsolve_durbin()

```
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 $RX = B$

Here is the call graph for this function:

6.11.1.3 mat_qr()

```
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

Here is the call graph for this function:

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 gen_error()

```
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)

Here is the caller graph for this function:

6.12.1.2 graph_error()

```
int graph_error (
             int err_ )
```

Generates error message for graph errors and exits.

Parameters

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

Here is the caller graph for this function:

6.12.1.3 int_vec_error()

```
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_VE↩ C_SIZEMISMATCH)

Here is the caller graph for this function:

6.12.1.4 int_vecstack_error()

```
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_FNOT←↩ GETINT_VEC/INT_VECSTACK_SIZEMISMATCH)

Here is the caller graph for this function:

6.12.1.5 mat_error()

```
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_error()

```
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/MAT←↩ STACK_SIZEMISMATCH/ MATSTACK_INVERSE_ERROR)

Here is the caller graph for this function:

6.12.1.7 pq_error()

```
int pq_error (
             int err_ )
```

Generates error message for priority queue errors and exits.

Parameters

in	*err*	Error type (PQ_MALLOC/PQ_EMPTY)

Here is the caller graph for this function:

6.12.1.8 queue_error()

```
int queue_error (
            int err_ )
```

Generates error message for queue errors and exits.

Parameters

in	*err*	Error type (QUEUE_MALLOC/QUEUE_EMPTY)

Here is the caller graph for this function:

6.12.1.9 stack_error()

```
int stack_error (
            int err_ )
```

Generates error message for stack errors and exits.

Parameters

in	*err*	Error type (STACK_MALLOC/STACK_EMPTY)

Here is the caller graph for this function:

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 mat_fft2()

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

Computes fast Fourier transform.

Parameters

in	*C*	Complex data matrix stack
in	*dir*	FFT direction (MAT_FFT2_FORWARD/MAT_FFT2_BACKWARD)
in	*result*	Matrix stack to store the result

Returns

Transformed matrix stack

Here is the call graph for this function:

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 mat_conv2()

```
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

Here is the call graph for this function:

6.15 matfit.c File Reference

Functions

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

  Performs 2-d polynomial model fitting using least squares.
- MATRIX mat_least_squares (MATRIX A, MATRIX Y, MATRIX result)

  Solves linear equations using least squares.
- MATRIX mat_w_least_squares (MATRIX A, MATRIX Y, MATRIX w, MATRIX result)

  Solves linear equations using weighted least squares.
- MATRIX mat_rob_least_squares (MATRIX A, MATRIX Y, int lossfunc, MATRIX result)

  Solves linear equations using robust reweighted least squares.
- MATRIX mat_robust_fit (MATRIX A, MATRIX Y, int deg, int lossfunc, MATRIX result)

  Performs 2-d polynomial model fitting using robust least squares.

6.15.1 Function Documentation

6.15.1.1 mat_least_squares()

```
MATRIX mat_least_squares (
            MATRIX A,
            MATRIX Y,
            MATRIX result )
```

Solves linear equations using least squares.

Parameters

in	*A*	Input data matrix
in	*Y*	Input observation matrix
in	*result*	Matrix to store the result

Returns

$$\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{Y}$$

Here is the call graph for this function: Here is the caller graph for this function:

6.15.1.2 mat_linear_ls_fit()

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

Performs 2-d polynomial model fitting using least squares.

Parameters

in	*A*	Input data column matrix
in	*Y*	Input observation column matrix
in	*deg*	Polynomial degree N
in	*result*	Matrix to store the result

Returns

Polynomial co-efficient matrix $\begin{bmatrix}\alpha_N & \cdots & \alpha_0\end{bmatrix}^T$

Here is the call graph for this function:

6.15.1.3 mat_rob_least_squares()

```
MATRIX mat_rob_least_squares (
            MATRIX A,
            MATRIX Y,
            int lossfunc,
            MATRIX result )
```

Solves linear equations using robust reweighted least squares.

Parameters

in	*A*	Input data matrix
in	*Y*	Input observation matrix
in	*lossfunc*	Loss function type (MAT_LOSS_BISQUARE/MAT_LOSS_HUBER)
in	*result*	Matrix to store the result

Returns

Robust $\mathbf{X}$

Here is the call graph for this function: Here is the caller graph for this function:

6.15.1.4 mat_robust_fit()

```
MATRIX mat_robust_fit (
            MATRIX A,
            MATRIX Y,
            int deg,
            int lossfunc,
            MATRIX result )
```

Performs 2-d polynomial model fitting using robust least squares.

Parameters

in	*A*	Input data column matrix
in	*Y*	Input observation column matrix
in	*deg*	Polynomial degree N
in	*lossfunc*	Loss function type (MAT_LOSS_BISQUARE/MAT_LOSS_HUBER)
in	*result*	Matrix to store the result

Returns

Polynomial co-efficient matrix $\begin{bmatrix}\alpha_N & \cdots & \alpha_0\end{bmatrix}^T$

Here is the call graph for this function:

6.15.1.5 mat_w_least_squares()

```
MATRIX mat_w_least_squares (
            MATRIX A,
            MATRIX Y,
            MATRIX w,
            MATRIX result )
```

Solves linear equations using weighted least squares.

Parameters

in	*A*	Input data matrix
in	*Y*	Input observation matrix
in	*w*	Input weight column matrix
in	*result*	Matrix to store the result

Returns

$$\left(\mathbf{A}^T\mathrm{diag}(w)\mathbf{A}\right)^{-1}\mathbf{A}^T\mathrm{diag}(w)\mathbf{Y}$$

Here is the call graph for this function: Here is the caller graph for this function:

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 mat_fliplr()

```
MATRIX mat_fliplr (
            MATRIX A,
            MATRIX result )
```

Here is the call graph for this function:

6.16.1.2 mat_flipud()

```
MATRIX mat_flipud (
            MATRIX A,
            MATRIX result )
```

Here is the call graph for this function:

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.

- mtype __mat_round_away_zero (mtype x)

  Rounds a number away from zero.
- mtype __mat_round_towards_zero (mtype x)

  Rounds a number towards zero.
- 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 __mat_addfunc()

```
mtype __mat_addfunc (
            mtype x,
            mtype y )
```

Computes addition function.

Parameters

in	*x*	
in	*y*	

Returns

$x + y$

6.17.1.2 __mat_arccosh()

```
mtype __mat_arccosh (
            mtype x )
```

Computes inverse hyperbolic cosine function.

Parameters

in	*x*	

Returns

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

6.17.1.3 __mat_arcsinh()

```
mtype __mat_arcsinh (
             mtype x )
```

Computes inverse hyperbolic sine function.

Parameters

in	*x*	

Returns

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

Here is the call graph for this function:

6.17.1.4 __mat_arctanh()

```
mtype __mat_arctanh (
             mtype x )
```

Computes inverse hyperbolic tangent function.

Parameters

in	*x*	

Returns

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

Here is the call graph for this function:

6.17.1.5 __mat_bisquare_wt()

```
mtype __mat_bisquare_wt (
             mtype x,
             mtype k )
```

Computes bisquare weight function.

Parameters

in	*x*	
in	*k*	

Returns

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

6.17.1.6 __mat_divfunc()

```
mtype __mat_divfunc (
             mtype x,
             mtype y )
```

Computes division function.

Parameters

in	*x*	
in	*y*	

Returns

$$\frac{x}{y}$$

6.17.1.7 __mat_huber_wt()

```
mtype __mat_huber_wt (
             mtype x,
             mtype k )
```

Computes Huber weight function.

Parameters

in	*x*	
in	*k*	

Returns

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

6.17.1.8 __mat_logplusone()

```
mtype __mat_logplusone (
             mtype x )
```

Computes logarithm plus one function.

Parameters

in	*x*	

Returns

$\log\left(1+x\right)$

Here is the caller graph for this function:

6.17.1.9 __mat_mulfunc()

```
mtype __mat_mulfunc (
             mtype x,
             mtype y )
```

Computes multiplication function.

Parameters

in	*x*	
in	*y*	

Returns

xy

Here is the caller graph for this function:

6.17.1.10 __mat_round_away_zero()

```
mtype __mat_round_away_zero (
             mtype x )
```

Rounds a number away from zero.

Parameters

in	*x*	Input value

Returns

$\mathrm{sgn}(x)\left\lfloor |x| + 0.5 \right\rfloor$

6.17.1.11 __mat_round_towards_zero()

```
mtype __mat_round_towards_zero (
             mtype x )
```

Rounds a number towards zero.

Parameters

in	*x*	Input value

Returns

$\mathrm{sgn}(x)\left\lceil |x| - 0.5 \right\rceil$

6.17.1.12 __mat_sqrfunc()

```
mtype __mat_sqrfunc (
             mtype x )
```

Computes square function.

Parameters

in	*x*	

Returns

x^2

Here is the caller graph for this function:

6.17.1.13 __mat_sqrtfunc()

```
mtype __mat_sqrtfunc (
             mtype x )
```

Computes square root function.

Parameters

in	*x*	

Returns

$\sqrt{x}$

Here is the caller graph for this function:

6.17.1.14 __mat_subfunc()

```
mtype __mat_subfunc (
             mtype x,
             mtype y )
```

Computes subtraction function.

Parameters

in	*x*	
in	*y*	

Returns

$x - y$

Here is the caller graph for this function:

6.17.1.15 mat_bisquare_wt()

```
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

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

Here is the call graph for this function: Here is the caller graph for this function:

6.17.1.16 mat_gfunc()

```
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

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

Here is the call graph for this function: Here is the caller graph for this function:

6.17.1.17 mat_huber_wt()

```
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

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

Here is the call graph for this function: Here is the caller graph for this 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 mat_graph_adjlist()

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

Here is the call graph for this function:

6.18.1.2 mat_graph_adjm_to_adjl()

```
void mat_graph_adjm_to_adjl (
            MAT_GRAPH g,
            MATRIX a )
```

Here is the call graph for this function:

6.18.1.3 mat_graph_creat()

```
MAT_GRAPH mat_graph_creat (
            void )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.18.1.4 mat_graph_dump()

```
void mat_graph_dump (
            MAT_GRAPH g,
            int mst )
```

Here is the call graph for this function:

6.18.1.5 mat_graph_dumpf()

```
void mat_graph_dumpf (
            MAT_GRAPH g,
            int mst,
            MAT_FILEPOINTER fp )
```

Here is the caller graph for this function:

6.18.1.6 mat_graph_reverse()

```
MAT_GRAPH mat_graph_reverse (
            MAT_GRAPH g,
            MAT_GRAPH r )
```

Here is the call graph for this function:

6.18.1.7 mat_graph_search()

```
MAT_INT_QUEUE mat_graph_search (
            MAT_GRAPH g,
            int connected,
            int mst )
```

Here is the call graph for this function:

6.18.1.8 mat_graph_visit()

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

Here is the call graph for this function: Here is the caller graph for this function:

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 mat_innerprod()

```
mtype mat_innerprod (
            MATRIX A,
            MATRIX B )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.19.1.2 mat_norm_inf()

```
mtype mat_norm_inf (
            MATRIX A )
```

6.19.1.3 mat_norm_one()

```
mtype mat_norm_one (
            MATRIX A )
```

Here is the caller graph for this function:

6.19.1.4 mat_norm_p()

```
mtype mat_norm_p (
            MATRIX A,
            mtype p )
```

Here is the caller graph for this function:

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 mat_int_qadrat()

```
mtype mat_int_qadrat (
            mtype(*)(mtype) func,
            mtype lower,
            mtype upper )
```

Computes Gauss quadrature integration.

Parameters

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

Returns

$\int_a^b f\left(x\right)dx$

Here is the call graph for this function:

6.20.1.2 mat_int_simpson()

```
mtype mat_int_simpson (
            mtype(*)(mtype) func,
            int n,
            mtype lower,
            mtype upper )
```

Computes Simpson's integration.

Parameters

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

Returns

$\int_a^b f\left(x\right)dx$

6.20.1.3 mat_int_trapezoid()

```
mtype mat_int_trapezoid (
            mtype(*)(mtype) func,
            int n,
            mtype lower,
            mtype upper )
```

Computes trapezoid integration.

Parameters

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

Returns

$\int_a^b f\left(x\right)dx$

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 mat_inv()

```
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}

Here is the call graph for this function: Here is the caller graph for this function:

6.21.1.2 mat_reg_inv()

```
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

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

Here is the call graph for this function: Here is the caller graph for this function:

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 mat_kdtree_free()

```
int mat_kdtree_free (
            MAT_KDTREE t )
```

Frees a k-d tree.

Parameters

in	*t*	Input k-d tree

Returns

Success

Here is the call graph for this function:

6.22.1.2 mat_kdtree_k_nearest()

```
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 $j = 1, 2, \cdots, N$

Here is the call graph for this function:

6.22.1.3 mat_kdtree_make_tree()

```
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

Here is the call graph for this function:

6.22.1.4 mat_kdtree_nearest()

```
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 $j = 1, 2, \cdots, N$

Here is the call graph for this function:

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 mat_max()

```
MATVEC_DPOINTER mat_max (
            MATRIX A,
            int dim )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.23.1.2 mat_min()

```
MATVEC_DPOINTER mat_min (
            MATRIX A,
            int dim )
```

Here is the call graph for this function: Here is the caller graph for this function:

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 __mat_mds_metric()

```
MATRIX __mat_mds_metric (
            MATRIX d,
            int dims,
            MATRIX result )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.24.1.2 __mat_mds_nonmetric()

```
MATRIX __mat_mds_nonmetric (
            MATRIX d,
            int dims,
            MATRIX result )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.24.1.3 mat_mds()

```
MATRIX mat_mds (
            MATRIX d,
            int dims,
            int type,
            MATRIX result )
```

Here is the call graph for this function:

6.25 matmean.c File Reference

Functions

- mtype mat_mean (MATRIX A)

  Computes the mean of a matrix.
- MATRIX mat_mean_row (MATRIX A, MATRIX result)

  Computes row-mean of a matrix.
- MATRIX mat_mean_col (MATRIX A, MATRIX result)

  Computes column-mean of a matrix.
- mtype int_vec_mean (INT_VECTOR A)

  Computes element-mean of an integer vector.

6.25.1 Function Documentation

6.25.1.1 int_vec_mean()

```
mtype int_vec_mean (
            INT_VECTOR A )
```

Computes element-mean of an integer vector.

Parameters

in	*A*	Input integer vector

Returns

$\mathrm{mean}(A)$

6.25.1.2 mat_mean()

```
mtype mat_mean (
            MATRIX A )
```

Computes the mean of a matrix.

Parameters

A	Input matrix

Returns

$\mathrm{mean}(\mathbf{A})$

Here is the call graph for this function:

6.25.1.3 mat_mean_col()

```
MATRIX mat_mean_col (
            MATRIX A,
            MATRIX result )
```

Computes column-mean of a matrix.

Parameters

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

Returns

$\mathbf{1}^T\mathbf{A}/\#\mathrm{rows}$

Here is the call graph for this function:

6.25.1.4 mat_mean_row()

```
MATRIX mat_mean_row (
             MATRIX A,
             MATRIX result )
```

Computes row-mean of a matrix.

Parameters

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

Returns

$\mathbf{A1}/\#\mathrm{cols}$

Here is the call graph for this function: Here is the caller graph for this function:

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 __mat_cart2pol()

```
void __mat_cart2pol (
             mtype x,
             mtype y,
             mtype ∗ rho,
             mtype ∗ th )
```

6.26.1.2 __mat_pol2cart()

```
void __mat_pol2cart (
             mtype rho,
             mtype th,
             mtype ∗ x,
             mtype ∗ y )
```

6.26.1.3 int_vec_permute_vect()

```
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

Here is the call graph for this function:

6.26.1.4 int_vec_unique()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.26.1.5 mat_bsxfun()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.26.1.6 mat_calc_dist_sq()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.26.1.7 mat_cart2pol()

```
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

Here is the call graph for this function:

6.26.1.8 mat_find_within_dist()

```
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

Here is the call graph for this function:

6.26.1.9 mat_fnextline()

```
void mat_fnextline (
            MAT_FILEPOINTER fp )
```

Prints nextline to file.

Parameters

in	*fp*	Pointer to opened file

Here is the caller graph for this function:

6.26.1.10 mat_get_sub_matrix_from_cols()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.26.1.11 mat_get_sub_matrix_from_rows()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.26.1.12 mat_get_sub_vector()

```
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

Here is the call graph for this function:

6.26.1.13 mat_nextline()

```
void mat_nextline (
             void )
```

Prints nextline to stdout.

Here is the call graph for this function:

6.26.1.14 mat_pick_col()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.26.1.15 mat_pick_row()

```
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

Here is the call graph for this function:

6.26.1.16 mat_pol2cart()

```
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

Here is the call graph for this function:

6.26.1.17 mats_isinf()

```
int mats_isinf (
             mtype x )
```

Checks if scalar is infinite.

Parameters

in	*x*	Input scalar

Returns

Zero/non-zero

6.26.1.18 mats_isnan()

```
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)

  Computes matrix multiplication.
- MATRIX mat_mul_fast (MATRIX A, MATRIX B, MATRIX result)

  Computes fast matrix multiplication (not implemented)
- MATRIX mat_muls (MATRIX A, mtype s, MATRIX result)

Multiplies a matrix by a scalar.

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

  Computes element-wise matrix multiplication.
- mtype mat_diagmul (MATRIX A)

  Computes matrix diagonal product.
- INT_VECTOR int_vec_mul (INT_VECTOR A, INT_VECTOR B, INT_VECTOR result)

  Computes element-wise integer vector multiplication.
- INT_VECTOR int_vec_muls (INT_VECTOR A, int x, INT_VECTOR result)

  Multiplies an integer vector by a scalar.

6.27.1 Function Documentation

6.27.1.1 int_vec_mul()

```
INT_VECTOR int_vec_mul (
            INT_VECTOR A,
            INT_VECTOR B,
            INT_VECTOR result )
```

Computes element-wise integer vector multiplication.

Parameters

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

Returns

$A. * B$

Here is the call graph for this function:

6.27.1.2 int_vec_muls()

```
INT_VECTOR int_vec_muls (
            INT_VECTOR A,
            int x,
            INT_VECTOR result )
```

Multiplies an integer vector by a scalar.

Parameters

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

Returns

sA

Here is the call graph for this function: Here is the caller graph for this function:

6.27.1.3 mat_diagmul()

```
mtype mat_diagmul (
            MATRIX A )
```

Computes matrix diagonal product.

Parameters

in	*A*	Input matrix

Returns

$\mathrm{prod}(\mathrm{diag}(\mathbf{A}))$

6.27.1.4 mat_mul()

```
MATRIX mat_mul (
            MATRIX A,
            MATRIX B,
            MATRIX result )
```

Computes matrix multiplication.

Parameters

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

Returns

$\mathbf{AB}$

Here is the call graph for this function: Here is the caller graph for this function:

6.27.1.5 mat_mul_dot()

```
MATRIX mat_mul_dot (
            MATRIX A,
            MATRIX B,
            MATRIX result )
```

Computes element-wise matrix multiplication.

Parameters

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

Returns

$\mathbf{A}. * \mathbf{B}$

Here is the call graph for this function: Here is the caller graph for this function:

6.27.1.6 mat_mul_fast()

```
MATRIX mat_mul_fast (
            MATRIX A,
            MATRIX B,
            MATRIX result )
```

Computes fast matrix multiplication (not implemented)

Parameters

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

Returns

$\mathbf{AB}$

Here is the call graph for this function: Here is the caller graph for this function:

6.27.1.7 mat_muls()

```
MATRIX mat_muls (
            MATRIX A,
            mtype s,
            MATRIX result )
```

Multiplies a matrix by a scalar.

Parameters

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

Returns

$s\mathbf{A}$

Here is the call graph for this function: Here is the caller graph for this function:

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 mat_corcol()

```
MATSTACK mat_corcol (
             MATRIX data )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.28.1.2 mat_covcol()

```
MATSTACK mat_covcol (
             MATRIX data )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.28.1.3 mat_eig_sym()

```
MATSTACK mat_eig_sym (
             MATRIX symmat,
             MATSTACK result )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.28.1.4 mat_pca()

```
MATSTACK mat_pca (
             MATRIX data,
             int pca_type )
```

Here is the call graph for this function:

6.28.1.5 mat_scpcol()

```
MATRIX mat_scpcol (
             MATRIX data )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.28.1.6 mat_tqli()

```
void mat_tqli (
             MATRIX d,
             MATRIX e,
             MATRIX z )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.28.1.7 mat_tred2()

```
void mat_tred2 (
             MATRIX a,
             MATRIX d,
             MATRIX e )
```

Here is the caller graph for this function:

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 mat_pinv()

```
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

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

Here is the call graph for this function: Here is the caller graph for this function:

6.29.1.2 mat_wpinv()

```
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

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

Here is the call graph for this function: Here is the caller graph for this function:

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 mat_binom()

```
mtype mat_binom (
              int n,
              int k )
```

Computes a binomial co-efficient.

Parameters

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

Returns

$\binom{n}{k}$

Here is the call graph for this function: Here is the caller graph for this function:

6.30.1.2 mat_binom_init()

```
void mat_binom_init ( )
```

Initializes the binomial series.

Here is the call graph for this function: Here is the caller graph for this function:

6.30.1.3 mat_cheby()

```
MATRIX mat_cheby (
             int n )
```

Computes the n^{th} Chebyshev polynomial.

Parameters

in	*n*	Polynomial series index

Returns

Output polynomial matrix

Here is the call graph for this function: Here is the caller graph for this function:

6.30.1.4 mat_cheby_approx()

```
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

Here is the call graph for this function:

6.30.1.5 mat_cheby_coeffs_to_poly()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.30.1.6 mat_cheby_init()

```
void mat_cheby_init ( )
```

Initializes the Chebyshev polynomial series.

Here is the call graph for this function:

6.30.1.7 mat_legendre()

```
MATRIX mat_legendre (
            int n )
```

Computes the n^{th} Legendre polynomial.

Parameters

in	*n*	Polynomial series index

Returns

Output polynomial matrix

Here is the call graph for this function: Here is the caller graph for this function:

6.30.1.8 mat_legendre_init()

```
void mat_legendre_init ( )
```

Initializes the Legendre polynomial series.

Here is the call graph for this function:

6.30.1.9 mat_poly_add()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.30.1.10 mat_poly_diff()

```
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

Here is the call graph for this function:

6.30.1.11 mat_poly_diff_eval()

```
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

Here is the call graph for this function:

6.30.1.12 mat_poly_div()

```
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

Here is the call graph for this function:

6.30.1.13 mat_poly_eval()

```
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

Here is the call graph for this function:

6.30.1.14 mat_poly_mul()

```
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

Here is the call graph for this function:

6.30.1.15 mat_poly_scale()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.30.1.16 mat_poly_shift()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.30.2 Variable Documentation

6.30.2.1 mat_binom_series_table

```
MATSTACK mat_binom_series_table
```

6.30.2.2 mat_cheby_series_table

```
MATSTACK mat_cheby_series_table
```

6.30.2.3 mat_legendre_series_table

```
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 mat_bayes_classifier_test()

```
INT_VECTOR mat_bayes_classifier_test (
             MATRIX data,
             MAT_BAYES_MODEL b_model )
```

Here is the call graph for this function:

6.31.1.2 mat_bayes_classifier_train()

```
MAT_BAYES_MODEL mat_bayes_classifier_train (
             MATRIX data,
             INT_VECTOR labels )
```

Here is the call graph for this function:

6.31.1.3 mat_kmeans()

```
MATVEC_DPOINTER mat_kmeans (
             MATRIX data,
             int k,
             int iters,
             MATVEC_DPOINTER result )
```

Here is the call graph for this function:

6.31.1.4 mat_perceptron_test()

```
INT_VECTOR mat_perceptron_test (
             MATRIX data,
             MAT_PERCEPTRON p_model )
```

Here is the call graph for this function:

6.31.1.5 mat_perceptron_train()

```
MAT_PERCEPTRON mat_perceptron_train (
             MATRIX data,
             INT_VECTOR labels,
             int num_of_iterations )
```

Here is the call graph for this function:

6.31.1.6 mat_perceptron_train_()

```
MAT_PERCEPTRON mat_perceptron_train_ (
             MATRIX data1,
             MATRIX data2,
             MAT_PERCEPTRON p_model,
             int class_num )
```

Here is the call graph for this function: Here is the caller graph for this function:

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 mat_omp()

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

Here is the call graph for this function:

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 __mat_rand()

```
mtype __mat_rand (
             void   )
```

Here is the call graph for this function:

6.33.1.2 __mat_randexp()

```
mtype __mat_randexp (
             mtype mu )
```

Here is the call graph for this function:

6.33.1.3 __mat_randfun()

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

Here is the call graph for this function: Here is the caller graph for this function:

6.33.1.4 __mat_randn()

```
mtype __mat_randn (
            void )
```

Here is the call graph for this function:

6.33.1.5 int_vec_randperm()

```
INT_VECTOR int_vec_randperm (
            int n,
            INT_VECTOR result )
```

Here is the call graph for this function:

6.33.1.6 mat_rand()

```
MATRIX mat_rand (
            int n,
            int m,
            MATRIX result )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.33.1.7 mat_randexp()

```
MATRIX mat_randexp (
            int n,
            int m,
            mtype mu,
            MATRIX result )
```

Here is the call graph for this function:

6.33.1.8 mat_randfun()

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

Here is the call graph for this function:

6.33.1.9 mat_randn()

```
MATRIX mat_randn (
             int n,
             int m,
             MATRIX result )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.33.1.10 mat_randperm()

```
MATRIX mat_randperm (
             int m,
             int n,
             MATRIX result )
```

Here is the call graph for this function:

6.33.1.11 mat_randperm_n()

```
MATRIX mat_randperm_n (
             int n,
             MATRIX result )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.33.1.12 mat_set_seed()

```
void mat_set_seed (
             int seed )
```

Here is the caller graph for this function:

6.33.2 Variable Documentation

6.33.2.1 MAT_SEED

```
unsigned int MAT_SEED = 0
```

6.33.2.2 MAT_SET_SEED

```
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

- __declspec (thread) clock_t MAT_CLOCK_TIME

  Starts stopwatch timer.
- 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.
- int int_vec_sum (INT_VECTOR A)

Computes element-sum of an integer vector.

- mtype int_vec_mean (INT_VECTOR A)

  Computes element-mean of an integer vector.
- mtype mat_sum (MATRIX A)

  Computes element-sum of a matrix.
- MATRIX mat_sum_row (MATRIX A, MATRIX result)

  Computes row-sum of a matrix.
- MATRIX mat_sum_col (MATRIX A, MATRIX result)

  Computes column-sum of a matrix.
- mtype mat_mean (MATRIX A)

  Computes the mean of a matrix.
- MATRIX mat_mean_row (MATRIX A, MATRIX result)

  Computes row-mean of a matrix.
- MATRIX mat_mean_col (MATRIX A, MATRIX result)

  Computes column-mean of a matrix.
- INT_VECTOR int_vec_abs (INT_VECTOR A, INT_VECTOR result)

  Computes absolute value of an integer vector.
- 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)

  Subtracts an integer vector from integer vector.
- INT_VECTOR int_vec_subs (INT_VECTOR A, int s, INT_VECTOR result)

  Subtracts an integer from integer vector.
- INT_VECTOR int_vec_subs_neg (INT_VECTOR A, int s, INT_VECTOR result)

  Subtracts an integer vector from an integer.
- INT_VECTOR int_vec_mul (INT_VECTOR A, INT_VECTOR B, INT_VECTOR result)

  Computes element-wise integer vector multiplication.
- INT_VECTOR int_vec_muls (INT_VECTOR A, int s, INT_VECTOR result)

  Multiplies an integer vector by a scalar.
- INT_VECTOR int_vec_inv (INT_VECTOR A, INT_VECTOR result)

  Computes element-wise integer vector inverse.
- 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.
- INT_VECTOR int_vec_divs_inv (INT_VECTOR A, int s, INT_VECTOR result)

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

  Computes absolute value of matrix.
- 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)

  Subtracts a matrix from another matrix.
- MATRIX mat_subs (MATRIX A, mtype s, MATRIX result)

  Subtracts a scalar from a matrix.
- MATRIX mat_subs_neg (MATRIX A, mtype s, MATRIX result)

  Subtracts a matrix from a scalar.

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

  Computes matrix multiplication.
- MATRIX mat_mul_fast (MATRIX A, MATRIX B, MATRIX result)

  Computes fast matrix multiplication (not implemented)
- MATRIX mat_mul_dot (MATRIX A, MATRIX B, MATRIX result)

  Computes element-wise matrix multiplication.
- MATRIX mat_muls (MATRIX A, mtype s, MATRIX result)

  Multiplies a matrix by a scalar.
- MATRIX mat_inv_dot (MATRIX A, MATRIX result)

  Computes element-wise matrix inverse.
- 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.
- MATRIX mat_divs_inv (MATRIX A, mtype s, MATRIX result)

  Divides a scalar by a matrix.
- 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)

  Computes matrix diagonal product.
- 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)

  Computes the symmetric Toeplitz matrix from a co-efficient matrix.
- int mat_lu (MATRIX A, MATRIX P)

  Computes LU decomposition of a matrix.
- void mat_backsubs1 (MATRIX A, MATRIX B, MATRIX C, MATRIX P, int xcol)
- MATRIX mat_lsolve (MATRIX A, MATRIX b, MATRIX result)

  Solves linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$.
- 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.
- MATSTACK mat_svd (MATRIX a, int niters, MATSTACK result)

  Computes the SVD of a matrix.
- 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)

  Solves linear equations using least squares.
- MATRIX mat_w_least_squares (MATRIX A, MATRIX Y, MATRIX w, MATRIX result)

  Solves linear equations using weighted least squares.
- MATRIX mat_rob_least_squares (MATRIX A, MATRIX Y, int lossfunc, MATRIX result)

  Solves linear equations using robust reweighted least squares.
- MATRIX mat_linear_ls_fit (MATRIX A, MATRIX Y, int deg, MATRIX result)

  Performs 2-d polynomial model fitting using least squares.
- MATRIX mat_robust_fit (MATRIX A, MATRIX Y, int deg, int lossfunc, MATRIX result)

  Performs 2-d polynomial model fitting using robust least squares.
- 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.
- mtype __mat_round_away_zero (mtype x)

  Rounds a number away from zero.
- mtype __mat_round_towards_zero (mtype x)

  Rounds a number towards zero.
- 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)

  Checks if greater than zero.
- int gen_lt (mtype a)

  Checks if less than zero.
- int gen_eq (mtype a)

  Checks if equals to zero.
- 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)
- 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_PRIO↩ RITYQUEUE pq, MATRIX bmax, MATRIX bmin)
- MATSTACK mat_omp (MATRIX A, MATRIX b, int k, mtype tol, MATSTACK result)

Variables

- _Thread_local 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 INT_VECSTACK

```
typedef INT_VECTOR∗ INT_VECSTACK
```

Integer Vector Stack

6.35.1.2 INT_VECTOR

```
typedef int∗ INT_VECTOR
```

Integer Vector

6.35.1.3 mat_bayes_model

```
typedef struct mat_bayes_model mat_bayes_model
```

Bayes Classifier Model Structure.

Bayes Classifier Model

6.35.1.4 MAT_BAYES_MODEL

```
typedef mat_bayes_model* MAT_BAYES_MODEL
```

Bayes Classifier Model Pointer

6.35.1.5 mat_gnode

```
typedef struct mat_gnode mat_gnode
```

Graph Node Structure.

Graph Node

6.35.1.6 MAT_GNODE

```
typedef mat_gnode* MAT_GNODE
```

Graph Node Pointer

6.35.1.7 mat_graph

```
typedef struct mat_graph mat_graph
```

Graph Structure.

6.35.1.8 MAT_GRAPH

```
typedef mat_graph* MAT_GRAPH
```

6.35.1.9 mat_int_priorityqueue

```
typedef struct mat_int_priorityqueue mat_int_priorityqueue
```

Integer Priority Queue Structure.

Integer Priority Queue

6.35.1.10 MAT_INT_PRIORITYQUEUE

```
typedef mat_int_priorityqueue* MAT_INT_PRIORITYQUEUE
```

Integer Priority Queue Pointer

6.35.1.11 mat_int_queue

```
typedef struct mat_int_queue mat_int_queue
```

Integer Queue Structure.

Integer Queue

6.35.1.12 MAT_INT_QUEUE

```
typedef mat_int_queue* MAT_INT_QUEUE
```

Integer Queue Pointer

6.35.1.13 mat_int_stack

```
typedef struct mat_int_stack mat_int_stack
```

Integer Stack Structure.

Integer Stack

6.35.1.14 MAT_INT_STACK

```
typedef mat_int_stack* MAT_INT_STACK
```

Integer Stack Pointer

6.35.1.15 mat_intpqnode

```
typedef struct mat_intpqnode mat_intpqnode
```

Integer Priority Queue Node Structure.

Integer Priority Queue Node

6.35.1.16 MAT_INTPQNODE

```
typedef mat_intpqnode* MAT_INTPQNODE
```

Integer Priority Queue Node Pointer

6.35.1.17 mat_kdnode

```
typedef struct mat_kdnode mat_kdnode
```

6.35.1.18 MAT_KDNODE

```
typedef mat_kdnode* MAT_KDNODE
```

6.35.1.19 mat_kdtree

```
typedef struct mat_kdtree mat_kdtree
```

6.35.1.20 MAT_KDTREE

```
typedef mat_kdtree* MAT_KDTREE
```

6.35.1.21 mat_mtype_priorityqueue

```
typedef struct mat_mtype_priorityqueue mat_mtype_priorityqueue
```

Mtype Priority Queue Structure.

Mtype Priority Queue

6.35.1.22 MAT_MTYPE_PRIORITYQUEUE

```
typedef mat_mtype_priorityqueue* MAT_MTYPE_PRIORITYQUEUE
```

Mtype Priority Queue Pointer

6.35.1.23 mat_mtype_queue

```
typedef struct mat_mtype_queue mat_mtype_queue
```

Mtype Queue Structure.

Mtype Queue

6.35.1.24 MAT_MTYPE_QUEUE

```
typedef mat_mtype_queue* MAT_MTYPE_QUEUE
```

Mtype Queue Pointer

6.35.1.25 mat_mtype_stack

```
typedef struct mat_mtype_stack mat_mtype_stack
```

Mtype Stack Structure.

Mtype Stack

6.35.1.26 MAT_MTYPE_STACK

```
typedef mat_mtype_stack* MAT_MTYPE_STACK
```

Mtype Stack Pointer

6.35.1.27 mat_mtypepqnode

```
typedef struct mat_mtypepqnode mat_mtypepqnode
```

Mtype Priority Queue Node Structure.

Mtype Priority Queue Node

6.35.1.28 MAT_MTYPEPQNODE

```
typedef mat_mtypepqnode* MAT_MTYPEPQNODE
```

Mtype Priority Queue Node Pointer

6.35.1.29 mat_perceptron

```
typedef struct mat_perceptron mat_perceptron
```

Perceptron Classifier Model Structure.

Perceptron Classifier Model

6.35.1.30 MAT_PERCEPTRON

```
typedef mat_perceptron* MAT_PERCEPTRON
```

Perceptron Classifier Model Pointer

6.35.1.31 mat_qintnode

```
typedef struct mat_qintnode mat_qintnode
```

Integer Queue Node Structure.

Integer Queue Node

6.35.1.32 MAT_QINTNODE

```
typedef mat_qintnode* MAT_QINTNODE
```

Integer Queue Node Pointer

6.35.1.33 mat_qmtypenode

```
typedef struct mat_qmtypenode mat_qmtypenode
```

Mtype Queue Node Structure.

Mtype Queue Node

6.35.1.34 MAT_QMTYPENODE

```
typedef mat_qmtypenode* MAT_QMTYPENODE
```

Mtype Queue Node Pointer

6.35.1.35 MAT_TREE

```
typedef mat_tree_node* MAT_TREE
```

Search Tree Pointer

6.35.1.36 mat_tree_node

```
typedef struct mat_tree_node mat_tree_node
```

Search Tree Node Structure.

Search Tree Node

6.35.1.37 MAT_TREE_NODE

```
typedef mat_tree_node* MAT_TREE_NODE
```

Search Tree Node Pointer

6.35.1.38 MATRIX

```
typedef mtype** MATRIX
```

Mtype Matrix

6.35.1.39 MATSTACK

```
typedef MATRIX* MATSTACK
```

Mtype Matrix Stack

6.35.1.40 MATVEC_DPOINTER

```
typedef void** MATVEC_DPOINTER
```

Mtype Matrix - Integer Vector Pair

6.35.2 Function Documentation

6.35.2.1 __declspec()

```
__declspec (
             thread )
```

Starts stopwatch timer.

6.35.2.2 __int_vec_creat()

```
INT_VECTOR __int_vec_creat (
             int len )
```

Here is the caller graph for this function:

6.35.2.3 __int_vecstack_creat()

```
INT_VECSTACK __int_vecstack_creat (
             int len )
```

Here is the caller graph for this function:

6.35.2.4 __mat_addfunc()

```
mtype __mat_addfunc (
             mtype x,
             mtype y )
```

Computes addition function.

Parameters

in	*x*	
in	*y*	

Returns

$x + y$

6.35.2.5 __mat_arccosh()

```
mtype __mat_arccosh (
            mtype x )
```

Computes inverse hyperbolic cosine function.

Parameters

in	*x*	

Returns

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

6.35.2.6 __mat_arcsinh()

```
mtype __mat_arcsinh (
            mtype x )
```

Computes inverse hyperbolic sine function.

Parameters

in	*x*	

Returns

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

Here is the call graph for this function:

6.35.2.7 __mat_arctanh()

```
mtype __mat_arctanh (
            mtype x )
```

Computes inverse hyperbolic tangent function.

Parameters

in	*x*	

Returns

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

Here is the call graph for this function:

6.35.2.8 __mat_bisquare_wt()

```
mtype __mat_bisquare_wt (
             mtype x,
             mtype k )
```

Computes bisquare weight function.

Parameters

in	*x*	
in	*k*	

Returns

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

6.35.2.9 __mat_cart2pol()

```
void __mat_cart2pol (
             mtype x,
             mtype y,
             mtype * rho,
             mtype * th )
```

6.35.2.10 __mat_creat()

```
MATRIX __mat_creat (
             int r,
             int c )
```

Here is the caller graph for this function:

6.35.2.11 __mat_divfunc()

```
mtype __mat_divfunc (
             mtype x,
             mtype y )
```

Computes division function.

Parameters

in	*x*	
in	*y*	

Returns

$\frac{x}{y}$

6.35.2.12 __mat_fft()

```
int __mat_fft (
             int dir,
             int m,
             mtype * x,
             mtype * y )
```

Here is the caller graph for this function:

6.35.2.13 __mat_huber_wt()

```
mtype __mat_huber_wt (
             mtype x,
             mtype k )
```

Computes Huber weight function.

Parameters

in	*x*	
in	*k*	

Returns

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

6.35.2.14 __mat_kd_find_median()

```
MAT_KDNODE __mat_kd_find_median (
             MAT_KDNODE kd_start,
             MAT_KDNODE kd_end,
             int idx )
```

6.35.2.15 __mat_kdtree_k_nearest()

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

Here is the caller graph for this function:

6.35.2.16 __mat_kdtree_make_tree()

```
MAT_KDNODE __mat_kdtree_make_tree (
             MAT_KDNODE t,
             int len,
             int i,
             int dim )
```

Here is the caller graph for this function:

6.35.2.17 __mat_kdtree_nearest()

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

Here is the caller graph for this function:

6.35.2.18 __mat_lint()

```
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 f14,
            mtype hmin,
            mtype hmax,
            mtype re,
            mtype ae )
```

Here is the caller graph for this function:

6.35.2.19 __mat_logplusone()

```
mtype __mat_logplusone (
            mtype x )
```

Computes logarithm plus one function.

Parameters

in	*x*	

Returns

$\log{(1+x)}$

Here is the caller graph for this function:

6.35.2.20 __mat_mds_metric()

```
MATRIX __mat_mds_metric (
            MATRIX d,
            int dims,
            MATRIX result )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.21 __mat_mds_nonmetric()

```
MATRIX __mat_mds_nonmetric (
            MATRIX d,
            int dims,
            MATRIX result )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.22 __mat_mulfunc()

```
mtype __mat_mulfunc (
             mtype x,
             mtype y )
```

Computes multiplication function.

Parameters

in	*x*	
in	*y*	

Returns

xy

Here is the caller graph for this function:

6.35.2.23 __mat_pol2cart()

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

6.35.2.24 __mat_powerof2()

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

Here is the caller graph for this function:

6.35.2.25 __mat_quicksort()

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

Here is the caller graph for this function:

6.35.2.26 __mat_rand()

```
mtype __mat_rand (
             void  )
```

Here is the call graph for this function:

6.35.2.27 __mat_randexp()

```
mtype __mat_randexp (
             mtype mu )
```

Here is the call graph for this function:

6.35.2.28 __mat_randfun()

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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.29 __mat_randn()

```
mtype __mat_randn (
             void  )
```

Here is the call graph for this function:

6.35.2.30 __mat_round_away_zero()

```
mtype __mat_round_away_zero (
             mtype x )
```

Rounds a number away from zero.

Parameters

in	*x*	Input value

Returns

$\mathrm{sgn}(x)\,\lfloor|x|+0.5\rfloor$

6.35.2.31 __mat_round_towards_zero()

```
mtype __mat_round_towards_zero (
             mtype x )
```

Rounds a number towards zero.

Parameters

in	*x*	Input value

Returns

$\mathrm{sgn}(x)\left\lceil |x| - 0.5 \right\rceil$

6.35.2.32 __mat_sqrfunc()

```
mtype __mat_sqrfunc (
             mtype x )
```

Computes square function.

Parameters

in	*x*	

Returns

x^2

Here is the caller graph for this function:

6.35.2.33 __mat_sqrtfunc()

```
mtype __mat_sqrtfunc (
             mtype x )
```

Computes square root function.

Parameters

in	*x*	

Returns

$\sqrt{x}$

Here is the caller graph for this function:

6.35.2.34 __mat_subfunc()

```
mtype __mat_subfunc (
            mtype x,
            mtype y )
```

Computes subtraction function.

Parameters

in	*x*	
in	*y*	

Returns

$x - y$

Here is the caller graph for this function:

6.35.2.35 __matstack_creat()

```
MATSTACK __matstack_creat (
            int len )
```

Here is the caller graph for this function:

6.35.2.36 gen_eq()

```
int gen_eq (
            mtype a )
```

Checks if equals to zero.

Parameters

in	*a*	Input value

Returns

int $a == 0$

6.35.2.37 gen_error()

```
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)

Here is the caller graph for this function:

6.35.2.38 gen_gt()

```
int gen_gt (
            mtype a )
```

Checks if greater than zero.

Parameters

in	*a*	Input value

Returns

int $a > 0$

6.35.2.39 gen_lt()

```
int gen_lt (
            mtype a )
```

Checks if less than zero.

Parameters

in	*a*	Input value

Returns

int $a < 0$

6.35.2.40 graph_error()

```
int graph_error (
            int err_ )
```

Generates error message for graph errors and exits.

Parameters

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

Here is the caller graph for this function:

6.35.2.41 int_vec2_mat()

```
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

Here is the call graph for this function:

6.35.2.42 int_vec_abs()

```
INT_VECTOR int_vec_abs (
            INT_VECTOR A,
            INT_VECTOR result )
```

Computes absolute value of an integer vector.

Parameters

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

Returns

$\mathrm{abs}(A)$

Here is the call graph for this function:

6.35.2.43 int_vec_add()

```
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

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

Here is the call graph for this function:

6.35.2.44 int_vec_adds()

```
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

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

Here is the call graph for this function:

6.35.2.45 int_vec_append()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.46 int_vec_concat()

```
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	*result*	Vector to store the result

Returns

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

Here is the call graph for this function:

6.35.2.47 int_vec_copy()

```
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

Here is the call graph for this function:

6.35.2.48 int_vec_creat()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.49 int_vec_div()

```
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

$A./B$

Here is the call graph for this function:

6.35.2.50 int_vec_divs()

```
INT_VECTOR int_vec_divs (
             INT_VECTOR A,
             int s,
             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

$\frac{A}{s}$

Here is the call graph for this function:

6.35.2.51 int_vec_divs_inv()

```
INT_VECTOR int_vec_divs_inv (
             INT_VECTOR A,
             int s,
             INT_VECTOR result )
```

Divides a scalar by an integer vector.

Parameters

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

Returns

$s1./A$

Here is the call graph for this function:

6.35.2.52 int_vec_dump()

```
void int_vec_dump (
             INT_VECTOR A )
```

Dumps an integer vector in the stdout.

Parameters

in	*A*	Input vector

Here is the call graph for this function:

6.35.2.53 int_vec_dumpf()

```
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

Here is the call graph for this function:

6.35.2.54 int_vec_error()

```
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_VE↩ C_SIZEMISMATCH)

Here is the caller graph for this function:

6.35.2.55 int_vec_fdump()

```
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

Here is the call graph for this function:

6.35.2.56 int_vec_fdumpf()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.57 int_vec_fill()

```
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

6.35.2.58 int_vec_fill_type()

```
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

Here is the caller graph for this function:

6.35.2.59 int_vec_find()

```
INT_VECTOR int_vec_find (
            INT_VECTOR a,
            int rel_type,
            int n )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.60 int_vec_free()

```
int int_vec_free (
            INT_VECTOR A )
```

Frees an integer vector.

Parameters

in	*A*	Input vector

Returns

Success

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.61 int_vec_inv()

```
INT_VECTOR int_vec_inv (
            INT_VECTOR A,
            INT_VECTOR result )
```

Computes element-wise integer vector inverse.

Parameters

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

Returns

$1./A$

Here is the call graph for this function:

6.35.2.62 int_vec_mean()

```
mtype int_vec_mean (
            INT_VECTOR A )
```

Computes element-mean of an integer vector.

Parameters

in	*A*	Input integer vector

Returns

$\text{mean}(A)$

6.35.2.63 int_vec_mul()

```
INT_VECTOR int_vec_mul (
             INT_VECTOR A,
             INT_VECTOR B,
             INT_VECTOR result )
```

Computes element-wise integer vector multiplication.

Parameters

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

Returns

$A. * B$

Here is the call graph for this function:

6.35.2.64 int_vec_muls()

```
INT_VECTOR int_vec_muls (
             INT_VECTOR A,
             int x,
             INT_VECTOR result )
```

Multiplies an integer vector by a scalar.

Parameters

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

Returns

sA

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.65 int_vec_permute_vect()

```
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

Here is the call graph for this function:

6.35.2.66 int_vec_randperm()

```
INT_VECTOR int_vec_randperm (
             int n,
             INT_VECTOR result )
```

Here is the call graph for this function:

6.35.2.67 int_vec_sub()

```
INT_VECTOR int_vec_sub (
             INT_VECTOR A,
             INT_VECTOR B,
             INT_VECTOR result )
```

Subtracts an integer vector from integer vector.

Parameters

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

Returns

$$\mathbf{A} - \mathbf{B}$$

Here is the call graph for this function:

6.35.2.68 int_vec_subs()

```
INT_VECTOR int_vec_subs (
              INT_VECTOR A,
              int s,
              INT_VECTOR result )
```

Subtracts an integer from integer vector.

Parameters

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

Returns

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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.69 int_vec_subs_neg()

```
INT_VECTOR int_vec_subs_neg (
              INT_VECTOR A,
              int s,
              INT_VECTOR result )
```

Subtracts an integer vector from an integer.

Parameters

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

Returns

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

Here is the call graph for this function:

6.35.2.70 int_vec_sum()

```
int int_vec_sum (
             INT_VECTOR A )
```

Computes element-sum of an integer vector.

Parameters

in	*A*	Input integer vector

Returns

$\text{sum}(A)$

6.35.2.71 int_vec_unique()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.72 int_vecstack_creat()

```
INT_VECSTACK int_vecstack_creat (
             int len )
```

Creates an integer vector stack.

Parameters

in	*len*	Length of the stack

Returns

Output vector stack

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.73 int_vecstack_error()

```
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_FNOT↩ GETINT_VEC/INT_VECSTACK_SIZEMISMATCH)

Here is the caller graph for this function:

6.35.2.74 int_vecstack_free()

```
int int_vecstack_free (
             INT_VECSTACK A )
```

Frees an integer vector stack.

Parameters

in	*A*	Input vector stack

Returns

Success

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.75 mat_2int_vec()

```
INT_VECTOR mat_2int_vec (
             MATRIX A )
```

Converts a matrix to an integer vector.

Parameters

in	*A*	Input matrix
in	*v*	Vector to store the result

Returns

Output vector

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.76 mat_abs()

```
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

$\text{abs}(\mathbf{A})$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.77 mat_add()

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

Adds two matrices.

Parameters

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

Returns

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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.78 mat_adds()

```
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

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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.79 mat_backsubs1()

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

Here is the caller graph for this function:

6.35.2.80 mat_bayes_classifier_test()

```
INT_VECTOR mat_bayes_classifier_test (
                 MATRIX data,
                 MAT_BAYES_MODEL b_model )
```

Here is the call graph for this function:

6.35.2.81 mat_bayes_classifier_train()

```
MAT_BAYES_MODEL mat_bayes_classifier_train (
                 MATRIX data,
                 INT_VECTOR labels )
```

Here is the call graph for this function:

6.35.2.82 mat_bayes_model_creat()

```
MAT_BAYES_MODEL mat_bayes_model_creat (
                 void  )
```

Creates a Bayes model.

Returns

Output Bayes model

Here is the caller graph for this function:

6.35.2.83 mat_bayes_model_free()

```
int mat_bayes_model_free (
            MAT_BAYES_MODEL a )
```

Frees a Bayes model.

Parameters

in	*a*	Input Bayes model

Returns

Success

Here is the call graph for this function:

6.35.2.84 mat_binom()

```
mtype mat_binom (
            int n,
            int k )
```

Computes a binomial co-efficient.

Parameters

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

Returns

$\binom{n}{k}$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.85 mat_binom_init()

```
void mat_binom_init ( )
```

Initializes the binomial series.

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.86 mat_bisquare_wt()

```
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

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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.87 mat_bs_delete()

```
MAT_TREE mat_bs_delete (
            mtype x,
            MAT_TREE T )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.88 mat_bs_find()

```
MAT_TREE mat_bs_find (
            mtype x,
            MAT_TREE T )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.89 mat_bs_find_max()

```
MAT_TREE mat_bs_find_max (
            MAT_TREE T )
```

6.35.2.90 mat_bs_find_min()

```
MAT_TREE mat_bs_find_min (
            MAT_TREE T )
```

Here is the caller graph for this function:

6.35.2.91 mat_bs_free()

```
MAT_TREE mat_bs_free (
            MAT_TREE T )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.92 mat_bs_inorder()

```
int mat_bs_inorder (
            MAT_TREE T,
            int index,
            mtype ** ordered )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.93 mat_bs_insert()

```
MAT_TREE mat_bs_insert (
            mtype x,
            MAT_TREE T )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.94 mat_bs_make_null()

```
MAT_TREE mat_bs_make_null (
            void )
```

Here is the caller graph for this function:

6.35.2.95 mat_bsxfun()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.96 mat_calc_dist_sq()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.97 mat_cart2pol()

```
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

Here is the call graph for this function:

6.35.2.98 mat_cheby()

```
MATRIX mat_cheby (
             int n )
```

Computes the n^{th} Chebyshev polynomial.

Parameters

in	*n*	Polynomial series index

Returns

Output polynomial matrix

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.99 mat_cheby_approx()

```
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

Here is the call graph for this function:

6.35.2.100 mat_cheby_coeffs_to_poly()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.101 mat_cheby_init()

```
void mat_cheby_init ( )
```

Initializes the Chebyshev polynomial series.

Here is the call graph for this function:

6.35.2.102 mat_cholesky()

```
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

Here is the call graph for this function:

6.35.2.103 mat_cofact()

```
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}

6.35.2.104 mat_colcopy()

```
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

Here is the caller graph for this function:

6.35.2.105 mat_concat()

```
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 $\begin{bmatrix} A \\ B \end{bmatrix}$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.106 mat_conjgrad()

```
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

Here is the call graph for this function:

6.35.2.107 mat_conv2()

```
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

Here is the call graph for this function:

6.35.2.108 mat_copy()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.109 mat_corcol()

```
MATSTACK mat_corcol (
             MATRIX data )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.110 mat_count_words_in_line()

```
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
in	*count*	Pointer to output count

Returns

EOF reached

Here is the caller graph for this function:

6.35.2.111 mat_covcol()

```
MATSTACK mat_covcol (
             MATRIX data )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.112 mat_creat()

```
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

Here is the call graph for this function:

6.35.2.113 mat_creat_diag()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.114 mat_det()

```
mtype mat_det (
            MATRIX A )
```

Computes the determinant of a matrix.

Parameters

in	*A*	Input matrix

Returns

$\det(A)$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.115 mat_diagmul()

```
mtype mat_diagmul (
            MATRIX A )
```

Computes matrix diagonal product.

Parameters

in	*A*	Input matrix

Returns

$\mathrm{prod}(\mathrm{diag}(\mathbf{A}))$

6.35.2.116 mat_div_dot()

```
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

$\mathbf{A}./\mathbf{B}$

Here is the call graph for this function:

6.35.2.117 mat_divs()

```
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

$$\frac{\mathbf{A}}{s}$$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.118 mat_divs_inv()

```
MATRIX mat_divs_inv (
             MATRIX A,
             mtype s,
             MATRIX result )
```

Divides a scalar by a matrix.

Parameters

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

Returns

$$s\mathbf{1}\mathbf{1}^T./\mathbf{A}$$

Here is the call graph for this function:

6.35.2.119 mat_dlmread()

```
MATRIX mat_dlmread (
             const char * fname )
```

Reads a matrix from a file.

Parameters

in	*fname*	Filename to read from

Returns

Output matrix

Here is the call graph for this function:

6.35.2.120 mat_dlmwrite()

```
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

Here is the call graph for this function:

6.35.2.121 mat_dump()

```
void mat_dump (
             MATRIX A )
```

Dumps a matrix in the stdout.

Parameters

in	*A*	Input matrix

Here is the call graph for this function:

6.35.2.122 mat_dumpf()

```
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

Here is the call graph for this function:

6.35.2.123 mat_durbin()

```
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 $B = \begin{bmatrix} R[1][0] & R[2][0] & \cdots & R[n][0] \end{bmatrix}$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.124 mat_eig_sym()

```
MATSTACK mat_eig_sym (
             MATRIX symmat,
             MATSTACK result )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.125 mat_error()

```
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)

6.35.2.126 mat_fdump()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.127 mat_fdumpf()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.128 mat_fft2()

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

Computes fast Fourier transform.

Parameters

in	*C*	Complex data matrix stack
in	*dir*	FFT direction (MAT_FFT2_FORWARD/MAT_FFT2_BACKWARD)
in	*result*	Matrix stack to store the result

Returns

Transformed matrix stack

Here is the call graph for this function:

6.35.2.129 mat_fgetmat()

```
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

6.35.2.130 mat_fill()

```
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

Here is the caller graph for this function:

6.35.2.131 mat_fill_type()

```
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

Here is the caller graph for this function:

6.35.2.132 mat_find()

```
INT_VECSTACK mat_find (
             MATRIX A,
             int rel_type,
             mtype x )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.133 mat_find_within_dist()

```
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

Here is the call graph for this function:

6.35.2.134 mat_fliplr()

```
MATRIX mat_fliplr (
             MATRIX A,
             MATRIX result )
```

Here is the call graph for this function:

6.35.2.135 mat_flipud()

```
MATRIX mat_flipud (
             MATRIX A,
             MATRIX result )
```

Here is the call graph for this function:

6.35.2.136 mat_fnextline()

```
void mat_fnextline (
             MAT_FILEPOINTER fp )
```

Prints nextline to file.

Parameters

in	*fp*	Pointer to opened file

Here is the caller graph for this function:

6.35.2.137 mat_free()

```
int mat_free (
             MATRIX A )
```

Frees a matrix.

Parameters

in	*A*	Input matrix

Returns

Success

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.138 mat_get_sub_matrix_from_cols()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.139 mat_get_sub_matrix_from_rows()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.140 mat_get_sub_vector()

```
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

Here is the call graph for this function:

6.35.2.141 mat_gfunc()

```
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

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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.142 mat_go_next_word()

```
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

6.35.2.143 mat_graph_adjlist()

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

Here is the call graph for this function:

6.35.2.144 mat_graph_adjm_to_adjl()

```
void mat_graph_adjm_to_adjl (
             MAT_GRAPH g,
             MATRIX a )
```

Here is the call graph for this function:

6.35.2.145 mat_graph_creat()

```
MAT_GRAPH mat_graph_creat (
             void   )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.146 mat_graph_dump()

```
void mat_graph_dump (
             MAT_GRAPH g,
             int mst )
```

Here is the call graph for this function:

6.35.2.147 mat_graph_dumpf()

```
void mat_graph_dumpf (
             MAT_GRAPH g,
             int mst,
             MAT_FILEPOINTER fp )
```

Here is the caller graph for this function:

6.35.2.148 mat_graph_reverse()

```
MAT_GRAPH mat_graph_reverse (
             MAT_GRAPH g,
             MAT_GRAPH r )
```

Here is the call graph for this function:

6.35.2.149 mat_graph_search()

```
MAT_INT_QUEUE mat_graph_search (
             MAT_GRAPH g,
             int connected,
             int mst )
```

Here is the call graph for this function:

6.35.2.150 mat_graph_visit()

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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.151 mat_huber_wt()

```
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

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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.152 mat_innerprod()

```
mtype mat_innerprod (
            MATRIX A,
            MATRIX B )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.153 mat_int_priorityqueue_creat()

```
MAT_INT_PRIORITYQUEUE mat_int_priorityqueue_creat (
            int type )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.154 mat_int_priorityqueue_dequeue()

```
mat_intpqnode mat_int_priorityqueue_dequeue (
            MAT_INT_PRIORITYQUEUE H )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.155 mat_int_priorityqueue_enqueue()

```
void mat_int_priorityqueue_enqueue (
            MAT_INT_PRIORITYQUEUE H,
            int data,
            int priority )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.156 mat_int_priorityqueue_free()

```
int mat_int_priorityqueue_free (
            MAT_INT_PRIORITYQUEUE H )
```

Here is the caller graph for this function:

6.35.2.157 mat_int_priorityqueue_is_empty()

```
int mat_int_priorityqueue_is_empty (
            MAT_INT_PRIORITYQUEUE H )
```

6.35.2.158 mat_int_priorityqueue_update()

```
int mat_int_priorityqueue_update (
            MAT_INT_PRIORITYQUEUE H,
            int data,
            int priority,
            int type )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.159 mat_int_qadrat()

```
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

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

Here is the call graph for this function:

6.35.2.160 mat_int_queue_creat()

```
MAT_INT_QUEUE mat_int_queue_creat (
             void  )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.161 mat_int_queue_dequeue()

```
int mat_int_queue_dequeue (
             MAT_INT_QUEUE s )
```

Here is the call graph for this function:

6.35.2.162 mat_int_queue_enqueue()

```
void mat_int_queue_enqueue (
             MAT_INT_QUEUE s,
             int value )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.163 mat_int_queue_free()

```
int mat_int_queue_free (
             MAT_INT_QUEUE s )
```

6.35.2.164 mat_int_queue_is_empty()

```
int mat_int_queue_is_empty (
             MAT_INT_QUEUE s )
```

6.35.2.165 mat_int_simpson()

```
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

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

6.35.2.166 mat_int_stack_creat()

```
MAT_INT_STACK mat_int_stack_creat (
             void )
```

Here is the call graph for this function:

6.35.2.167 mat_int_stack_free()

```
int mat_int_stack_free (
             MAT_INT_STACK s )
```

6.35.2.168 mat_int_stack_is_empty()

```
int mat_int_stack_is_empty (
             MAT_INT_STACK s )
```

6.35.2.169 mat_int_stack_pop()

```
int mat_int_stack_pop (
             MAT_INT_STACK s )
```

Here is the call graph for this function:

6.35.2.170 mat_int_stack_push()

```
void mat_int_stack_push (
             MAT_INT_STACK s,
             int value )
```

Here is the call graph for this function:

6.35.2.171 mat_int_trapezoid()

```
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

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

6.35.2.172 mat_inv()

```
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}

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.173 mat_inv_dot()

```
MATRIX mat_inv_dot (
            MATRIX A,
            MATRIX result )
```

Computes element-wise matrix inverse.

Parameters

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

Returns

$\mathbf{1}\mathbf{1}^T./\mathbf{A}$

Here is the call graph for this function:

6.35.2.174 mat_isnumeric()

```
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

Here is the caller graph for this function:

6.35.2.175 mat_kdtree_free()

```
int mat_kdtree_free (
            MAT_KDTREE t )
```

Frees a k-d tree.

Parameters

in	*t*	Input k-d tree

Returns

Success

Here is the call graph for this function:

6.35.2.176 mat_kdtree_k_nearest()

```
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 $j = 1, 2, \cdots, N$

Here is the call graph for this function:

6.35.2.177 mat_kdtree_make_tree()

```
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

Here is the call graph for this function:

6.35.2.178 mat_kdtree_nearest()

```
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 $j = 1, 2, \cdots, N$

Here is the call graph for this function:

6.35.2.179 mat_kmeans()

```
MATVEC_DPOINTER mat_kmeans (
             MATRIX data,
             int k,
             int iters,
             MATVEC_DPOINTER result )
```

Here is the call graph for this function:

6.35.2.180 mat_least_squares()

```
MATRIX mat_least_squares (
            MATRIX A,
            MATRIX Y,
            MATRIX result )
```

Solves linear equations using least squares.

Parameters

in	*A*	Input data matrix
in	*Y*	Input observation matrix
in	*result*	Matrix to store the result

Returns

$$\left(\mathbf{A}^T\mathbf{A}\right)^{-1}\mathbf{A}^T\mathbf{Y}$$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.181 mat_legendre()

```
MATRIX mat_legendre (
            int n )
```

Computes the n^{th} Legendre polynomial.

Parameters

in	*n*	Polynomial series index

Returns

Output polynomial matrix

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.182 mat_legendre_init()

```
void mat_legendre_init ( )
```

Initializes the Legendre polynomial series.

Here is the call graph for this function:

6.35.2.183 mat_linear_ls_fit()

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

Performs 2-d polynomial model fitting using least squares.

Parameters

in	*A*	Input data column matrix
in	*Y*	Input observation column matrix
in	*deg*	Polynomial degree N
in	*result*	Matrix to store the result

Returns

Polynomial co-efficient matrix $\begin{bmatrix}\alpha_N & \cdots & \alpha_0\end{bmatrix}^T$

Here is the call graph for this function:

6.35.2.184 mat_lsolve()

```
MATRIX mat_lsolve (
             MATRIX A,
             MATRIX b,
             MATRIX result )
```

Solves linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$.

Parameters

in	*A*	Input matrix $\mathbf{A}$
in	*b*	Input matrix $\mathbf{b}$
in	*result*	Matrix to store the result

Returns

Output matrix $\mathbf{x}$

Here is the call graph for this function:

6.35.2.185 mat_lsolve_durbin()

```
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 $RX = B$

Here is the call graph for this function:

6.35.2.186 mat_lu()

```
int mat_lu (
            MATRIX A,
            MATRIX P )
```

Computes LU decomposition of a matrix.

Parameters

in	*A*	Input matrix overwritten by matrices L and U
in	*P*	Matrix to store permutation matrix P

Returns

p Status

Here is the caller graph for this function:

6.35.2.187 mat_max()

```
MATVEC_DPOINTER mat_max (
            MATRIX A,
            int dim )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.188 mat_mds()

```
MATRIX mat_mds (
             MATRIX d,
             int dims,
             int type,
             MATRIX result )
```

Here is the call graph for this function:

6.35.2.189 mat_mean()

```
mtype mat_mean (
             MATRIX A )
```

Computes the mean of a matrix.

Parameters

A	Input matrix

Returns

$\text{mean}(\mathbf{A})$

Here is the call graph for this function:

6.35.2.190 mat_mean_col()

```
MATRIX mat_mean_col (
             MATRIX A,
             MATRIX result )
```

Computes column-mean of a matrix.

Parameters

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

Returns

$\mathbf{1}^T\mathbf{A}/\#\text{rows}$

Here is the call graph for this function:

6.35.2.191 mat_mean_row()

```
MATRIX mat_mean_row (
             MATRIX A,
             MATRIX result )
```

Computes row-mean of a matrix.

Parameters

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

Returns

$\mathbf{A1}/\#\mathrm{cols}$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.192 mat_median()

```
mtype mat_median (
            MATRIX A )
```

Computes the median of elements of a given matrix.

Parameters

in	*A*	Input matrix

Returns

$\mathrm{med}\left(\{a_{ij}\}\right)$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.193 mat_min()

```
MATVEC_DPOINTER mat_min (
            MATRIX A,
            int dim )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.194 mat_minor()

```
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}

Here is the call graph for this function:

6.35.2.195 mat_mtype_priorityqueue_creat()

```
MAT_MTYPE_PRIORITYQUEUE mat_mtype_priorityqueue_creat (
             int type )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.196 mat_mtype_priorityqueue_dequeue()

```
mat_mtypepqnode mat_mtype_priorityqueue_dequeue (
             MAT_MTYPE_PRIORITYQUEUE H )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.197 mat_mtype_priorityqueue_enqueue()

```
void mat_mtype_priorityqueue_enqueue (
             MAT_MTYPE_PRIORITYQUEUE H,
             mtype data,
             mtype priority )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.198 mat_mtype_priorityqueue_free()

```
int mat_mtype_priorityqueue_free (
             MAT_MTYPE_PRIORITYQUEUE H )
```

Here is the caller graph for this function:

6.35.2.199 mat_mtype_priorityqueue_is_empty()

```
int mat_mtype_priorityqueue_is_empty (
             MAT_MTYPE_PRIORITYQUEUE H )
```

6.35.2.200 mat_mtype_priorityqueue_update()

```
int mat_mtype_priorityqueue_update (
             MAT_MTYPE_PRIORITYQUEUE H,
             mtype data,
             mtype priority,
             int type )
```

Here is the call graph for this function:

6.35.2.201 mat_mtype_queue_creat()

```
MAT_MTYPE_QUEUE mat_mtype_queue_creat (
             void )
```

Here is the call graph for this function:

6.35.2.202 mat_mtype_queue_dequeue()

```
mtype mat_mtype_queue_dequeue (
             MAT_MTYPE_QUEUE s )
```

Here is the call graph for this function:

6.35.2.203 mat_mtype_queue_enqueue()

```
void mat_mtype_queue_enqueue (
             MAT_MTYPE_QUEUE s,
             mtype value )
```

Here is the call graph for this function:

6.35.2.204 mat_mtype_queue_free()

```
int mat_mtype_queue_free (
             MAT_MTYPE_QUEUE s )
```

6.35.2.205 mat_mtype_queue_is_empty()

```
int mat_mtype_queue_is_empty (
             MAT_MTYPE_QUEUE s )
```

6.35.2.206 mat_mtype_stack_creat()

```
MAT_MTYPE_STACK mat_mtype_stack_creat (
             void )
```

Here is the call graph for this function:

6.35.2.207 mat_mtype_stack_free()

```
int mat_mtype_stack_free (
             MAT_MTYPE_STACK s )
```

6.35.2.208 mat_mtype_stack_is_empty()

```
int mat_mtype_stack_is_empty (
             MAT_MTYPE_STACK s )
```

6.35.2.209 mat_mtype_stack_pop()

```
mtype mat_mtype_stack_pop (
             MAT_MTYPE_STACK s )
```

Here is the call graph for this function:

6.35.2.210 mat_mtype_stack_push()

```
void mat_mtype_stack_push (
             MAT_MTYPE_STACK s,
             mtype value )
```

Here is the call graph for this function:

6.35.2.211 mat_mul()

```
MATRIX mat_mul (
             MATRIX A,
             MATRIX B,
             MATRIX result )
```

Computes matrix multiplication.

Parameters

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

Returns

$\mathbf{AB}$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.212 mat_mul_dot()

```
MATRIX mat_mul_dot (
            MATRIX A,
            MATRIX B,
            MATRIX result )
```

Computes element-wise matrix multiplication.

Parameters

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

Returns

$\mathbf{A}. * \mathbf{B}$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.213 mat_mul_fast()

```
MATRIX mat_mul_fast (
            MATRIX A,
            MATRIX B,
            MATRIX result )
```

Computes fast matrix multiplication (not implemented)

Parameters

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

Returns

$\mathbf{AB}$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.214 mat_muls()

```
MATRIX mat_muls (
             MATRIX A,
             mtype s,
             MATRIX result )
```

Multiplies a matrix by a scalar.

Parameters

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

Returns

$s\mathbf{A}$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.215 mat_nextline()

```
void mat_nextline (
             void )
```

Prints nextline to stdout.

Here is the call graph for this function:

6.35.2.216 mat_norm_inf()

```
mtype mat_norm_inf (
             MATRIX A )
```

6.35.2.217 mat_norm_one()

```
mtype mat_norm_one (
             MATRIX A )
```

Here is the caller graph for this function:

6.35.2.218 mat_norm_p()

```
mtype mat_norm_p (
             MATRIX A,
             mtype p )
```

Here is the caller graph for this function:

6.35.2.219 mat_omp()

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

Here is the call graph for this function:

6.35.2.220 mat_order_statistic()

```
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

$\mathrm{O}_k\left(\{a_{ij}\}\right)$

Here is the call graph for this function:

6.35.2.221 mat_pca()

```
MATSTACK mat_pca (
            MATRIX data,
            int pca_type )
```

Here is the call graph for this function:

6.35.2.222 mat_perceptron_creat()

```
MAT_PERCEPTRON mat_perceptron_creat (
            void )
```

Creates a perceptron.

Returns

Output perceptron

Here is the caller graph for this function:

6.35.2.223 mat_perceptron_free()

```
int mat_perceptron_free (
            MAT_PERCEPTRON a )
```

Frees a perceptron.

Parameters

in	*a*	Input perceptron

Returns

Success

Here is the call graph for this function:

6.35.2.224 mat_perceptron_test()

```
INT_VECTOR mat_perceptron_test (
             MATRIX data,
             MAT_PERCEPTRON p_model )
```

Here is the call graph for this function:

6.35.2.225 mat_perceptron_train()

```
MAT_PERCEPTRON mat_perceptron_train (
             MATRIX data,
             INT_VECTOR labels,
             int num_of_iterations )
```

Here is the call graph for this function:

6.35.2.226 mat_perceptron_train_()

```
MAT_PERCEPTRON mat_perceptron_train_ (
             MATRIX data1,
             MATRIX data2,
             MAT_PERCEPTRON p_model,
             int class_num )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.227 mat_pick_col()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.228 mat_pick_row()

```
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

Here is the call graph for this function:

6.35.2.229 mat_pinv()

```
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

$\left(A^T A\right)^{-1} A^T$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.230 mat_pol2cart()

```
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

Here is the call graph for this function:

6.35.2.231 mat_poly_add()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.232 mat_poly_diff()

```
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

Here is the call graph for this function:

6.35.2.233 mat_poly_diff_eval()

```
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

Here is the call graph for this function:

6.35.2.234 mat_poly_div()

```
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

Here is the call graph for this function:

6.35.2.235 mat_poly_eval()

```
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

Here is the call graph for this function:

6.35.2.236 mat_poly_mul()

```
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

Here is the call graph for this function:

6.35.2.237 mat_poly_scale()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.238 mat_poly_shift()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.239 mat_qr()

```
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

Here is the call graph for this function:

6.35.2.240 mat_qsort()

```
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)
in	*result*	Matrix stack to store the result

Returns

Output matrix stack of sorted A and their positions

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.241 mat_rand()

```
MATRIX mat_rand (
            int r,
            int c,
            MATRIX result )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.242 mat_randexp()

```
MATRIX mat_randexp (
            int r,
            int c,
            mtype mu,
            MATRIX result )
```

Here is the call graph for this function:

6.35.2.243 mat_randfun()

```
MATRIX mat_randfun (
             int r,
             int c,
             mtype(*)(mtype) fun,
             mtype xmin,
             mtype xmax,
             MATRIX result )
```

Here is the call graph for this function:

6.35.2.244 mat_randn()

```
MATRIX mat_randn (
             int r,
             int c,
             MATRIX result )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.245 mat_randperm()

```
MATRIX mat_randperm (
             int m,
             int n,
             MATRIX result )
```

Here is the call graph for this function:

6.35.2.246 mat_randperm_n()

```
MATRIX mat_randperm_n (
             int n,
             MATRIX result )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.247 mat_read_word()

```
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
in	*c_word*	Pointer to word read

Returns

EOF reached

Here is the caller graph for this function:

6.35.2.248 mat_reg_inv()

```
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

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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.249 mat_rob_least_squares()

```
MATRIX mat_rob_least_squares (
            MATRIX A,
            MATRIX Y,
            int lossfunc,
            MATRIX result )
```

Solves linear equations using robust reweighted least squares.

Parameters

in	*A*	Input data matrix
in	*Y*	Input observation matrix
in	*lossfunc*	Loss function type (MAT_LOSS_BISQUARE/MAT_LOSS_HUBER)
in	*result*	Matrix to store the result

Returns

Robust $\mathbf{X}$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.250 mat_robust_fit()

```
MATRIX mat_robust_fit (
            MATRIX A,
            MATRIX Y,
            int deg,
            int lossfunc,
            MATRIX result )
```

Performs 2-d polynomial model fitting using robust least squares.

Parameters

in	*A*	Input data column matrix
in	*Y*	Input observation column matrix
in	*deg*	Polynomial degree N
in	*lossfunc*	Loss function type (MAT_LOSS_BISQUARE/MAT_LOSS_HUBER)
in	*result*	Matrix to store the result

Returns

Polynomial co-efficient matrix $\begin{bmatrix}\alpha_N & \cdots & \alpha_0\end{bmatrix}^T$

Here is the call graph for this function:

6.35.2.251 mat_rowcopy()

```
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

6.35.2.252 mat_scpcol()

```
MATRIX mat_scpcol (
             MATRIX data )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.253 mat_set_seed()

```
void mat_set_seed (
             int seed )
```

Here is the caller graph for this function:

6.35.2.254 mat_sub()

```
MATRIX mat_sub (
             MATRIX A,
             MATRIX B,
             MATRIX result )
```

Subtracts a matrix from another matrix.

Parameters

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

Returns

$$\mathbf{A} - \mathbf{B}$$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.255 mat_submat()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.256 mat_subs()

```
MATRIX mat_subs (
            MATRIX A,
            mtype s,
            MATRIX result )
```

Subtracts a scalar from a matrix.

Parameters

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

Returns

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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.257 mat_subs_neg()

```
MATRIX mat_subs_neg (
            MATRIX A,
            mtype s,
            MATRIX result )
```

Subtracts a matrix from a scalar.

Parameters

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

Returns

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

Here is the call graph for this function:

6.35.2.258 mat_sum()

```
mtype mat_sum (
            MATRIX A )
```

Computes element-sum of a matrix.

Parameters

in	*A*	Input matrix

Returns

$\mathrm{sum}(\mathbf{A})$

Here is the caller graph for this function:

6.35.2.259 mat_sum_col()

```
MATRIX mat_sum_col (
            MATRIX A,
            MATRIX result )
```

Computes column-sum of a matrix.

Parameters

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

Returns

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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.260 mat_sum_row()

```
MATRIX mat_sum_row (
            MATRIX A,
            MATRIX result )
```

Computes row-sum of a matrix.

Parameters

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

Returns

$\mathbf{A1}$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.261 mat_svd()

```
MATSTACK mat_svd (
            MATRIX a,
            int niters,
            MATSTACK result )
```

Computes the SVD of a matrix.

Parameters

in	*a*	Input matrix
in	*niters*	Iterations to use
	result	Matrix stack to store the result

Returns

MATSTACK ($\mathbf{U}, \mathbf{S}, \mathbf{V}$)

Here is the call graph for this function:

6.35.2.262 mat_symtoeplz()

```
MATRIX mat_symtoeplz (
            MATRIX R,
            MATRIX result )
```

Computes the symmetric Toeplitz matrix from a co-efficient matrix.

Parameters

in	*R*	Input coefficient matrix
in	*result*	Matrix to store the result

Returns

$\mathrm{symtoep}(\mathbf{R})$

Here is the call graph for this function:

6.35.2.263 mat_tic()

```
void mat_tic (
            void )
```

6.35.2.264 mat_toc()

```
double mat_toc (
             void  )
```

Computes elapsed time from last start of timer.

Returns

Elapsed time

6.35.2.265 mat_toc_print()

```
void mat_toc_print (
             void  )
```

Computes and prints elapsed time from last start of timer on the stdout.

6.35.2.266 mat_tqli()

```
void mat_tqli (
             MATRIX d,
             MATRIX e,
             MATRIX z )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.267 mat_tran()

```
MATRIX mat_tran (
             MATRIX A,
             MATRIX result )
```

Computes the transpose of a matrix.

Parameters

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

Returns

$\mathbf{A}^T$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.268 mat_tred2()

```
void mat_tred2 (
            MATRIX a,
            MATRIX d,
            MATRIX e )
```

Here is the caller graph for this function:

6.35.2.269 mat_vectorize()

```
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

$vec(\mathbf{A})$

Here is the call graph for this function:

6.35.2.270 mat_vectorize_tr()

```
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

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

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.271 mat_w_least_squares()

```
MATRIX mat_w_least_squares (
            MATRIX A,
            MATRIX Y,
            MATRIX w,
            MATRIX result )
```

Solves linear equations using weighted least squares.

Parameters

in	*A*	Input data matrix
in	*Y*	Input observation matrix
in	*w*	Input weight column matrix
in	*result*	Matrix to store the result

Returns

$$\left(\mathbf{A}^T\mathrm{diag}(w)\mathbf{A}\right)^{-1}\mathbf{A}^T\mathrm{diag}(w)\mathbf{Y}$$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.272 mat_wpinv()

```
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

$$\left(A^TWA\right)^{-1}A^TW$$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.273 mat_xcopy()

```
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 $A_{s_i:e_i,s_j:e_j}$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.274 mat_xjoin()

```
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 $\left[\begin{array}{cc} A_{11} & A_{12} \\ A_{21} & A_{22} \end{array}\right]$

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.275 mats_isinf()

```
int mats_isinf (
             mtype x )
```

Checks if scalar is infinite.

Parameters

in	*x*	Input scalar

Returns

Zero/non-zero

6.35.2.276 mats_isnan()

```
int mats_isnan (
             mtype x )
```

Checks if scalar is NaN.

Parameters

in	*x*	Input scalar

Returns

Zero/non-zero

6.35.2.277 matstack_append()

```
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

Here is the call graph for this function:

6.35.2.278 matstack_creat()

```
MATSTACK matstack_creat (
             int len )
```

Creates a matrix stack.

Parameters

in	*len*	Length of the stack

Returns

Output matrix stack

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.279 matstack_error()

```
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/MAT← STACK_SIZEMISMATCH/ MATSTACK_INVERSE_ERROR)

Here is the caller graph for this function:

6.35.2.280 matstack_free()

```
int matstack_free (
             MATSTACK A )
```

Frees a matrix stack.

Parameters

in	*A*	Input matrix stack

Returns

Success

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.281 matvec_creat()

```
MATVEC_DPOINTER matvec_creat (
             void  )
```

Creates a matrix-vector pair.

Returns

Output matrix-vector pair

Here is the caller graph for this function:

6.35.2.282 matvec_free()

```
int matvec_free (
             MATVEC_DPOINTER a )
```

Frees a matrix-vector pair.

Parameters

in	*a*	Input matrix-vector pair

Returns

Success

Here is the call graph for this function: Here is the caller graph for this function:

6.35.2.283 pq_error()

```
int pq_error (
             int err_ )
```

Generates error message for priority queue errors and exits.

Parameters

in	*err*	Error type (PQ_MALLOC/PQ_EMPTY)

Here is the caller graph for this function:

6.35.2.284 queue_error()

```
int queue_error (
             int err_ )
```

Generates error message for queue errors and exits.

Parameters

in	*err*	Error type (QUEUE_MALLOC/QUEUE_EMPTY)

Here is the caller graph for this function:

6.35.2.285 stack_error()

```
int stack_error (
            int err_ )
```

Generates error message for stack errors and exits.

Parameters

in	*err*	Error type (STACK_MALLOC/STACK_EMPTY)

Here is the caller graph for this function:

6.35.3 Variable Documentation

6.35.3.1 mat_binom_series_table

```
MATSTACK mat_binom_series_table
```

6.35.3.2 mat_cheby_series_table

```
MATSTACK mat_cheby_series_table
```

6.35.3.3 MAT_CLOCK_TIME

```
__thread clock_t MAT_CLOCK_TIME
```

6.35.3.4 mat_legendre_series_table

```
MATSTACK mat_legendre_series_table
```

6.35.3.5 MAT_SEED

```
unsigned int MAT_SEED
```

6.35.3.6 MAT_SET_SEED

```
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_vec_find()

```
INT_VECTOR int_vec_find (
             INT_VECTOR a,
             int rel_type,
             int n )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.36.1.2 mat_find()

```
INT_VECSTACK mat_find (
             MATRIX A,
             int rel_type,
             mtype x )
```

Here is the call graph for this function: Here is the caller graph for this function:

6.37 matsolve.c File Reference

Functions

- int mat_lu (MATRIX A, MATRIX P)

  Computes LU decomposition of a matrix.
- void mat_backsubs1 (MATRIX A, MATRIX B, MATRIX X, MATRIX P, int xcol)
- MATRIX mat_lsolve (MATRIX A, MATRIX b, MATRIX result)

  Solves linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$.
- 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 mat_backsubs1()

```
void mat_backsubs1 (
             MATRIX A,
             MATRIX B,
             MATRIX X,
             MATRIX P,
             int xcol )
```

Here is the caller graph for this function:

6.37.1.2 mat_cholesky()

```
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

Here is the call graph for this function:

6.37.1.3 mat_conjgrad()

```
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

Here is the call graph for this function:

6.37.1.4 mat_lsolve()

```
MATRIX mat_lsolve (
            MATRIX A,
            MATRIX b,
            MATRIX result )
```

Solves linear equations $\mathbf{Ax} = \mathbf{b}$.

Parameters

in	*A*	Input matrix $\mathbf{A}$
in	*b*	Input matrix $\mathbf{b}$
in	*result*	Matrix to store the result

Returns

Output matrix $\mathbf{x}$

Here is the call graph for this function:

6.37.1.5 mat_lu()

```
int mat_lu (
            MATRIX A,
            MATRIX P )
```

Computes LU decomposition of a matrix.

Parameters

in	*A*	Input matrix overwritten by matrices L and U
in	*P*	Matrix to store permutation matrix P

Returns

p Status

Here is the caller graph for this function:

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 mat_median()

```
mtype mat_median (
            MATRIX A )
```

Computes the median of elements of a given matrix.

Parameters

in	*A*	Input matrix

Returns

$\text{med}\left(\{a_{ij}\}\right)$

Here is the call graph for this function: Here is the caller graph for this function:

6.38.1.2 mat_order_statistic()

```
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

$\mathrm{O}_k\left(\{a_{ij}\}\right)$

Here is the call graph for this function:

6.38.1.3 mat_qsort()

```
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)
in	*result*	Matrix stack to store the result

Returns

Output matrix stack of sorted A and their positions

Here is the call graph for this function: Here is the caller graph for this function:

6.39 matstdrels.c File Reference

Functions

- int gen_gt (mtype a)

  Checks if greater than zero.
- int gen_lt (mtype a)

  Checks if less than zero.
- int gen_eq (mtype a)

  Checks if equals to zero.

6.39.1 Function Documentation

6.39.1.1 gen_eq()

```
int gen_eq (
            mtype a )
```

Checks if equals to zero.

Parameters

in	*a*	Input value

Returns

int $a == 0$

6.39.1.2 gen_gt()

```
int gen_gt (
            mtype a )
```

Checks if greater than zero.

Parameters

in	*a*	Input value

Returns

int $a > 0$

6.39.1.3 gen_lt()

```
int gen_lt (
            mtype a )
```

Checks if less than zero.

Parameters

in	*a*	Input value

Returns

int $a < 0$

6.40 matsub.c File Reference

Functions

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

  Subtracts a matrix from another matrix.
- MATRIX mat_subs (MATRIX A, mtype s, MATRIX result)

  Subtracts a scalar from a matrix.
- MATRIX mat_subs_neg (MATRIX A, mtype s, MATRIX result)

  Subtracts a matrix from a scalar.
- INT_VECTOR int_vec_sub (INT_VECTOR A, INT_VECTOR B, INT_VECTOR result)

  Subtracts an integer vector from integer vector.
- INT_VECTOR int_vec_subs (INT_VECTOR A, int s, INT_VECTOR result)

  Subtracts an integer from integer vector.
- INT_VECTOR int_vec_subs_neg (INT_VECTOR A, int s, INT_VECTOR result)

  Subtracts an integer vector from an integer.

6.40.1 Function Documentation

6.40.1.1 int_vec_sub()

```
INT_VECTOR int_vec_sub (
            INT_VECTOR A,
            INT_VECTOR B,
            INT_VECTOR result )
```

Subtracts an integer vector from integer vector.

Parameters

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

Returns

$\mathbf{A} - \mathbf{B}$

Here is the call graph for this function:

6.40.1.2 int_vec_subs()

```
INT_VECTOR int_vec_subs (
            INT_VECTOR A,
            int s,
            INT_VECTOR result )
```

Subtracts an integer from integer vector.

Parameters

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

Returns

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

Here is the call graph for this function: Here is the caller graph for this function:

6.40.1.3 int_vec_subs_neg()

```
INT_VECTOR int_vec_subs_neg (
            INT_VECTOR A,
            int s,
            INT_VECTOR result )
```

Subtracts an integer vector from an integer.

Parameters

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

Returns

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

Here is the call graph for this function:

6.40.1.4 mat_sub()

```
MATRIX mat_sub (
            MATRIX A,
            MATRIX B,
            MATRIX result )
```

Subtracts a matrix from another matrix.

Parameters

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

Returns

$\mathbf{A} - \mathbf{B}$

Here is the call graph for this function: Here is the caller graph for this function:

6.40.1.5 mat_subs()

```
MATRIX mat_subs (
            MATRIX A,
            mtype s,
            MATRIX result )
```

Subtracts a scalar from a matrix.

Parameters

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

Returns

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

Here is the call graph for this function: Here is the caller graph for this function:

6.40.1.6 mat_subs_neg()

```
MATRIX mat_subs_neg (
            MATRIX A,
            mtype s,
            MATRIX result )
```

Subtracts a matrix from a scalar.

Parameters

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

Returns

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

Here is the call graph for this function:

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 mat_submat()

```
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

Here is the call graph for this function: Here is the caller graph for this function:

6.42 matsum.c File Reference

Functions

- mtype mat_sum (MATRIX A)

Computes element-sum of a matrix.

- MATRIX mat_sum_row (MATRIX A, MATRIX result)

  Computes row-sum of a matrix.
- MATRIX mat_sum_col (MATRIX A, MATRIX result)

  Computes column-sum of a matrix.
- int int_vec_sum (INT_VECTOR A)

  Computes element-sum of an integer vector.

6.42.1 Function Documentation

6.42.1.1 int_vec_sum()

```
int int_vec_sum (
             INT_VECTOR A )
```

Computes element-sum of an integer vector.

Parameters

in	*A*	Input integer vector

Returns

$\mathrm{sum}(A)$

6.42.1.2 mat_sum()

```
mtype mat_sum (
             MATRIX A )
```

Computes element-sum of a matrix.

Parameters

in	*A*	Input matrix

Returns

$\mathrm{sum}(\mathbf{A})$

Here is the caller graph for this function:

6.42.1.3 mat_sum_col()

```
MATRIX mat_sum_col (
            MATRIX A,
            MATRIX result )
```

Computes column-sum of a matrix.

Parameters

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

Returns

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

Here is the call graph for this function: Here is the caller graph for this function:

6.42.1.4 mat_sum_row()

```
MATRIX mat_sum_row (
            MATRIX A,
            MATRIX result )
```

Computes row-sum of a matrix.

Parameters

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

Returns

$\mathbf{A}\mathbf{1}$

Here is the call graph for this function: Here is the caller graph for this function:

6.43 matsvd.c File Reference

Functions

- MATSTACK mat_svd (MATRIX a, int niters, MATSTACK result)

  Computes the SVD of a matrix.

6.43.1 Function Documentation

6.43.1.1 mat_svd()

```
MATSTACK mat_svd (
            MATRIX a,
            int niters,
            MATSTACK result )
```

Computes the SVD of a matrix.

Parameters

in	*a*	Input matrix
in	*niters*	Iterations to use
	result	Matrix stack to store the result

Returns

MATSTACK ($\mathbf{U}, \mathbf{S}, \mathbf{V}$)

Here is the call graph for this function:

6.44 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.

6.44.1 Function Documentation

6.44.1.1 mat_count_words_in_line()

```
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
in	*count*	Pointer to output count

Returns

EOF reached

Here is the caller graph for this function:

6.44.1.2 mat_dlmread()

```
MATRIX mat_dlmread (
             const char * fname )
```

Reads a matrix from a file.

Parameters

in	*fname*	Filename to read from

Returns

Output matrix

Here is the call graph for this function:

6.44.1.3 mat_dlmwrite()

```
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

Here is the call graph for this function:

6.44.1.4 mat_go_next_word()

```
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

6.44.1.5 mat_isnumeric()

```
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

Here is the caller graph for this function:

6.44.1.6 mat_read_word()

```
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
in	*c_word*	Pointer to word read

Returns

EOF reached

Here is the caller graph for this function:

6.45 mattimers.c File Reference

Functions

- __declspec (thread)

  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

- _Thread_local clock_t MAT_CLOCK_TIME

6.45.1 Function Documentation

6.45.1.1 __declspec()

```
__declspec (
             thread  )
```

Starts stopwatch timer.

6.45.1.2 mat_toc()

```
double mat_toc (
             void  )
```

Computes elapsed time from last start of timer.

Returns

Elapsed time

6.45.1.3 mat_toc_print()

```
void mat_toc_print (
             void  )
```

Computes and prints elapsed time from last start of timer on the stdout.

6.45.2 Variable Documentation

6.45.2.1 MAT_CLOCK_TIME

```
__thread clock_t MAT_CLOCK_TIME
```

6.46 mattoepz.c File Reference

Functions

- MATRIX mat_symtoeplz (MATRIX R, MATRIX result)

  Computes the symmetric Toeplitz matrix from a co-efficient matrix.

6.46.1 Function Documentation

6.46.1.1 mat_symtoeplz()

```
MATRIX mat_symtoeplz (
            MATRIX R,
            MATRIX result )
```

Computes the symmetric Toeplitz matrix from a co-efficient matrix.

Parameters

in	*R*	Input coefficient matrix
in	*result*	Matrix to store the result

Returns

$\mathrm{symtoep}(\mathbf{R})$

Here is the call graph for this function:

6.47 mattran.c File Reference

Functions

- MATRIX mat_tran (MATRIX A, MATRIX result)

  Computes the transpose of a matrix.

6.47.1 Function Documentation

6.47.1.1 mat_tran()

```
MATRIX mat_tran (
             MATRIX A,
             MATRIX result )
```

Computes the transpose of a matrix.

Parameters

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

Returns

$\mathbf{A}^T$

Here is the call graph for this function: Here is the caller graph for this function:

6.48 README.md File Reference

Index