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_calc_norma_inf(), mat2D_copy(), mat2D_copy_src_to_des_window(), mat2D_copy_src_window_to_des(), mat2D_cross(), mat2D_det(), mat2D_det_2x2_mat(), mat2D_dot(), mat2D_dot_product(), mat2D_eig_check(), mat2D_eig_power_iteration(), mat2D_fill(), mat2D_fill_sequence(), mat2D_find_first_non_zero_value(), mat2D_get_col(), mat2D_get_row(), mat2D_inner_product(), mat2D_invert(), mat2D_LUP_decomposition_with_swap(), mat2D_mat_is_all_digit(), mat2D_minor_alloc_fill_from_mat(), mat2D_mult(), mat2D_mult_row(), mat2D_offset2d(), mat2D_outer_product(), mat2D_power_iterate(), mat2D_print(), mat2D_print_as_col(), mat2D_rand(), mat2D_reduce(), mat2D_row_is_all_digit(), mat2D_set_identity(), mat2D_set_rot_mat_x(), mat2D_set_rot_mat_y(), mat2D_set_rot_mat_z(), mat2D_shift(), 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_calc_norma_inf(), mat2D_col_is_all_digit(), mat2D_copy(), mat2D_copy_src_to_des_window(), mat2D_copy_src_window_to_des(), mat2D_cross(), mat2D_det(), mat2D_det_2x2_mat(), mat2D_dot(), mat2D_dot_product(), mat2D_eig_check(), mat2D_eig_power_iteration(), mat2D_fill(), mat2D_fill_sequence(), mat2D_get_col(), mat2D_get_row(), mat2D_inner_product(), mat2D_invert(), mat2D_LUP_decomposition_with_swap(), mat2D_mat_is_all_digit(), mat2D_minor_alloc_fill_from_mat(), mat2D_mult(), mat2D_offset2d(), mat2D_outer_product(), mat2D_power_iterate(), mat2D_print(), mat2D_print_as_col(), mat2D_rand(), mat2D_reduce(), mat2D_set_identity(), mat2D_set_rot_mat_x(), mat2D_set_rot_mat_y(), mat2D_set_rot_mat_z(), mat2D_shift(), 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(), mat2D_eig_check(), mat2D_eig_power_iteration(), and mat2D_offset2d().

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

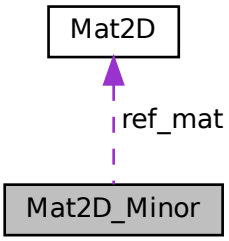
- Matrix2D.h

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

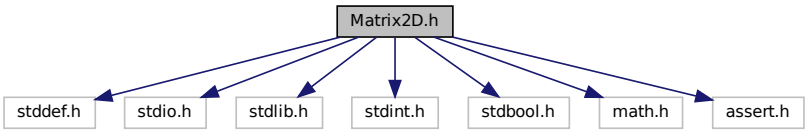
File Documentation

4.1 Matrix2D.h File Reference

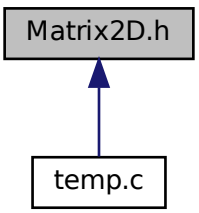
Lightweight 2D matrix helpers (double / uint32_t).

```
#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_MAX_POWER_ITERATION 100
- #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.
- #define mat2D_normalize_inf(m) mat2D_mult((m), 1.0 / mat2D_calc_norma_inf((m)))
- #define mat2D_dprintDOUBLE(expr) printf(#expr " = %#g\n", expr)
- #define mat2D_dprintSIZE_T(expr) printf(#expr " = %zu\n", expr)
- #define mat2D_dprintINT(expr) printf(#expr " = %d\n", expr)

Functions

- void mat2D_add (Mat2D dst, Mat2D a)

  In-place addition: dst += a.
- 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_add_row_to_row (Mat2D des, size_t des_row, Mat2D src, size_t src_row)

  Add a source row into a destination 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).*

- 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.
- double mat2D_calc_norma (Mat2D m)

  Compute the Frobenius norm of a matrix, sqrt(sum(m_ij^2)).
- double mat2D_calc_norma_inf (Mat2D m)

  Compute the maximum absolute element value of a matrix.
- bool mat2D_col_is_all_digit (Mat2D m, double digit, size_t c)

  Check if all elements of a column equal a given digit.
- void mat2D_copy (Mat2D des, Mat2D src)

  Copy all elements from src to des.
- void mat2D_copy_src_to_des_window (Mat2D des, Mat2D src, size_t is, size_t js, size_t ie, size_t je)

  Copy `src` into a window of `des`.
- void mat2D_copy_src_window_to_des (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_cross (Mat2D dst, Mat2D v1, Mat2D v2)

  3D cross product: dst = a x b for 3x1 vectors.
- void mat2D_dot (Mat2D dst, Mat2D a, Mat2D b)

  *Matrix product: dst = a * b.*
- double mat2D_dot_product (Mat2D v1, Mat2D v2)

  Dot product between two vectors.
- double mat2D_det (Mat2D m)

  Determinant of a square matrix via Gaussian elimination.
- double mat2D_det_2x2_mat (Mat2D m)

  Determinant of a 2x2 matrix.
- double mat2D_det_2x2_mat_minor (Mat2D_Minor mm)

  Determinant of a 2x2 minor.
- void mat2D_eig_check (Mat2D A, Mat2D eigenvalues, Mat2D eigenvectors, Mat2D res)

  Check an eigen-decomposition by forming the residual (A V - V \Lambda).
- void mat2D_eig_power_iteration (Mat2D A, Mat2D eigenvalues, Mat2D eigenvectors, Mat2D init_vector, bool norm_inf_vectors)

  Estimate eigenvalues/eigenvectors using repeated power iteration with deflation.
- 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.
- bool mat2D_find_first_non_zero_value (Mat2D m, size_t r, size_t *non_zero_col)

  Find the first non-zero (per MAT2D_EPS) element in a row.
- 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.
- 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_get_row (Mat2D des, size_t des_row, Mat2D src, size_t src_row)

  Copy a row from src into a row of des.
- double mat2D_inner_product (Mat2D v)

Compute the inner product of a vector with itself: dot(v, v).

- void mat2D_invert (Mat2D des, Mat2D src)

  Invert a square matrix using Gauss-Jordan 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.
- bool mat2D_mat_is_all_digit (Mat2D m, double digit)

  Check if all elements of a matrix equal a given digit.
- 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.
- double mat2D_minor_det (Mat2D_Minor mm)

  Determinant of a minor via recursive expansion by minors.
- 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.
- 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.
- 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_outer_product (Mat2D des, Mat2D v)

  Compute the outer product of a vector with itself: des = v ∗ v^T.
- int mat2D_power_iterate (Mat2D A, Mat2D v, double ∗lambda, double shift, bool norm_inf_v)

  Approximate an eigenpair using (shifted) power iteration.
- 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_rand (Mat2D m, double low, double high)

  Fill a matrix with pseudo-random doubles in [low, high].
- double mat2D_rand_double (void)

  Return a pseudo-random double in the range [0, 1].
- size_t mat2D_reduce (Mat2D m)

  Reduce a matrix in-place to reduced row echelon form (RREF) and return its rank.
- bool mat2D_row_is_all_digit (Mat2D m, double digit, size_t r)

  Check if all elements of a row equal a given digit.
- 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_set_identity (Mat2D m)

  Set a square matrix to the identity matrix.
- 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_shift (Mat2D m, double shift)

  Add a scalar shift to the diagonal: m[i,i] += shift.
- 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.
- void mat2D_sub (Mat2D dst, Mat2D a)

  In-place subtraction: dst -= a.
- 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_sub_row_to_row (Mat2D des, size_t des_row, Mat2D src, size_t src_row)

  Subtract a source row from a destination 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).
- void mat2D_swap_rows (Mat2D m, size_t r1, size_t r2)

  Swap two rows of a matrix in-place.
- void mat2D_transpose (Mat2D des, Mat2D src)

  Transpose a matrix: des = src^T.
- double mat2D_upper_triangulate (Mat2D m)

  Transform a matrix to (row-echelon) upper triangular form by forward elimination.

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_dprintDOUBLE

```
#define mat2D_dprintDOUBLE(
              expr ) printf(#expr " = %#g\n", expr)
```

Definition at line 243 of file Matrix2D.h.

4.1.2.5 mat2D_dprintINT

```
#define mat2D_dprintINT(
              expr ) printf(#expr " = %d\n", expr)
```

Definition at line 247 of file Matrix2D.h.

4.1.2.6 mat2D_dprintSIZE_T

```
#define mat2D_dprintSIZE_T(
            expr ) printf(#expr " = %zu\n", expr)
```

Definition at line 245 of file Matrix2D.h.

4.1.2.7 MAT2D_EPS

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

Definition at line 188 of file Matrix2D.h.

4.1.2.8 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.

4.1.2.9 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 200 of file Matrix2D.h.

4.1.2.10 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.

4.1.2.11 MAT2D_MAX_POWER_ITERATION

```
#define MAT2D_MAX_POWER_ITERATION 100
```

Definition at line 190 of file Matrix2D.h.

4.1.2.12 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 209 of file Matrix2D.h.

4.1.2.13 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 227 of file Matrix2D.h.

4.1.2.14 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 239 of file Matrix2D.h.

4.1.2.15 mat2D_normalize_inf

```
#define mat2D_normalize_inf(
            m ) mat2D_mult((m), 1.0 / mat2D_calc_norma_inf((m)))
```

Definition at line 241 of file Matrix2D.h.

4.1.2.16 MAT2D_PI

```
#define MAT2D_PI 3.14159265358979323846
```

Definition at line 186 of file Matrix2D.h.

4.1.2.17 MAT2D_PRINT

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

Convenience macro to print a matrix with its variable name.

Definition at line 215 of file Matrix2D.h.

4.1.2.18 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 221 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 341 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 362 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 406 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.

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.

Precondition

des.cols == src.cols

Definition at line 386 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 425 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_eig_check(), mat2D_eig_power_iteration(), mat2D_invert(), mat2D_power_iterate(), 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 449 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 466 of file Matrix2D.h.

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

Referenced by mat2D_power_iterate().

4.1.3.8 mat2D_calc_norma_inf()

```
double mat2D_calc_norma_inf (
            Mat2D m )
```

Compute the maximum absolute element value of a matrix.

Parameters

m	Matrix.

Returns

The element-wise maximum: (\max_{i,j} |m_{ij}|).

Note

Despite the name, this is not the induced matrix infinity norm (maximum row sum). It is the max-absolute-entry metric.

Definition at line 487 of file Matrix2D.h.

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

Referenced by mat2D_eig_power_iteration(), and mat2D_power_iterate().

4.1.3.9 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 511 of file Matrix2D.h.

References MAT2D_AT, and Mat2D::rows.

Referenced by mat2D_det().

4.1.3.10 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 529 of file Matrix2D.h.

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

Referenced by mat2D_det(), mat2D_eig_check(), mat2D_eig_power_iteration(), mat2D_LUP_decomposition_with_swap(), and mat2D_power_iterate().

4.1.3.11 mat2D_copy_src_to_des_window()

```
void mat2D_copy_src_to_des_window (
            Mat2D des,
            Mat2D src,
            size_t is,
            size_t js,
            size_t ie,
            size_t je )
```

Copy `src` into a window of `des`.

Copies the entire `src` matrix into `des` at the rectangular region: rows [`is`, `ie`] and columns [`js`, `je`] (inclusive).

Parameters

des	Destination matrix.
src	Source matrix copied into the destination window.
is	Start row index in destination (inclusive).
js	Start column index in destination (inclusive).
ie	End row index in destination (inclusive).
je	End column index in destination (inclusive).

Precondition

(je - js + 1) == src.cols and (ie - is + 1) == src.rows.

Definition at line 557 of file Matrix2D.h.

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

Referenced by mat2D_eig_power_iteration(), and mat2D_invert().

4.1.3.12 mat2D_copy_src_window_to_des()

```
void mat2D_copy_src_window_to_des (
            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 582 of file Matrix2D.h.

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

Referenced by mat2D_invert().

4.1.3.13 mat2D_cross()

```
void mat2D_cross (
            Mat2D dst,
            Mat2D v1,
            Mat2D v2 )
```

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 604 of file Matrix2D.h.

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

4.1.3.14 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 693 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 main().

4.1.3.15 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 745 of file Matrix2D.h.

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

4.1.3.16 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 756 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().

4.1.3.17 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 630 of file Matrix2D.h.

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

Referenced by mat2D_eig_check(), mat2D_power_iterate(), mat2D_set_DCM_zyx(), and mat2D_solve_linear_sys_LUP_decompositi

4.1.3.18 mat2D_dot_product()

```
double mat2D_dot_product (
            Mat2D v1,
            Mat2D v2 )
```

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 659 of file Matrix2D.h.

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

Referenced by mat2D_power_iterate().

4.1.3.19 mat2D_eig_check()

```
void mat2D_eig_check (
            Mat2D A,
            Mat2D eigenvalues,
            Mat2D eigenvectors,
            Mat2D res )
```

Check an eigen-decomposition by forming the residual (A V - V \Lambda).

Parameters

A	Square matrix (N x N).
eigenvalues	Diagonal matrix (\Lambda) (N x N).
eigenvectors	Matrix of eigenvectors (V) (N x N), typically with eigenvectors stored as columns.
res	Destination matrix (N x N) receiving the residual.

Postcondition

`res` is overwritten with (A V - V \Lambda).

Precondition

All inputs are N x N and shapes match.

Definition at line 775 of file Matrix2D.h.

References Mat2D::cols, mat2D_alloc(), MAT2D_ASSERT, MAT2D_AT, mat2D_copy(), mat2D_dot(), mat2D_free(), mat2D_mult(), mat2D_sub(), Mat2D::rows, and Mat2D::stride_r.

Referenced by main().

4.1.3.20 mat2D_eig_power_iteration()

```
void mat2D_eig_power_iteration (
            Mat2D A,
            Mat2D eigenvalues,
            Mat2D eigenvectors,
            Mat2D init_vector,
            bool norm_inf_vectors )
```

Estimate eigenvalues/eigenvectors using repeated power iteration with deflation.

Repeatedly applies mat2D_power_iterate() to estimate an eigenpair of the current matrix B, stores it into `eigenvalues` and `eigenvectors`, then deflates B by subtracting (\lambda v v^T).

The vector `init_vector` is copied into each eigenvector slot as the initial guess before running iteration.

Parameters

	A	Input square matrix (N x N).
out	*eigenvalues*	Destination (N x N) written as a diagonal matrix.
out	*eigenvectors*	Destination (N x N) whose columns are the estimated eigenvectors.
	init_vector	Initial guess (N x 1), must have non-zero norm.
	norm_inf_vectors	If true, each output eigenvector column is normalized by mat2D_normalize_inf().

Warning

This implementation is primarily educational and makes strong assumptions; it may fail or be inaccurate for matrices that do not satisfy the power-iteration convergence conditions.

Precondition

A is square; eigenvalues/eigenvectors are N x N; init_vector is N x 1.

Conditions:

- The eigenvectors must form an orthonormal basis
- The largest eigenvalue must be positive and unique

Definition at line 840 of file Matrix2D.h.

References Mat2D::cols, mat2D_alloc(), MAT2D_ASSERT, MAT2D_AT, mat2D_calc_norma_inf(), mat2D_copy(), mat2D_copy_src_to_des_window(), mat2D_free(), mat2D_mult(), mat2D_normalize_inf, mat2D_outer_product(), mat2D_power_iterate(), mat2D_set_identity(), mat2D_sub(), Mat2D::rows, and Mat2D::stride_r.

Referenced by main().

4.1.3.21 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 900 of file Matrix2D.h.

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

Referenced by mat2D_LUP_decomposition_with_swap(), and mat2D_solve_linear_sys_LUP_decomposition().

4.1.3.22 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 916 of file Matrix2D.h.

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

4.1.3.23 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 929 of file Matrix2D.h.

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

4.1.3.24 mat2D_find_first_non_zero_value()

```
bool mat2D_find_first_non_zero_value (
            Mat2D m,
```

```
            size_t r,
            size_t * non_zero_col )
```

Find the first non-zero (per MAT2D_EPS) element in a row.

Parameters

	m	Matrix to search.
	r	Row index to search (0-based).
out	*non_zero_col*	On success, receives the column index of the first element in row `r` such that !MAT2D_IS_ZERO(value).

Returns

true if a non-zero element was found, false if the row is all zeros (within MAT2D_EPS).

Note

Scans columns from 0 to m.cols-1 (left to right).

Definition at line 950 of file Matrix2D.h.

References Mat2D::cols, MAT2D_AT, and MAT2D_IS_ZERO.

Referenced by mat2D_reduce().

4.1.3.25 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 969 of file Matrix2D.h.

References Mat2D::elements, and MAT2D_FREE.

Referenced by main(), mat2D_det(), mat2D_eig_check(), mat2D_eig_power_iteration(), mat2D_invert(), mat2D_power_iterate(), mat2D_set_DCM_zyx(), and mat2D_solve_linear_sys_LUP_decomposition().

4.1.3.26 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 982 of file Matrix2D.h.

References Mat2D_uint32::elements, and MAT2D_FREE.

4.1.3.27 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 997 of file Matrix2D.h.

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

4.1.3.28 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 1017 of file Matrix2D.h.

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

4.1.3.29 mat2D_inner_product()

```
double mat2D_inner_product (
            Mat2D v )
```

Compute the inner product of a vector with itself: dot(v, v).

Parameters

v	Vector (shape n x 1 or 1 x n).

Returns

(\sum_k v_k^2) (the squared Euclidean norm).

Precondition

v.cols == 1 || v.rows == 1

Definition at line 1036 of file Matrix2D.h.

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

4.1.3.30 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.

Forms an augmented matrix [src | I], performs Gauss-Jordan style reduction in-place (via mat2D_reduce()), and then copies the right half into `des`.

Warning

This routine does not explicitly detect singular matrices. If `src` is singular (or nearly singular), mat2D_reduce() may assert on a near-zero pivot or produce unstable/undefined results.

Definition at line 1072 of file Matrix2D.h.

References Mat2D::cols, mat2D_alloc(), MAT2D_ASSERT, mat2D_copy_src_to_des_window(), mat2D_copy_src_window_to_des(), mat2D_free(), mat2D_reduce(), mat2D_set_identity(), and Mat2D::rows.

Referenced by mat2D_solve_linear_sys_LUP_decomposition().

4.1.3.31 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 1106 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().

4.1.3.32 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 1151 of file Matrix2D.h.

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

4.1.3.33 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 1174 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().

4.1.3.34 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 1216 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().

4.1.3.35 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 1253 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().

4.1.3.36 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 1279 of file Matrix2D.h.

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

Referenced by mat2D_det(), and mat2D_minor_det().

4.1.3.37 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 1291 of file Matrix2D.h.

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

4.1.3.38 mat2D_mult()

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

In-place scalar multiplication: m *= factor.

Parameters

m	Matrix.
factor	Scalar multiplier.

Definition at line 1309 of file Matrix2D.h.

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

Referenced by mat2D_eig_check(), mat2D_eig_power_iteration(), and mat2D_power_iterate().

4.1.3.39 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 1326 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_reduce().

4.1.3.40 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 1343 of file Matrix2D.h.

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

Referenced by mat2D_fill_sequence().

4.1.3.41 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 1359 of file Matrix2D.h.

References Mat2D_uint32::cols, MAT2D_ASSERT, Mat2D_uint32::rows, and Mat2D_uint32::stride_r.

4.1.3.42 mat2D_outer_product()

```
void mat2D_outer_product (
            Mat2D des,
            Mat2D v )
```

Compute the outer product of a vector with itself: des = v ∗ v^T.

Parameters

des	Destination square matrix (n x n).
v	Vector (shape n x 1 or 1 x n).

Postcondition

`des` is fully overwritten with (v v^T).

Precondition

des.rows == des.cols

(v.cols == 1 && des.rows == v.rows) || (v.rows == 1 && des.cols == v.cols)

Definition at line 1376 of file Matrix2D.h.

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

Referenced by mat2D_eig_power_iteration().

4.1.3.43 mat2D_power_iterate()

```
int mat2D_power_iterate (
            Mat2D A,
            Mat2D v,
            double * lambda,
            double shift,
            bool norm_inf_v )
```

Approximate an eigenpair using (shifted) power iteration.

Runs power iteration on the shifted matrix (B = A - \text{shift} \cdot I). The input/output vector `v` is iteratively updated (in-place) and normalized. An eigenvalue estimate is written to `lambda` (if non-NULL) as: (\lambda \approx \lambda(B) + \text{shift}).

Parameters

	A	Square matrix (N x N).
`in,out`	*v*	Initial guess vector (N x 1). Overwritten with the estimated dominant eigenvector of the shifted matrix.
`out`	*lambda*	Optional output for the eigenvalue estimate (may be NULL).
	shift	Diagonal shift applied as described above.
	norm_inf_v	If true, normalize `v` at the end by mat2D_normalize_inf().

Return values

0	Converged (difference below MAT2D_EPS within MAT2D_MAX_POWER_ITERATION).
1	Did not converge (often corresponds to sign-flip/alternation behavior).

Precondition

A is square and v has shape (A.rows x 1).

Conditions:

- The eigenvectors must form a basis
- The largest eigenvalue must be positive and unique

Definition at line 1419 of file Matrix2D.h.

References Mat2D::cols, mat2D_alloc(), MAT2D_ASSERT, mat2D_calc_norma(), mat2D_calc_norma_inf(), mat2D_copy(), mat2D_dot(), mat2D_dot_product(), MAT2D_EPS, mat2D_free(), MAT2D_MAX_POWER_ITERATION, mat2D_mult(), mat2D_normalize, mat2D_normalize_inf, mat2D_shift(), mat2D_sub(), and Mat2D::rows.

Referenced by mat2D_eig_power_iteration().

4.1.3.44 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 1482 of file Matrix2D.h.

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

4.1.3.45 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 1501 of file Matrix2D.h.

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

4.1.3.46 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 1521 of file Matrix2D.h.

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

Referenced by main().

4.1.3.47 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 1539 of file Matrix2D.h.

Referenced by mat2D_rand().

4.1.3.48 mat2D_reduce()

```
size_t mat2D_reduce (
            Mat2D m )
```

Reduce a matrix in-place to reduced row echelon form (RREF) and return its rank.

Parameters

m	Matrix modified in-place.

Returns

The computed rank (number of pivot rows found).

Internally calls mat2D_upper_triangulate() and then performs backward elimination and row scaling to produce a reduced row echelon form.

Note

When used on an augmented matrix (e.g. [A | I]), this can be used as part of Gauss-Jordan inversion when A is nonsingular.

Definition at line 1557 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, mat2D_find_first_non_zero_value(), MAT2D_IS_ZERO, mat2D_mult_row(), mat2D_sub_row_time_factor_to_row(), mat2D_upper_triangulate(), and Mat2D::rows.

Referenced by mat2D_invert().

4.1.3.49 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 1595 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_det().

4.1.3.50 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 1616 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().

4.1.3.51 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 1641 of file Matrix2D.h.

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

Referenced by mat2D_eig_power_iteration(), mat2D_invert(), mat2D_LUP_decomposition_with_swap(), mat2D_set_rot_mat_x(), mat2D_set_rot_mat_y(), and mat2D_set_rot_mat_z().

4.1.3.52 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 1669 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().

4.1.3.53 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 1693 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().

4.1.3.54 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 1717 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().

4.1.3.55 mat2D_shift()

```
void mat2D_shift (
            Mat2D m,
            double shift )
```

Add a scalar shift to the diagonal: m[i,i] += shift.

Parameters

m	Square matrix modified in-place.
shift	Value added to each diagonal element.

Precondition

m.rows == m.cols

Definition at line 1737 of file Matrix2D.h.

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

Referenced by mat2D_power_iterate().

4.1.3.56 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 1761 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.

4.1.3.57 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 1803 of file Matrix2D.h.

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

Referenced by mat2D_eig_check(), mat2D_eig_power_iteration(), and mat2D_power_iterate().

4.1.3.58 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 1824 of file Matrix2D.h.

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

4.1.3.59 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 1865 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_LUP_decomposition_with_swap(), mat2D_reduce(), and mat2D_upper_triangulate().

4.1.3.60 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 1845 of file Matrix2D.h.

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

4.1.3.61 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 1880 of file Matrix2D.h.

References Mat2D::cols, and MAT2D_AT.

Referenced by mat2D_LUP_decomposition_with_swap(), and mat2D_upper_triangulate().

4.1.3.62 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 1896 of file Matrix2D.h.

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

4.1.3.63 mat2D_upper_triangulate()

```
double mat2D_upper_triangulate (
            Mat2D m )
```

Transform a matrix to (row-echelon) upper triangular form by forward elimination.

Parameters

m	Matrix transformed in-place.

Returns

A determinant sign factor caused by row swaps: +1.0 or -1.0.

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. Performs partial pivoting by selecting the row with the largest absolute pivot candidate in each column. Uses elimination operations that (in exact arithmetic) do not change the determinant. Each row swap flips the determinant sign; the cumulative sign is returned.

Warning

Not robust for linearly dependent rows or very small pivots.

Definition at line 1926 of file Matrix2D.h.

References Mat2D::cols, MAT2D_ASSERT, MAT2D_AT, MAT2D_IS_ZERO, mat2D_sub_row_time_factor_to_row(), mat2D_swap_rows(), and Mat2D::rows.

Referenced by mat2D_det(), and mat2D_reduce().

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 #define MAT2D_PI 3.14159265358979323846
00187
00188 #define MAT2D_EPS 1e-15
00189
00190 #define MAT2D_MAX_POWER_ITERATION 100
00191
00192
00200 #define MAT2D_IS_ZERO(x) (fabs(x) < MAT2D_EPS)
00201
00209 #define MAT2D_MINOR_AT(mm, i, j) MAT2D_AT((mm).ref_mat, (mm).rows_list[i], (mm).cols_list[j])
00210
00215 #define MAT2D_PRINT(m) mat2D_print(m, #m, 0)
00216
00221 #define MAT2D_PRINT_AS_COL(m) mat2D_print_as_col(m, #m, 0)
00222
00227 #define MAT2D_MINOR_PRINT(mm) mat2D_minor_print(mm, #mm, 0)
00228
00239 #define mat2D_normalize(m) mat2D_mult((m), 1.0 / mat2D_calc_norma((m)))
00240
00241 #define mat2D_normalize_inf(m) mat2D_mult((m), 1.0 / mat2D_calc_norma_inf((m)))
00242
00243 #define mat2D_dprintDOUBLE(expr) printf(#expr " = %#g\n", expr)
00244
00245 #define mat2D_dprintSIZE_T(expr) printf(#expr " = %zu\n", expr)
00246
00247 #define mat2D_dprintINT(expr) printf(#expr " = %d\n", expr)
00248
00249 void            mat2D_add(Mat2D dst, Mat2D a);
00250 void            mat2D_add_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00251 void            mat2D_add_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
```

```
00252 void            mat2D_add_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor);
00253 Mat2D          mat2D_alloc(size_t rows, size_t cols);
00254 Mat2D_uint32   mat2D_alloc_uint32(size_t rows, size_t cols);
00255
00256 double         mat2D_calc_norma(Mat2D m);
00257 double         mat2D_calc_norma_inf(Mat2D m);
00258 bool           mat2D_col_is_all_digit(Mat2D m, double digit, size_t c);
00259 void           mat2D_copy(Mat2D des, Mat2D src);
00260 void           mat2D_copy_src_to_des_window(Mat2D des, Mat2D src, size_t is, size_t js, size_t ie,
       size_t je);
00261 void           mat2D_copy_src_window_to_des(Mat2D des, Mat2D src, size_t is, size_t js, size_t ie,
       size_t je);
00262 void           mat2D_cross(Mat2D dst, Mat2D v1, Mat2D v2);
00263
00264 void           mat2D_dot(Mat2D dst, Mat2D a, Mat2D b);
00265 double         mat2D_dot_product(Mat2D v1, Mat2D v2);
00266 double         mat2D_det(Mat2D m);
00267 double         mat2D_det_2x2_mat(Mat2D m);
00268 double         mat2D_det_2x2_mat_minor(Mat2D_Minor mm);
00269
00270 void           mat2D_eig_check(Mat2D A, Mat2D eigenvalues, Mat2D eigenvectors, Mat2D res);
00271 void           mat2D_eig_power_iteration(Mat2D A, Mat2D eigenvalues, Mat2D eigenvectors, Mat2D
       init_vector, bool norm_inf_vectors);
00272
00273 void           mat2D_fill(Mat2D m, double x);
00274 void           mat2D_fill_sequence(Mat2D m, double start, double step);
00275 void           mat2D_fill_uint32(Mat2D_uint32 m, uint32_t x);
00276 bool           mat2D_find_first_non_zero_value(Mat2D m, size_t r, size_t *non_zero_col);
00277 void           mat2D_free(Mat2D m);
00278 void           mat2D_free_uint32(Mat2D_uint32 m);
00279
00280 void           mat2D_get_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00281 void           mat2D_get_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
00282
00283 double         mat2D_inner_product(Mat2D v);
00284 void           mat2D_invert(Mat2D des, Mat2D src);
00285
00286 void           mat2D_LUP_decomposition_with_swap(Mat2D src, Mat2D l, Mat2D p, Mat2D u);
00287
00288 bool           mat2D_mat_is_all_digit(Mat2D m, double digit);
00289 Mat2D_Minor    mat2D_minor_alloc_fill_from_mat(Mat2D ref_mat, size_t i, size_t j);
00290 Mat2D_Minor    mat2D_minor_alloc_fill_from_mat_minor(Mat2D_Minor ref_mm, size_t i, size_t j);
00291 double         mat2D_minor_det(Mat2D_Minor mm);
00292 void           mat2D_minor_free(Mat2D_Minor mm);
00293 void           mat2D_minor_print(Mat2D_Minor mm, const char *name, size_t padding);
00294 void           mat2D_mult(Mat2D m, double factor);
00295 void           mat2D_mult_row(Mat2D m, size_t r, double factor);
00296
00297 size_t         mat2D_offset2d(Mat2D m, size_t i, size_t j);
00298 size_t         mat2D_offset2d_uint32(Mat2D_uint32 m, size_t i, size_t j);
00299 void           mat2D_outer_product(Mat2D des, Mat2D v);
00300
00301 int            mat2D_power_iterate(Mat2D A, Mat2D v, double *lambda, double shift, bool norm_inf_v);
00302 void           mat2D_print(Mat2D m, const char *name, size_t padding);
00303 void           mat2D_print_as_col(Mat2D m, const char *name, size_t padding);
00304
00305 void           mat2D_rand(Mat2D m, double low, double high);
00306 double         mat2D_rand_double(void);
00307 size_t         mat2D_reduce(Mat2D m);
00308 bool           mat2D_row_is_all_digit(Mat2D m, double digit, size_t r);
00309
00310 void           mat2D_set_DCM_zyx(Mat2D DCM, float yaw_deg, float pitch_deg, float roll_deg);
00311 void           mat2D_set_identity(Mat2D m);
00312 void           mat2D_set_rot_mat_x(Mat2D m, float angle_deg);
00313 void           mat2D_set_rot_mat_y(Mat2D m, float angle_deg);
00314 void           mat2D_set_rot_mat_z(Mat2D m, float angle_deg);
00315 void           mat2D_shift(Mat2D m, double shift);
00316 void           mat2D_solve_linear_sys_LUP_decomposition(Mat2D A, Mat2D x, Mat2D B);
00317 void           mat2D_sub(Mat2D dst, Mat2D a);
00318 void           mat2D_sub_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col);
00319 void           mat2D_sub_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row);
00320 void           mat2D_sub_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor);
00321 void           mat2D_swap_rows(Mat2D m, size_t r1, size_t r2);
00322
00323 void           mat2D_transpose(Mat2D des, Mat2D src);
00324
00325 double         mat2D_upper_triangulate(Mat2D m);
00326
00327 #endif // MATRIX2D_H_
00328
00329 #ifdef MATRIX2D_IMPLEMENTATION
00330 #undef MATRIX2D_IMPLEMENTATION
00331
00332
00341 void mat2D_add(Mat2D dst, Mat2D a)
00342 {
00343     MAT2D_ASSERT(dst.rows == a.rows);
```

```
00344     MAT2D_ASSERT(dst.cols == a.cols);
00345     for (size_t i = 0; i < dst.rows; ++i) {
00346         for (size_t j = 0; j < dst.cols; ++j) {
00347             MAT2D_AT(dst, i, j) += MAT2D_AT(a, i, j);
00348         }
00349     }
00350 }
00351
00362 void mat2D_add_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
00363 {
00364     MAT2D_ASSERT(src_col < src.cols);
00365     MAT2D_ASSERT(des.rows == src.rows);
00366     MAT2D_ASSERT(des_col < des.cols);
00367
00368     for (size_t i = 0; i < des.rows; i++) {
00369         MAT2D_AT(des, i, des_col) += MAT2D_AT(src, i, src_col);
00370     }
00371 }
00372
00386 void mat2D_add_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
00387 {
00388     MAT2D_ASSERT(src_row < src.rows);
00389     MAT2D_ASSERT(des.cols == src.cols);
00390     MAT2D_ASSERT(des_row < des.rows);
00391
00392     for (size_t j = 0; j < des.cols; j++) {
00393         MAT2D_AT(des, des_row, j) += MAT2D_AT(src, src_row, j);
00394     }
00395 }
00396
00406 void mat2D_add_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor)
00407 {
00408     for (size_t j = 0; j < m.cols; ++j) {
00409         MAT2D_AT(m, des_r, j) += factor * MAT2D_AT(m, src_r, j);
00410     }
00411 }
00412
00425 Mat2D mat2D_alloc(size_t rows, size_t cols)
00426 {
00427     Mat2D m;
00428     m.rows = rows;
00429     m.cols = cols;
00430     m.stride_r = cols;
00431     m.elements = (double*)MAT2D_MALLOC(sizeof(double)*rows*cols);
00432     MAT2D_ASSERT(m.elements != NULL);
00433
00434     return m;
00435 }
00436
00449 Mat2D_uint32 mat2D_alloc_uint32(size_t rows, size_t cols)
00450 {
00451     Mat2D_uint32 m;
00452     m.rows = rows;
00453     m.cols = cols;
00454     m.stride_r = cols;
00455     m.elements = (uint32_t*)MAT2D_MALLOC(sizeof(uint32_t)*rows*cols);
00456     MAT2D_ASSERT(m.elements != NULL);
00457
00458     return m;
00459 }
00460
00466 double mat2D_calc_norma(Mat2D m)
00467 {
00468     double sum = 0;
00469
00470     for (size_t i = 0; i < m.rows; ++i) {
00471         for (size_t j = 0; j < m.cols; ++j) {
00472             sum += MAT2D_AT(m, i, j) * MAT2D_AT(m, i, j);
00473         }
00474     }
00475     return sqrt(sum);
00476 }
00477
00487 double mat2D_calc_norma_inf(Mat2D m)
00488 {
00489     double max = 0;
00490     for (size_t i = 0; i < m.rows; ++i) {
00491         for (size_t j = 0; j < m.cols; ++j) {
00492             double current = fabs(MAT2D_AT(m, i, j));
00493             if (current > max) {
00494                 max = current;
00495             }
00496         }
00497     }
00498
00499     return max;
00500 }
```

```
00501
00511 bool mat2D_col_is_all_digit(Mat2D m, double digit, size_t c)
00512 {
00513     for (size_t i = 0; i < m.rows; ++i) {
00514         if (MAT2D_AT(m, i, c) != digit) {
00515             return false;
00516         }
00517     }
00518     return true;
00519 }
00520
00529 void mat2D_copy(Mat2D des, Mat2D src)
00530 {
00531     MAT2D_ASSERT(des.cols == src.cols);
00532     MAT2D_ASSERT(des.rows == src.rows);
00533
00534     for (size_t i = 0; i < des.rows; ++i) {
00535         for (size_t j = 0; j < des.cols; ++j) {
00536             MAT2D_AT(des, i, j) = MAT2D_AT(src, i, j);
00537         }
00538     }
00539 }
00540
00557 void mat2D_copy_src_to_des_window(Mat2D des, Mat2D src, size_t is, size_t js, size_t ie, size_t je)
00558 {
00559     MAT2D_ASSERT(je >= js && ie >= is);
00560     MAT2D_ASSERT(je-js+1 == src.cols);
00561     MAT2D_ASSERT(ie-is+1 == src.rows);
00562     MAT2D_ASSERT(je-js+1 <= des.cols);
00563     MAT2D_ASSERT(ie-is+1 <= des.rows);
00564
00565     for (size_t index = 0; index < src.rows; ++index) {
00566         for (size_t jndex = 0; jndex < src.cols; ++jndex) {
00567             MAT2D_AT(des, is+index, js+jndex) = MAT2D_AT(src, index, jndex);
00568         }
00569     }
00570 }
00571
00582 void mat2D_copy_src_window_to_des(Mat2D des, Mat2D src, size_t is, size_t js, size_t ie, size_t je)
00583 {
00584     MAT2D_ASSERT(je >= js && ie >= is);
00585     MAT2D_ASSERT(je-js+1 == des.cols);
00586     MAT2D_ASSERT(ie-is+1 == des.rows);
00587     MAT2D_ASSERT(je-js+1 <= src.cols);
00588     MAT2D_ASSERT(ie-is+1 <= src.rows);
00589
00590     for (size_t index = 0; index < des.rows; ++index) {
00591         for (size_t jndex = 0; jndex < des.cols; ++jndex) {
00592             MAT2D_AT(des, index, jndex) = MAT2D_AT(src, is+index, js+jndex);
00593         }
00594     }
00595 }
00596
00604 void mat2D_cross(Mat2D dst, Mat2D v1, Mat2D v2)
00605 {
00606     MAT2D_ASSERT(3 == dst.rows && 1 == dst.cols);
00607     MAT2D_ASSERT(3 == v1.rows && 1 == v1.cols);
00608     MAT2D_ASSERT(3 == v2.rows && 1 == v2.cols);
00609
00610     MAT2D_AT(dst, 0, 0) = MAT2D_AT(v1, 1, 0) * MAT2D_AT(v2, 2, 0) - MAT2D_AT(v1, 2, 0) * MAT2D_AT(v2,
       1, 0);
00611     MAT2D_AT(dst, 1, 0) = MAT2D_AT(v1, 2, 0) * MAT2D_AT(v2, 0, 0) - MAT2D_AT(v1, 0, 0) * MAT2D_AT(v2,
       2, 0);
00612     MAT2D_AT(dst, 2, 0) = MAT2D_AT(v1, 0, 0) * MAT2D_AT(v2, 1, 0) - MAT2D_AT(v1, 1, 0) * MAT2D_AT(v2,
       0, 0);
00613 }
00614
00630 void mat2D_dot(Mat2D dst, Mat2D a, Mat2D b)
00631 {
00632     MAT2D_ASSERT(a.cols == b.rows);
00633     MAT2D_ASSERT(a.rows == dst.rows);
00634     MAT2D_ASSERT(b.cols == dst.cols);
00635
00636     size_t i, j, k;
00637
00638     for (i = 0; i < dst.rows; i++) {
00639         for (j = 0; j < dst.cols; j++) {
00640             MAT2D_AT(dst, i, j) = 0;
00641             for (k = 0; k < a.cols; k++) {
00642                 MAT2D_AT(dst, i, j) += MAT2D_AT(a, i, k)*MAT2D_AT(b, k, j);
00643             }
00644         }
00645     }
00646
00647 }
00648
00659 double mat2D_dot_product(Mat2D v1, Mat2D v2)
```

```
00660 {
00661     MAT2D_ASSERT(v1.rows == v2.rows);
00662     MAT2D_ASSERT(v1.cols == v2.cols);
00663     MAT2D_ASSERT((1 == v1.cols && 1 == v2.cols) || (1 == v1.rows && 1 == v2.rows));
00664 
00665     double dot_product = 0;
00666 
00667     if (1 == v1.cols) {
00668         for (size_t i = 0; i < v1.rows; i++) {
00669             dot_product += MAT2D_AT(v1, i, 0) * MAT2D_AT(v2, i, 0);
00670         }
00671     } else {
00672         for (size_t j = 0; j < v1.cols; j++) {
00673             dot_product += MAT2D_AT(v1, 0, j) * MAT2D_AT(v2, 0, j);
00674         }
00675     }
00676 
00677     return dot_product;
00678 }
00679 
00693 double mat2D_det(Mat2D m)
00694 {
00695     MAT2D_ASSERT(m.cols == m.rows && "should be a square matrix");
00696 
00697     /* checking if there is a row or column with all zeros */
00698     /* checking rows */
00699     for (size_t i = 0; i < m.rows; i++) {
00700         if (mat2D_row_is_all_digit(m, 0, i)) {
00701             return 0;
00702         }
00703     }
00704     /* checking cols */
00705     for (size_t j = 0; j < m.rows; j++) {
00706         if (mat2D_col_is_all_digit(m, 0, j)) {
00707             return 0;
00708         }
00709     }
00710 
00711     #if 0/* This is an implementation of naive determinant calculation using minors. This is too slow
     */
00712     double det = 0;
00713     /* TODO: finding beast row or col? */
00714     for (size_t i = 0, j = 0; i < m.rows; i++) { /* first column */
00715         if (MAT2D_AT(m, i, j) < 1e-10) continue;
00716         Mat2D_Minor sub_mm = mat2D_minor_alloc_fill_from_mat(m, i, j);
00717         int factor = (i+j)%2 ? -1 : 1;
00718         if (sub_mm.cols != 2) {
00719             MAT2D_ASSERT(sub_mm.cols == sub_mm.rows && "should be a square matrix");
00720             det += MAT2D_AT(m, i, j) * (factor) * mat2D_minor_det(sub_mm);
00721         } else if (sub_mm.cols == 2 && sub_mm.rows == 2) {
00722             det += MAT2D_AT(m, i, j) * (factor) * mat2D_det_2x2_mat_minor(sub_mm);;
00723         }
00724         mat2D_minor_free(sub_mm);
00725     }
00726     #endif
00727 
00728     Mat2D temp_m = mat2D_alloc(m.rows, m.cols);
00729     mat2D_copy(temp_m, m);
00730     double factor = mat2D_upper_triangulate(temp_m);
00731     double diag_mul = 1;
00732     for (size_t i = 0; i < temp_m.rows; i++) {
00733         diag_mul *= MAT2D_AT(temp_m, i, i);
00734     }
00735     mat2D_free(temp_m);
00736 
00737     return diag_mul / factor;
00738 }
00739 
00745 double mat2D_det_2x2_mat(Mat2D m)
00746 {
00747     MAT2D_ASSERT(2 == m.cols && 2 == m.rows && "Not a 2x2 matrix");
00748     return MAT2D_AT(m, 0, 0) * MAT2D_AT(m, 1, 1) - MAT2D_AT(m, 0, 1) * MAT2D_AT(m, 1, 0);
00749 }
00750 
00756 double mat2D_det_2x2_mat_minor(Mat2D_Minor mm)
00757 {
00758     MAT2D_ASSERT(2 == mm.cols && 2 == mm.rows && "Not a 2x2 matrix");
00759     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);
00760 }
00761 
00775 void mat2D_eig_check(Mat2D A, Mat2D eigenvalues, Mat2D eigenvectors, Mat2D res)
00776 {
00777     MAT2D_ASSERT(A.cols == A.rows);
00778     MAT2D_ASSERT(eigenvalues.cols == A.cols);
00779     MAT2D_ASSERT(eigenvalues.rows == A.rows);
00780     MAT2D_ASSERT(eigenvectors.cols == A.cols);
```

```
00781     MAT2D_ASSERT(eigenvectors.rows == A.rows);
00782     MAT2D_ASSERT(res.cols == A.cols);
00783     MAT2D_ASSERT(res.rows == A.rows);
00784 
00785     #if 1
00786     mat2D_dot(res, A, eigenvectors);
00787     Mat2D VL = mat2D_alloc(A.rows, A.cols);
00788     mat2D_dot(VL, eigenvectors, eigenvalues);
00789 
00790     mat2D_sub(res, VL);
00791 
00792     mat2D_free(VL);
00793     #else
00794     Mat2D temp_v = mat2D_alloc(A.rows, 1);
00795     for (size_t i = 0; i < A.rows; i++) {
00796         Mat2D eig_vector = {.cols = 1,
00797                             .elements = &MAT2D_AT(eigenvectors, 0, i),
00798                             .rows = A.rows,
00799                             .stride_r = eigenvectors.stride_r};
00800         Mat2D v = {.cols = 1,
00801                    .elements = &MAT2D_AT(res, 0, i),
00802                    .rows = A.rows,
00803                    .stride_r = res.stride_r};
00804 
00805         mat2D_dot(temp_v, A, eig_vector);
00806         mat2D_copy(v, eig_vector);
00807         mat2D_mult(v, MAT2D_AT(eigenvalues, i, i ));
00808 
00809         mat2D_sub(v, temp_v);
00810     }
00811     mat2D_free(temp_v);
00812     #endif
00813 }
00814 
00840 void mat2D_eig_power_iteration(Mat2D A, Mat2D eigenvalues, Mat2D eigenvectors, Mat2D init_vector, bool
     norm_inf_vectors)
00841 {
00842     /* https://www.youtube.com/watch?v=c8DIOzuZqBs */
00843 
00849     MAT2D_ASSERT(A.cols == A.rows);
00850     MAT2D_ASSERT(eigenvalues.cols == A.cols);
00851     MAT2D_ASSERT(eigenvalues.rows == A.rows);
00852     MAT2D_ASSERT(eigenvectors.cols == A.cols);
00853     MAT2D_ASSERT(eigenvectors.rows == A.rows);
00854     MAT2D_ASSERT(init_vector.cols == 1);
00855     MAT2D_ASSERT(init_vector.rows == A.rows);
00856     MAT2D_ASSERT(mat2D_calc_norma_inf(init_vector) > 0);
00857 
00858     mat2D_set_identity(eigenvalues);
00859     Mat2D B = mat2D_alloc(A.rows, A.cols);
00860     Mat2D temp_mat = mat2D_alloc(A.rows, A.cols);
00861     mat2D_copy(B, A);
00862 
00863     for (int i = 0, shift_value = 0; i < (int)A.rows; i++) {
00864         mat2D_copy_src_to_des_window(eigenvectors, init_vector, 0, i, init_vector.rows-1, i);
00865         Mat2D v = {.cols = init_vector.cols,
00866                    .elements = &MAT2D_AT(eigenvectors, 0, i),
00867                    .rows = init_vector.rows,
00868                    .stride_r = eigenvectors.stride_r};
00869         if (mat2D_power_iterate(B, v, &MAT2D_AT(eigenvalues, i, i), shift_value, 0)) { /* norm_inf_v
     must be zero*/
00870             shift_value++;
00871             i--;
00872             continue;
00873         } else {
00874             shift_value = 0;
00875         }
00876         mat2D_outer_product(temp_mat, v);
00877         mat2D_mult(temp_mat, MAT2D_AT(eigenvalues, i, i));
00878         mat2D_sub(B, temp_mat);
00879     }
00880 
00881     if (norm_inf_vectors) {
00882         for (size_t c = 0; c < eigenvectors.cols; c++) {
00883             Mat2D v = {.cols = init_vector.cols,
00884                        .elements = &MAT2D_AT(eigenvectors, 0, c),
00885                        .rows = init_vector.rows,
00886                        .stride_r = eigenvectors.stride_r};
00887             mat2D_normalize_inf(v);
00888         }
00889     }
00890 
00891     mat2D_free(B);
00892     mat2D_free(temp_mat);
00893 }
00894 
00900 void mat2D_fill(Mat2D m, double x)
```

```
00901 {
00902     for (size_t i = 0; i < m.rows; ++i) {
00903         for (size_t j = 0; j < m.cols; ++j) {
00904             MAT2D_AT(m, i, j) = x;
00905         }
00906     }
00907 }
00908
00916 void mat2D_fill_sequence(Mat2D m, double start, double step) {
00917     for (size_t i = 0; i < m.rows; i++) {
00918         for (size_t j = 0; j < m.cols; j++) {
00919             MAT2D_AT(m, i, j) = start + step * mat2D_offset2d(m, i, j);
00920         }
00921     }
00922 }
00923
00929 void mat2D_fill_uint32(Mat2D_uint32 m, uint32_t x)
00930 {
00931     for (size_t i = 0; i < m.rows; ++i) {
00932         for (size_t j = 0; j < m.cols; ++j) {
00933             MAT2D_AT_UINT32(m, i, j) = x;
00934         }
00935     }
00936 }
00937
00950 bool mat2D_find_first_non_zero_value(Mat2D m, size_t r, size_t *non_zero_col)
00951 {
00952     for (size_t c = 0; c < m.cols; ++c) {
00953         if (!MAT2D_IS_ZERO(MAT2D_AT(m, r, c))) {
00954             *non_zero_col = c;
00955             return true;
00956         }
00957     }
00958     return false;
00959 }
00960
00969 void mat2D_free(Mat2D m)
00970 {
00971     MAT2D_FREE(m.elements);
00972 }
00973
00982 void mat2D_free_uint32(Mat2D_uint32 m)
00983 {
00984     MAT2D_FREE(m.elements);
00985 }
00986
00997 void mat2D_get_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
00998 {
00999     MAT2D_ASSERT(src_col < src.cols);
01000     MAT2D_ASSERT(des.rows == src.rows);
01001     MAT2D_ASSERT(des_col < des.cols);
01002
01003     for (size_t i = 0; i < des.rows; i++) {
01004         MAT2D_AT(des, i, des_col) = MAT2D_AT(src, i, src_col);
01005     }
01006 }
01007
01017 void mat2D_get_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
01018 {
01019     MAT2D_ASSERT(src_row < src.rows);
01020     MAT2D_ASSERT(des.cols == src.cols);
01021     MAT2D_ASSERT(des_row < des.rows);
01022
01023     for (size_t j = 0; j < des.cols; j++) {
01024         MAT2D_AT(des, des_row, j) = MAT2D_AT(src, src_row, j);
01025     }
01026 }
01027
01036 double mat2D_inner_product(Mat2D v)
01037 {
01038     MAT2D_ASSERT((1 == v.cols) || (1 == v.rows));
01039
01040     double dot_product = 0;
01041
01042     if (1 == v.cols) {
01043         for (size_t i = 0; i < v.rows; i++) {
01044             dot_product += MAT2D_AT(v, i, 0) * MAT2D_AT(v, i, 0);
01045         }
01046     } else {
01047         for (size_t j = 0; j < v.cols; j++) {
01048             dot_product += MAT2D_AT(v, 0, j) * MAT2D_AT(v, 0, j);
01049         }
01050     }
01051
01052     return dot_product;
01053 }
01054
```

```
01072 void mat2D_invert(Mat2D des, Mat2D src)
01073 {
01074     MAT2D_ASSERT(src.cols == src.rows && "Must be an NxN matrix");
01075     MAT2D_ASSERT(des.cols == src.cols && des.rows == des.cols);
01076 
01077     Mat2D m = mat2D_alloc(src.rows, src.cols * 2);
01078     mat2D_copy_src_to_des_window(m, src, 0, 0, src.rows-1, src.cols-1);
01079 
01080     mat2D_set_identity(des);
01081     mat2D_copy_src_to_des_window(m, des, 0, src.cols, des.rows-1, 2 * des.cols-1);
01082 
01083     mat2D_reduce(m);
01084 
01085     mat2D_copy_src_window_to_des(des, m, 0, src.cols, des.rows-1, 2 * des.cols-1);
01086 
01087     mat2D_free(m);
01088 }
01089 
01106 void mat2D_LUP_decomposition_with_swap(Mat2D src, Mat2D l, Mat2D p, Mat2D u)
01107 {
01108     /* performing LU decomposition Following the Wikipedia page:
      https://en.wikipedia.org/wiki/LU_decomposition */
01109 
01110     mat2D_copy(u, src);
01111     mat2D_set_identity(p);
01112     mat2D_fill(l, 0);
01113 
01114     for (size_t i = 0; i < (size_t)fmin(u.rows-1, u.cols); i++) {
01115         if (MAT2D_IS_ZERO(MAT2D_AT(u, i, i))) {   /* swapping only if it is zero */
01116             /* finding biggest first number (absolute value) */
01117             size_t biggest_r = i;
01118             for (size_t index = i; index < u.rows; index++) {
01119                 if (fabs(MAT2D_AT(u, index, i)) > fabs(MAT2D_AT(u, biggest_r, i))) {
01120                     biggest_r = index;
01121                 }
01122             }
01123             if (i != biggest_r) {
01124                 mat2D_swap_rows(u, i, biggest_r);
01125                 mat2D_swap_rows(p, i, biggest_r);
01126                 mat2D_swap_rows(l, i, biggest_r);
01127             }
01128         }
01129         for (size_t j = i+1; j < u.cols; j++) {
01130             double factor = 1 / MAT2D_AT(u, i, i);
01131             if (!isfinite(factor)) {
01132                 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__);
01133             }
01134             double mat_value = MAT2D_AT(u, j, i);
01135             mat2D_sub_row_time_factor_to_row(u, j, i, mat_value * factor);
01136             MAT2D_AT(l, j, i) = mat_value * factor;
01137         }
01138         MAT2D_AT(l, i, i) = 1;
01139     }
01140     MAT2D_AT(l, l.rows-1, l.cols-1) = 1;
01141 }
01142 
01151 bool mat2D_mat_is_all_digit(Mat2D m, double digit)
01152 {
01153     for (size_t i = 0; i < m.rows; ++i) {
01154         for (size_t j = 0; j < m.cols; ++j) {
01155             if (MAT2D_AT(m, i, j) != digit) {
01156                 return false;
01157             }
01158         }
01159     }
01160     return true;
01161 }
01162 
01174 Mat2D_Minor mat2D_minor_alloc_fill_from_mat(Mat2D ref_mat, size_t i, size_t j)
01175 {
01176     MAT2D_ASSERT(ref_mat.cols == ref_mat.rows && "minor is defined only for square matrix");
01177 
01178     Mat2D_Minor mm;
01179     mm.cols = ref_mat.cols-1;
01180     mm.rows = ref_mat.rows-1;
01181     mm.stride_r = ref_mat.cols-1;
01182     mm.cols_list = (size_t*)MAT2D_MALLOC(sizeof(size_t)*(ref_mat.cols-1));
01183     mm.rows_list = (size_t*)MAT2D_MALLOC(sizeof(size_t)*(ref_mat.rows-1));
01184     mm.ref_mat = ref_mat;
01185 
01186     MAT2D_ASSERT(mm.cols_list != NULL && mm.rows_list != NULL);
01187 
01188     for (size_t index = 0, temp_index = 0; index < ref_mat.rows; index++) {
01189         if (index != i) {
01190             mm.rows_list[temp_index] = index;
01191             temp_index++;
```

```
01192         }
01193     }
01194     for (size_t jndex = 0, temp_jndex = 0; jndex < ref_mat.cols; jndex++) {
01195         if (jndex != j) {
01196             mm.cols_list[temp_jndex] = jndex;
01197             temp_jndex++;
01198         }
01199     }
01200
01201     return mm;
01202 }
01203
01216 Mat2D_Minor mat2D_minor_alloc_fill_from_mat_minor(Mat2D_Minor ref_mm, size_t i, size_t j)
01217 {
01218     MAT2D_ASSERT(ref_mm.cols == ref_mm.rows && "minor is defined only for square matrix");
01219
01220     Mat2D_Minor mm;
01221     mm.cols = ref_mm.cols-1;
01222     mm.rows = ref_mm.rows-1;
01223     mm.stride_r = ref_mm.cols-1;
01224     mm.cols_list = (size_t*)MAT2D_MALLOC(sizeof(size_t)*(ref_mm.cols-1));
01225     mm.rows_list = (size_t*)MAT2D_MALLOC(sizeof(size_t)*(ref_mm.rows-1));
01226     mm.ref_mat = ref_mm.ref_mat;
01227
01228     MAT2D_ASSERT(mm.cols_list != NULL && mm.rows_list != NULL);
01229
01230     for (size_t index = 0, temp_index = 0; index < ref_mm.rows; index++) {
01231         if (index != i) {
01232             mm.rows_list[temp_index] = ref_mm.rows_list[index];
01233             temp_index++;
01234         }
01235     }
01236     for (size_t jndex = 0, temp_jndex = 0; jndex < ref_mm.cols; jndex++) {
01237         if (jndex != j) {
01238             mm.cols_list[temp_jndex] = ref_mm.cols_list[jndex];
01239             temp_jndex++;
01240         }
01241     }
01242
01243     return mm;
01244 }
01245
01253 double mat2D_minor_det(Mat2D_Minor mm)
01254 {
01255     MAT2D_ASSERT(mm.cols == mm.rows && "should be a square matrix");
01256
01257     double det = 0;
01258     /* TODO: finding beast row or col? */
01259     for (size_t i = 0, j = 0; i < mm.rows; i++) { /* first column */
01260         if (fabs(MAT2D_MINOR_AT(mm, i, j)) < 1e-10) continue;
01261         Mat2D_Minor sub_mm = mat2D_minor_alloc_fill_from_mat_minor(mm, i, j);
01262         int factor = (i+j)%2 ? -1 : 1;
01263         if (sub_mm.cols != 2) {
01264             MAT2D_ASSERT(sub_mm.cols == sub_mm.rows && "should be a square matrix");
01265             det += MAT2D_MINOR_AT(mm, i, j) * (factor) * mat2D_minor_det(sub_mm);
01266         } else if (sub_mm.cols == 2 && sub_mm.rows == 2) {
01267             det += MAT2D_MINOR_AT(mm, i, j) * (factor) * mat2D_det_2x2_mat_minor(sub_mm);;
01268         }
01269         mat2D_minor_free(sub_mm);
01270     }
01271     return det;
01272 }
01273
01279 void mat2D_minor_free(Mat2D_Minor mm)
01280 {
01281     MAT2D_FREE(mm.cols_list);
01282     MAT2D_FREE(mm.rows_list);
01283 }
01284
01291 void mat2D_minor_print(Mat2D_Minor mm, const char *name, size_t padding)
01292 {
01293     printf("%*s%s = [\n", (int) padding, "", name);
01294     for (size_t i = 0; i < mm.rows; ++i) {
01295         printf("%*s    ", (int) padding, "");
01296         for (size_t j = 0; j < mm.cols; ++j) {
01297             printf("%f ", MAT2D_MINOR_AT(mm, i, j));
01298         }
01299         printf("\n");
01300     }
01301     printf("%*s]\n", (int) padding, "");
01302 }
01303
01309 void mat2D_mult(Mat2D m, double factor)
01310 {
01311     for (size_t i = 0; i < m.rows; ++i) {
01312         for (size_t j = 0; j < m.cols; ++j) {
01313             MAT2D_AT(m, i, j) *= factor;
```

```
01314         }
01315     }
01316 }
01317
01326 void mat2D_mult_row(Mat2D m, size_t r, double factor)
01327 {
01328     for (size_t j = 0; j < m.cols; ++j) {
01329         MAT2D_AT(m, r, j) *= factor;
01330     }
01331 }
01332
01343 size_t mat2D_offset2d(Mat2D m, size_t i, size_t j)
01344 {
01345     MAT2D_ASSERT(i < m.rows && j < m.cols);
01346     return i * m.stride_r + j;
01347 }
01348
01359 size_t mat2D_offset2d_uint32(Mat2D_uint32 m, size_t i, size_t j)
01360 {
01361     MAT2D_ASSERT(i < m.rows && j < m.cols);
01362     return i * m.stride_r + j;
01363 }
01364
01376 void mat2D_outer_product(Mat2D des, Mat2D v)
01377 {
01378     MAT2D_ASSERT(des.cols == des.rows);
01379     MAT2D_ASSERT((1 == v.cols && des.rows == v.rows) || (1 == v.rows && des.cols == v.cols));
01380
01381     // mat2D_fill(des, 0);
01382
01383     if (1 == v.cols) {
01384         for (size_t i = 0; i < des.rows; i++) {
01385             for (size_t j = 0; j < des.cols; j++) {
01386                 MAT2D_AT(des, i, j) = MAT2D_AT(v, i, 0) * MAT2D_AT(v, j, 0);
01387             }
01388         }
01389     } else {
01390         for (size_t i = 0; i < des.rows; i++) {
01391             for (size_t j = 0; j < des.cols; j++) {
01392                 MAT2D_AT(des, i, j) = MAT2D_AT(v, 0, i) * MAT2D_AT(v, 0, j);
01393             }
01394         }
01395     }
01396 }
01397
01419 int mat2D_power_iterate(Mat2D A, Mat2D v, double *lambda, double shift, bool norm_inf_v)
01420 {
01421     /* https://www.youtube.com/watch?v=SkPusgctgpI */
01422
01428     MAT2D_ASSERT(A.cols == A.rows);
01429     MAT2D_ASSERT(v.cols == 1);
01430     MAT2D_ASSERT(v.rows == A.rows);
01431     MAT2D_ASSERT(mat2D_calc_norma_inf(v) > 0);
01432
01433     Mat2D current_v = mat2D_alloc(v.rows, v.cols);
01434     Mat2D temp_v = mat2D_alloc(v.rows, v.cols);
01435     Mat2D B = mat2D_alloc(A.rows, A.cols);
01436     mat2D_copy(B, A);
01437     mat2D_shift(B, shift * -1.0);
01438
01439     double temp_lambda = 0;
01440     double diff = 0;
01441
01442     /* Rayleigh quotient */
01443     mat2D_dot(temp_v, B, v);
01444     temp_lambda = mat2D_dot_product(temp_v, v) / (mat2D_calc_norma(v) * mat2D_calc_norma(v));
01445     int i = 0;
01446     for (i = 0; i < MAT2D_MAX_POWER_ITERATION; i++) {
01447         mat2D_copy(current_v, v);
01448         mat2D_dot(v, B, current_v);
01449         mat2D_normalize(v);
01450         mat2D_mult(v, temp_lambda > 0 ? 1 : -1);
01451         // mat2D_mult(v, fabs(lambda) / lambda);
01452         mat2D_dot(temp_v, B, v);
01453         temp_lambda = mat2D_dot_product(temp_v, v);
01454
01455         mat2D_sub(current_v, v);
01456         diff = mat2D_calc_norma_inf(current_v);
01457         if (diff < MAT2D_EPS) {
01458             break;
01459         }
01460     }
01461
01462     mat2D_free(current_v);
01463     mat2D_free(temp_v);
01464     mat2D_free(B);
01465
```

```
01466     if (norm_inf_v) mat2D_normalize_inf(v);
01467     if (lambda) *lambda = temp_lambda + shift;
01468
01469     if (diff > MAT2D_EPS) {
01470         return 1; /* eigenvector alternating between two options */
01471     } else {
01472         return 0;
01473     }
01474 }
01475
01482 void mat2D_print(Mat2D m, const char *name, size_t padding)
01483 {
01484     printf("%*s%s = [\n", (int) padding, "", name);
01485     for (size_t i = 0; i < m.rows; ++i) {
01486         printf("%*s    ", (int) padding, "");
01487         for (size_t j = 0; j < m.cols; ++j) {
01488             printf("%9.6f ", MAT2D_AT(m, i, j));
01489         }
01490         printf("\n");
01491     }
01492     printf("%*s]\n", (int) padding, "");
01493 }
01494
01501 void mat2D_print_as_col(Mat2D m, const char *name, size_t padding)
01502 {
01503     printf("%*s%s = [\n", (int) padding, "", name);
01504     for (size_t i = 0; i < m.rows*m.cols; ++i) {
01505             printf("%*s    ", (int) padding, "");
01506             printf("%f\n", m.elements[i]);
01507     }
01508     printf("%*s]\n", (int) padding, "");
01509 }
01510
01521 void mat2D_rand(Mat2D m, double low, double high)
01522 {
01523     for (size_t i = 0; i < m.rows; ++i) {
01524         for (size_t j = 0; j < m.cols; ++j) {
01525             MAT2D_AT(m, i, j) = mat2D_rand_double()*(high - low) + low;
01526         }
01527     }
01528 }
01529
01539 double mat2D_rand_double(void)
01540 {
01541     return (double) rand() / (double) RAND_MAX;
01542 }
01543
01557 size_t mat2D_reduce(Mat2D m)
01558 {
01559     /* preforming Gauss-Jordan reduction to Reduced Row Echelon Form (RREF) */
01560     /* Gauss elimination: https://en.wikipedia.org/wiki/Gaussian_elimination */
01561
01562     mat2D_upper_triangulate(m);
01563
01564     size_t rank = 0;
01565
01566     for (int r = m.rows-1; r >= 0; r--) {
01567         size_t c = m.cols-1;
01568         if (!mat2D_find_first_non_zero_value(m, r, &c)) {
01569             continue; /* row of zeros */
01570         }
01571
01572         double pivot = MAT2D_AT(m, r, c);
01573         MAT2D_ASSERT(!MAT2D_IS_ZERO(pivot));
01574         mat2D_mult_row(m, r, 1.0 / pivot);
01575
01576         for (int i = 0; i < r; i++) {
01577             double factor = MAT2D_AT(m, i, c);
01578             mat2D_sub_row_time_factor_to_row(m, i, r, factor);
01579         }
01580         rank++;
01581     }
01582
01583     return rank;
01584 }
01585
01595 bool mat2D_row_is_all_digit(Mat2D m, double digit, size_t r)
01596 {
01597     for (size_t j = 0; j < m.cols; ++j) {
01598         if (MAT2D_AT(m, r, j) != digit) {
01599             return false;
01600         }
01601     }
01602     return true;
01603 }
01604
01616 void mat2D_set_DCM_zyx(Mat2D DCM, float yaw_deg, float pitch_deg, float roll_deg)
```

```
01617 {
01618     Mat2D RotZ = mat2D_alloc(3,3);
01619     mat2D_set_rot_mat_z(RotZ, yaw_deg);
01620     Mat2D RotY = mat2D_alloc(3,3);
01621     mat2D_set_rot_mat_y(RotY, pitch_deg);
01622     Mat2D RotX = mat2D_alloc(3,3);
01623     mat2D_set_rot_mat_x(RotX, roll_deg);
01624     Mat2D temp = mat2D_alloc(3,3);
01625 
01626     mat2D_dot(temp, RotY, RotZ);
01627     mat2D_dot(DCM, RotX, temp); /* I have a DCM */
01628 
01629     mat2D_free(RotZ);
01630     mat2D_free(RotY);
01631     mat2D_free(RotX);
01632     mat2D_free(temp);
01633 }
01634 
01641 void mat2D_set_identity(Mat2D m)
01642 {
01643     MAT2D_ASSERT(m.cols == m.rows);
01644     for (size_t i = 0; i < m.rows; ++i) {
01645         for (size_t j = 0; j < m.cols; ++j) {
01646             MAT2D_AT(m, i, j) = i == j ? 1 : 0;
01647             // if (i == j) {
01648             //     MAT2D_AT(m, i, j) = 1;
01649             // }
01650             // else {
01651             //     MAT2D_AT(m, i, j) = 0;
01652             // }
01653         }
01654     }
01655 }
01656 
01669 void mat2D_set_rot_mat_x(Mat2D m, float angle_deg)
01670 {
01671     MAT2D_ASSERT(3 == m.cols && 3 == m.rows);
01672 
01673     float angle_rad = angle_deg * MAT2D_PI / 180;
01674     mat2D_set_identity(m);
01675     MAT2D_AT(m, 1, 1) =  cos(angle_rad);
01676     MAT2D_AT(m, 1, 2) =  sin(angle_rad);
01677     MAT2D_AT(m, 2, 1) = -sin(angle_rad);
01678     MAT2D_AT(m, 2, 2) =  cos(angle_rad);
01679 }
01680 
01693 void mat2D_set_rot_mat_y(Mat2D m, float angle_deg)
01694 {
01695     MAT2D_ASSERT(3 == m.cols && 3 == m.rows);
01696 
01697     float angle_rad = angle_deg * MAT2D_PI / 180;
01698     mat2D_set_identity(m);
01699     MAT2D_AT(m, 0, 0) =  cos(angle_rad);
01700     MAT2D_AT(m, 0, 2) = -sin(angle_rad);
01701     MAT2D_AT(m, 2, 0) =  sin(angle_rad);
01702     MAT2D_AT(m, 2, 2) =  cos(angle_rad);
01703 }
01704 
01717 void mat2D_set_rot_mat_z(Mat2D m, float angle_deg)
01718 {
01719     MAT2D_ASSERT(3 == m.cols && 3 == m.rows);
01720 
01721     float angle_rad = angle_deg * MAT2D_PI / 180;
01722     mat2D_set_identity(m);
01723     MAT2D_AT(m, 0, 0) =  cos(angle_rad);
01724     MAT2D_AT(m, 0, 1) =  sin(angle_rad);
01725     MAT2D_AT(m, 1, 0) = -sin(angle_rad);
01726     MAT2D_AT(m, 1, 1) =  cos(angle_rad);
01727 }
01728 
01737 void mat2D_shift(Mat2D m, double shift)
01738 {
01739     MAT2D_ASSERT(m.cols == m.rows);
01740     for (size_t i = 0; i < m.rows; i++) {
01741         MAT2D_AT(m, i, i) += shift;
01742     }
01743 }
01744 
01761 void mat2D_solve_linear_sys_LUP_decomposition(Mat2D A, Mat2D x, Mat2D B)
01762 {
01763     MAT2D_ASSERT(A.cols == x.rows);
01764     MAT2D_ASSERT(1 == x.cols);
01765     MAT2D_ASSERT(A.rows == B.rows);
01766     MAT2D_ASSERT(1 == B.cols);
01767 
01768     Mat2D y     = mat2D_alloc(x.rows, x.cols);
01769     Mat2D l     = mat2D_alloc(A.rows, A.cols);
```

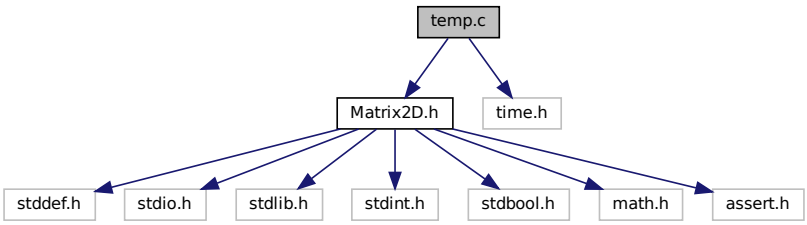
```
01770     Mat2D p     = mat2D_alloc(A.rows, A.cols);
01771     Mat2D u     = mat2D_alloc(A.rows, A.cols);
01772     Mat2D inv_l = mat2D_alloc(l.rows, l.cols);
01773     Mat2D inv_u = mat2D_alloc(u.rows, u.cols);
01774
01775     mat2D_LUP_decomposition_with_swap(A, l, p, u);
01776
01777     mat2D_invert(inv_l, l);
01778     mat2D_invert(inv_u, u);
01779
01780     mat2D_fill(x, 0);   /* x here is only a temp mat*/
01781     mat2D_fill(y, 0);
01782     mat2D_dot(x, p, B);
01783     mat2D_dot(y, inv_l, x);
01784
01785     mat2D_fill(x, 0);
01786     mat2D_dot(x, inv_u, y);
01787
01788     mat2D_free(y);
01789     mat2D_free(l);
01790     mat2D_free(p);
01791     mat2D_free(u);
01792     mat2D_free(inv_l);
01793     mat2D_free(inv_u);
01794 }
01795
01803 void mat2D_sub(Mat2D dst, Mat2D a)
01804 {
01805     MAT2D_ASSERT(dst.rows == a.rows);
01806     MAT2D_ASSERT(dst.cols == a.cols);
01807     for (size_t i = 0; i < dst.rows; ++i) {
01808         for (size_t j = 0; j < dst.cols; ++j) {
01809             MAT2D_AT(dst, i, j) -= MAT2D_AT(a, i, j);
01810         }
01811     }
01812 }
01813
01824 void mat2D_sub_col_to_col(Mat2D des, size_t des_col, Mat2D src, size_t src_col)
01825 {
01826     MAT2D_ASSERT(src_col < src.cols);
01827     MAT2D_ASSERT(des.rows == src.rows);
01828     MAT2D_ASSERT(des_col < des.cols);
01829
01830     for (size_t i = 0; i < des.rows; i++) {
01831         MAT2D_AT(des, i, des_col) -= MAT2D_AT(src, i, src_col);
01832     }
01833 }
01834
01845 void mat2D_sub_row_to_row(Mat2D des, size_t des_row, Mat2D src, size_t src_row)
01846 {
01847     MAT2D_ASSERT(src_row < src.rows);
01848     MAT2D_ASSERT(des.cols == src.cols);
01849     MAT2D_ASSERT(des_row < des.rows);
01850
01851     for (size_t j = 0; j < des.cols; j++) {
01852         MAT2D_AT(des, des_row, j) -= MAT2D_AT(src, src_row, j);
01853     }
01854 }
01855
01865 void mat2D_sub_row_time_factor_to_row(Mat2D m, size_t des_r, size_t src_r, double factor)
01866 {
01867     for (size_t j = 0; j < m.cols; ++j) {
01868         MAT2D_AT(m, des_r, j) -= factor * MAT2D_AT(m, src_r, j);
01869     }
01870 }
01871
01880 void mat2D_swap_rows(Mat2D m, size_t r1, size_t r2)
01881 {
01882     for (size_t j = 0; j < m.cols; j++) {
01883         double temp = MAT2D_AT(m, r1, j);
01884         MAT2D_AT(m, r1, j) = MAT2D_AT(m, r2, j);
01885         MAT2D_AT(m, r2, j) = temp;
01886     }
01887 }
01888
01896 void mat2D_transpose(Mat2D des, Mat2D src)
01897 {
01898     MAT2D_ASSERT(des.cols == src.rows);
01899     MAT2D_ASSERT(des.rows == src.cols);
01900
01901     for (size_t index = 0; index < des.rows; ++index) {
01902         for (size_t jndex = 0; jndex < des.cols; ++jndex) {
01903             MAT2D_AT(des, index, jndex) = MAT2D_AT(src, jndex, index);
01904         }
01905     }
01906 }
01907
```

```
01926 double mat2D_upper_triangulate(Mat2D m)
01927 {
01928     /* preforming Gauss elimination: https://en.wikipedia.org/wiki/Gaussian_elimination */
01929     /* returns the factor multiplying the determinant */
01930
01931     double factor_to_return = 1;
01932
01933     size_t r = 0;
01934     for (size_t c = 0; c < m.cols && r < m.rows; c++) {
01935         /* finding biggest first number (absolute value); partial pivoting */
01936         size_t piv = r;
01937         double best = fabs(MAT2D_AT(m, r, c));
01938         for (size_t i = r + 1; i < m.rows; i++) {
01939             double v = fabs(MAT2D_AT(m, i, c));
01940             if (v > best) {
01941                 best = v;
01942                 piv = i;
01943             }
01944         }
01945         if (MAT2D_IS_ZERO(best)) {
01946             continue; /* move to next column, same pivot row r */
01947         }
01948         if (piv != r) {
01949             mat2D_swap_rows(m, piv, r);
01950             factor_to_return *= -1.0;
01951         }
01952
01953         /* Eliminate entries below pivot in column c */
01954         double pivot = MAT2D_AT(m, r, c);
01955         MAT2D_ASSERT(!MAT2D_IS_ZERO(pivot));
01956         for (size_t i = r + 1; i < m.rows; i++) {
01957             double f = MAT2D_AT(m, i, c) / pivot;
01958             mat2D_sub_row_time_factor_to_row(m, i, r, f);
01959         }
01960         r++;
01961     }
01962     return factor_to_return;
01963 }
01964
01965 #endif // MATRIX2D_IMPLEMENTATION
```

4.3 temp.c File Reference

```
#include "Matrix2D.h"
#include <time.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 5 of file temp.c.

References mat2D_alloc(), MAT2D_AT, mat2D_det(), mat2D_dprintDOUBLE, mat2D_eig_check(), mat2D_eig_power_iteration(), mat2D_free(), MAT2D_PRINT, and mat2D_rand().

4.4 temp.c

```
00001 #define MATRIX2D_IMPLEMENTATION
00002 #include "Matrix2D.h"
00003 #include <time.h>
00004
00005 int main(void)
00006 {
00007     srand(time(0));
00008
00009     int n = 5;
00010     Mat2D A = mat2D_alloc(n, n);
00011     Mat2D res = mat2D_alloc(n, n);
00012     Mat2D eigenvalues = mat2D_alloc(n, n);
00013     Mat2D eigenvectors = mat2D_alloc(n, n);
00014     Mat2D v = mat2D_alloc(n, 1);
00015
00016     MAT2D_AT(v, 0, 0) = 23;
00017     MAT2D_AT(v, 1, 0) = 13;
00018     MAT2D_AT(v, 2, 0) = -13;
00019     MAT2D_AT(v, 3, 0) = -31;
00020     MAT2D_AT(v, 4, 0) = 3;
00021
00022     mat2D_rand(v, 0, 1);
00023
00024     MAT2D_AT(A, 0, 0) = 14;
00025     MAT2D_AT(A, 1, 0) = -6;
00026     MAT2D_AT(A, 2, 0) = 14;
00027     MAT2D_AT(A, 3, 0) = 6;
00028     MAT2D_AT(A, 4, 0) = 8;
00029
00030     MAT2D_AT(A, 0, 1) = -6;
00031     MAT2D_AT(A, 1, 1) = -30;
00032     MAT2D_AT(A, 2, 1) = 6;
00033     MAT2D_AT(A, 3, 1) = -30;
00034     MAT2D_AT(A, 4, 1) = 0;
00035
00036     MAT2D_AT(A, 0, 2) = 14;
00037     MAT2D_AT(A, 1, 2) = 6;
```

```
00038     MAT2D_AT(A, 2, 2) = 14;
00039     MAT2D_AT(A, 3, 2) = -6;
00040     MAT2D_AT(A, 4, 2) = 8;
00041
00042     MAT2D_AT(A, 0, 3) = 6;
00043     MAT2D_AT(A, 1, 3) = -30;
00044     MAT2D_AT(A, 2, 3) = -6;
00045     MAT2D_AT(A, 3, 3) = -30;
00046     MAT2D_AT(A, 4, 3) = 0;
00047
00048     MAT2D_AT(A, 0, 4) = 8;
00049     MAT2D_AT(A, 1, 4) = 0;
00050     MAT2D_AT(A, 2, 4) = 8;
00051     MAT2D_AT(A, 3, 4) = 0;
00052     MAT2D_AT(A, 4, 4) = 20;
00053
00054     MAT2D_PRINT(v);
00055     MAT2D_PRINT(A);
00056
00057     mat2D_eig_power_iteration(A, eigenvalues, eigenvectors, v, 1);
00058
00059     MAT2D_PRINT(eigenvalues);
00060     MAT2D_PRINT(eigenvectors);
00061
00062     mat2D_eig_check(A, eigenvalues, eigenvectors, res);
00063
00064     MAT2D_PRINT(res);
00065
00066
00067     mat2D_dprintDOUBLE(mat2D_det(eigenvectors));
00068
00069
00070
00071     mat2D_free(A);
00072     mat2D_free(res);
00073     mat2D_free(eigenvalues);
00074     mat2D_free(eigenvectors);
00075     mat2D_free(v);
00076
00077     return 0;
00078 }
```

Index