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 double.

`#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 double.

- 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 a contiguous buffer of rows * cols doubles.

Note

This type is a shallow handle; copying Mat2D copies the pointer, not the underlying data.

Definition at line 117 of file Matrix2D.h.

3.1.2 Member Data Documentation

3.1.2.1 cols

```
size_t Mat2D::cols
```

Definition at line 119 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_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_upper_triangulate().

3.1.2.2 elements

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

Definition at line 121 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 118 of file Matrix2D.h.

Referenced by mat2D_add(), mat2D_add_col_to_col(), 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_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_upper_triangulate().

3.1.2.4 stride_r

```
size_t Mat2D::stride_r
```

Definition at line 120 of file Matrix2D.h.

Referenced by mat2D_alloc(), and mat2D_offset2d().

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

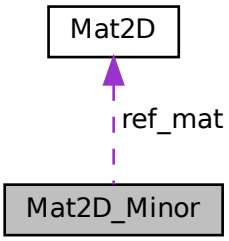
- Matrix2D.h

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. The minor does not own the reference matrix data; instead it stores two index arrays (rows_list, cols_list) mapping minor coordinates to the reference matrix coordinates.

Memory ownership:

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

Definition at line 152 of file Matrix2D.h.

3.2.2 Member Data Documentation

3.2.2.1 cols

```
size_t Mat2D_Minor::cols
```

Definition at line 154 of file Matrix2D.h.

Referenced by mat2D_det(), mat2D_det_2x2_mat_minor(), mat2D_minor_alloc_fill_from_mat(), mat2D_minor_alloc_fill_from_mat_m
mat2D_minor_det(), and mat2D_minor_print().

3.2.2.2 cols_list

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

Definition at line 157 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 158 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 153 of file Matrix2D.h.

Referenced by mat2D_det(), mat2D_det_2x2_mat_minor(), mat2D_minor_alloc_fill_from_mat(), mat2D_minor_alloc_fill_from_mat_m
mat2D_minor_det(), and mat2D_minor_print().

3.2.2.5 rows_list

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

Definition at line 156 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 155 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.

Same layout rules as Mat2D, but with uint32_t elements.

Definition at line 130 of file Matrix2D.h.

3.3.2 Member Data Documentation

3.3.2.1 cols

```
size_t Mat2D_uint32::cols
```

Definition at line 132 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 134 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 131 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 133 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

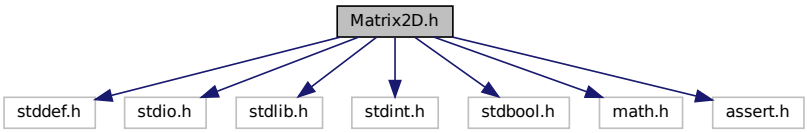
File Documentation

4.1 Matrix2D.h File Reference

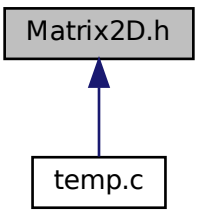
Lightweight 2D matrix helpers (double / uint32_t).

```
#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 double.
- struct Mat2D_uint32

  Dense row-major matrix of uint32_t.
- struct Mat2D_Minor

  A minor "view" into a reference matrix.

Macros

- #define MAT2D_MALLOC malloc

  Allocation function used by this library.
- #define MAT2D_FREE free

  Deallocation function used by this library.
- #define MAT2D_ASSERT assert

  Assertion macro used by this library for parameter validation.
- #define MAT2D_AT(m, i, j) (m).elements[mat2D_offset2d((m), (i), (j))]

  Access element (i, j) of a Mat2D (0-based).
- #define MAT2D_AT_UINT32(m, i, j) (m).elements[mat2D_offset2d_uint32((m), (i), (j))]

  Access element (i, j) of a Mat2D_uint32 (0-based).
- #define MAT2D_PI 3.14159265358979323846
- #define MAT2D_EPS 1e-15
- #define MAT2D_IS_ZERO(x) (fabs(x) $<$ MAT2D_EPS)

  Test whether a floating-point value is “near zero”.
- #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).
- #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)))

  Normalize a matrix in-place to unit Frobenius norm.

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-by-cols matrix of double.
- Mat2D_uint32 mat2D_alloc_uint32 (size_t rows, size_t cols)

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

  Free the buffer owned by a Mat2D.
- void mat2D_free_uint32 (Mat2D_uint32 m)

  Free the buffer owned by a Mat2D_uint32.
- 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 pseudo-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 using Gauss-Jordan style elimination.
- 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.

- 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.
- 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)
- 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.
- 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_upper_triangulate (Mat2D m)

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

  Determinant of a square 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 an LUP-based approach.
- 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

Lightweight 2D matrix helpers (double / uint32_t).

This single-header module provides small utilities for dense row-major matrices:

- Allocation/free for Mat2D (double) and Mat2D_uint32
- Basic arithmetic and row/column operations
- Matrix multiplication, transpose, dot and cross products
- Determinant and inversion (Gaussian / Gauss-Jordan style)
- A simple LUP decomposition helper and a linear system solver
- Rotation matrix helpers (X/Y/Z) and a Z-Y-X DCM builder (as implemented)
- “Minor” views (index lists into a reference matrix) for educational determinant-by-minors computation

Storage model

- Matrices are dense and row-major (C-style).
- Element at row i and column j (0-based) is: elements[i ∗ stride_r + j]
- For matrices created by mat2D_alloc(), stride_r == cols.

Usage

- In exactly one translation unit, define MATRIX2D_IMPLEMENTATION before including this header to compile the implementation.
- In all other files, include the header without that macro to get declarations only.

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

Notes and limitations

- 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/L1TbWe8b↩VOc?list=PLpM-Dvs8t0VZPZKggcql-MmjaBdZKeDMw
- All APIs assume the caller provides correctly-sized destination matrices. Shape mismatches are checked with MAT2D_ASSERT in many routines.
- This library does not try to be numerically robust:
  - Pivoting is limited (only performed when a pivot is “near zero” per MAT2D_EPS in several routines).
  - Ill-conditioned matrices may produce inaccurate determinants/inverses.
- RNG uses C rand(); it is not cryptographically secure.

Warning

Numerical stability and correctness

- mat2D_minor_det() is factorial-time and is intended only for very small matrices (educational use).
- mat2D_invert() uses Gauss-Jordan elimination and may be unstable for ill-conditioned matrices. Consider a more robust decomposition for production use (full pivoting / QR / SVD).
- Several routines do not guard against aliasing (e.g. dst == a). Unless documented otherwise, assume inputs and outputs must not overlap.

Definition in file Matrix2D.h.

4.1.2 Macro Definition Documentation

4.1.2.1 MAT2D_ASSERT

```
#define MAT2D_ASSERT assert
```

Assertion macro used by this library for parameter validation.

Defaults to assert(). Override by defining MAT2D_ASSERT before including this header to customize validation behavior.

Definition at line 101 of file Matrix2D.h.

4.1.2.2 MAT2D_AT

```
#define MAT2D_AT(
              m,
              i,
              j ) (m).elements[mat2D_offset2d((m), (i), (j))]
```

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

Expands to row-major indexing using stride_r: (m).elements[(i) ∗ (m).stride_r + (j)]

Warning

In the “fast” configuration this macro performs no bounds checking.

Definition at line 179 of file Matrix2D.h.

4.1.2.3 MAT2D_AT_UINT32

```
#define MAT2D_AT_UINT32(
              m,
              i,
              j ) (m).elements[mat2D_offset2d_uint32((m), (i), (j))]
```

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

Warning

In the “fast” configuration this macro performs no bounds checking.

Definition at line 180 of file Matrix2D.h.

4.1.2.4 MAT2D_EPS

```
#define MAT2D_EPS 1e-15
```

Definition at line 191 of file Matrix2D.h.

4.1.2.5 MAT2D_FREE

```
#define MAT2D_FREE free
```

Deallocation function used by this library.

Defaults to free(). Override by defining MAT2D_FREE before including this header to match a custom allocator.

Definition at line 88 of file Matrix2D.h.

4.1.2.6 MAT2D_IS_ZERO

```
#define MAT2D_IS_ZERO(
              x ) (fabs(x) < MAT2D_EPS)
```

Test whether a floating-point value is “near zero”.

Uses fabs(x) $<$ MAT2D_EPS.

Definition at line 201 of file Matrix2D.h.

4.1.2.7 MAT2D_MALLOC

```
#define MAT2D_MALLOC malloc
```

Allocation function used by this library.

Defaults to malloc(). Override by defining MAT2D_MALLOC before including this header to use a custom allocator.

Definition at line 76 of file Matrix2D.h.

4.1.2.8 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).

Dereferences into the underlying reference matrix using rows_list/cols_list.

Definition at line 210 of file Matrix2D.h.

4.1.2.9 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 228 of file Matrix2D.h.

4.1.2.10 mat2D_normalize

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

Normalize a matrix in-place to unit Frobenius norm.

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

Warning

If the Frobenius norm is 0, this performs a division by zero.

Definition at line 240 of file Matrix2D.h.

4.1.2.11 MAT2D_PI

```
#define MAT2D_PI 3.14159265358979323846
```

Definition at line 187 of file Matrix2D.h.

4.1.2.12 MAT2D_PRINT

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

Convenience macro to print a matrix with its variable name.

Definition at line 216 of file Matrix2D.h.

4.1.2.13 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 222 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

dst and a have identical shape.

Definition at line 591 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, 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.

Performs: 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 953 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, 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.
des_r	Destination row index.
src_r	Source row index.
factor	Scalar multiplier.

Warning

Indices are not bounds-checked in this routine.

Definition at line 611 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 )
```

Parameters

src_row	Row index in source.

Precondition

des.cols == src.cols

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 1031 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, and Mat2D::rows.

4.1.3.5 mat2D_alloc()

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

Allocate a rows-by-cols matrix of double.

Parameters

rows	Number of rows. Must be > 0.
cols	Number of columns. Must be > 0.

Returns

A Mat2D owning a contiguous buffer of rows $*$ cols elements.

Postcondition

The returned matrix has stride_r == cols.

The returned matrix must be released with mat2D_free().

Warning

This function asserts allocation success via MAT2D_ASSERT.

Definition at line 344 of file Matrix2D.h.

References Mat2D::cols, Mat2D::elements, MAT2D_ASSERT, MAT2D_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-by-cols matrix of uint32_t.

Parameters

rows	Number of rows. Must be > 0.
cols	Number of columns. Must be > 0.

Returns

A Mat2D_uint32 owning a contiguous buffer of rows $*$ cols elements.

Postcondition

The returned matrix has stride_r == cols.

The returned matrix must be released with mat2D_free_uint32().

Warning

This function asserts allocation success via MAT2D_ASSERT.

Definition at line 368 of file Matrix2D.h.

References Mat2D_uint32::cols, Mat2D_uint32::elements, MAT2D_ASSERT, MAT2D_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 1068 of file Matrix2D.h.

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

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.

Warning

Uses exact floating-point equality.

Definition at line 1128 of file Matrix2D.h.

References MAT2D_AT, and Mat2D::rows.

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.

des and src have identical shape.

Definition at line 887 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, and Mat2D::rows.

Referenced by 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 909 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, 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 572 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, and Mat2D::rows.

4.1.3.12 mat2D_det()

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

Determinant of a square matrix via Gaussian elimination.

Parameters

m	Square matrix.

Returns

det(m).

Copies `m` internally, transforms the copy to upper triangular form, and returns the product of diagonal elements adjusted by the row-swap factor.

Warning

The early “all-zero row/column” check uses exact comparisons to 0.

Limited pivoting may cause poor numerical results for some inputs.

Definition at line 1215 of file Matrix2D.h.

References Mat2D::cols, Mat2D_Minor::cols, mat2D_alloc(), MAT2D_ASSERT, MAT2D_AT, mat2D_col_is_all_digit(), mat2D_copy(), mat2D_det_2x2_mat_minor(), mat2D_free(), mat2D_minor_alloc_fill_from_mat(), mat2D_minor_det(), mat2D_minor_free(), mat2D_row_is_all_digit(), mat2D_upper_triangulate(), Mat2D::rows, and Mat2D_Minor::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) = m00∗m11 - m01∗m10.

Definition at line 1143 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, 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 1582 of file Matrix2D.h.

References Mat2D_Minor::cols, MAT2D_ASSERT, MAT2D_MINOR_AT, and Mat2D_Minor::rows.

Referenced by mat2D_det(), and 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 fully overwritten.

Warning

dst must not alias a or b (overlap is not handled).

Definition at line 515 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, and Mat2D::rows.

Referenced by 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 and a.cols == b.cols

(a.cols == 1 && b.cols == 1) || (a.rows == 1 && b.rows == 1)

Definition at line 544 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, 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 443 of file Matrix2D.h.

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

Referenced by 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 459 of file Matrix2D.h.

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

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 472 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 buffer owned by a Mat2D.

Parameters

m	Matrix whose elements were allocated via MAT2D_MALLOC.

Note

This does not modify m (it is passed by value).

It is safe to call with m.elements == NULL.

Definition at line 388 of file Matrix2D.h.

References Mat2D::elements, and MAT2D_FREE.

Referenced by main(), 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 buffer owned by a Mat2D_uint32.

Parameters

m	Matrix whose elements were allocated via MAT2D_MALLOC.

Note

This does not modify m (it is passed by value).

It is safe to call with m.elements == NULL.

Definition at line 401 of file Matrix2D.h.

References Mat2D_uint32::elements, and MAT2D_FREE.

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.

Precondition

des.rows == src.rows

des_col $<$ des.cols and src_col $<$ src.cols

Definition at line 932 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, 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.

Precondition

des.cols == src.cols

Definition at line 1011 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, 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.

des is allocated as the same shape as src.

On singular matrices (det == 0), prints an error message, writes all zeros into `des`, and returns.

Warning

This implementation performs limited pivoting (only when a pivot is “near zero” per MAT2D_EPS). It may be unstable for ill-conditioned matrices.

This routine computes det(src) first, which performs an additional elimination pass and can amplify numerical issues.

Definition at line 1353 of file Matrix2D.h.

References Mat2D::cols, mat2D_alloc(), MAT2D_ASSERT, MAT2D_AT, mat2D_copy(), mat2D_det(), mat2D_fill(), mat2D_free(), MAT2D_IS_ZERO, mat2D_mult_row(), mat2D_set_identity(), mat2D_sub_row_time_factor_to_row(), mat2D_swap_rows(), 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 by this function).
l	Output lower-triangular-like matrix (intended to have unit diagonal).
p	Output permutation matrix.
u	Output upper-triangular-like matrix.

Precondition

src is square.

l, p, u are allocated with the same shape as src.

Warning

Pivoting is limited: a row swap is performed only when the pivot is “near zero” (MAT2D_IS_ZERO()).

This routine swaps rows of L during decomposition; for a standard LUP implementation, care is required when swapping partially-built L.

Definition at line 1278 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_copy(), mat2D_fill(), MAT2D_IS_ZERO, mat2D_set_identity(), mat2D_sub_row_time_factor_to_row(), mat2D_swap_rows(), and Mat2D::rows.

Referenced by mat2D_solve_linear_sys_LUP_decomposition().

4.1.3.26 mat2D_make_identity()

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

Reduce a matrix to identity using Gauss-Jordan style elimination.

Parameters

m	Matrix modified in-place.

Returns

A multiplicative factor that tracks the effect of row swaps and row scalings performed inside this routine (useful when relating the transformation to determinants).

Internally calls mat2D_upper_triangulate() and then performs backward elimination and row scaling to reach identity (if the matrix is nonsingular and pivots are usable).

Warning

No full pivoting is performed. Ill-conditioned matrices may produce poor results or floating-point exceptions.

Definition at line 754 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, mat2D_mult_row(), mat2D_sub_row_time_factor_to_row(), mat2D_upper_triangulate(), 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.

Warning

Uses exact floating-point equality.

Definition at line 1088 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

The returned minor owns rows_list and cols_list and must be released with mat2D_minor_free().

The returned minor does not own ref_mat.elements.

Definition at line 1475 of file Matrix2D.h.

References Mat2D::cols, Mat2D_Minor::cols, Mat2D_Minor::cols_list, MAT2D_ASSERT, MAT2D_MALLOC, Mat2D_Minor::ref_mat, Mat2D::rows, Mat2D_Minor::rows, Mat2D_Minor::rows_list, and Mat2D_Minor::stride_r.

Referenced by mat2D_det().

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

The returned minor owns rows_list and cols_list and must be released with mat2D_minor_free().

The returned minor does not own the underlying reference matrix data.

Definition at line 1517 of file Matrix2D.h.

References Mat2D_Minor::cols, Mat2D_Minor::cols_list, MAT2D_ASSERT, MAT2D_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 1595 of file Matrix2D.h.

References Mat2D_Minor::cols, MAT2D_ASSERT, mat2D_det_2x2_mat_minor(), mat2D_minor_alloc_fill_from_mat_minor(), MAT2D_MINOR_AT, mat2D_minor_free(), and Mat2D_Minor::rows.

Referenced by mat2D_det().

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 1552 of file Matrix2D.h.

References Mat2D_Minor::cols_list, MAT2D_FREE, and Mat2D_Minor::rows_list.

Referenced by mat2D_det(), and 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 1564 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 657 of file Matrix2D.h.

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

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.

Warning

Indices are not bounds-checked in this routine.

Definition at line 674 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 < m.rows and 0 <= j < m.cols (checked by MAT2D_ASSERT).

Definition at line 416 of file Matrix2D.h.

References Mat2D::cols, MAT2D_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 < m.rows and 0 <= j < m.cols (checked by MAT2D_ASSERT).

Definition at line 432 of file Matrix2D.h.

References Mat2D_uint32::cols, MAT2D_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 687 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 706 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 pseudo-random doubles in [low, high].

Parameters

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

Precondition

high $>$ low (not checked here; caller responsibility).

Note

Uses mat2D_rand_double() (rand()).

Definition at line 491 of file Matrix2D.h.

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

Referenced by main().

4.1.3.40 mat2D_rand_double()

```
double mat2D_rand_double (
            void  )
```

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

Uses rand() / RAND_MAX from the C standard library.

Note

This RNG is not cryptographically secure and may have weak statistical properties depending on the platform.

Definition at line 327 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.

Warning

Uses exact floating-point equality.

Definition at line 1109 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).

Note

This routine allocates temporary 3x3 matrices internally.

Definition at line 860 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 (checked by MAT2D_ASSERT).

Definition at line 722 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, and Mat2D::rows.

Referenced by 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.

The matrix written is: [1, 0 , 0] [0, cos(a), sin(a)] [0,-sin(a), cos(a)]

Definition at line 789 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, MAT2D_PI, mat2D_set_identity(), 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.

The matrix written is: [cos(a), 0,-sin(a)] [0 , 1, 0] [sin(a), 0, cos(a)]

Definition at line 813 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, MAT2D_PI, mat2D_set_identity(), 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.

The matrix written is: [cos(a), sin(a), 0] [-sin(a), cos(a), 0] [0 , 0 , 1]

Definition at line 837 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, MAT2D_PI, mat2D_set_identity(), 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 an LUP-based approach.

Parameters

A	Coefficient matrix (N x N).
x	Solution vector (N x 1). Written on success.
B	Right-hand side vector (N x 1).

This routine computes an LUP decomposition and then forms explicit inverses of L and U (inv(L), inv(U)) to compute: x = inv(U) $*$ inv(L) $*$ (P $*$ B)

Warning

Explicitly inverting L and U is typically less stable and slower than forward/back substitution. Prefer substitution for production-quality solvers.

Definition at line 1429 of file Matrix2D.h.

References Mat2D::cols, mat2D_alloc(), MAT2D_ASSERT, mat2D_dot(), mat2D_fill(), mat2D_free(), mat2D_invert(), mat2D_LUP_decomposition_with_swap(), and Mat2D::rows.

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

dst and a have identical shape.

Definition at line 625 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, and Mat2D::rows.

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.

Performs: 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 974 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, 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.

Warning

Indices are not bounds-checked in this routine.

Definition at line 645 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_invert(), mat2D_LUP_decomposition_with_swap(), mat2D_make_identity(), and mat2D_upper_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.

Performs: 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 1052 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, 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.

Warning

Row indices are not bounds-checked in this routine.

Definition at line 993 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_invert(), mat2D_LUP_decomposition_with_swap(), and mat2D_upper_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.

Warning

If des aliases src, results are undefined (no in-place transpose).

Definition at line 1322 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, and Mat2D::rows.

4.1.3.54 mat2D_upper_triangulate()

```
double mat2D_upper_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.

Returns

A factor tracking the effect of row swaps on the determinant. Currently this is ±1 depending on the number of row swaps performed.

This routine performs Gaussian elimination using row operations of the form: row_j = row_j - (m[j,i] / m[i,i]) ∗ row_i which do not change the determinant. Row swaps flip the determinant sign and are tracked by the returned factor.

Warning

Pivoting is limited: a row swap is attempted only when the pivot is “near zero” per MAT2D_IS_ZERO(). No full pivoting is used.

Definition at line 1168 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, MAT2D_IS_ZERO, mat2D_sub_row_time_factor_to_row(), mat2D_swap_rows(), and Mat2D::rows.

Referenced by main(), mat2D_det(), and mat2D_make_identity().

4.2 Matrix2D.h

```
00001
00057 #ifndef MATRIX2D_H_
00058 #define MATRIX2D_H_
00059
00060 #include <stddef.h>
00061 #include <stdio.h>
00062 #include <stdlib.h>
00063 #include <stdint.h>
00064 #include <stdbool.h>
00065 #include <math.h>
00066
00075 #ifndef MAT2D_MALLOC
00076 #define MAT2D_MALLOC malloc
00077 #endif //MAT2D_MALLOC
00078
00087 #ifndef MAT2D_FREE
00088 #define MAT2D_FREE free
00089 #endif //MAT2D_FREE
00090
00099 #ifndef MAT2D_ASSERT
00100 #include <assert.h>
00101 #define MAT2D_ASSERT assert
00102 #endif //MAT2D_ASSERT
00103
00117 typedef struct {
00118     size_t rows;
00119     size_t cols;
00120     size_t stride_r; /* elements to traverse to reach the next row */
00121     double *elements;
00122 } Mat2D;
00123
00130 typedef struct {
00131     size_t rows;
00132     size_t cols;
00133     size_t stride_r; /* elements to traverse to reach the next row */
00134     uint32_t *elements;
00135 } Mat2D_uint32;
00136
00152 typedef struct {
00153     size_t rows;
00154     size_t cols;
00155     size_t stride_r; /* logical stride for the minor shape (not used for access) */
00156     size_t *rows_list;
00157     size_t *cols_list;
00158     Mat2D ref_mat;
00159 } Mat2D_Minor;
00160
00178 #if 1
00179 #define MAT2D_AT(m, i, j) (m).elements[mat2D_offset2d((m), (i), (j))]
00180 #define MAT2D_AT_UINT32(m, i, j) (m).elements[mat2D_offset2d_uint32((m), (i), (j))]
00181 #else /* use this macro for batter performance but no assertion */
00182 #define MAT2D_AT(m, i, j) (m).elements[(i) * (m).stride_r + (j)]
00183 #define MAT2D_AT_UINT32(m, i, j) (m).elements[(i) * (m).stride_r + (j)]
00184 #endif
00185
00186 #ifndef MAT2D_PI
00187     #define MAT2D_PI 3.14159265358979323846
00188 #endif
00189
00190 #ifndef MAT2D_EPS
00191     #define MAT2D_EPS 1e-15
00192 #endif
00193
00201 #define MAT2D_IS_ZERO(x) (fabs(x) < MAT2D_EPS)
00202
00210 #define MAT2D_MINOR_AT(mm, i, j) MAT2D_AT((mm).ref_mat, (mm).rows_list[i], (mm).cols_list[j])
00211
00216 #define MAT2D_PRINT(m) mat2D_print(m, #m, 0)
00217
00222 #define MAT2D_PRINT_AS_COL(m) mat2D_print_as_col(m, #m, 0)
00223
00228 #define MAT2D_MINOR_PRINT(mm) mat2D_minor_print(mm, #mm, 0)
00229
00240 #define mat2D_normalize(m) mat2D_mult((m), 1.0 / mat2D_calc_norma((m)))
00241
00242 double          mat2D_rand_double(void);
00243
00244 Mat2D           mat2D_alloc(size_t rows, size_t cols);
00245 Mat2D_uint32    mat2D_alloc_uint32(size_t rows, size_t cols);
00246 void            mat2D_free(Mat2D m);
00247 void            mat2D_free_uint32(Mat2D_uint32 m);
00248 size_t          mat2D_offset2d(Mat2D m, size_t i, size_t j);
00249 size_t          mat2D_offset2d_uint32(Mat2D_uint32 m, size_t i, size_t j);
00250
00251 void            mat2D_fill(Mat2D m, double x);
```

```
00252 void            mat2D_fill_sequence(Mat2D m, double start, double step);
00253 void            mat2D_fill_uint32(Mat2D_uint32 m, uint32_t x);
00254 void            mat2D_rand(Mat2D m, double low, double high);
00255
00256 void            mat2D_dot(Mat2D dst, Mat2D a, Mat2D b);
00257 double          mat2D_dot_product(Mat2D a, Mat2D b);
00258 void            mat2D_cross(Mat2D dst, Mat2D a, Mat2D b);
00259
00260 void            mat2D_add(Mat2D dst, Mat2D a);
00261 void            mat2D_add_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor);
00262
00263 void            mat2D_sub(Mat2D dst, Mat2D a);
00264 void            mat2D_sub_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor);
00265
00266 void            mat2D_mult(Mat2D m, double factor);
00267 void            mat2D_mult_row(Mat2D m, size_t r, double factor);
00268
00269 void            mat2D_print(Mat2D m, const char *name, size_t padding);
00270 void            mat2D_print_as_col(Mat2D m, const char *name, size_t padding);
00271
00272 void            mat2D_set_identity(Mat2D m);
00273 double          mat2D_make_identity(Mat2D m);
00274 void            mat2D_set_rot_mat_x(Mat2D m, float angle_deg);
00275 void            mat2D_set_rot_mat_y(Mat2D m, float angle_deg);
00276 void            mat2D_set_rot_mat_z(Mat2D m, float angle_deg);
00277 void            mat2D_set_DCM_zyx(Mat2D DCM, float yaw_deg, float pitch_deg, float roll_deg);
00278
00279 void            mat2D_copy(Mat2D des, Mat2D src);
00280 void            mat2D_copy_mat_to_mat_at_window(Mat2D des, Mat2D src, size_t is, size_t js, size_t ie,
      size_t je);
00281
00282 void            mat2D_get_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00283 void            mat2D_add_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00284 void            mat2D_sub_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00285
00286 void            mat2D_swap_rows(Mat2D m, size_t r1, size_t r2);
00287 void            mat2D_get_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
00288 void            mat2D_add_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
00289 void            mat2D_sub_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
00290
00291 double          mat2D_calc_norma(Mat2D m);
00292
00293 bool            mat2D_mat_is_all_digit(Mat2D m, double digit);
00294 bool            mat2D_row_is_all_digit(Mat2D m, double digit, size_t r);
00295 bool            mat2D_col_is_all_digit(Mat2D m, double digit, size_t c);
00296
00297 double          mat2D_det_2x2_mat(Mat2D m);
00298 double          mat2D_upper_triangulate(Mat2D m);
00299 double          mat2D_det(Mat2D m);
00300 void            mat2D_LUP_decomposition_with_swap(Mat2D src, Mat2D l, Mat2D p, Mat2D u);
00301 void            mat2D_transpose(Mat2D des, Mat2D src);
00302 void            mat2D_invert(Mat2D des, Mat2D src);
00303 void            mat2D_solve_linear_sys_LUP_decomposition(Mat2D A, Mat2D x, Mat2D B);
00304
00305 Mat2D_Minor     mat2D_minor_alloc_fill_from_mat(Mat2D ref_mat, size_t i, size_t j);
00306 Mat2D_Minor     mat2D_minor_alloc_fill_from_mat_minor(Mat2D_Minor ref_mm, size_t i, size_t j);
00307 void            mat2D_minor_free(Mat2D_Minor mm);
00308 void            mat2D_minor_print(Mat2D_Minor mm, const char *name, size_t padding);
00309 double          mat2D_det_2x2_mat_minor(Mat2D_Minor mm);
00310 double          mat2D_minor_det(Mat2D_Minor mm);
00311
00312 #endif // MATRIX2D_H_
00313
00314 #ifdef MATRIX2D_IMPLEMENTATION
00315 #undef MATRIX2D_IMPLEMENTATION
00316
00317
00327 double mat2D_rand_double(void)
00328 {
00329     return (double) rand() / (double) RAND_MAX;
00330 }
00331
00344 Mat2D mat2D_alloc(size_t rows, size_t cols)
00345 {
00346     Mat2D m;
00347     m.rows = rows;
00348     m.cols = cols;
00349     m.stride_r = cols;
00350     m.elements = (double*)MAT2D_MALLOC(sizeof(double)*rows*cols);
00351     MAT2D_ASSERT(m.elements != NULL);
00352
00353     return m;
00354 }
00355
00368 Mat2D_uint32 mat2D_alloc_uint32(size_t rows, size_t cols)
00369 {
00370     Mat2D_uint32 m;
```

```
00371     m.rows = rows;
00372     m.cols = cols;
00373     m.stride_r = cols;
00374     m.elements = (uint32_t*)MAT2D_MALLOC(sizeof(uint32_t)*rows*cols);
00375     MAT2D_ASSERT(m.elements != NULL);
00376
00377     return m;
00378 }
00379
00388 void mat2D_free(Mat2D m)
00389 {
00390     MAT2D_FREE(m.elements);
00391 }
00392
00401 void mat2D_free_uint32(Mat2D_uint32 m)
00402 {
00403     MAT2D_FREE(m.elements);
00404 }
00405
00416 size_t mat2D_offset2d(Mat2D m, size_t i, size_t j)
00417 {
00418     MAT2D_ASSERT(i < m.rows && j < m.cols);
00419     return i * m.stride_r + j;
00420 }
00421
00432 size_t mat2D_offset2d_uint32(Mat2D_uint32 m, size_t i, size_t j)
00433 {
00434     MAT2D_ASSERT(i < m.rows && j < m.cols);
00435     return i * m.stride_r + j;
00436 }
00437
00443 void mat2D_fill(Mat2D m, double x)
00444 {
00445     for (size_t i = 0; i < m.rows; ++i) {
00446         for (size_t j = 0; j < m.cols; ++j) {
00447             MAT2D_AT(m, i, j) = x;
00448         }
00449     }
00450 }
00451
00459 void mat2D_fill_sequence(Mat2D m, double start, double step) {
00460     for (size_t i = 0; i < m.rows; i++) {
00461         for (size_t j = 0; j < m.cols; j++) {
00462             MAT2D_AT(m, i, j) = start + step * mat2D_offset2d(m, i, j);
00463         }
00464     }
00465 }
00466
00472 void mat2D_fill_uint32(Mat2D_uint32 m, uint32_t x)
00473 {
00474     for (size_t i = 0; i < m.rows; ++i) {
00475         for (size_t j = 0; j < m.cols; ++j) {
00476             MAT2D_AT_UINT32(m, i, j) = x;
00477         }
00478     }
00479 }
00480
00491 void mat2D_rand(Mat2D m, double low, double high)
00492 {
00493     for (size_t i = 0; i < m.rows; ++i) {
00494         for (size_t j = 0; j < m.cols; ++j) {
00495             MAT2D_AT(m, i, j) = mat2D_rand_double()*(high - low) + low;
00496         }
00497     }
00498 }
00499
00515 void mat2D_dot(Mat2D dst, Mat2D a, Mat2D b)
00516 {
00517     MAT2D_ASSERT(a.cols == b.rows);
00518     MAT2D_ASSERT(a.rows == dst.rows);
00519     MAT2D_ASSERT(b.cols == dst.cols);
00520
00521     size_t i, j, k;
00522
00523     for (i = 0; i < dst.rows; i++) {
00524         for (j = 0; j < dst.cols; j++) {
00525             MAT2D_AT(dst, i, j) = 0;
00526             for (k = 0; k < a.cols; k++) {
00527                 MAT2D_AT(dst, i, j) += MAT2D_AT(a, i, k)*MAT2D_AT(b, k, j);
00528             }
00529         }
00530     }
00531
00532 }
00533
00544 double mat2D_dot_product(Mat2D a, Mat2D b)
00545 {
```

```
00546     MAT2D_ASSERT(a.rows == b.rows);
00547     MAT2D_ASSERT(a.cols == b.cols);
00548     MAT2D_ASSERT((1 == a.cols && 1 == b.cols) || (1 == a.rows && 1 == b.rows));
00549
00550     double dot_product = 0;
00551
00552     if (1 == a.cols) {
00553         for (size_t i = 0; i < a.rows; i++) {
00554             dot_product += MAT2D_AT(a, i, 0) * MAT2D_AT(b, i, 0);
00555         }
00556     } else {
00557         for (size_t j = 0; j < a.cols; j++) {
00558             dot_product += MAT2D_AT(a, 0, j) * MAT2D_AT(b, 0, j);
00559         }
00560     }
00561
00562     return dot_product;
00563 }
00564
00572 void mat2D_cross(Mat2D dst, Mat2D a, Mat2D b)
00573 {
00574     MAT2D_ASSERT(3 == dst.rows && 1 == dst.cols);
00575     MAT2D_ASSERT(3 == a.rows && 1 == a.cols);
00576     MAT2D_ASSERT(3 == b.rows && 1 == b.cols);
00577
00578     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);
00579     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);
00580     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);
00581 }
00582
00591 void mat2D_add(Mat2D dst, Mat2D a)
00592 {
00593     MAT2D_ASSERT(dst.rows == a.rows);
00594     MAT2D_ASSERT(dst.cols == a.cols);
00595     for (size_t i = 0; i < dst.rows; ++i) {
00596         for (size_t j = 0; j < dst.cols; ++j) {
00597             MAT2D_AT(dst, i, j) += MAT2D_AT(a, i, j);
00598         }
00599     }
00600 }
00601
00611 void mat2D_add_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor)
00612 {
00613     for (size_t j = 0; j < m.cols; ++j) {
00614         MAT2D_AT(m, des_r, j) += factor * MAT2D_AT(m, src_r, j);
00615     }
00616 }
00617
00625 void mat2D_sub(Mat2D dst, Mat2D a)
00626 {
00627     MAT2D_ASSERT(dst.rows == a.rows);
00628     MAT2D_ASSERT(dst.cols == a.cols);
00629     for (size_t i = 0; i < dst.rows; ++i) {
00630         for (size_t j = 0; j < dst.cols; ++j) {
00631             MAT2D_AT(dst, i, j) -= MAT2D_AT(a, i, j);
00632         }
00633     }
00634 }
00635
00645 void mat2D_sub_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor)
00646 {
00647     for (size_t j = 0; j < m.cols; ++j) {
00648         MAT2D_AT(m, des_r, j) -= factor * MAT2D_AT(m, src_r, j);
00649     }
00650 }
00651
00657 void mat2D_mult(Mat2D m, double factor)
00658 {
00659     for (size_t i = 0; i < m.rows; ++i) {
00660         for (size_t j = 0; j < m.cols; ++j) {
00661             MAT2D_AT(m, i, j) *= factor;
00662         }
00663     }
00664 }
00665
00674 void mat2D_mult_row(Mat2D m, size_t r, double factor)
00675 {
00676     for (size_t j = 0; j < m.cols; ++j) {
00677         MAT2D_AT(m, r, j) *= factor;
00678     }
00679 }
00680
00687 void mat2D_print(Mat2D m, const char *name, size_t padding)
00688 {
```

```
00689     printf("%*s%s = [\n", (int) padding, "", name);
00690     for (size_t i = 0; i < m.rows; ++i) {
00691         printf("%*s    ", (int) padding, "");
00692         for (size_t j = 0; j < m.cols; ++j) {
00693             printf("%9.6f ", MAT2D_AT(m, i, j));
00694         }
00695         printf("\n");
00696     }
00697     printf("%*s]\n", (int) padding, "");
00698 }
00699
00706 void mat2D_print_as_col(Mat2D m, const char *name, size_t padding)
00707 {
00708     printf("%*s%s = [\n", (int) padding, "", name);
00709     for (size_t i = 0; i < m.rows*m.cols; ++i) {
00710             printf("%*s    ", (int) padding, "");
00711             printf("%f\n", m.elements[i]);
00712     }
00713     printf("%*s]\n", (int) padding, "");
00714 }
00715
00722 void mat2D_set_identity(Mat2D m)
00723 {
00724     MAT2D_ASSERT(m.cols == m.rows);
00725     for (size_t i = 0; i < m.rows; ++i) {
00726         for (size_t j = 0; j < m.cols; ++j) {
00727             MAT2D_AT(m, i, j) = i == j ? 1 : 0;
00728             // if (i == j) {
00729             //     MAT2D_AT(m, i, j) = 1;
00730             // }
00731             // else {
00732             //     MAT2D_AT(m, i, j) = 0;
00733             // }
00734         }
00735     }
00736 }
00737
00754 double mat2D_make_identity(Mat2D m)
00755 {
00756     /* make identity matrix using Gauss elimination */
00757     /* preforming Gauss elimination: https://en.wikipedia.org/wiki/Gaussian_elimination */
00758     /* returns the factor multiplying the determinant */
00759
00760     double factor_to_return = mat2D_upper_triangulate(m);
00761
00762     double factor = 1 / MAT2D_AT(m, m.rows-1, m.cols-1);
00763     mat2D_mult_row(m, m.rows-1, factor);
00764     factor_to_return *= factor;
00765     for (size_t c = m.cols-1; c > 0; c--) {
00766         double factor = 1 / MAT2D_AT(m, c, c);
00767         mat2D_mult_row(m, c, factor);
00768         for (int r = c-1; r >= 0; r--) {
00769             mat2D_sub_row_time_factor_to_row(m, r, c, MAT2D_AT(m, r, c));
00770         }
00771     }
00772
00773
00774     return factor_to_return;
00775 }
00776
00789 void mat2D_set_rot_mat_x(Mat2D m, float angle_deg)
00790 {
00791     MAT2D_ASSERT(3 == m.cols && 3 == m.rows);
00792
00793     float angle_rad = angle_deg * MAT2D_PI / 180;
00794     mat2D_set_identity(m);
00795     MAT2D_AT(m, 1, 1) =  cos(angle_rad);
00796     MAT2D_AT(m, 1, 2) =  sin(angle_rad);
00797     MAT2D_AT(m, 2, 1) = -sin(angle_rad);
00798     MAT2D_AT(m, 2, 2) =  cos(angle_rad);
00799 }
00800
00813 void mat2D_set_rot_mat_y(Mat2D m, float angle_deg)
00814 {
00815     MAT2D_ASSERT(3 == m.cols && 3 == m.rows);
00816
00817     float angle_rad = angle_deg * MAT2D_PI / 180;
00818     mat2D_set_identity(m);
00819     MAT2D_AT(m, 0, 0) =  cos(angle_rad);
00820     MAT2D_AT(m, 0, 2) = -sin(angle_rad);
00821     MAT2D_AT(m, 2, 0) =  sin(angle_rad);
00822     MAT2D_AT(m, 2, 2) =  cos(angle_rad);
00823 }
00824
00837 void mat2D_set_rot_mat_z(Mat2D m, float angle_deg)
00838 {
00839     MAT2D_ASSERT(3 == m.cols && 3 == m.rows);
```

```
00840
00841     float angle_rad = angle_deg * MAT2D_PI / 180;
00842     mat2D_set_identity(m);
00843     MAT2D_AT(m, 0, 0) =  cos(angle_rad);
00844     MAT2D_AT(m, 0, 1) =  sin(angle_rad);
00845     MAT2D_AT(m, 1, 0) = -sin(angle_rad);
00846     MAT2D_AT(m, 1, 1) =  cos(angle_rad);
00847 }
00848
00860 void mat2D_set_DCM_zyx(Mat2D DCM, float yaw_deg, float pitch_deg, float roll_deg)
00861 {
00862     Mat2D RotZ = mat2D_alloc(3,3);
00863     mat2D_set_rot_mat_z(RotZ, yaw_deg);
00864     Mat2D RotY = mat2D_alloc(3,3);
00865     mat2D_set_rot_mat_y(RotY, pitch_deg);
00866     Mat2D RotX = mat2D_alloc(3,3);
00867     mat2D_set_rot_mat_x(RotX, roll_deg);
00868     Mat2D temp = mat2D_alloc(3,3);
00869
00870     mat2D_dot(temp, RotY, RotZ);
00871     mat2D_dot(DCM, RotX, temp); /* I have a DCM */
00872
00873     mat2D_free(RotZ);
00874     mat2D_free(RotY);
00875     mat2D_free(RotX);
00876     mat2D_free(temp);
00877 }
00878
00887 void mat2D_copy(Mat2D des, Mat2D src)
00888 {
00889     MAT2D_ASSERT(des.cols == src.cols);
00890     MAT2D_ASSERT(des.rows == src.rows);
00891
00892     for (size_t i = 0; i < des.rows; ++i) {
00893         for (size_t j = 0; j < des.cols; ++j) {
00894             MAT2D_AT(des, i, j) = MAT2D_AT(src, i, j);
00895         }
00896     }
00897 }
00898
00909 void mat2D_copy_mat_to_mat_at_window(Mat2D des, Mat2D src, size_t is, size_t js, size_t ie, size_t je)
00910 {
00911     MAT2D_ASSERT(je >= js && ie >= is);
00912     MAT2D_ASSERT(je-js+1 == des.cols);
00913     MAT2D_ASSERT(ie-is+1 == des.rows);
00914
00915     for (size_t index = 0; index < des.rows; ++index) {
00916         for (size_t jndex = 0; jndex < des.cols; ++jndex) {
00917             MAT2D_AT(des, index, jndex) = MAT2D_AT(src, is+index, js+jndex);
00918         }
00919     }
00920 }
00921
00932 void mat2D_get_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
00933 {
00934     MAT2D_ASSERT(src_col < src.cols);
00935     MAT2D_ASSERT(des.rows == src.rows);
00936     MAT2D_ASSERT(des_col < des.cols);
00937
00938     for (size_t i = 0; i < des.rows; i++) {
00939         MAT2D_AT(des, i, des_col) = MAT2D_AT(src, i, src_col);
00940     }
00941 }
00942
00953 void mat2D_add_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
00954 {
00955     MAT2D_ASSERT(src_col < src.cols);
00956     MAT2D_ASSERT(des.rows == src.rows);
00957     MAT2D_ASSERT(des_col < des.cols);
00958
00959     for (size_t i = 0; i < des.rows; i++) {
00960         MAT2D_AT(des, i, des_col) += MAT2D_AT(src, i, src_col);
00961     }
00962 }
00963
00974 void mat2D_sub_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
00975 {
00976     MAT2D_ASSERT(src_col < src.cols);
00977     MAT2D_ASSERT(des.rows == src.rows);
00978     MAT2D_ASSERT(des_col < des.cols);
00979
00980     for (size_t i = 0; i < des.rows; i++) {
00981         MAT2D_AT(des, i, des_col) -= MAT2D_AT(src, i, src_col);
00982     }
00983 }
00984
00993 void mat2D_swap_rows(Mat2D m, size_t r1, size_t r2)
```

```
00994 {
00995     for (size_t j = 0; j < m.cols; j++) {
00996         double temp = MAT2D_AT(m, r1, j);
00997         MAT2D_AT(m, r1, j) = MAT2D_AT(m, r2, j);
00998         MAT2D_AT(m, r2, j) = temp;
00999     }
01000 }
01001
01011 void mat2D_get_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
01012 {
01013     MAT2D_ASSERT(src_row < src.rows);
01014     MAT2D_ASSERT(des.cols == src.cols);
01015     MAT2D_ASSERT(des_row < des.rows);
01016
01017     for (size_t j = 0; j < des.cols; j++) {
01018         MAT2D_AT(des, des_row, j) = MAT2D_AT(src, src_row, j);
01019     }
01020 }
01021
01031 void mat2D_add_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
01032 {
01033     MAT2D_ASSERT(src_row < src.rows);
01034     MAT2D_ASSERT(des.cols == src.cols);
01035     MAT2D_ASSERT(des_row < des.rows);
01036
01037     for (size_t j = 0; j < des.cols; j++) {
01038         MAT2D_AT(des, des_row, j) += MAT2D_AT(src, src_row, j);
01039     }
01040 }
01041
01052 void mat2D_sub_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
01053 {
01054     MAT2D_ASSERT(src_row < src.rows);
01055     MAT2D_ASSERT(des.cols == src.cols);
01056     MAT2D_ASSERT(des_row < des.rows);
01057
01058     for (size_t j = 0; j < des.cols; j++) {
01059         MAT2D_AT(des, des_row, j) -= MAT2D_AT(src, src_row, j);
01060     }
01061 }
01062
01068 double mat2D_calc_norma(Mat2D m)
01069 {
01070     double sum = 0;
01071
01072     for (size_t i = 0; i < m.rows; ++i) {
01073         for (size_t j = 0; j < m.cols; ++j) {
01074             sum += MAT2D_AT(m, i, j) * MAT2D_AT(m, i, j);
01075         }
01076     }
01077     return sqrt(sum);
01078 }
01079
01088 bool mat2D_mat_is_all_digit(Mat2D m, double digit)
01089 {
01090     for (size_t i = 0; i < m.rows; ++i) {
01091         for (size_t j = 0; j < m.cols; ++j) {
01092             if (MAT2D_AT(m, i, j) != digit) {
01093                 return false;
01094             }
01095         }
01096     }
01097     return true;
01098 }
01099
01109 bool mat2D_row_is_all_digit(Mat2D m, double digit, size_t r)
01110 {
01111     for (size_t j = 0; j < m.cols; ++j) {
01112         if (MAT2D_AT(m, r, j) != digit) {
01113             return false;
01114         }
01115     }
01116     return true;
01117 }
01118
01128 bool mat2D_col_is_all_digit(Mat2D m, double digit, size_t c)
01129 {
01130     for (size_t i = 0; i < m.rows; ++i) {
01131         if (MAT2D_AT(m, i, c) != digit) {
01132             return false;
01133         }
01134     }
01135     return true;
01136 }
01137
01143 double mat2D_det_2x2_mat(Mat2D m)
01144 {
```

```
01145     MAT2D_ASSERT(2 == m.cols && 2 == m.rows && "Not a 2x2 matrix");
01146     return MAT2D_AT(m, 0, 0) * MAT2D_AT(m, 1, 1) - MAT2D_AT(m, 0, 1) * MAT2D_AT(m, 1, 0);
01147 }
01148
01168 double mat2D_upper_triangulate(Mat2D m)
01169 {
01170     /* preforming Gauss elimination: https://en.wikipedia.org/wiki/Gaussian_elimination */
01171     /* returns the factor multiplying the determinant */
01172
01173     double factor_to_return = 1;
01174
01175     size_t size = (size_t)fmin(m.rows, m.cols);
01176     for (size_t i = 0; i < size; i++) {
01177         if (MAT2D_IS_ZERO(MAT2D_AT(m, i, i))) {   /* swapping only if it is zero */
01178             /* finding biggest first number (absolute value) */
01179             size_t biggest_r = i;
01180             for (size_t index = i; index < m.rows; index++) {
01181                 if (fabs(MAT2D_AT(m, index, i)) > fabs(MAT2D_AT(m, biggest_r, i))) {
01182                     biggest_r = index;
01183                 }
01184             }
01185             if (i != biggest_r) {
01186                 mat2D_swap_rows(m, i, biggest_r);
01187                 factor_to_return *= -1;
01188             }
01189         }
01190         for (size_t j = i+1; j < m.rows; j++) {
01191             double factor = 1 / MAT2D_AT(m, i, i);
01192             if (!isfinite(factor)) {
01193                 printf("%s:%d:\n%s:\n[Error] unable to transfrom into uperr triangular matrix.
      Probably some of the rows are not independent.\n", __FILE__, __LINE__, __func__);
01194             }
01195             double mat_value = MAT2D_AT(m, j, i);
01196             mat2D_sub_row_time_factor_to_row(m, j, i, mat_value * factor);
01197         }
01198     }
01199     return factor_to_return;
01200 }
01201
01215 double mat2D_det(Mat2D m)
01216 {
01217     MAT2D_ASSERT(m.cols == m.rows && "should be a square matrix");
01218
01219     /* checking if there is a row or column with all zeros */
01220     /* checking rows */
01221     for (size_t i = 0; i < m.rows; i++) {
01222         if (mat2D_row_is_all_digit(m, 0, i)) {
01223             return 0;
01224         }
01225     }
01226     /* checking cols */
01227     for (size_t j = 0; j < m.rows; j++) {
01228         if (mat2D_col_is_all_digit(m, 0, j)) {
01229             return 0;
01230         }
01231     }
01232
01233     #if 0/* This is an implementation of naive determinant calculation using minors. This is too slow
    */
01234     double det = 0;
01235     /* TODO: finding beast row or col? */
01236     for (size_t i = 0, j = 0; i < m.rows; i++) { /* first column */
01237         if (MAT2D_AT(m, i, j) < 1e-10) continue;
01238         Mat2D_Minor sub_mm = mat2D_minor_alloc_fill_from_mat(m, i, j);
01239         int factor = (i+j)%2 ? -1 : 1;
01240         if (sub_mm.cols != 2) {
01241             MAT2D_ASSERT(sub_mm.cols == sub_mm.rows && "should be a square matrix");
01242             det += MAT2D_AT(m, i, j) * (factor) * mat2D_minor_det(sub_mm);
01243         } else if (sub_mm.cols == 2 && sub_mm.rows == 2) {
01244             det += MAT2D_AT(m, i, j) * (factor) * mat2D_det_2x2_mat_minor(sub_mm);;
01245         }
01246         mat2D_minor_free(sub_mm);
01247     }
01248     #endif
01249
01250     Mat2D temp_m = mat2D_alloc(m.rows, m.cols);
01251     mat2D_copy(temp_m, m);
01252     double factor = mat2D_upper_triangulate(temp_m);
01253     double diag_mul = 1;
01254     for (size_t i = 0; i < temp_m.rows; i++) {
01255         diag_mul *= MAT2D_AT(temp_m, i, i);
01256     }
01257     mat2D_free(temp_m);
01258
01259     return diag_mul / factor;
01260 }
01261
```

```
01278 void mat2D_LUP_decomposition_with_swap(Mat2D src, Mat2D l, Mat2D p, Mat2D u)
01279 {
01280     /* performing LU decomposition Following the Wikipedia page:
      https://en.wikipedia.org/wiki/LU_decomposition */
01281
01282     mat2D_copy(u, src);
01283     mat2D_set_identity(p);
01284     mat2D_fill(l, 0);
01285
01286     for (size_t i = 0; i < (size_t)fmin(u.rows-1, u.cols); i++) {
01287         if (MAT2D_IS_ZERO(MAT2D_AT(u, i, i))) {   /* swapping only if it is zero */
01288             /* finding biggest first number (absolute value) */
01289             size_t biggest_r = i;
01290             for (size_t index = i; index < u.rows; index++) {
01291                 if (fabs(MAT2D_AT(u, index, i)) > fabs(MAT2D_AT(u, biggest_r, i))) {
01292                     biggest_r = index;
01293                 }
01294             }
01295             if (i != biggest_r) {
01296                 mat2D_swap_rows(u, i, biggest_r);
01297                 mat2D_swap_rows(p, i, biggest_r);
01298                 mat2D_swap_rows(l, i, biggest_r);
01299             }
01300         }
01301         for (size_t j = i+1; j < u.cols; j++) {
01302             double factor = 1 / MAT2D_AT(u, i, i);
01303             if (!isfinite(factor)) {
01304                 printf("%s:%d:\n%s:\n[Error] unable to transfrom into uper triangular matrix. Probably
      some of the rows are not independent.\n", __FILE__, __LINE__, __func__);
01305             }
01306             double mat_value = MAT2D_AT(u, j, i);
01307             mat2D_sub_row_time_factor_to_row(u, j, i, mat_value * factor);
01308             MAT2D_AT(l, j, i) = mat_value * factor;
01309         }
01310         MAT2D_AT(l, i, i) = 1;
01311     }
01312     MAT2D_AT(l, l.rows-1, l.cols-1) = 1;
01313 }
01314
01322 void mat2D_transpose(Mat2D des, Mat2D src)
01323 {
01324     MAT2D_ASSERT(des.cols == src.rows);
01325     MAT2D_ASSERT(des.rows == src.cols);
01326
01327     for (size_t index = 0; index < des.rows; ++index) {
01328         for (size_t jndex = 0; jndex < des.cols; ++jndex) {
01329             MAT2D_AT(des, index, jndex) = MAT2D_AT(src, jndex, index);
01330         }
01331     }
01332 }
01333
01353 void mat2D_invert(Mat2D des, Mat2D src)
01354 {
01355     MAT2D_ASSERT(src.cols == src.rows && "should be an NxN matrix");
01356     MAT2D_ASSERT(des.cols == src.cols && des.rows == des.cols);
01357
01358     Mat2D m = mat2D_alloc(src.rows, src.cols);
01359     mat2D_copy(m, src);
01360
01361     mat2D_set_identity(des);
01362
01363     if (!(mat2D_det(m))) {
01364         mat2D_fill(des, 0);
01365         printf("%s:%d:\n%s:\n[Error] Can't invert the matrix. Determinant is zero! Set the inverse
      matrix to all zeros\n", __FILE__, __LINE__, __func__);
01366         mat2D_free(m);
01367         return;
01368     }
01369
01370     size_t size = (size_t)fmin(m.rows, m.cols);
01371     for (size_t i = 0; i < size; i++) {
01372         if (MAT2D_IS_ZERO(MAT2D_AT(m, i, i))) {   /* swapping only if it is zero */
01373             /* finding biggest first number (absolute value) */
01374             size_t biggest_r = i;
01375             for (size_t index = i; index < m.rows; index++) {
01376                 if (fabs(MAT2D_AT(m, index, i)) > fabs(MAT2D_AT(m, biggest_r, i))) {
01377                     biggest_r = index;
01378                 }
01379             }
01380             if (i != biggest_r) {
01381                 mat2D_swap_rows(m, i, biggest_r);
01382                 mat2D_swap_rows(des, i, biggest_r);
01383                 printf("%s:%d:\n%s:\n[INFO] swapping row %zu with row %zu.\n", __FILE__, __LINE__,
      __func__, i, biggest_r);
01384             } else {
01385                 MAT2D_ASSERT(0 && "can't inverse");
01386             }
```

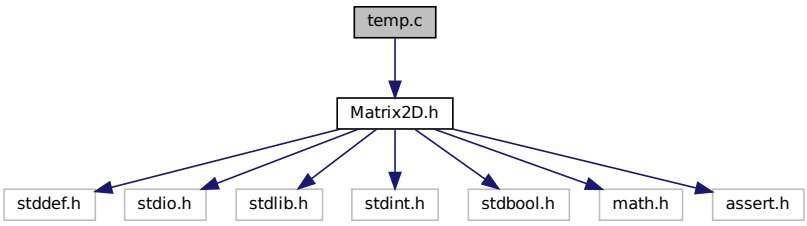
```
01387         }
01388         for (size_t j = i+1; j < size; j++) {
01389             double factor = 1 / MAT2D_AT(m, i, i);
01390             double mat_value = MAT2D_AT(m, j, i);
01391             mat2D_sub_row_time_factor_to_row(m, j, i, mat_value * factor);
01392 
01393             mat2D_sub_row_time_factor_to_row(des, j, i, mat_value * factor);
01394         }
01395     }
01396     double factor = 1 / MAT2D_AT(m, m.rows-1, m.cols-1);
01397     mat2D_mult_row(m, m.rows-1, factor);
01398     mat2D_mult_row(des, des.rows-1, factor);
01399     for (size_t c = m.cols-1; c > 0; c--) {
01400         double factor = 1 / MAT2D_AT(m, c, c);
01401         mat2D_mult_row(m, c, factor);
01402         mat2D_mult_row(des, c, factor);
01403         for (int r = c-1; r >= 0; r--) {
01404             double mat_value = MAT2D_AT(m, r, c);
01405             mat2D_sub_row_time_factor_to_row(m, r, c, mat_value);
01406             mat2D_sub_row_time_factor_to_row(des, r, c, mat_value);
01407         }
01408     }
01409 
01410     mat2D_free(m);
01411 }
01412 
01429 void mat2D_solve_linear_sys_LUP_decomposition(Mat2D A, Mat2D x, Mat2D B)
01430 {
01431     MAT2D_ASSERT(A.cols == x.rows);
01432     MAT2D_ASSERT(1 == x.cols);
01433     MAT2D_ASSERT(A.rows == B.rows);
01434     MAT2D_ASSERT(1 == B.cols);
01435 
01436     Mat2D y     = mat2D_alloc(x.rows, x.cols);
01437     Mat2D l     = mat2D_alloc(A.rows, A.cols);
01438     Mat2D p     = mat2D_alloc(A.rows, A.cols);
01439     Mat2D u     = mat2D_alloc(A.rows, A.cols);
01440     Mat2D inv_l = mat2D_alloc(l.rows, l.cols);
01441     Mat2D inv_u = mat2D_alloc(u.rows, u.cols);
01442 
01443     mat2D_LUP_decomposition_with_swap(A, l, p, u);
01444 
01445     mat2D_invert(inv_l, l);
01446     mat2D_invert(inv_u, u);
01447 
01448     mat2D_fill(x, 0);   /* x here is only a temp mat*/
01449     mat2D_fill(y, 0);
01450     mat2D_dot(x, p, B);
01451     mat2D_dot(y, inv_l, x);
01452 
01453     mat2D_fill(x, 0);
01454     mat2D_dot(x, inv_u, y);
01455 
01456     mat2D_free(y);
01457     mat2D_free(l);
01458     mat2D_free(p);
01459     mat2D_free(u);
01460     mat2D_free(inv_l);
01461     mat2D_free(inv_u);
01462 }
01463 
01475 Mat2D_Minor mat2D_minor_alloc_fill_from_mat(Mat2D ref_mat, size_t i, size_t j)
01476 {
01477     MAT2D_ASSERT(ref_mat.cols == ref_mat.rows && "minor is defined only for square matrix");
01478 
01479     Mat2D_Minor mm;
01480     mm.cols = ref_mat.cols-1;
01481     mm.rows = ref_mat.rows-1;
01482     mm.stride_r = ref_mat.cols-1;
01483     mm.cols_list = (size_t*)MAT2D_MALLOC(sizeof(size_t)*(ref_mat.cols-1));
01484     mm.rows_list = (size_t*)MAT2D_MALLOC(sizeof(size_t)*(ref_mat.rows-1));
01485     mm.ref_mat = ref_mat;
01486 
01487     MAT2D_ASSERT(mm.cols_list != NULL && mm.rows_list != NULL);
01488 
01489     for (size_t index = 0, temp_index = 0; index < ref_mat.rows; index++) {
01490         if (index != i) {
01491             mm.rows_list[temp_index] = index;
01492             temp_index++;
01493         }
01494     }
01495     for (size_t jndex = 0, temp_jndex = 0; jndex < ref_mat.cols; jndex++) {
01496         if (jndex != j) {
01497             mm.cols_list[temp_jndex] = jndex;
01498             temp_jndex++;
01499         }
01500     }
```

```
01501
01502     return mm;
01503 }
01504
01517 Mat2D_Minor mat2D_minor_alloc_fill_from_mat_minor(Mat2D_Minor ref_mm, size_t i, size_t j)
01518 {
01519     MAT2D_ASSERT(ref_mm.cols == ref_mm.rows && "minor is defined only for square matrix");
01520
01521     Mat2D_Minor mm;
01522     mm.cols = ref_mm.cols-1;
01523     mm.rows = ref_mm.rows-1;
01524     mm.stride_r = ref_mm.cols-1;
01525     mm.cols_list = (size_t*)MAT2D_MALLOC(sizeof(size_t)*(ref_mm.cols-1));
01526     mm.rows_list = (size_t*)MAT2D_MALLOC(sizeof(size_t)*(ref_mm.rows-1));
01527     mm.ref_mat = ref_mm.ref_mat;
01528
01529     MAT2D_ASSERT(mm.cols_list != NULL && mm.rows_list != NULL);
01530
01531     for (size_t index = 0, temp_index = 0; index < ref_mm.rows; index++) {
01532         if (index != i) {
01533             mm.rows_list[temp_index] = ref_mm.rows_list[index];
01534             temp_index++;
01535         }
01536     }
01537     for (size_t jndex = 0, temp_jndex = 0; jndex < ref_mm.cols; jndex++) {
01538         if (jndex != j) {
01539             mm.cols_list[temp_jndex] = ref_mm.cols_list[jndex];
01540             temp_jndex++;
01541         }
01542     }
01543
01544     return mm;
01545 }
01546
01552 void mat2D_minor_free(Mat2D_Minor mm)
01553 {
01554     MAT2D_FREE(mm.cols_list);
01555     MAT2D_FREE(mm.rows_list);
01556 }
01557
01564 void mat2D_minor_print(Mat2D_Minor mm, const char *name, size_t padding)
01565 {
01566     printf("%*s%s = [\n", (int) padding, "", name);
01567     for (size_t i = 0; i < mm.rows; ++i) {
01568         printf("%*s    ", (int) padding, "");
01569         for (size_t j = 0; j < mm.cols; ++j) {
01570             printf("%f ", MAT2D_MINOR_AT(mm, i, j));
01571         }
01572         printf("\n");
01573     }
01574     printf("%*s]\n", (int) padding, "");
01575 }
01576
01582 double mat2D_det_2x2_mat_minor(Mat2D_Minor mm)
01583 {
01584     MAT2D_ASSERT(2 == mm.cols && 2 == mm.rows && "Not a 2x2 matrix");
01585     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);
01586 }
01587
01595 double mat2D_minor_det(Mat2D_Minor mm)
01596 {
01597     MAT2D_ASSERT(mm.cols == mm.rows && "should be a square matrix");
01598
01599     double det = 0;
01600     /* TODO: finding beast row or col? */
01601     for (size_t i = 0, j = 0; i < mm.rows; i++) { /* first column */
01602         if (fabs(MAT2D_MINOR_AT(mm, i, j)) < 1e-10) continue;
01603         Mat2D_Minor sub_mm = mat2D_minor_alloc_fill_from_mat_minor(mm, i, j);
01604         int factor = (i+j)%2 ? -1 : 1;
01605         if (sub_mm.cols != 2) {
01606             MAT2D_ASSERT(sub_mm.cols == sub_mm.rows && "should be a square matrix");
01607             det += MAT2D_MINOR_AT(mm, i, j) * (factor) * mat2D_minor_det(sub_mm);
01608         } else if (sub_mm.cols == 2 && sub_mm.rows == 2) {
01609             det += MAT2D_MINOR_AT(mm, i, j) * (factor) * mat2D_det_2x2_mat_minor(sub_mm);;
01610         }
01611         mat2D_minor_free(sub_mm);
01612     }
01613     return det;
01614 }
01615
01616 #endif // MATRIX2D_IMPLEMENTATION
```

4.3 temp.c File Reference

```
#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 1 of file temp.c.

4.3.2 Function Documentation

4.3.2.1 main()

```
int main (
            void  )
```

Definition at line 4 of file temp.c.

References mat2D_alloc(), mat2D_free(), MAT2D_PRINT, mat2D_rand(), and mat2D_upper_triangulate().

4.4 temp.c

```
00001 #define MATRIX2D_IMPLEMENTATION
00002 #include "Matrix2D.h"
00003
00004 int main(void)
00005 {
00006     Mat2D m = mat2D_alloc(5, 4);
00007
00008     mat2D_rand(m, 0, 1);
00009
00010     MAT2D_PRINT(m);
00011
00012     mat2D_upper_triangulate(m);
00013
00014     MAT2D_PRINT(m);
00015
00016
00017     mat2D_free(m);
00018
00019     return 0;
00020 }
```

Index