Matrix2D

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 Mat2D Struct Reference

Dense row-major matrix of doubles.

`#include <Matrix2D.h>`

Public Attributes

- size_t rows
- size_t cols
- size_t stride_r
- double * elements

3.1.1 Detailed Description

Dense row-major matrix of doubles.

- rows: number of rows (height)
- cols: number of columns (width)
- stride_r: number of elements between successive rows in memory (for contiguous storage, stride_r == cols)
- elements: pointer to contiguous storage of size rows * cols

Definition at line 82 of file Matrix2D.h.

3.1.2 Member Data Documentation

3.1.2.1 cols

```
size_t Mat2D::cols
```

Definition at line 84 of file Matrix2D.h.

Referenced by mat2D_add(), mat2D_add_col_to_col(), mat2D_add_row_time_factor_to_row(), mat2D_add_row_to_row(), mat2D_alloc(), mat2D_calc_norma(), mat2D_col_is_all_digit(), mat2D_copy(), mat2D_copy_mat_to_mat_at_window(), mat2D_cross(), mat2D_det(), mat2D_det_2x2_mat(), mat2D_dot(), mat2D_dot_product(), mat2D_fill(), mat2D_fill_sequence(), mat2D_get_col(), mat2D_get_row(), mat2D_invert(), mat2D_LUP_decomposition_with_swap(), mat2D_make_identity(), mat2D_mat_is_all_digit(), mat2D_minor_alloc_fill_from_mat(), mat2D_mult(), mat2D_mult_row(), mat2D_offset2d(), mat2D_print(), mat2D_print_as_col(), mat2D_rand(), mat2D_row_is_all_digit(), mat2D_set_identity(), mat2D_set_rot_mat_x(), mat2D_set_rot_mat_y(), mat2D_set_rot_mat_z(), mat2D_solve_linear_sys_LUP_decomposition(), mat2D_sub(), mat2D_sub_col_to_col(), mat2D_sub_row_time_factor_to_row(), mat2D_sub_row_to_row(), mat2D_swap_rows(), mat2D_transpose(), and mat2D_triangulate().

3.1.2.2 elements

```
double* Mat2D::elements
```

Definition at line 86 of file Matrix2D.h.

Referenced by mat2D_alloc(), mat2D_free(), and mat2D_print_as_col().

3.1.2.3 rows

```
size_t Mat2D::rows
```

Definition at line 83 of file Matrix2D.h.

Referenced by mat2D_add(), mat2D_add_col_to_col(), mat2D_add_row_to_row(), mat2D_alloc(), mat2D_calc_norma(), mat2D_copy(), mat2D_copy_mat_to_mat_at_window(), mat2D_cross(), mat2D_det(), mat2D_det_2x2_mat(), mat2D_dot(), mat2D_dot_product(), mat2D_fill(), mat2D_fill_sequence(), mat2D_get_col(), mat2D_get_row(), mat2D_invert(), mat2D_LUP_decomposition_with_swap(), mat2D_make_identity(), mat2D_mat_is_all_digit(), mat2D_minor_alloc_fill_from_mat(), mat2D_mult(), mat2D_offset2d(), mat2D_print(), mat2D_print_as_col(), mat2D_rand(), mat2D_set_identity(), mat2D_set_rot_mat_x(), mat2D_set_rot_mat_y(), mat2D_set_rot_mat_z(), mat2D_solve_linear_sys_LUP_decomposition(), mat2D_sub(), mat2D_sub_col_to_col(), mat2D_sub_row_to_row(), mat2D_transpose(), and mat2D_triangulate().

3.1.2.4 stride_r

```
size_t Mat2D::stride_r
```

Definition at line 85 of file Matrix2D.h.

Referenced by mat2D_alloc(), and mat2D_offset2d().

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

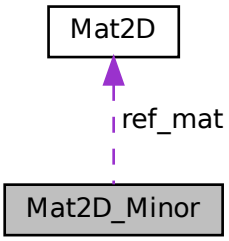
- Matrix2D.h

3.2 Mat2D_Minor Struct Reference

A minor "view" into a reference matrix.

`#include <Matrix2D.h>`

Collaboration diagram for Mat2D_Minor:



Public Attributes

- size_t rows
- size_t cols
- size_t stride_r
- size_t * rows_list
- size_t * cols_list
- Mat2D ref_mat

3.2.1 Detailed Description

A minor "view" into a reference matrix.

Represents a minor by excluding one row and one column of a reference matrix. It holds index lists mapping into the reference matrix, without owning the data of the reference matrix itself.

Memory ownership:

- rows_list and cols_list are heap-allocated by minor allocators and must be freed with mat2D_minor_free.
- The underlying matrix data (ref_mat.elements) is not owned by the minor and must not be freed by the minor functions.

Definition at line 120 of file Matrix2D.h.

3.2.2 Member Data Documentation

3.2.2.1 cols

```
size_t Mat2D_Minor::cols
```

Definition at line 122 of file Matrix2D.h.

Referenced by mat2D_det_2x2_mat_minor(), mat2D_minor_alloc_fill_from_mat(), mat2D_minor_alloc_fill_from_mat_minor(), mat2D_minor_det(), and mat2D_minor_print().

3.2.2.2 cols_list

```
size_t* Mat2D_Minor::cols_list
```

Definition at line 125 of file Matrix2D.h.

Referenced by mat2D_minor_alloc_fill_from_mat(), mat2D_minor_alloc_fill_from_mat_minor(), and mat2D_minor_free().

3.2.2.3 ref_mat

```
Mat2D Mat2D_Minor::ref_mat
```

Definition at line 126 of file Matrix2D.h.

Referenced by mat2D_minor_alloc_fill_from_mat(), and mat2D_minor_alloc_fill_from_mat_minor().

3.2.2.4 rows

```
size_t Mat2D_Minor::rows
```

Definition at line 121 of file Matrix2D.h.

Referenced by mat2D_det_2x2_mat_minor(), mat2D_minor_alloc_fill_from_mat(), mat2D_minor_alloc_fill_from_mat_minor(), mat2D_minor_det(), and mat2D_minor_print().

3.2.2.5 rows_list

```
size_t* Mat2D_Minor::rows_list
```

Definition at line 124 of file Matrix2D.h.

Referenced by mat2D_minor_alloc_fill_from_mat(), mat2D_minor_alloc_fill_from_mat_minor(), and mat2D_minor_free().

3.2.2.6 stride_r

```
size_t Mat2D_Minor::stride_r
```

Definition at line 123 of file Matrix2D.h.

Referenced by mat2D_minor_alloc_fill_from_mat(), and mat2D_minor_alloc_fill_from_mat_minor().

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

- Matrix2D.h

3.3 Mat2D_uint32 Struct Reference

Dense row-major matrix of uint32_t.

```
#include <Matrix2D.h>
```

Public Attributes

- size_t rows
- size_t cols
- size_t stride_r
- uint32_t * elements

3.3.1 Detailed Description

Dense row-major matrix of uint32_t.

- rows: number of rows (height)
- cols: number of columns (width)
- stride_r: number of elements between successive rows in memory (for contiguous storage, stride_r == cols)
- elements: pointer to contiguous storage of size rows * cols

Definition at line 99 of file Matrix2D.h.

3.3.2 Member Data Documentation

3.3.2.1 cols

```
size_t Mat2D_uint32::cols
```

Definition at line 101 of file Matrix2D.h.

Referenced by mat2D_alloc_uint32(), mat2D_fill_uint32(), and mat2D_offset2d_uint32().

3.3.2.2 elements

```
uint32_t* Mat2D_uint32::elements
```

Definition at line 103 of file Matrix2D.h.

Referenced by mat2D_alloc_uint32(), and mat2D_free_uint32().

3.3.2.3 rows

```
size_t Mat2D_uint32::rows
```

Definition at line 100 of file Matrix2D.h.

Referenced by mat2D_alloc_uint32(), mat2D_fill_uint32(), and mat2D_offset2d_uint32().

3.3.2.4 stride_r

```
size_t Mat2D_uint32::stride_r
```

Definition at line 102 of file Matrix2D.h.

Referenced by mat2D_alloc_uint32(), and mat2D_offset2d_uint32().

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

- Matrix2D.h

Chapter 4

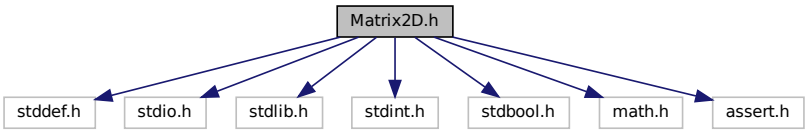
File Documentation

4.1 Matrix2D.h File Reference

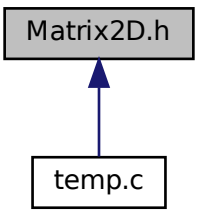
A single-header C library for simple 2D matrix operations on doubles and uint32_t, including allocation, basic arithmetic, linear algebra, and helpers (LUP, inverse, determinant, DCM, etc.).

```
#include <stddef.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <stdbool.h>
#include <math.h>
#include <assert.h>
```

Include dependency graph for Matrix2D.h:



This graph shows which files directly or indirectly include this file:



Classes

- struct Mat2D

  Dense row-major matrix of doubles.
- struct Mat2D_uint32

  Dense row-major matrix of uint32_t.
- struct Mat2D_Minor

  A minor "view" into a reference matrix.

Macros

- #define MATRIX2D_MALLOC malloc

  Allocation function used by the library.
- #define MATRIX2D_ASSERT assert

  Assertion macro used by the library for parameter validation.
- #define MAT2D_AT(m, i, j) (m).elements[i ∗ m.stride_r + j]

  Access element (i, j) of a Mat2D (0-based).
- #define MAT2D_AT_UINT32(m, i, j) (m).elements[i ∗ m.stride_r + j]

  Access element (i, j) of a Mat2D_uint32 (0-based).
- #define PI M_PI
- #define MAT2D_MINOR_AT(mm, i, j) MAT2D_AT(mm.ref_mat, mm.rows_list[i], mm.cols_list[j])

  Access element (i, j) of a Mat2D_Minor (0-based), dereferencing into the underlying reference matrix.
- #define MAT2D_PRINT(m) mat2D_print(m, #m, 0)

  Convenience macro to print a matrix with its variable name.
- #define MAT2D_PRINT_AS_COL(m) mat2D_print_as_col(m, #m, 0)

  Convenience macro to print a matrix as a single column with its name.
- #define MAT2D_MINOR_PRINT(mm) mat2D_minor_print(mm, #mm, 0)

  Convenience macro to print a minor with its variable name.
- #define mat2D_normalize(m) mat2D_mult((m), 1.0 / mat2D_calc_norma((m)))

  In-place normalization of all elements so that the Frobenius norm becomes 1.

Functions

- double mat2D_rand_double (void)

  Return a pseudo-random double in the range [0, 1].
- Mat2D mat2D_alloc (size_t rows, size_t cols)

  Allocate a rows x cols matrix of doubles.
- Mat2D_uint32 mat2D_alloc_uint32 (size_t rows, size_t cols)

  Allocate a rows x cols matrix of uint32_t.
- void mat2D_free (Mat2D m)

  Free the memory owned by a Mat2D (elements pointer).
- void mat2D_free_uint32 (Mat2D_uint32 m)

  Free the memory owned by a Mat2D_uint32 (elements pointer).
- size_t mat2D_offset2d (Mat2D m, size_t i, size_t j)

  Compute the linear offset of element (i, j) in a Mat2D.
- size_t mat2D_offset2d_uint32 (Mat2D_uint32 m, size_t i, size_t j)

  Compute the linear offset of element (i, j) in a Mat2D_uint32.
- void mat2D_fill (Mat2D m, double x)

  Fill all elements of a matrix of doubles with a scalar value.
- void mat2D_fill_sequence (Mat2D m, double start, double step)

Fill a matrix with an arithmetic sequence laid out in row-major order.

- void mat2D_fill_uint32 (Mat2D_uint32 m, uint32_t x)

  Fill all elements of a matrix of uint32_t with a scalar value.
- void mat2D_rand (Mat2D m, double low, double high)

  Fill a matrix with random doubles in [low, high).
- void mat2D_dot (Mat2D dst, Mat2D a, Mat2D b)

  *Matrix product: dst = a * b.*
- double mat2D_dot_product (Mat2D a, Mat2D b)

  Dot product between two vectors.
- void mat2D_cross (Mat2D dst, Mat2D a, Mat2D b)

  3D cross product: dst = a x b for 3x1 vectors.
- void mat2D_add (Mat2D dst, Mat2D a)

  In-place addition: dst += a.
- void mat2D_add_row_time_factor_to_row (Mat2D m, size_t des_r, size_t src_r, double factor)

  *Row operation: row(des_r) += factor * row(src_r).*
- void mat2D_sub (Mat2D dst, Mat2D a)

  In-place subtraction: dst -= a.
- void mat2D_sub_row_time_factor_to_row (Mat2D m, size_t des_r, size_t src_r, double factor)

  *Row operation: row(des_r) -= factor * row(src_r).*
- void mat2D_mult (Mat2D m, double factor)

  *In-place scalar multiplication: m *= factor.*
- void mat2D_mult_row (Mat2D m, size_t r, double factor)

  *In-place row scaling: row(r) *= factor.*
- void mat2D_print (Mat2D m, const char *name, size_t padding)

  Print a matrix to stdout with a name and indentation padding.
- void mat2D_print_as_col (Mat2D m, const char *name, size_t padding)

  Print a matrix as a flattened column vector to stdout.
- void mat2D_set_identity (Mat2D m)

  Set a square matrix to the identity matrix.
- double mat2D_make_identity (Mat2D m)

  Reduce a matrix to identity via Gauss-Jordan elimination and return the cumulative scaling factor.
- void mat2D_set_rot_mat_x (Mat2D m, float angle_deg)

  Set a 3x3 rotation matrix for rotation about the X-axis.
- void mat2D_set_rot_mat_y (Mat2D m, float angle_deg)

  Set a 3x3 rotation matrix for rotation about the Y-axis.
- void mat2D_set_rot_mat_z (Mat2D m, float angle_deg)

  Set a 3x3 rotation matrix for rotation about the Z-axis.
- void mat2D_set_DCM_zyx (Mat2D DCM, float yaw_deg, float pitch_deg, float roll_deg)

  Build a 3x3 direction cosine matrix (DCM) from Z-Y-X Euler angles.
- void mat2D_copy (Mat2D des, Mat2D src)

  Copy all elements from src to des.
- void mat2D_copy_mat_to_mat_at_window (Mat2D des, Mat2D src, size_t is, size_t js, size_t ie, size_t je)

  Copy a rectangular window from src into des.
- void mat2D_get_col (Mat2D des, size_t des_col, Mat2D src, size_t src_col)

  Copy a column from src into a column of des.
- void mat2D_add_col_to_col (Mat2D des, size_t des_col, Mat2D src, size_t src_col)

  Add a source column into a destination column: des[:, des_col] += src[:, src_col].
- void mat2D_sub_col_to_col (Mat2D des, size_t des_col, Mat2D src, size_t src_col)

  Subtract a source column from a destination column: des[:, des_col] -= src[:, src_col].
- void mat2D_swap_rows (Mat2D m, size_t r1, size_t r2)

  Swap two rows of a matrix in-place.

- void mat2D_get_row (Mat2D des, size_t des_row, Mat2D src, size_t src_row)

  Copy a row from src into a row of des.
- void mat2D_add_row_to_row (Mat2D des, size_t des_row, Mat2D src, size_t src_row)

  Add a source row into a destination row: des[des_row, :] += src[src_row, :].
- void mat2D_sub_row_to_row (Mat2D des, size_t des_row, Mat2D src, size_t src_row)

  Subtract a source row from a destination row: des[des_row, :] -= src[src_row, :].
- double mat2D_calc_norma (Mat2D m)

  Compute the Frobenius norm of a matrix, sqrt(sum(m_ij^ 2)).
- bool mat2D_mat_is_all_digit (Mat2D m, double digit)

  Check if all elements of a matrix equal a given digit.
- bool mat2D_row_is_all_digit (Mat2D m, double digit, size_t r)

  Check if all elements of a row equal a given digit.
- bool mat2D_col_is_all_digit (Mat2D m, double digit, size_t c)

  Check if all elements of a column equal a given digit.
- double mat2D_det_2x2_mat (Mat2D m)

  Determinant of a 2x2 matrix.
- double mat2D_triangulate (Mat2D m)

  Forward elimination to transform a matrix to upper triangular form.
- double mat2D_det (Mat2D m)

  Determinant of an NxN matrix via Gaussian elimination.
- void mat2D_LUP_decomposition_with_swap (Mat2D src, Mat2D l, Mat2D p, Mat2D u)

  Compute LUP decomposition: P∗A = L∗U with L unit diagonal.
- void mat2D_transpose (Mat2D des, Mat2D src)

  Transpose a matrix: des = src^ T.
- void mat2D_invert (Mat2D des, Mat2D src)

  Invert a square matrix using Gauss-Jordan elimination.
- void mat2D_solve_linear_sys_LUP_decomposition (Mat2D A, Mat2D x, Mat2D B)

  Solve the linear system A x = B using LUP decomposition.
- Mat2D_Minor mat2D_minor_alloc_fill_from_mat (Mat2D ref_mat, size_t i, size_t j)

  Allocate a minor view by excluding row i and column j of ref_mat.
- Mat2D_Minor mat2D_minor_alloc_fill_from_mat_minor (Mat2D_Minor ref_mm, size_t i, size_t j)

  Allocate a nested minor view from an existing minor by excluding row i and column j of the minor.
- void mat2D_minor_free (Mat2D_Minor mm)

  Free the index arrays owned by a minor.
- void mat2D_minor_print (Mat2D_Minor mm, const char ∗name, size_t padding)

  Print a minor matrix to stdout with a name and indentation padding.
- double mat2D_det_2x2_mat_minor (Mat2D_Minor mm)

  Determinant of a 2x2 minor.
- double mat2D_minor_det (Mat2D_Minor mm)

  Determinant of a minor via recursive expansion by minors.

4.1.1 Detailed Description

A single-header C library for simple 2D matrix operations on doubles and uint32_t, including allocation, basic arithmetic, linear algebra, and helpers (LUP, inverse, determinant, DCM, etc.).

- Storage is contiguous row-major (C-style). The element at row i, column j (0-based) is located at `elements[i * stride_r + j]`.

- Dense matrices of `double` are represented by Mat2D, and dense matrices of `uint32_t` are represented by Mat2D_uint32.
- Some routines assert shape compatibility using MATRIX2D_ASSERT.
- Random number generation uses the C library `rand()`; it is not cryptographically secure.
- Inversion is done via Gauss-Jordan elimination with partial pivoting only when a pivot is zero; this can be numerically unstable for ill-conditioned matrices. See notes below.
- To compile the implementation, define MATRIX2D_IMPLEMENTATION in exactly one translation unit before including this header.

Example: #define MATRIX2D_IMPLEMENTATION #include "matrix2d.h"

Note

This one-file library is heavily inspired by Tsoding's nn.h implementation of matrix creation and operations: `https://github.com/tsoding/nn.h` and the video: `https://youtu.be/L1TbWe8bVOc?list=PLpM-Dvs8t0VZPZKggcql-MmjaBdZKeDMw`

Warning

Numerical stability:

- There is a set of functions for minors that can be used to compute the determinant, but that approach is factorial in complexity and too slow for larger matrices. This library uses Gaussian elimination instead.
- The inversion function can fail or be unstable if pivot values become very small. Consider preconditioning or using a more robust decomposition (e.g., full pivoting, SVD) for ill-conditioned problems.

Definition in file Matrix2D.h.

4.1.2 Macro Definition Documentation

4.1.2.1 MAT2D_AT

```
#define MAT2D_AT(
                m,
                i,
                j ) (m).elements[i * m.stride_r + j]
```

Access element (i, j) of a Mat2D (0-based).

Warning

This macro does not perform bounds checking in the fast configuration. Use carefully.

Definition at line 146 of file Matrix2D.h.

4.1.2.2 MAT2D_AT_UINT32

```
#define MAT2D_AT_UINT32(
              m,
              i,
              j ) (m).elements[i * m.stride_r + j]
```

Access element (i, j) of a Mat2D_uint32 (0-based).

Warning

This macro does not perform bounds checking in the fast configuration. Use carefully.

Definition at line 147 of file Matrix2D.h.

4.1.2.3 MAT2D_MINOR_AT

```
#define MAT2D_MINOR_AT(
              mm,
              i,
              j ) MAT2D_AT(mm.ref_mat, mm.rows_list[i], mm.cols_list[j])
```

Access element (i, j) of a Mat2D_Minor (0-based), dereferencing into the underlying reference matrix.

Definition at line 159 of file Matrix2D.h.

4.1.2.4 MAT2D_MINOR_PRINT

```
#define MAT2D_MINOR_PRINT(
              mm ) mat2D_minor_print(mm, #mm, 0)
```

Convenience macro to print a minor with its variable name.

Definition at line 174 of file Matrix2D.h.

4.1.2.5 mat2D_normalize

```
#define mat2D_normalize(
              m ) mat2D_mult((m), 1.0 / mat2D_calc_norma((m)))
```

In-place normalization of all elements so that the Frobenius norm becomes 1.

Equivalent to: m *= 1.0 / mat2D_calc_norma(m).

Definition at line 181 of file Matrix2D.h.

4.1.2.6 MAT2D_PRINT

```
#define MAT2D_PRINT(
             m ) mat2D_print(m, #m, 0)
```

Convenience macro to print a matrix with its variable name.

Definition at line 164 of file Matrix2D.h.

4.1.2.7 MAT2D_PRINT_AS_COL

```
#define MAT2D_PRINT_AS_COL(
             m ) mat2D_print_as_col(m, #m, 0)
```

Convenience macro to print a matrix as a single column with its name.

Definition at line 169 of file Matrix2D.h.

4.1.2.8 MATRIX2D_ASSERT

```
#define MATRIX2D_ASSERT assert
```

Assertion macro used by the library for parameter validation.

Defaults to C `assert`. Override by defining MATRIX2D_ASSERT before including this header if you want custom behavior.

Definition at line 69 of file Matrix2D.h.

4.1.2.9 MATRIX2D_MALLOC

```
#define MATRIX2D_MALLOC malloc
```

Allocation function used by the library.

Defaults to `malloc`. Override by defining MATRIX2D_MALLOC before including this header if you want to use a custom allocator.

Definition at line 57 of file Matrix2D.h.

4.1.2.10 PI

```
#define PI M_PI
```

Definition at line 151 of file Matrix2D.h.

4.1.3 Function Documentation

4.1.3.1 mat2D_add()

```
void mat2D_add (
             Mat2D dst,
             Mat2D a )
```

In-place addition: dst += a.

Parameters

dst	Destination matrix to be incremented.
a	Summand of same shape as dst.

Precondition

Shapes match.

Definition at line 493 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.1.3.2 mat2D_add_col_to_col()

```
void mat2D_add_col_to_col (
            Mat2D des,
            size_t des_col,
            Mat2D src,
            size_t src_col )
```

Add a source column into a destination column: des[:, des_col] += src[:, src_col].

Parameters

des	Destination matrix (same row count as src).
des_col	Column index in destination.
src	Source matrix.
src_col	Column index in source.

Definition at line 825 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.1.3.3 mat2D_add_row_time_factor_to_row()

```
void mat2D_add_row_time_factor_to_row (
            Mat2D m,
            size_t des_r,
            size_t src_r,
            double factor )
```

Row operation: row(des_r) += factor * row(src_r).

Parameters

m	Matrix.

Parameters

des_r	Destination row index.
src_r	Source row index.
factor	Scalar multiplier.

Definition at line 511 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

4.1.3.4 mat2D_add_row_to_row()

```
void mat2D_add_row_to_row (
            Mat2D des,
            size_t des_row,
            Mat2D src,
            size_t src_row )
```

Add a source row into a destination row: des[des_row, :] += src[src_row, :].

Parameters

des	Destination matrix (same number of columns as src).
des_row	Row index in destination.
src	Source matrix.
src_row	Row index in source.

Definition at line 894 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.1.3.5 mat2D_alloc()

```
Mat2D mat2D_alloc (
            size_t rows,
            size_t cols )
```

Allocate a rows x cols matrix of doubles.

Parameters

rows	Number of rows (>= 1).
cols	Number of columns (>= 1).

Returns

A Mat2D with contiguous storage; must be freed with mat2D_free.

Postcondition

m.stride_r == cols.

Definition at line 275 of file Matrix2D.h.

References Mat2D::cols, Mat2D::elements, MATRIX2D_ASSERT, MATRIX2D_MALLOC, Mat2D::rows, and Mat2D::stride_r.

Referenced by main(), mat2D_det(), mat2D_invert(), mat2D_set_DCM_zyx(), and mat2D_solve_linear_sys_LUP_decomposition().

4.1.3.6 mat2D_alloc_uint32()

```
Mat2D_uint32 mat2D_alloc_uint32 (
            size_t rows,
            size_t cols )
```

Allocate a rows x cols matrix of uint32_t.

Parameters

rows	Number of rows (>= 1).
cols	Number of columns (>= 1).

Returns

A Mat2D_uint32 with contiguous storage; free with mat2D_free_uint32.

Postcondition

m.stride_r == cols.

Definition at line 294 of file Matrix2D.h.

References Mat2D_uint32::cols, Mat2D_uint32::elements, MATRIX2D_ASSERT, MATRIX2D_MALLOC, Mat2D_uint32::rows, and Mat2D_uint32::stride_r.

4.1.3.7 mat2D_calc_norma()

```
double mat2D_calc_norma (
            Mat2D m )
```

Compute the Frobenius norm of a matrix, sqrt(sum(m_ij^2)).

Parameters

m	Matrix.

Returns

Frobenius norm.

Definition at line 928 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, and Mat2D::rows.

Referenced by main().

4.1.3.8 mat2D_col_is_all_digit()

```
bool mat2D_col_is_all_digit (
            Mat2D m,
            double digit,
            size_t c )
```

Check if all elements of a column equal a given digit.

Parameters

m	Matrix.
digit	Value to compare.
c	Column index.

Returns

true if every element equals digit, false otherwise.

Definition at line 982 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_det().

4.1.3.9 mat2D_copy()

```
void mat2D_copy (
            Mat2D des,
            Mat2D src )
```

Copy all elements from src to des.

Parameters

des	Destination matrix.
src	Source matrix.

Precondition

Shapes match.

Definition at line 765 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

Referenced by main(), mat2D_det(), mat2D_invert(), and mat2D_LUP_decomposition_with_swap().

4.1.3.10 mat2D_copy_mat_to_mat_at_window()

```
void mat2D_copy_mat_to_mat_at_window (
            Mat2D des,
            Mat2D src,
            size_t is,
            size_t js,
            size_t ie,
            size_t je )
```

Copy a rectangular window from src into des.

Parameters

des	Destination matrix. Must have size (ie - is + 1) x (je - js + 1).
src	Source matrix.
is	Start row index in src (inclusive).
js	Start column index in src (inclusive).
ie	End row index in src (inclusive).
je	End column index in src (inclusive).

Precondition

0 <= is <= ie < src.rows, 0 <= js <= je < src.cols.

Definition at line 787 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.1.3.11 mat2D_cross()

```
void mat2D_cross (
            Mat2D dst,
            Mat2D a,
            Mat2D b )
```

3D cross product: dst = a x b for 3x1 vectors.

Parameters

dst	3x1 destination vector.
a	3x1 input vector.
b	3x1 input vector.

Precondition

All matrices have shape 3x1.

Definition at line 476 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.1.3.12 mat2D_det()

```
double mat2D_det (
            Mat2D m )
```

Determinant of an NxN matrix via Gaussian elimination.

Parameters

m	Square matrix.

Returns

det(m).

Copies m internally, triangulates it, and returns the product of diagonal elements (adjusted by any scaling factor as implemented).

Definition at line 1049 of file Matrix2D.h.

References Mat2D::cols, mat2D_alloc(), MAT2D_AT, mat2D_col_is_all_digit(), mat2D_copy(), mat2D_free(), mat2D_row_is_all_digit(), mat2D_triangulate(), MATRIX2D_ASSERT, and Mat2D::rows.

Referenced by mat2D_invert().

4.1.3.13 mat2D_det_2x2_mat()

```
double mat2D_det_2x2_mat (
            Mat2D m )
```

Determinant of a 2x2 matrix.

Parameters

m	Matrix (must be 2x2).

Returns

det(m) = a11 a22 - a12 a21.

Definition at line 997 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.1.3.14 mat2D_det_2x2_mat_minor()

```
double mat2D_det_2x2_mat_minor (
            Mat2D_Minor mm )
```

Determinant of a 2x2 minor.

Parameters

mm	Minor (must be 2x2).

Returns

det(mm).

Definition at line 1380 of file Matrix2D.h.

References Mat2D_Minor::cols, MAT2D_MINOR_AT, MATRIX2D_ASSERT, and Mat2D_Minor::rows.

Referenced by mat2D_minor_det().

4.1.3.15 mat2D_dot()

```
void mat2D_dot (
            Mat2D dst,
            Mat2D a,
            Mat2D b )
```

Matrix product: dst = a ∗ b.

Parameters

dst	Destination matrix (size a.rows x b.cols).
a	Left matrix (size a.rows x a.cols).
b	Right matrix (size a.cols x b.cols).

Precondition

a.cols == b.rows, dst.rows == a.rows, dst.cols == b.cols.

Postcondition

dst is overwritten.

Definition at line 421 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

Referenced by main(), mat2D_set_DCM_zyx(), and mat2D_solve_linear_sys_LUP_decomposition().

4.1.3.16 mat2D_dot_product()

```
double mat2D_dot_product (
            Mat2D a,
            Mat2D b )
```

Dot product between two vectors.

Parameters

a	Vector (shape n x 1 or 1 x n).
b	Vector (same shape as a).

Returns

The scalar dot product sum.

Precondition

a.rows == b.rows, a.cols == b.cols, and one dimension equals 1.

Definition at line 447 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.1.3.17 mat2D_fill()

```
void mat2D_fill (
            Mat2D m,
            double x )
```

Fill all elements of a matrix of doubles with a scalar value.

Parameters

m	Matrix to fill.
x	Value to assign to every element.

Definition at line 359 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, and Mat2D::rows.

Referenced by main(), mat2D_invert(), mat2D_LUP_decomposition_with_swap(), and mat2D_solve_linear_sys_LUP_decomposition(

4.1.3.18 mat2D_fill_sequence()

```
void mat2D_fill_sequence (
            Mat2D m,
            double start,
            double step )
```

Fill a matrix with an arithmetic sequence laid out in row-major order.

Parameters

m	Matrix to fill.
start	First value in the sequence.
step	Increment between consecutive elements.

Element at linear index k gets value start + step $*$ k.

Definition at line 375 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_offset2d(), and Mat2D::rows.

Referenced by main().

4.1.3.19 mat2D_fill_uint32()

```
void mat2D_fill_uint32 (
            Mat2D_uint32 m,
            uint32_t x )
```

Fill all elements of a matrix of uint32_t with a scalar value.

Parameters

m	Matrix to fill.
x	Value to assign to every element.

Definition at line 388 of file Matrix2D.h.

References Mat2D_uint32::cols, MAT2D_AT_UINT32, and Mat2D_uint32::rows.

4.1.3.20 mat2D_free()

```
void mat2D_free (
            Mat2D m )
```

Free the memory owned by a Mat2D (elements pointer).

Parameters

m	Matrix whose elements were allocated via MATRIX2D_MALLOC.

Note

Safe to call with m.elements == NULL.

Definition at line 311 of file Matrix2D.h.

References Mat2D::elements.

Referenced by mat2D_det(), mat2D_invert(), mat2D_set_DCM_zyx(), and mat2D_solve_linear_sys_LUP_decomposition().

4.1.3.21 mat2D_free_uint32()

```
void mat2D_free_uint32 (
            Mat2D_uint32 m )
```

Free the memory owned by a Mat2D_uint32 (elements pointer).

Parameters

m	Matrix whose elements were allocated via MATRIX2D_MALLOC.

Note

Safe to call with m.elements == NULL.

Definition at line 321 of file Matrix2D.h.

References Mat2D_uint32::elements.

4.1.3.22 mat2D_get_col()

```
void mat2D_get_col (
            Mat2D des,
            size_t des_col,
            Mat2D src,
            size_t src_col )
```

Copy a column from src into a column of des.

Parameters

des	Destination matrix (same row count as src).
des_col	Column index in destination.
src	Source matrix.
src_col	Column index in source.

Definition at line 807 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.1.3.23 mat2D_get_row()

```
void mat2D_get_row (
            Mat2D des,
            size_t des_row,
            Mat2D src,
            size_t src_row )
```

Copy a row from src into a row of des.

Parameters

des	Destination matrix (same number of columns as src).
des_row	Row index in destination.
src	Source matrix.
src_row	Row index in source.

Definition at line 876 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.1.3.24 mat2D_invert()

```
void mat2D_invert (
            Mat2D des,
            Mat2D src )
```

Invert a square matrix using Gauss-Jordan elimination.

Parameters

des	Destination matrix (same shape as src).
src	Source square matrix.

Precondition

src is square and nonsingular.

If det(src) == 0, prints an error and sets des to all zeros.

Warning

May be numerically unstable for ill-conditioned matrices.

Definition at line 1166 of file Matrix2D.h.

References Mat2D::cols, mat2D_alloc(), MAT2D_AT, mat2D_copy(), mat2D_det(), mat2D_fill(), mat2D_free(), mat2D_mult_row(), mat2D_set_identity(), mat2D_sub_row_time_factor_to_row(), mat2D_swap_rows(), MATRIX2D_ASSERT, and Mat2D::rows.

Referenced by mat2D_solve_linear_sys_LUP_decomposition().

4.1.3.25 mat2D_LUP_decomposition_with_swap()

```
void mat2D_LUP_decomposition_with_swap (
            Mat2D src,
            Mat2D l,
            Mat2D p,
            Mat2D u )
```

Compute LUP decomposition: P∗A = L∗U with L unit diagonal.

Parameters

src	Input matrix A (not modified).
l	Lower triangular matrix with unit diagonal (output).
p	Permutation matrix (output).
u	Upper triangular matrix (output).

Precondition

l, p, u are allocated to match src shape; src is square.

Definition at line 1104 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_copy(), mat2D_fill(), mat2D_set_identity(), mat2D_sub_row_time_factor_to_row(), mat2D_swap_rows(), and Mat2D::rows.

Referenced by main(), and mat2D_solve_linear_sys_LUP_decomposition().

4.1.3.26 mat2D_make_identity()

```
double mat2D_make_identity (
            Mat2D m )
```

Reduce a matrix to identity via Gauss-Jordan elimination and return the cumulative scaling factor.

Parameters

m	Matrix reduced in-place to identity (if nonsingular).

Returns

The product of row scaling factors applied during elimination.

Note

Intended as a helper for determinant-related operations.

Warning

Not robust to singular or ill-conditioned matrices.

Definition at line 640 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_mult_row(), mat2D_sub_row_time_factor_to_row(), mat2D_swap_rows(), and Mat2D::rows.

4.1.3.27 mat2D_mat_is_all_digit()

```
bool mat2D_mat_is_all_digit (
            Mat2D m,
            double digit )
```

Check if all elements of a matrix equal a given digit.

Parameters

m	Matrix.
digit	Value to compare.

Returns

true if every element equals digit, false otherwise.

Definition at line 946 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, and Mat2D::rows.

4.1.3.28 mat2D_minor_alloc_fill_from_mat()

```
Mat2D_Minor mat2D_minor_alloc_fill_from_mat (
            Mat2D ref_mat,
            size_t i,
            size_t j )
```

Allocate a minor view by excluding row i and column j of ref_mat.

Parameters

ref_mat	Reference square matrix.
i	Excluded row index in ref_mat.
j	Excluded column index in ref_mat.

Returns

A Mat2D_Minor that references ref_mat.

Note

Free rows_list and cols_list with mat2D_minor_free when done.

Definition at line 1276 of file Matrix2D.h.

References Mat2D::cols, Mat2D_Minor::cols, Mat2D_Minor::cols_list, MATRIX2D_ASSERT, MATRIX2D_MALLOC, Mat2D_Minor::ref_mat, Mat2D::rows, Mat2D_Minor::rows, Mat2D_Minor::rows_list, and Mat2D_Minor::stride_r.

4.1.3.29 mat2D_minor_alloc_fill_from_mat_minor()

```
Mat2D_Minor mat2D_minor_alloc_fill_from_mat_minor (
            Mat2D_Minor ref_mm,
            size_t i,
            size_t j )
```

Allocate a nested minor view from an existing minor by excluding row i and column j of the minor.

Parameters

ref_mm	Reference minor.
i	Excluded row index in the minor.
j	Excluded column index in the minor.

Returns

A new Mat2D_Minor that references the same underlying matrix.

Note

Free rows_list and cols_list with mat2D_minor_free when done.

Definition at line 1315 of file Matrix2D.h.

References Mat2D_Minor::cols, Mat2D_Minor::cols_list, MATRIX2D_ASSERT, MATRIX2D_MALLOC, Mat2D_Minor::ref_mat, Mat2D_Minor::rows, Mat2D_Minor::rows_list, and Mat2D_Minor::stride_r.

Referenced by mat2D_minor_det().

4.1.3.30 mat2D_minor_det()

```
double mat2D_minor_det (
            Mat2D_Minor mm )
```

Determinant of a minor via recursive expansion by minors.

Parameters

mm	Square minor.

Returns

det(mm).

Warning

Exponential complexity (factorial). Intended for educational or very small matrices only.

Definition at line 1393 of file Matrix2D.h.

References Mat2D_Minor::cols, mat2D_det_2x2_mat_minor(), mat2D_minor_alloc_fill_from_mat_minor(), MAT2D_MINOR_AT, mat2D_minor_free(), MATRIX2D_ASSERT, and Mat2D_Minor::rows.

4.1.3.31 mat2D_minor_free()

```
void mat2D_minor_free (
            Mat2D_Minor mm )
```

Free the index arrays owned by a minor.

Parameters

mm	Minor to free.

Note

After this call, mm.rows_list and mm.cols_list are invalid.

Definition at line 1350 of file Matrix2D.h.

References Mat2D_Minor::cols_list, and Mat2D_Minor::rows_list.

Referenced by mat2D_minor_det().

4.1.3.32 mat2D_minor_print()

```
void mat2D_minor_print (
            Mat2D_Minor mm,
            const char * name,
            size_t padding )
```

Print a minor matrix to stdout with a name and indentation padding.

Parameters

mm	Minor to print.
name	Label to print.
padding	Left padding in spaces.

Definition at line 1362 of file Matrix2D.h.

References Mat2D_Minor::cols, MAT2D_MINOR_AT, and Mat2D_Minor::rows.

4.1.3.33 mat2D_mult()

```
void mat2D_mult (
            Mat2D m,
            double factor )
```

In-place scalar multiplication: m *= factor.

Parameters

m	Matrix.
factor	Scalar multiplier.

Definition at line 554 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, and Mat2D::rows.

Referenced by main().

4.1.3.34 mat2D_mult_row()

```
void mat2D_mult_row (
            Mat2D m,
            size_t r,
            double factor )
```

In-place row scaling: row(r) ∗= factor.

Parameters

m	Matrix.
r	Row index.
factor	Scalar multiplier.

Definition at line 569 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_invert(), and mat2D_make_identity().

4.1.3.35 mat2D_offset2d()

```
size_t mat2D_offset2d (
            Mat2D m,
            size_t i,
            size_t j )
```

Compute the linear offset of element (i, j) in a Mat2D.

Parameters

m	Matrix.
i	Row index (0-based).
j	Column index (0-based).

Returns

The linear offset i * stride_r + j.

Precondition

0 <= i < rows, 0 <= j < cols (asserted).

Definition at line 334 of file Matrix2D.h.

References Mat2D::cols, MATRIX2D_ASSERT, Mat2D::rows, and Mat2D::stride_r.

Referenced by mat2D_fill_sequence().

4.1.3.36 mat2D_offset2d_uint32()

```
size_t mat2D_offset2d_uint32 (
            Mat2D_uint32 m,
            size_t i,
            size_t j )
```

Compute the linear offset of element (i, j) in a Mat2D_uint32.

Parameters

m	Matrix.
i	Row index (0-based).
j	Column index (0-based).

Returns

The linear offset i * stride_r + j.

Precondition

0 <= i < rows, 0 <= j < cols (asserted).

Definition at line 348 of file Matrix2D.h.

References Mat2D_uint32::cols, MATRIX2D_ASSERT, Mat2D_uint32::rows, and Mat2D_uint32::stride_r.

4.1.3.37 mat2D_print()

```
void mat2D_print (
            Mat2D m,
            const char * name,
            size_t padding )
```

Print a matrix to stdout with a name and indentation padding.

Parameters

m	Matrix to print.
name	Label to print.
padding	Left padding in spaces.

Definition at line 582 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, and Mat2D::rows.

4.1.3.38 mat2D_print_as_col()

```
void mat2D_print_as_col (
            Mat2D m,
            const char * name,
            size_t padding )
```

Print a matrix as a flattened column vector to stdout.

Parameters

m	Matrix to print (flattened in row-major).
name	Label to print.
padding	Left padding in spaces.

Definition at line 601 of file Matrix2D.h.

References Mat2D::cols, Mat2D::elements, and Mat2D::rows.

4.1.3.39 mat2D_rand()

```
void mat2D_rand (
            Mat2D m,
            double low,
            double high )
```

Fill a matrix with random doubles in [low, high).

Parameters

m	Matrix to fill.
low	Lower bound (inclusive).
high	Upper bound (exclusive).

Precondition

high $>$ low.

Definition at line 404 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_rand_double(), and Mat2D::rows.

4.1.3.40 mat2D_rand_double()

```
double mat2D_rand_double (
            void  )
```

Return a pseudo-random double in the range [0, 1].

Note

Uses C library rand() and RAND_MAX. Not cryptographically secure.

Definition at line 263 of file Matrix2D.h.

Referenced by mat2D_rand().

4.1.3.41 mat2D_row_is_all_digit()

```
bool mat2D_row_is_all_digit (
            Mat2D m,
            double digit,
            size_t r )
```

Check if all elements of a row equal a given digit.

Parameters

m	Matrix.
digit	Value to compare.
r	Row index.

Returns

true if every element equals digit, false otherwise.

Definition at line 965 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_det().

4.1.3.42 mat2D_set_DCM_zyx()

```
void mat2D_set_DCM_zyx (
            Mat2D DCM,
            float yaw_deg,
            float pitch_deg,
            float roll_deg )
```

Build a 3x3 direction cosine matrix (DCM) from Z-Y-X Euler angles.

Parameters

DCM	3x3 destination matrix.
yaw_deg	Rotation about Z in degrees.
pitch_deg	Rotation about Y in degrees.
roll_deg	Rotation about X in degrees.

Computes DCM = R_x(roll) ∗ R_y(pitch) ∗ R_z(yaw).

Definition at line 740 of file Matrix2D.h.

References mat2D_alloc(), mat2D_dot(), mat2D_free(), mat2D_set_rot_mat_x(), mat2D_set_rot_mat_y(), and mat2D_set_rot_mat_z().

4.1.3.43 mat2D_set_identity()

```
void mat2D_set_identity (
            Mat2D m )
```

Set a square matrix to the identity matrix.

Parameters

m	Matrix (must be square).

Precondition

m.rows == m.cols.

Definition at line 616 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

Referenced by main(), mat2D_invert(), mat2D_LUP_decomposition_with_swap(), mat2D_set_rot_mat_x(), mat2D_set_rot_mat_y(), and mat2D_set_rot_mat_z().

4.1.3.44 mat2D_set_rot_mat_x()

```
void mat2D_set_rot_mat_x (
            Mat2D m,
            float angle_deg )
```

Set a 3x3 rotation matrix for rotation about the X-axis.

Parameters

m	3x3 destination matrix.
angle_deg	Angle in degrees.

Definition at line 686 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_set_identity(), MATRIX2D_ASSERT, PI, and Mat2D::rows.

Referenced by mat2D_set_DCM_zyx().

4.1.3.45 mat2D_set_rot_mat_y()

```
void mat2D_set_rot_mat_y (
            Mat2D m,
            float angle_deg )
```

Set a 3x3 rotation matrix for rotation about the Y-axis.

Parameters

m	3x3 destination matrix.
angle_deg	Angle in degrees.

Definition at line 703 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_set_identity(), MATRIX2D_ASSERT, PI, and Mat2D::rows.

Referenced by mat2D_set_DCM_zyx().

4.1.3.46 mat2D_set_rot_mat_z()

```
void mat2D_set_rot_mat_z (
            Mat2D m,
            float angle_deg )
```

Set a 3x3 rotation matrix for rotation about the Z-axis.

Parameters

m	3x3 destination matrix.
angle_deg	Angle in degrees.

Definition at line 720 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_set_identity(), MATRIX2D_ASSERT, PI, and Mat2D::rows.

Referenced by mat2D_set_DCM_zyx().

4.1.3.47 mat2D_solve_linear_sys_LUP_decomposition()

```
void mat2D_solve_linear_sys_LUP_decomposition (
            Mat2D A,
            Mat2D x,
            Mat2D B )
```

Solve the linear system A x = B using LUP decomposition.

Parameters

A	Coefficient matrix (NxN).
x	Solution vector (N x 1) (output).
B	Right-hand side vector (N x 1).

Internally computes LUP and uses explicit inverses of L and U.

Warning

Forming inverses explicitly can be less stable; a forward/backward substitution would be preferable for production-quality code.

Definition at line 1233 of file Matrix2D.h.

References Mat2D::cols, mat2D_alloc(), mat2D_dot(), mat2D_fill(), mat2D_free(), mat2D_invert(), mat2D_LUP_decomposition_with_s
MATRIX2D_ASSERT, and Mat2D::rows.

Referenced by main().

4.1.3.48 mat2D_sub()

```
void mat2D_sub (
            Mat2D dst,
            Mat2D a )
```

In-place subtraction: dst -= a.

Parameters

dst	Destination matrix to be decremented.
a	Subtrahend of same shape as dst.

Precondition

Shapes match.

Definition at line 524 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

Referenced by main().

4.1.3.49 mat2D_sub_col_to_col()

```
void mat2D_sub_col_to_col (
            Mat2D des,
            size_t des_col,
            Mat2D src,
            size_t src_col )
```

Subtract a source column from a destination column: des[:, des_col] -= src[:, src_col].

Parameters

des	Destination matrix (same row count as src).
des_col	Column index in destination.
src	Source matrix.
src_col	Column index in source.

Definition at line 843 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.1.3.50 mat2D_sub_row_time_factor_to_row()

```
void mat2D_sub_row_time_factor_to_row (
            Mat2D m,
            size_t des_r,
            size_t src_r,
            double factor )
```

Row operation: row(des_r) -= factor * row(src_r).

Parameters

m	Matrix.
des_r	Destination row index.
src_r	Source row index.
factor	Scalar multiplier.

Definition at line 542 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_invert(), mat2D_LUP_decomposition_with_swap(), mat2D_make_identity(), and mat2D_triangulate().

4.1.3.51 mat2D_sub_row_to_row()

```
void mat2D_sub_row_to_row (
            Mat2D des,
            size_t des_row,
            Mat2D src,
            size_t src_row )
```

Subtract a source row from a destination row: des[des_row, :] -= src[src_row, :].

Parameters

des	Destination matrix (same number of columns as src).
des_row	Row index in destination.
src	Source matrix.
src_row	Row index in source.

Definition at line 912 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.1.3.52 mat2D_swap_rows()

```
void mat2D_swap_rows (
            Mat2D m,
            size_t r1,
            size_t r2 )
```

Swap two rows of a matrix in-place.

Parameters

m	Matrix.
r1	First row index.
r2	Second row index.

Definition at line 860 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_invert(), mat2D_LUP_decomposition_with_swap(), mat2D_make_identity(), and mat2D_triangulate().

4.1.3.53 mat2D_transpose()

```
void mat2D_transpose (
            Mat2D des,
            Mat2D src )
```

Transpose a matrix: des = src^T.

Parameters

des	Destination matrix (shape src.cols x src.rows).
src	Source matrix.

Definition at line 1146 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MATRIX2D_ASSERT, and Mat2D::rows.

4.1.3.54 mat2D_triangulate()

```
double mat2D_triangulate (
            Mat2D m )
```

Forward elimination to transform a matrix to upper triangular form.

Parameters

m	Matrix transformed in-place.

Returns

Product of row scaling factors (currently 1 in this implementation).

Note

Used as part of determinant computation via triangularization.

Warning

Not robust for linearly dependent rows or tiny pivots.

Definition at line 1010 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_sub_row_time_factor_to_row(), mat2D_swap_rows(), and Mat2D::rows.

Referenced by mat2D_det().

4.2 Matrix2D.h

```
00001
00039 #ifndef MATRIX2D_H_
00040 #define MATRIX2D_H_
00041
00042 #include <stddef.h>
00043 #include <stdio.h>
00044 #include <stdlib.h>
00045 #include <stdint.h>
00046 #include <stdbool.h>
00047     #include <math.h>
00048
00056 #ifndef MATRIX2D_MALLOC
00057 #define MATRIX2D_MALLOC malloc
00058 #endif //MATRIX2D_MALLOC
00059
00067 #ifndef MATRIX2D_ASSERT
00068 #include <assert.h>
00069 #define MATRIX2D_ASSERT assert
00070 #endif //MATRIX2D_ASSERT
00071
00082 typedef struct {
00083     size_t rows;
00084     size_t cols;
00085     size_t stride_r; /* how many element you need to traves to get to the element underneath */
00086     double *elements;
00087 } Mat2D;
00088
00099 typedef struct {
00100     size_t rows;
00101     size_t cols;
00102     size_t stride_r; /* how many element you need to traves to get to the element underneath */
00103     uint32_t *elements;
00104 } Mat2D_uint32;
00105
00120 typedef struct {
00121     size_t rows;
00122     size_t cols;
00123     size_t stride_r; /* how many element you need to traves to get to the element underneath */
00124     size_t *rows_list;
00125     size_t *cols_list;
00126     Mat2D ref_mat;
00127 } Mat2D_Minor;
00128
00142 #if 0
00143 #define MAT2D_AT(m, i, j) (m).elements[mat2D_offset2d((m), (i), (j))]
00144 #define MAT2D_AT_UINT32(m, i, j) (m).elements[mat2D_offset2d_uint32((m), (i), (j))]
00145 #else /* use this macro for batter performance but no assertion */
00146 #define MAT2D_AT(m, i, j) (m).elements[i * m.stride_r + j]
00147 #define MAT2D_AT_UINT32(m, i, j) (m).elements[i * m.stride_r + j]
00148 #endif
00149
00150 #ifndef PI
00151     #define PI M_PI
00152 #endif
00153
00159 #define MAT2D_MINOR_AT(mm, i, j) MAT2D_AT(mm.ref_mat, mm.rows_list[i], mm.cols_list[j])
00164 #define MAT2D_PRINT(m) mat2D_print(m, #m, 0)
00169 #define MAT2D_PRINT_AS_COL(m) mat2D_print_as_col(m, #m, 0)
```

```
00174 #define MAT2D_MINOR_PRINT(mm) mat2D_minor_print(mm, #mm, 0)
00181 #define mat2D_normalize(m) mat2D_mult((m), 1.0 / mat2D_calc_norma((m)))
00182
00183 double mat2D_rand_double(void);
00184
00185 Mat2D mat2D_alloc(size_t rows, size_t cols);
00186 Mat2D_uint32 mat2D_alloc_uint32(size_t rows, size_t cols);
00187 void mat2D_free(Mat2D m);
00188 void mat2D_free_uint32(Mat2D_uint32 m);
00189 size_t mat2D_offset2d(Mat2D m, size_t i, size_t j);
00190 size_t mat2D_offset2d_uint32(Mat2D_uint32 m, size_t i, size_t j);
00191
00192 void mat2D_fill(Mat2D m, double x);
00193 void mat2D_fill_sequence(Mat2D m, double start, double step);
00194 void mat2D_fill_uint32(Mat2D_uint32 m, uint32_t x);
00195 void mat2D_rand(Mat2D m, double low, double high);
00196
00197 void mat2D_dot(Mat2D dst, Mat2D a, Mat2D b);
00198 double mat2D_dot_product(Mat2D a, Mat2D b);
00199 void mat2D_cross(Mat2D dst, Mat2D a, Mat2D b);
00200
00201 void mat2D_add(Mat2D dst, Mat2D a);
00202 void mat2D_add_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor);
00203
00204 void mat2D_sub(Mat2D dst, Mat2D a);
00205 void mat2D_sub_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor);
00206
00207 void mat2D_mult(Mat2D m, double factor);
00208 void mat2D_mult_row(Mat2D m, size_t r, double factor);
00209
00210 void mat2D_print(Mat2D m, const char *name, size_t padding);
00211 void mat2D_print_as_col(Mat2D m, const char *name, size_t padding);
00212
00213 void mat2D_set_identity(Mat2D m);
00214 double mat2D_make_identity(Mat2D m);
00215 void mat2D_set_rot_mat_x(Mat2D m, float angle_deg);
00216 void mat2D_set_rot_mat_y(Mat2D m, float angle_deg);
00217 void mat2D_set_rot_mat_z(Mat2D m, float angle_deg);
00218 void mat2D_set_DCM_zyx(Mat2D DCM, float yaw_deg, float pitch_deg, float roll_deg);
00219
00220 void mat2D_copy(Mat2D des, Mat2D src);
00221 void mat2D_copy_mat_to_mat_at_window(Mat2D des, Mat2D src, size_t is, size_t js, size_t ie, size_t
       je);
00222
00223 void mat2D_get_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00224 void mat2D_add_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00225 void mat2D_sub_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00226
00227 void mat2D_swap_rows(Mat2D m, size_t r1, size_t r2);
00228 void mat2D_get_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
00229 void mat2D_add_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
00230 void mat2D_sub_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
00231
00232 double mat2D_calc_norma(Mat2D m);
00233
00234 bool mat2D_mat_is_all_digit(Mat2D m, double digit);
00235 bool mat2D_row_is_all_digit(Mat2D m, double digit, size_t r);
00236 bool mat2D_col_is_all_digit(Mat2D m, double digit, size_t c);
00237
00238 double mat2D_det_2x2_mat(Mat2D m);
00239 double mat2D_triangulate(Mat2D m);
00240 double mat2D_det(Mat2D m);
00241 void mat2D_LUP_decomposition_with_swap(Mat2D src, Mat2D l, Mat2D p, Mat2D u);
00242 void mat2D_transpose(Mat2D des, Mat2D src);
00243 void mat2D_invert(Mat2D des, Mat2D src);
00244 void mat2D_solve_linear_sys_LUP_decomposition(Mat2D A, Mat2D x, Mat2D B);
00245
00246 Mat2D_Minor mat2D_minor_alloc_fill_from_mat(Mat2D ref_mat, size_t i, size_t j);
00247 Mat2D_Minor mat2D_minor_alloc_fill_from_mat_minor(Mat2D_Minor ref_mm, size_t i, size_t j);
00248 void mat2D_minor_free(Mat2D_Minor mm);
00249 void mat2D_minor_print(Mat2D_Minor mm, const char *name, size_t padding);
00250 double mat2D_det_2x2_mat_minor(Mat2D_Minor mm);
00251 double mat2D_minor_det(Mat2D_Minor mm);
00252
00253 #endif // MATRIX2D_H_
00254
00255 #ifdef MATRIX2D_IMPLEMENTATION
00256 #undef MATRIX2D_IMPLEMENTATION
00257
00258
00263 double mat2D_rand_double(void)
00264 {
00265     return (double) rand() / (double) RAND_MAX;
00266 }
00267
00275 Mat2D mat2D_alloc(size_t rows, size_t cols)
00276 {
```

```
00277     Mat2D m;
00278     m.rows = rows;
00279     m.cols = cols;
00280     m.stride_r = cols;
00281     m.elements = (double*)MATRIX2D_MALLOC(sizeof(double)*rows*cols);
00282     MATRIX2D_ASSERT(m.elements != NULL);
00283 
00284     return m;
00285 }
00286 
00294 Mat2D_uint32 mat2D_alloc_uint32(size_t rows, size_t cols)
00295 {
00296     Mat2D_uint32 m;
00297     m.rows = rows;
00298     m.cols = cols;
00299     m.stride_r = cols;
00300     m.elements = (uint32_t*)MATRIX2D_MALLOC(sizeof(uint32_t)*rows*cols);
00301     MATRIX2D_ASSERT(m.elements != NULL);
00302 
00303     return m;
00304 }
00305 
00311 void mat2D_free(Mat2D m)
00312 {
00313     free(m.elements);
00314 }
00315 
00321 void mat2D_free_uint32(Mat2D_uint32 m)
00322 {
00323     free(m.elements);
00324 }
00325 
00334 size_t mat2D_offset2d(Mat2D m, size_t i, size_t j)
00335 {
00336     MATRIX2D_ASSERT(i < m.rows && j < m.cols);
00337     return i * m.stride_r + j;
00338 }
00339 
00348 size_t mat2D_offset2d_uint32(Mat2D_uint32 m, size_t i, size_t j)
00349 {
00350     MATRIX2D_ASSERT(i < m.rows && j < m.cols);
00351     return i * m.stride_r + j;
00352 }
00353 
00359 void mat2D_fill(Mat2D m, double x)
00360 {
00361     for (size_t i = 0; i < m.rows; ++i) {
00362         for (size_t j = 0; j < m.cols; ++j) {
00363             MAT2D_AT(m, i, j) = x;
00364         }
00365     }
00366 }
00367 
00375 void mat2D_fill_sequence(Mat2D m, double start, double step) {
00376     for (size_t i = 0; i < m.rows; i++) {
00377         for (size_t j = 0; j < m.cols; j++) {
00378             MAT2D_AT(m, i, j) = start + step * mat2D_offset2d(m, i, j);
00379         }
00380     }
00381 }
00382 
00388 void mat2D_fill_uint32(Mat2D_uint32 m, uint32_t x)
00389 {
00390     for (size_t i = 0; i < m.rows; ++i) {
00391         for (size_t j = 0; j < m.cols; ++j) {
00392             MAT2D_AT_UINT32(m, i, j) = x;
00393         }
00394     }
00395 }
00396 
00404 void mat2D_rand(Mat2D m, double low, double high)
00405 {
00406     for (size_t i = 0; i < m.rows; ++i) {
00407         for (size_t j = 0; j < m.cols; ++j) {
00408             MAT2D_AT(m, i, j) = mat2D_rand_double()*(high - low) + low;
00409         }
00410     }
00411 }
00412 
00421 void mat2D_dot(Mat2D dst, Mat2D a, Mat2D b)
00422 {
00423     MATRIX2D_ASSERT(a.cols == b.rows);
00424     MATRIX2D_ASSERT(a.rows == dst.rows);
00425     MATRIX2D_ASSERT(b.cols == dst.cols);
00426 
00427     size_t i, j, k;
00428 
```

```
00429     for (i = 0; i < dst.rows; i++) {
00430         for (j = 0; j < dst.cols; j++) {
00431             MAT2D_AT(dst, i, j) = 0;
00432             for (k = 0; k < a.cols; k++) {
00433                 MAT2D_AT(dst, i, j) += MAT2D_AT(a, i, k)*MAT2D_AT(b, k, j);
00434             }
00435         }
00436     }
00437
00438 }
00439
00447 double mat2D_dot_product(Mat2D a, Mat2D b)
00448 {
00449     MATRIX2D_ASSERT(a.rows == b.rows);
00450     MATRIX2D_ASSERT(a.cols == b.cols);
00451     MATRIX2D_ASSERT((1 == a.cols && 1 == b.cols) || (1 == a.rows && 1 == b.rows));
00452
00453     double dot_product = 0;
00454
00455     if (1 == a.cols) {
00456         for (size_t i = 0; i < a.rows; i++) {
00457             dot_product += MAT2D_AT(a, i, 0) * MAT2D_AT(b, i, 0);
00458         }
00459     } else {
00460         for (size_t j = 0; j < a.cols; j++) {
00461             dot_product += MAT2D_AT(a, 0, j) * MAT2D_AT(b, 0, j);
00462         }
00463     }
00464
00465     return dot_product;
00466
00467 }
00468
00476 void mat2D_cross(Mat2D dst, Mat2D a, Mat2D b)
00477 {
00478     MATRIX2D_ASSERT(3 == dst.rows && 1 == dst.cols);
00479     MATRIX2D_ASSERT(3 == a.rows && 1 == a.cols);
00480     MATRIX2D_ASSERT(3 == b.rows && 1 == b.cols);
00481
00482     MAT2D_AT(dst, 0, 0) = MAT2D_AT(a, 1, 0) * MAT2D_AT(b, 2, 0) - MAT2D_AT(a, 2, 0) * MAT2D_AT(b, 1,
      0);
00483     MAT2D_AT(dst, 1, 0) = MAT2D_AT(a, 2, 0) * MAT2D_AT(b, 0, 0) - MAT2D_AT(a, 0, 0) * MAT2D_AT(b, 2,
      0);
00484     MAT2D_AT(dst, 2, 0) = MAT2D_AT(a, 0, 0) * MAT2D_AT(b, 1, 0) - MAT2D_AT(a, 1, 0) * MAT2D_AT(b, 0,
      0);
00485 }
00486
00493 void mat2D_add(Mat2D dst, Mat2D a)
00494 {
00495     MATRIX2D_ASSERT(dst.rows == a.rows);
00496     MATRIX2D_ASSERT(dst.cols == a.cols);
00497     for (size_t i = 0; i < dst.rows; ++i) {
00498         for (size_t j = 0; j < dst.cols; ++j) {
00499             MAT2D_AT(dst, i, j) += MAT2D_AT(a, i, j);
00500         }
00501     }
00502 }
00503
00511 void mat2D_add_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor)
00512 {
00513     for (size_t j = 0; j < m.cols; ++j) {
00514         MAT2D_AT(m, des_r, j) += factor * MAT2D_AT(m, src_r, j);
00515     }
00516 }
00517
00524 void mat2D_sub(Mat2D dst, Mat2D a)
00525 {
00526     MATRIX2D_ASSERT(dst.rows == a.rows);
00527     MATRIX2D_ASSERT(dst.cols == a.cols);
00528     for (size_t i = 0; i < dst.rows; ++i) {
00529         for (size_t j = 0; j < dst.cols; ++j) {
00530             MAT2D_AT(dst, i, j) -= MAT2D_AT(a, i, j);
00531         }
00532     }
00533 }
00534
00542 void mat2D_sub_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor)
00543 {
00544     for (size_t j = 0; j < m.cols; ++j) {
00545         MAT2D_AT(m, des_r, j) -= factor * MAT2D_AT(m, src_r, j);
00546     }
00547 }
00548
00554 void mat2D_mult(Mat2D m, double factor)
00555 {
00556     for (size_t i = 0; i < m.rows; ++i) {
00557         for (size_t j = 0; j < m.cols; ++j) {
```

```
00558             MAT2D_AT(m, i, j) *= factor;
00559         }
00560     }
00561 }
00562
00569 void mat2D_mult_row(Mat2D m, size_t r, double factor)
00570 {
00571     for (size_t j = 0; j < m.cols; ++j) {
00572         MAT2D_AT(m, r, j) *= factor;
00573     }
00574 }
00575
00582 void mat2D_print(Mat2D m, const char *name, size_t padding)
00583 {
00584     printf("%*s%s = [\n", (int) padding, "", name);
00585     for (size_t i = 0; i < m.rows; ++i) {
00586         printf("%*s    ", (int) padding, "");
00587         for (size_t j = 0; j < m.cols; ++j) {
00588             printf("%9.6f ", MAT2D_AT(m, i, j));
00589         }
00590         printf("\n");
00591     }
00592     printf("%*s]\n", (int) padding, "");
00593 }
00594
00601 void mat2D_print_as_col(Mat2D m, const char *name, size_t padding)
00602 {
00603     printf("%*s%s = [\n", (int) padding, "", name);
00604     for (size_t i = 0; i < m.rows*m.cols; ++i) {
00605             printf("%*s    ", (int) padding, "");
00606             printf("%f\n", m.elements[i]);
00607     }
00608     printf("%*s]\n", (int) padding, "");
00609 }
00610
00616 void mat2D_set_identity(Mat2D m)
00617 {
00618     MATRIX2D_ASSERT(m.cols == m.rows);
00619     for (size_t i = 0; i < m.rows; ++i) {
00620         for (size_t j = 0; j < m.cols; ++j) {
00621             MAT2D_AT(m, i, j) = i == j ? 1 : 0;
00622             // if (i == j) {
00623             //     MAT2D_AT(m, i, j) = 1;
00624             // }
00625             // else {
00626             //     MAT2D_AT(m, i, j) = 0;
00627             // }
00628         }
00629     }
00630 }
00631
00640 double mat2D_make_identity(Mat2D m)
00641 {
00642     /* make identity matrix using Gauss elimination */
00643     /* preforming Gauss elimination: https://en.wikipedia.org/wiki/Gaussian_elimination */
00644     /* returns the factor multiplying the determinant */
00645
00646     double factor_to_return = 1;
00647
00648     for (size_t i = 0; i < (size_t)fmin(m.rows-1, m.cols); i++) {
00649         /* check if it is the biggest first number (absolute value) */
00650         size_t biggest_r = i;
00651         for (size_t index = i; index < m.rows; index++) {
00652             if (fabs(MAT2D_AT(m, index, index)) > fabs(MAT2D_AT(m, biggest_r, 0))) {
00653                 biggest_r = index;
00654             }
00655         }
00656         if (i != biggest_r) {
00657             mat2D_swap_rows(m, i, biggest_r);
00658             factor_to_return *= -1;
00659         }
00660         for (size_t j = i+1; j < m.cols; j++) {
00661             double factor = 1 / MAT2D_AT(m, i, i);
00662             mat2D_sub_row_time_factor_to_row(m, j, i, MAT2D_AT(m, j, i) * factor);
00663             mat2D_mult_row(m, i, factor);
00664             factor_to_return *= factor;
00665         }
00666     }
00667     double factor = 1 / MAT2D_AT(m, m.rows-1, m.cols-1);
00668     mat2D_mult_row(m, m.rows-1, factor);
00669     factor_to_return *= factor;
00670     for (size_t c = m.cols-1; c > 0; c--) {
00671         for (int r = c-1; r >= 0; r--) {
00672             double factor = 1 / MAT2D_AT(m, c, c);
00673             mat2D_sub_row_time_factor_to_row(m, r, c, MAT2D_AT(m, r, c) * factor);
00674         }
00675     }
```

```
00676
00677
00678     return factor_to_return;
00679 }
00680
00686 void mat2D_set_rot_mat_x(Mat2D m, float angle_deg)
00687 {
00688     MATRIX2D_ASSERT(3 == m.cols && 3 == m.rows);
00689
00690     float angle_rad = angle_deg * PI / 180;
00691     mat2D_set_identity(m);
00692     MAT2D_AT(m, 1, 1) =  cos(angle_rad);
00693     MAT2D_AT(m, 1, 2) =  sin(angle_rad);
00694     MAT2D_AT(m, 2, 1) = -sin(angle_rad);
00695     MAT2D_AT(m, 2, 2) =  cos(angle_rad);
00696 }
00697
00703 void mat2D_set_rot_mat_y(Mat2D m, float angle_deg)
00704 {
00705     MATRIX2D_ASSERT(3 == m.cols && 3 == m.rows);
00706
00707     float angle_rad = angle_deg * PI / 180;
00708     mat2D_set_identity(m);
00709     MAT2D_AT(m, 0, 0) =  cos(angle_rad);
00710     MAT2D_AT(m, 0, 2) = -sin(angle_rad);
00711     MAT2D_AT(m, 2, 0) =  sin(angle_rad);
00712     MAT2D_AT(m, 2, 2) =  cos(angle_rad);
00713 }
00714
00720 void mat2D_set_rot_mat_z(Mat2D m, float angle_deg)
00721 {
00722     MATRIX2D_ASSERT(3 == m.cols && 3 == m.rows);
00723
00724     float angle_rad = angle_deg * PI / 180;
00725     mat2D_set_identity(m);
00726     MAT2D_AT(m, 0, 0) =  cos(angle_rad);
00727     MAT2D_AT(m, 0, 1) =  sin(angle_rad);
00728     MAT2D_AT(m, 1, 0) = -sin(angle_rad);
00729     MAT2D_AT(m, 1, 1) =  cos(angle_rad);
00730 }
00731
00740 void mat2D_set_DCM_zyx(Mat2D DCM, float yaw_deg, float pitch_deg, float roll_deg)
00741 {
00742     Mat2D RotZ = mat2D_alloc(3,3);
00743     mat2D_set_rot_mat_z(RotZ, yaw_deg);
00744     Mat2D RotY = mat2D_alloc(3,3);
00745     mat2D_set_rot_mat_y(RotY, pitch_deg);
00746     Mat2D RotX = mat2D_alloc(3,3);
00747     mat2D_set_rot_mat_x(RotX, roll_deg);
00748     Mat2D temp = mat2D_alloc(3,3);
00749
00750     mat2D_dot(temp, RotY, RotZ);
00751     mat2D_dot(DCM, RotX, temp); /* I have a DCM */
00752
00753     mat2D_free(RotZ);
00754     mat2D_free(RotY);
00755     mat2D_free(RotX);
00756     mat2D_free(temp);
00757 }
00758
00765 void mat2D_copy(Mat2D des, Mat2D src)
00766 {
00767     MATRIX2D_ASSERT(des.cols == src.cols);
00768     MATRIX2D_ASSERT(des.rows == src.rows);
00769
00770     for (size_t i = 0; i < des.rows; ++i) {
00771         for (size_t j = 0; j < des.cols; ++j) {
00772             MAT2D_AT(des, i, j) = MAT2D_AT(src, i, j);
00773         }
00774     }
00775 }
00776
00787 void mat2D_copy_mat_to_mat_at_window(Mat2D des, Mat2D src, size_t is, size_t js, size_t ie, size_t je)
00788 {
00789     MATRIX2D_ASSERT(je > js && ie > is);
00790     MATRIX2D_ASSERT(je-js+1 == des.cols);
00791     MATRIX2D_ASSERT(ie-is+1 == des.rows);
00792
00793     for (size_t index = 0; index < des.rows; ++index) {
00794         for (size_t jndex = 0; jndex < des.cols; ++jndex) {
00795             MAT2D_AT(des, index, jndex) = MAT2D_AT(src, is+index, js+jndex);
00796         }
00797     }
00798 }
00799
00807 void mat2D_get_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
00808 {
```

```
00809     MATRIX2D_ASSERT(src_col < src.cols);
00810     MATRIX2D_ASSERT(des.rows == src.rows);
00811     MATRIX2D_ASSERT(des_col < des.cols);
00812 
00813     for (size_t i = 0; i < des.rows; i++) {
00814         MAT2D_AT(des, i, des_col) = MAT2D_AT(src, i, src_col);
00815     }
00816 }
00817 
00825 void mat2D_add_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
00826 {
00827     MATRIX2D_ASSERT(src_col < src.cols);
00828     MATRIX2D_ASSERT(des.rows == src.rows);
00829     MATRIX2D_ASSERT(des_col < des.cols);
00830 
00831     for (size_t i = 0; i < des.rows; i++) {
00832         MAT2D_AT(des, i, des_col) += MAT2D_AT(src, i, src_col);
00833     }
00834 }
00835 
00843 void mat2D_sub_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
00844 {
00845     MATRIX2D_ASSERT(src_col < src.cols);
00846     MATRIX2D_ASSERT(des.rows == src.rows);
00847     MATRIX2D_ASSERT(des_col < des.cols);
00848 
00849     for (size_t i = 0; i < des.rows; i++) {
00850         MAT2D_AT(des, i, des_col) -= MAT2D_AT(src, i, src_col);
00851     }
00852 }
00853 
00860 void mat2D_swap_rows(Mat2D m, size_t r1, size_t r2)
00861 {
00862     for (size_t j = 0; j < m.cols; j++) {
00863         double temp = MAT2D_AT(m, r1, j);
00864         MAT2D_AT(m, r1, j) = MAT2D_AT(m, r2, j);
00865         MAT2D_AT(m, r2, j) = temp;
00866     }
00867 }
00868 
00876 void mat2D_get_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
00877 {
00878     MATRIX2D_ASSERT(src_row < src.rows);
00879     MATRIX2D_ASSERT(des.cols == src.cols);
00880     MATRIX2D_ASSERT(des_row < des.rows);
00881 
00882     for (size_t j = 0; j < des.cols; j++) {
00883         MAT2D_AT(des, des_row, j) = MAT2D_AT(src, src_row, j);
00884     }
00885 }
00886 
00894 void mat2D_add_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
00895 {
00896     MATRIX2D_ASSERT(src_row < src.rows);
00897     MATRIX2D_ASSERT(des.cols == src.cols);
00898     MATRIX2D_ASSERT(des_row < des.rows);
00899 
00900     for (size_t j = 0; j < des.cols; j++) {
00901         MAT2D_AT(des, des_row, j) += MAT2D_AT(src, src_row, j);
00902     }
00903 }
00904 
00912 void mat2D_sub_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
00913 {
00914     MATRIX2D_ASSERT(src_row < src.rows);
00915     MATRIX2D_ASSERT(des.cols == src.cols);
00916     MATRIX2D_ASSERT(des_row < des.rows);
00917 
00918     for (size_t j = 0; j < des.cols; j++) {
00919         MAT2D_AT(des, des_row, j) -= MAT2D_AT(src, src_row, j);
00920     }
00921 }
00922 
00928 double mat2D_calc_norma(Mat2D m)
00929 {
00930     double sum = 0;
00931 
00932     for (size_t i = 0; i < m.rows; ++i) {
00933         for (size_t j = 0; j < m.cols; ++j) {
00934             sum += MAT2D_AT(m, i, j) * MAT2D_AT(m, i, j);
00935         }
00936     }
00937     return sqrt(sum);
00938 }
00939 
00946 bool mat2D_mat_is_all_digit(Mat2D m, double digit)
00947 {
```

```
00948     for (size_t i = 0; i < m.rows; ++i) {
00949         for (size_t j = 0; j < m.cols; ++j) {
00950             if (MAT2D_AT(m, i, j) != digit) {
00951                 return false;
00952             }
00953         }
00954     }
00955     return true;
00956 }
00957
00965 bool mat2D_row_is_all_digit(Mat2D m, double digit, size_t r)
00966 {
00967     for (size_t j = 0; j < m.cols; ++j) {
00968         if (MAT2D_AT(m, r, j) != digit) {
00969             return false;
00970         }
00971     }
00972     return true;
00973 }
00974
00982 bool mat2D_col_is_all_digit(Mat2D m, double digit, size_t c)
00983 {
00984     for (size_t i = 0; i < m.cols; ++i) {
00985         if (MAT2D_AT(m, i, c) != digit) {
00986             return false;
00987         }
00988     }
00989     return true;
00990 }
00991
00997 double mat2D_det_2x2_mat(Mat2D m)
00998 {
00999     MATRIX2D_ASSERT(2 == m.cols && 2 == m.rows && "Not a 2x2 matrix");
01000     return MAT2D_AT(m, 0, 0) * MAT2D_AT(m, 1, 1) - MAT2D_AT(m, 0, 1) * MAT2D_AT(m, 1, 0);
01001 }
01002
01010 double mat2D_triangulate(Mat2D m)
01011 {
01012     /* preforming Gauss elimination: https://en.wikipedia.org/wiki/Gaussian_elimination */
01013     /* returns the factor multiplying the determinant */
01014
01015     double factor_to_return = 1;
01016
01017     for (size_t i = 0; i < (size_t)fmin(m.rows-1, m.cols); i++) {
01018         if (!MAT2D_AT(m, i, i)) {   /* swapping only if it is zero */
01019             /* finding biggest first number (absolute value) */
01020             size_t biggest_r = i;
01021             for (size_t index = i; index < m.rows; index++) {
01022                 if (fabs(MAT2D_AT(m, index, i)) > fabs(MAT2D_AT(m, biggest_r, i))) {
01023                     biggest_r = index;
01024                 }
01025             }
01026             if (i != biggest_r) {
01027                 mat2D_swap_rows(m, i, biggest_r);
01028             }
01029         }
01030         for (size_t j = i+1; j < m.cols; j++) {
01031             double factor = 1 / MAT2D_AT(m, i, i);
01032             if (!isfinite(factor)) {
01033                 printf("%s:%d: [Error] unable to transfrom into uperr triangular matrix. Probably some
      of the rows are not independent.\n", __FILE__, __LINE__);
01034             }
01035             double mat_value = MAT2D_AT(m, j, i);
01036             mat2D_sub_row_time_factor_to_row(m, j, i, mat_value * factor);
01037         }
01038     }
01039     return factor_to_return;
01040 }
01041
01049 double mat2D_det(Mat2D m)
01050 {
01051     MATRIX2D_ASSERT(m.cols == m.rows && "should be a square matrix");
01052
01053     /* checking if there is a row or column with all zeros */
01054     /* checking rows */
01055     for (size_t i = 0; i < m.rows; i++) {
01056         if (mat2D_row_is_all_digit(m, 0, i)) {
01057             return 0;
01058         }
01059     }
01060     /* checking cols */
01061     for (size_t j = 0; j < m.rows; j++) {
01062         if (mat2D_col_is_all_digit(m, 0, j)) {
01063             return 0;
01064         }
01065     }
01066
```

```
01067     /* This is an implementation of naive determinant calculation using minors. This is too slow */
01068
01069     // double det = 0;
01070     // /* TODO: finding beast row or col? */
01071     // for (size_t i = 0, j = 0; i < m.rows; i++) { /* first column */
01072     //     if (MAT2D_AT(m, i, j) < 1e-10) continue;
01073     //     Mat2D_Minor sub_mm = mat2D_minor_alloc_fill_from_mat(m, i, j);
01074     //     int factor = (i+j)%2 ? -1 : 1;
01075     //     if (sub_mm.cols != 2) {
01076     //         MATRIX2D_ASSERT(sub_mm.cols == sub_mm.rows && "should be a square matrix");
01077     //         det += MAT2D_AT(m, i, j) * (factor) * mat2D_minor_det(sub_mm);
01078     //     } else if (sub_mm.cols == 2 && sub_mm.rows == 2) {
01079     //         det += MAT2D_AT(m, i, j) * (factor) * mat2D_det_2x2_mat_minor(sub_mm);;
01080     //     }
01081     //     mat2D_minor_free(sub_mm);
01082     // }
01083
01084     Mat2D temp_m = mat2D_alloc(m.rows, m.cols);
01085     mat2D_copy(temp_m, m);
01086     double factor = mat2D_triangulate(temp_m);
01087     double diag_mul = 1;
01088     for (size_t i = 0; i < temp_m.rows; i++) {
01089         diag_mul *= MAT2D_AT(temp_m, i, i);
01090     }
01091     mat2D_free(temp_m);
01092
01093     return diag_mul / factor;
01094 }
01095
01104 void mat2D_LUP_decomposition_with_swap(Mat2D src, Mat2D l, Mat2D p, Mat2D u)
01105 {
01106     /* performing LU decomposition Following the Wikipedia page:
       https://en.wikipedia.org/wiki/LU_decomposition */
01107
01108     mat2D_copy(u, src);
01109     mat2D_set_identity(p);
01110     mat2D_fill(l, 0);
01111
01112     for (size_t i = 0; i < (size_t)fmin(u.rows-1, u.cols); i++) {
01113         if (!MAT2D_AT(u, i, i)) {   /* swapping only if it is zero */
01114             /* finding biggest first number (absolute value) */
01115             size_t biggest_r = i;
01116             for (size_t index = i; index < u.rows; index++) {
01117                 if (fabs(MAT2D_AT(u, index, i)) > fabs(MAT2D_AT(u, biggest_r, i))) {
01118                     biggest_r = index;
01119                 }
01120             }
01121             if (i != biggest_r) {
01122                 mat2D_swap_rows(u, i, biggest_r);
01123                 mat2D_swap_rows(p, i, biggest_r);
01124                 mat2D_swap_rows(l, i, biggest_r);
01125             }
01126         }
01127         for (size_t j = i+1; j < u.cols; j++) {
01128             double factor = 1 / MAT2D_AT(u, i, i);
01129             if (!isfinite(factor)) {
01130                 printf("%s:%d: [Error] unable to transfrom into uper triangular matrix. Probably some
       of the rows are not independent.\n", __FILE__, __LINE__);
01131             }
01132             double mat_value = MAT2D_AT(u, j, i);
01133             mat2D_sub_row_time_factor_to_row(u, j, i, mat_value * factor);
01134             MAT2D_AT(l, j, i) = mat_value * factor;
01135         }
01136         MAT2D_AT(l, i, i) = 1;
01137     }
01138     MAT2D_AT(l, l.rows-1, l.cols-1) = 1;
01139 }
01140
01146 void mat2D_transpose(Mat2D des, Mat2D src)
01147 {
01148     MATRIX2D_ASSERT(des.cols == src.rows);
01149     MATRIX2D_ASSERT(des.rows == src.cols);
01150
01151     for (size_t index = 0; index < des.rows; ++index) {
01152         for (size_t jndex = 0; jndex < des.cols; ++jndex) {
01153             MAT2D_AT(des, index, jndex) = MAT2D_AT(src, jndex, index);
01154         }
01155     }
01156 }
01157
01166 void mat2D_invert(Mat2D des, Mat2D src)
01167 {
01168     MATRIX2D_ASSERT(src.cols == src.rows && "should be an NxN matrix");
01169     MATRIX2D_ASSERT(des.cols == src.cols && des.rows == des.cols);
01170
01171     Mat2D m = mat2D_alloc(src.rows, src.cols);
01172     mat2D_copy(m, src);
```

```
01173
01174     mat2D_set_identity(des);
01175
01176     if (!mat2D_det(m)) {
01177         mat2D_fill(des, 0);
01178         printf("%s:%d: [Error] Can't invert the matrix. Determinant is zero! Set the inverse matrix to
      all zeros\n", __FILE__, __LINE__);
01179         return;
01180     }
01181
01182     for (size_t i = 0; i < (size_t)fmin(m.rows-1, m.cols); i++) {
01183         if (!MAT2D_AT(m, i, i)) {   /* swapping only if it is zero */
01184             /* finding biggest first number (absolute value) */
01185             size_t biggest_r = i;
01186             for (size_t index = i; index < m.rows; index++) {
01187                 if (fabs(MAT2D_AT(m, index, i)) > fabs(MAT2D_AT(m, biggest_r, i))) {
01188                     biggest_r = index;
01189                 }
01190             }
01191             if (i != biggest_r) {
01192                 mat2D_swap_rows(m, i, biggest_r);
01193                 mat2D_swap_rows(des, i, biggest_r);
01194                 printf("%s:%d: [INFO] swapping row %zu with row %zu.\n", __FILE__, __LINE__, i,
      biggest_r);
01195             } else {
01196                 MATRIX2D_ASSERT(0 && "can't inverse");
01197             }
01198         }
01199         for (size_t j = i+1; j < m.cols; j++) {
01200             double factor = 1 / MAT2D_AT(m, i, i);
01201             double mat_value = MAT2D_AT(m, j, i);
01202             mat2D_sub_row_time_factor_to_row(m, j, i, mat_value * factor);
01203             mat2D_mult_row(m, i, factor);
01204
01205             mat2D_sub_row_time_factor_to_row(des, j, i, mat_value * factor);
01206             mat2D_mult_row(des, i, factor);
01207         }
01208     }
01209     double factor = 1 / MAT2D_AT(m, m.rows-1, m.cols-1);
01210     mat2D_mult_row(m, m.rows-1, factor);
01211     mat2D_mult_row(des, des.rows-1, factor);
01212     for (size_t c = m.cols-1; c > 0; c--) {
01213         for (int r = c-1; r >= 0; r--) {
01214             double factor = 1 / MAT2D_AT(m, c, c);
01215             double mat_value = MAT2D_AT(m, r, c);
01216             mat2D_sub_row_time_factor_to_row(m, r, c, mat_value * factor);
01217             mat2D_sub_row_time_factor_to_row(des, r, c, mat_value * factor);
01218         }
01219     }
01220
01221     mat2D_free(m);
01222 }
01223
01233 void mat2D_solve_linear_sys_LUP_decomposition(Mat2D A, Mat2D x, Mat2D B)
01234 {
01235     MATRIX2D_ASSERT(A.cols == x.rows);
01236     MATRIX2D_ASSERT(1 == x.cols);
01237     MATRIX2D_ASSERT(A.rows == B.rows);
01238     MATRIX2D_ASSERT(1 == B.cols);
01239
01240     Mat2D y     = mat2D_alloc(x.rows, x.cols);
01241     Mat2D l     = mat2D_alloc(A.rows, A.cols);
01242     Mat2D p     = mat2D_alloc(A.rows, A.cols);
01243     Mat2D u     = mat2D_alloc(A.rows, A.cols);
01244     Mat2D inv_l = mat2D_alloc(l.rows, l.cols);
01245     Mat2D inv_u = mat2D_alloc(u.rows, u.cols);
01246
01247     mat2D_LUP_decomposition_with_swap(A, l, p, u);
01248
01249     mat2D_invert(inv_l, l);
01250     mat2D_invert(inv_u, u);
01251
01252     mat2D_fill(x, 0);   /* x here is only a temp mat*/
01253     mat2D_fill(y, 0);
01254     mat2D_dot(x, p, B);
01255     mat2D_dot(y, inv_l, x);
01256
01257     mat2D_fill(x, 0);
01258     mat2D_dot(x, inv_u, y);
01259
01260     mat2D_free(y);
01261     mat2D_free(l);
01262     mat2D_free(p);
01263     mat2D_free(u);
01264     mat2D_free(inv_l);
01265     mat2D_free(inv_u);
01266 }
```

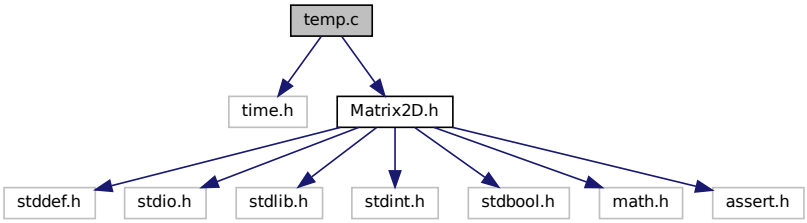
```
01267
01276 Mat2D_Minor mat2D_minor_alloc_fill_from_mat(Mat2D ref_mat, size_t i, size_t j)
01277 {
01278     MATRIX2D_ASSERT(ref_mat.cols == ref_mat.rows && "minor is defined only for square matrix");
01279
01280     Mat2D_Minor mm;
01281     mm.cols = ref_mat.cols-1;
01282     mm.rows = ref_mat.rows-1;
01283     mm.stride_r = ref_mat.cols-1;
01284     mm.cols_list = (size_t*)MATRIX2D_MALLOC(sizeof(double)*(ref_mat.cols-1));
01285     mm.rows_list = (size_t*)MATRIX2D_MALLOC(sizeof(double)*(ref_mat.rows-1));
01286     mm.ref_mat = ref_mat;
01287
01288     MATRIX2D_ASSERT(mm.cols_list != NULL && mm.rows_list != NULL);
01289
01290     for (size_t index = 0, temp_index = 0; index < ref_mat.rows; index++) {
01291         if (index != i) {
01292             mm.rows_list[temp_index] = index;
01293             temp_index++;
01294         }
01295     }
01296     for (size_t jndex = 0, temp_jndex = 0; jndex < ref_mat.rows; jndex++) {
01297         if (jndex != j) {
01298             mm.cols_list[temp_jndex] = jndex;
01299             temp_jndex++;
01300         }
01301     }
01302
01303     return mm;
01304 }
01305
01315 Mat2D_Minor mat2D_minor_alloc_fill_from_mat_minor(Mat2D_Minor ref_mm, size_t i, size_t j)
01316 {
01317     MATRIX2D_ASSERT(ref_mm.cols == ref_mm.rows && "minor is defined only for square matrix");
01318
01319     Mat2D_Minor mm;
01320     mm.cols = ref_mm.cols-1;
01321     mm.rows = ref_mm.rows-1;
01322     mm.stride_r = ref_mm.cols-1;
01323     mm.cols_list = (size_t*)MATRIX2D_MALLOC(sizeof(double)*(ref_mm.cols-1));
01324     mm.rows_list = (size_t*)MATRIX2D_MALLOC(sizeof(double)*(ref_mm.rows-1));
01325     mm.ref_mat = ref_mm.ref_mat;
01326
01327     MATRIX2D_ASSERT(mm.cols_list != NULL && mm.rows_list != NULL);
01328
01329     for (size_t index = 0, temp_index = 0; index < ref_mm.rows; index++) {
01330         if (index != i) {
01331             mm.rows_list[temp_index] = ref_mm.rows_list[index];
01332             temp_index++;
01333         }
01334     }
01335     for (size_t jndex = 0, temp_jndex = 0; jndex < ref_mm.rows; jndex++) {
01336         if (jndex != j) {
01337             mm.cols_list[temp_jndex] = ref_mm.cols_list[jndex];
01338             temp_jndex++;
01339         }
01340     }
01341
01342     return mm;
01343 }
01344
01350 void mat2D_minor_free(Mat2D_Minor mm)
01351 {
01352     free(mm.cols_list);
01353     free(mm.rows_list);
01354 }
01355
01362 void mat2D_minor_print(Mat2D_Minor mm, const char *name, size_t padding)
01363 {
01364     printf("%*s%s = [\n", (int) padding, "", name);
01365     for (size_t i = 0; i < mm.rows; ++i) {
01366         printf("%*s    ", (int) padding, "");
01367         for (size_t j = 0; j < mm.cols; ++j) {
01368             printf("%f ", MAT2D_MINOR_AT(mm, i, j));
01369         }
01370         printf("\n");
01371     }
01372     printf("%*s]\n", (int) padding, "");
01373 }
01374
01380 double mat2D_det_2x2_mat_minor(Mat2D_Minor mm)
01381 {
01382     MATRIX2D_ASSERT(2 == mm.cols && 2 == mm.rows && "Not a 2x2 matrix");
01383     return MAT2D_MINOR_AT(mm, 0, 0) * MAT2D_MINOR_AT(mm, 1, 1) - MAT2D_MINOR_AT(mm, 0, 1) *
       MAT2D_MINOR_AT(mm, 1, 0);
01384 }
01385
```

```
01393 double mat2D_minor_det(Mat2D_Minor mm)
01394 {
01395     MATRIX2D_ASSERT(mm.cols == mm.rows && "should be a square matrix");
01396 
01397     double det = 0;
01398     /* TODO: finding beast row or col? */
01399     for (size_t i = 0, j = 0; i < mm.rows; i++) { /* first column */
01400         if (MAT2D_MINOR_AT(mm, i, j) < 1e-10) continue;
01401         Mat2D_Minor sub_mm = mat2D_minor_alloc_fill_from_mat_minor(mm, i, j);
01402         int factor = (i+j)%2 ? -1 : 1;
01403         if (sub_mm.cols != 2) {
01404             MATRIX2D_ASSERT(sub_mm.cols == sub_mm.rows && "should be a square matrix");
01405             det += MAT2D_MINOR_AT(mm, i, j) * (factor) * mat2D_minor_det(sub_mm);
01406         } else if (sub_mm.cols == 2 && sub_mm.rows == 2) {
01407             det += MAT2D_MINOR_AT(mm, i, j) * (factor) * mat2D_det_2x2_mat_minor(sub_mm);;
01408         }
01409         mat2D_minor_free(sub_mm);
01410     }
01411     return det;
01412 }
01413 
01414 
01415 #endif // MATRIX2D_IMPLEMENTATION
```

4.3 temp.c File Reference

```
#include "time.h"
#include "Matrix2D.h"
```

Include dependency graph for temp.c:



Macros

- #define MATRIX2D_IMPLEMENTATION

Functions

- int main (void)

4.3.1 Macro Definition Documentation

4.3.1.1 MATRIX2D_IMPLEMENTATION

```
#define MATRIX2D_IMPLEMENTATION
```

Definition at line 2 of file temp.c.

4.3.2 Function Documentation

4.3.2.1 main()

```
int main (
            void  )
```

Definition at line 5 of file temp.c.

References mat2D_alloc(), MAT2D_AT, mat2D_calc_norma(), mat2D_copy(), mat2D_dot(), mat2D_fill(), mat2D_fill_sequence(), mat2D_LUP_decomposition_with_swap(), mat2D_mult(), MAT2D_PRINT, mat2D_set_identity(), mat2D_solve_linear_sys_LUP_decomposition(), and mat2D_sub().

4.4 temp.c

```
00001 #include "time.h"
00002 #define MATRIX2D_IMPLEMENTATION
00003 #include "Matrix2D.h"
00004
00005 int main(void)
00006 {
00007     int n = 3;
00008     Mat2D a     = mat2D_alloc(n, n);
00009     Mat2D l     = mat2D_alloc(n, n);
00010     Mat2D p     = mat2D_alloc(n, n);
00011     Mat2D u     = mat2D_alloc(n, n);
00012     Mat2D current_A  = mat2D_alloc(n, n);
00013     Mat2D previous_A = mat2D_alloc(n, n);
00014     Mat2D diff       = mat2D_alloc(n, n);
00015     Mat2D x          = mat2D_alloc(n, 1);
00016     Mat2D B          = mat2D_alloc(n, 1);
00017
00018     // srand(time(0));
00019     // mat2D_rand(a, 0, 1);
00020     mat2D_fill_sequence(a, 1, 1);
00021     MAT2D_PRINT(a);
00022
00023     mat2D_LUP_decomposition_with_swap(a, l, p, u);
00024
00025     MAT2D_PRINT(l);
00026     MAT2D_PRINT(p);
00027     MAT2D_PRINT(u);
00028
00029     mat2D_dot(current_A, l, u);
00030     MAT2D_PRINT(current_A);
00031
00032     for (int i = 0; i < 25; i++) {
00033         mat2D_dot(current_A, l, u);
00034         mat2D_dot(previous_A, u, l);
00035         mat2D_LUP_decomposition_with_swap(previous_A, l, p, u);
00036         mat2D_copy(diff, current_A);
00037         mat2D_sub(diff, previous_A);
00038     }
00039     MAT2D_PRINT(diff);
00040     mat2D_copy(current_A, previous_A);
00041
00042     mat2D_set_identity(previous_A);
00043     mat2D_mult(previous_A, MAT2D_AT(current_A, 1, 1));
00044     mat2D_sub(a, previous_A);
00045
00046     mat2D_fill(B, 0);
00047     mat2D_solve_linear_sys_LUP_decomposition(a, x, B);
00048
00049     MAT2D_PRINT(a);
00050     MAT2D_PRINT(x);
00051     printf("\n%g\n", mat2D_calc_norma(diff));
00052
00053     return 0;
00054 }
```

Index