# matrix_decomposition Documentation

***Release 1.2.post0.dev0+dirty.g7610094***

**Joscha Reimer**

**Feb 27, 2020**

# Contents

This is *matrix-decomposition*, a library to approximate Hermitian (dense and sparse) matrices by positive definite matrices. Furthermore it allows to decompose (factorize) positive definite matrices and solve associated systems of linear equations.

CHAPTER 1

# Release info

There are several ways to obtain and install this package.

## 1.1 Conda

To install this package with *conda* run:

```
conda install -c jore matrix-decomposition
```

https://anaconda.org/jore/matrix-decomposition

## 1.2 pip

To install this package with *pip* run:

```
pip install 'matrix-decomposition'
```

https://pypi.python.org/pypi/matrix-decomposition

## 1.3 GitHub

To clone this package with *git* run:

```
git clone https://github.com/jor-/matrix-decomposition.git
```

To install this package after that with *python* run:

```
cd matrix-decomposition; python setup.py install
```

https://github.com/jor-/matrix-decomposition

## 1.4 Zenodo

CHAPTER 2

# Documentation

https://matrix-decomposition.readthedocs.io

CHAPTER 3

# Test status

CHAPTER 4

# Contents

## 4.1 Functions

Several functions are included in this package. The most important ones are summarized here.

### 4.1.1 Positive semidefinite approximation of a matrix

matrix.approximation.positive_semidefinite.**positive_semidefinite_matrix**(*A*, *min_diag_B=None*, *max_diag_B=None*, *min_diag_D=None*, *max_diag_D=None*, *min_abs_value_D=None*, *min_abs_value_L=None*, *permutation=None*, *overwrite_A=False*)

Computes an approximation of $A$ which has a $LDL^H$ decomposition with the specified properties.

Returns $A$ if $A$ has such a decomposition and otherwise an approximation of $A$.

**Parameters**

- **A** (*numpy.ndarray or scipy.sparse.spmatrix*) – The matrix that should be approximated. $A$ must be Hermitian.
- **min_diag_B** (*numpy.ndarray or float*) – Each component of the diagonal of the returned matrix is forced to be greater or equal to *min_diag_B*. optional, default : No minimal value is forced.

- **max_diag_B** (*numpy.ndarray or float*) – Each component of the diagonal of the returned matrix is forced to be lower or equal to *max_diag_B*. optional, default : No maximal value is forced.
- **min_diag_D** (*float*) – Each component of the diagonal of the matrix $D$ in a $LDL^H$ decomposition of the returned matrix is forced to be greater or equal to *min_diag_D*. *min_diag_D* must be greater or equal to 0. optional, default : Is chosen by the algorithm.
- **max_diag_D** (*float*) – Each component of the diagonal of the matrix $D$ in a $LDL^H$ decomposition of the returned matrix is forced to be lower or equal to *max_diag_D*. optional, default : No maximal value is forced.
- **min_abs_value_D** (*float*) – Absolute values below *min_abs_value_D* are considered as zero in the matrix $D$ in a $LDL^H$ decomposition of the returned matrix. *min_abs_value_D* must be greater or equal to 0. optional, default : The square root of the resolution of the underlying data type.
- **min_abs_value_L** (*float*) – Absolute values below *min_abs_value_L* are considered as zero in the matrix $L$ of an approximated $LDL^H$ decomposition. *min_abs_value_L* must be greater or equal to 0. optional, default : The resolution of the underlying data type.
- **permutation** (*str or numpy.ndarray*) – The symmetric permutation method that is applied to the matrix before it is decomposed. It has to be a value in *matrix.UNIVERSAL_PERMUTATION_METHODS* or *APPROXIMATION_ONLY_PERMUTATION_METHODS*. If $A$ is sparse, it can also be a value in *matrix.SPARSE_ONLY_PERMUTATION_METHODS*. It is also possible to directly pass a permutation vector. optional, default: The permutation is chosen by the algorithm.
- **overwrite_A** (*bool*) – Whether it is allowed to overwrite $A$. Enabling may result in performance gain. optional, default: False

**Returns** **B** – An approximation of $A$ which has a $LDL^H$ decomposition.

**Return type** numpy.ndarray or scipy.sparse.spmatrix (same type as $A$)

**Raises**

- *matrix.errors.MatrixNotSquareError* – If $A$ is not a square matrix.
- *matrix.errors.MatrixComplexDiagonalValueError* – If $A$ has complex diagonal values.

matrix.approximation.positive_semidefinite.**decomposition**(*A*, *min_diag_B=None*, *max_diag_B=None*, *min_diag_D=None*, *max_diag_D=None*, *min_abs_value_D=None*, *min_abs_value_L=None*, *permutation=None*, *overwrite_A=False*, *return_type=None*)

Computes an approximative decomposition of a matrix with the specified properties.

Returns a decomposition of $A$ if has such a decomposition and otherwise a decomposition of an approximation of $A$.

**Parameters**

- **A** (*numpy.ndarray or scipy.sparse.spmatrix*) – The matrix that should be approximated by a decomposition. $A$ must be Hermitian.

- **min_diag_B** (*numpy.ndarray or float*) – Each component of the diagonal of the composed matrix $B$ of an approximated $LDL^H$ decomposition is forced to be greater or equal to *min_diag_B*. optional, default : No minimal value is forced.
- **max_diag_B** (*numpy.ndarray or float*) – Each component of the diagonal of the composed matrix $B$ of an approximated $LDL^H$ decomposition is forced to be lower or equal to *max_diag_B*. optional, default : No maximal value is forced.
- **min_diag_D** (*float*) – Each component of the diagonal of the matrix $D$ in an approximated $LDL^H$ decomposition is forced to be greater or equal to *min_diag_D*. *min_diag_D* must be greater or equal to 0. optional, default : Is chosen by the algorithm.
- **max_diag_D** (*float*) – Each component of the diagonal of the matrix $D$ in an approximated $LDL^H$ decomposition is forced to be lower or equal to *max_diag_D*. optional, default : No maximal value is forced.
- **min_abs_value_D** (*float*) – Absolute values below *min_abs_value_D* are considered as zero in the matrix $D$ of an approximated $LDL^H$ decomposition. *min_abs_value_D* must be greater or equal to 0. optional, default : The square root of the resolution of the underlying data type.
- **min_abs_value_L** (*float*) – Absolute values below *min_abs_value_L* are considered as zero in the matrix $L$ of an approximated $LDL^H$ decomposition. *min_abs_value_L* must be greater or equal to 0. optional, default : The resolution of the underlying data type.
- **permutation** (*str or numpy.ndarray*) – The symmetric permutation method that is applied to the matrix before it is decomposed. It has to be a value in *matrix.UNIVERSAL_PERMUTATION_METHODS* or *APPROXIMATION_ONLY_PERMUTATION_METHODS*. If $A$ is sparse, it can also be a value in *matrix.SPARSE_ONLY_PERMUTATION_METHODS*. It is also possible to directly pass a permutation vector. optional, default: The permutation is chosen by the algorithm.
- **overwrite_A** (*bool*) – Whether it is allowed to overwrite $A$. Enabling may result in performance gain. optional, default: False
- **return_type** (*str*) – The type of the decomposition that should be returned. It has to be a value in *matrix.DECOMPOSITION_TYPES*. optional, default : The type of the decomposition is chosen by the function itself.

**Returns** An (approximative) decomposition of $A$ of type *return_type*.

**Return type** *matrix.decompositions.DecompositionBase*

**Raises**

- *matrix.errors.MatrixNotSquareError* – If $A$ is not a square matrix.
- *matrix.errors.MatrixComplexDiagonalValueError* – If $A$ has complex diagonal values.

matrix.approximation.positive_semidefinite.**APPROXIMATION_ONLY_PERMUTATION_METHODS = ('minir**

Permutation methods supported only by the decomposition and the positive_semidefinite_matrix algorithm.

matrix.approximation.positive_semidefinite.**GMW_81**(*A*, *min_diag_B=None*, *max_diag_B=None*, *min_diag_D=None*, *overwrite_A=False*)

Computes a positive definite approximation of $A$.

Returns $A$ if $A$ is positive definite and meets the constrains and otherwise a positive definite approximation of $A$.

**Parameters**

- **A** (*numpy.ndarray*) – The matrix that should be approximated. *A* must be symmetric.
- **min_diag_B** (*numpy.ndarray or float*) – Each component of the diagonal of the returned matrix is forced to be greater or equal to *min_diag_B*. optional, default : No minimal value is forced.
- **max_diag_B** (*numpy.ndarray or float*) – Each component of the diagonal of the returned matrix is forced to be lower or equal to *max_diag_B*. optional, default : No maximal value is forced.
- **min_diag_D** (*float*) – Each component of the diagonal of the matrix $D$ in a $LDL^T$ decomposition of the returned matrix is forced to be greater or equal to *min_diag_D*. *min_diag_D* must be positive. optional, default : Is chosen by the algorithm.
- **overwrite_A** (*bool*) – Whether it is allowed to overwrite *A*. Enabling may result in performance gain. optional, default: False

**Returns** **B** – An approximation of *A* which is positive definite.

**Return type** numpy.ndarray

**Raises** *matrix.errors.MatrixNotSquareError* – If *A* is not a square matrix.

### Notes

The algorithm is introduced in [1]. Is is also described in [2]. This discription has been used for this implementation. The implementation has been extended to allow restrictions on the diagonal values.

### References

**[1] Gill, P. E.; Murray, W. & Wright, M. H.,** Practical optimization, Academic press, 1981

**[2] Fang, H.-r. & O'Leary, D. P.,** Modified Cholesky algorithms: a catalog with new approaches, Mathematical Programming, 2008, 115, 319-349

matrix.approximation.positive_semidefinite.**GMW_T1**(*A*, *min_diag_B=None*, *max_diag_B=None*, *min_diag_D=None*, *overwrite_A=False*)

Computes a positive definite approximation of *A*.

Returns *A* if *A* is positive definite and meets the constrains and otherwise a positive definite approximation of *A*.

**Parameters**

- **A** (*numpy.ndarray*) – The matrix that should be approximated. *A* must be symmetric.
- **min_diag_B** (*numpy.ndarray or float*) – Each component of the diagonal of the returned matrix is forced to be greater or equal to *min_diag_B*. optional, default : No minimal value is forced.
- **max_diag_B** (*numpy.ndarray or float*) – Each component of the diagonal of the returned matrix is forced to be lower or equal to *max_diag_B*. optional, default : No maximal value is forced.
- **min_diag_D** (*float*) – Each component of the diagonal of the matrix $D$ in a $LDL^T$ decomposition of the returned matrix is forced to be greater or equal to *min_diag_D*. *min_diag_D* must be positive. optional, default : Is chosen by the algorithm.

- **overwrite_A** (*bool*) – Whether it is allowed to overwrite *A*. Enabling may result in performance gain. optional, default: False

**Returns** **B** – An approximation of *A* which is positive definite.

**Return type** numpy.ndarray

**Raises** *matrix.errors.MatrixNotSquareError* – If *A* is not a square matrix.

### Notes

The algorithm is introduced in [1]. This discription has been used for this implementation. The algorithm is based on [2]. The implementation has been extended to allow restrictions on the diagonal values.

### References

**[1] Fang, H.-r. & O'Leary, D. P.,** Modified Cholesky algorithms: a catalog with new approaches, Mathematical Programming, 2008, 115, 319-349

**[2] Gill, P. E.; Murray, W. & Wright, M. H.,** Practical optimization, Academic press, 1981

matrix.approximation.positive_semidefinite.**GMW_T2**(*A*, *min_diag_B=None*, *max_diag_B=None*, *min_diag_D=None*, *overwrite_A=False*)

Computes a positive definite approximation of *A*.

Returns *A* if *A* is positive definite and meets the constrains and otherwise a positive definite approximation of *A*.

**Parameters**

- **A** (*numpy.ndarray*) – The matrix that should be approximated. *A* must be symmetric.
- **min_diag_B** (*numpy.ndarray or float*) – Each component of the diagonal of the returned matrix is forced to be greater or equal to *min_diag_B*. optional, default : No minimal value is forced.
- **max_diag_B** (*numpy.ndarray or float*) – Each component of the diagonal of the returned matrix is forced to be lower or equal to *max_diag_B*. optional, default : No maximal value is forced.
- **min_diag_D** (*float*) – Each component of the diagonal of the matrix $D$ in a $LDL^T$ decomposition of the returned matrix is forced to be greater or equal to *min_diag_D*. *min_diag_D* must be positive. optional, default : Is chosen by the algorithm.
- **overwrite_A** (*bool*) – Whether it is allowed to overwrite *A*. Enabling may result in performance gain. optional, default: False

**Returns** **B** – An approximation of *A* which is positive definite.

**Return type** numpy.ndarray

**Raises** *matrix.errors.MatrixNotSquareError* – If *A* is not a square matrix.

### Notes

The algorithm is introduced in [1]. This discription has been used for this implementation. The algorithm is based on [2]. The implementation has been extended to allow restrictions on the diagonal values.

### References

**[1] Fang, H.-r. & O'Leary, D. P.,** Modified Cholesky algorithms: a catalog with new approaches, Mathematical Programming, 2008, 115, 319-349

**[2] Gill, P. E.; Murray, W. & Wright, M. H.,** Practical optimization, Academic press, 1981

matrix.approximation.positive_semidefinite.**SE_90**(*A*, *min_diag_B=None*, *max_diag_B=None*, *min_diag_D=None*, *overwrite_A=False*)

Computes a positive definite approximation of *A*.

Returns *A* if *A* is positive definite and meets the constrains and otherwise a positive definite approximation of *A*.

**Parameters**

- **A** (*numpy.ndarray*) – The matrix that should be approximated. *A* must be symmetric.
- **min_diag_B** (*numpy.ndarray or float*) – Each component of the diagonal of the returned matrix is forced to be greater or equal to *min_diag_B*. optional, default : No minimal value is forced.
- **max_diag_B** (*numpy.ndarray or float*) – Each component of the diagonal of the returned matrix is forced to be lower or equal to *max_diag_B*. optional, default : No maximal value is forced.
- **min_diag_D** (*float*) – Each component of the diagonal of the matrix $D$ in a $LDL^T$ decomposition of the returned matrix is forced to be greater or equal to *min_diag_D*. *min_diag_D* must be positive. optional, default : Is chosen by the algorithm.
- **overwrite_A** (*bool*) – Whether it is allowed to overwrite *A*. Enabling may result in performance gain. optional, default: False

**Returns** **B** – An approximation of *A* which is positive definite.

**Return type** numpy.ndarray

**Raises** *matrix.errors.MatrixNotSquareError* – If *A* is not a square matrix.

### Notes

The algorithm is introduced in [1]. Is is also described in [2]. This discription has been used for this implementation. The implementation has been extended to allow restrictions on the diagonal values.

### References

**[1] Schnabel, R. & Eskow, E.,** A New Modified Cholesky Factorization, SIAM Journal on Scientific and Statistical Computing, 1990, 11, 1136-1158

**[2] Fang, H.-r. & O'Leary, D. P.,** Modified Cholesky algorithms: a catalog with new approaches, Mathematical Programming, 2008, 115, 319-349

matrix.approximation.positive_semidefinite.**SE_99**(*A*, *min_diag_B=None*, *max_diag_B=None*, *min_diag_D=None*, *overwrite_A=False*)

Computes a positive definite approximation of *A*.

Returns *A* if *A* is positive definite and meets the constrains and otherwise a positive definite approximation of *A*.

**Parameters**

- **A** (*numpy.ndarray*) – The matrix that should be approximated. *A* must be symmetric.
- **min_diag_B** (*numpy.ndarray or float*) – Each component of the diagonal of the returned matrix is forced to be greater or equal to *min_diag_B*. optional, default : No minimal value is forced.
- **max_diag_B** (*numpy.ndarray or float*) – Each component of the diagonal of the returned matrix is forced to be lower or equal to *max_diag_B*. optional, default : No maximal value is forced.
- **min_diag_D** (*float*) – Each component of the diagonal of the matrix $D$ in a $LDL^T$ decomposition of the returned matrix is forced to be greater or equal to *min_diag_D*. *min_diag_D* must be positive. optional, default : Is chosen by the algorithm.
- **overwrite_A** (*bool*) – Whether it is allowed to overwrite *A*. Enabling may result in performance gain. optional, default: False

**Returns** **B** – An approximation of *A* which is positive definite.

**Return type** numpy.ndarray

**Raises** *matrix.errors.MatrixNotSquareError* – If *A* is not a square matrix.

### Notes

The algorithm is introduced in [1]. Is is also described in [2]. This discription has been used for this implementation. The algorithm is based on [3]. The implementation has been extended to allow restrictions on the diagonal values.

### References

**[1] Schnabel, R. & Eskow, E.,** A Revised Modified Cholesky Factorization Algorithm, SIAM Journal on Optimization, 1999, 9, 1135-1148

**[2] Fang, H.-r. & O'Leary, D. P.,** Modified Cholesky algorithms: a catalog with new approaches, Mathematical Programming, 2008, 115, 319-349

**[3] Schnabel, R. & Eskow, E.,** A New Modified Cholesky Factorization, SIAM Journal on Scientific and Statistical Computing, 1990, 11, 1136-1158

matrix.approximation.positive_semidefinite.**SE_T1**(*A*, *min_diag_B=None*, *max_diag_B=None*, *min_diag_D=None*, *overwrite_A=False*)

Computes a positive definite approximation of *A*.

Returns *A* if *A* is positive definite and meets the constrains and otherwise a positive definite approximation of *A*.

**Parameters**

- **A** (*numpy.ndarray*) – The matrix that should be approximated. *A* must be symmetric.
- **min_diag_B** (*numpy.ndarray or float*) – Each component of the diagonal of the returned matrix is forced to be greater or equal to *min_diag_B*. optional, default : No minimal value is forced.

- **max_diag_B** (*numpy.ndarray or float*) – Each component of the diagonal of the returned matrix is forced to be lower or equal to *max_diag_B*. optional, default : No maximal value is forced.
- **min_diag_D** (*float*) – Each component of the diagonal of the matrix $D$ in a $LDL^T$ decomposition of the returned matrix is forced to be greater or equal to *min_diag_D*. *min_diag_D* must be positive. optional, default : Is chosen by the algorithm.
- **overwrite_A** (*bool*) – Whether it is allowed to overwrite *A*. Enabling may result in performance gain. optional, default: False

**Returns** **B** – An approximation of *A* which is positive definite.

**Return type** numpy.ndarray

**Raises** *matrix.errors.MatrixNotSquareError* – If *A* is not a square matrix.

#### Notes

The algorithm is introduced in [1]. This discription has been used for this implementation. The algorithm is based on [2]. The implementation has been extended to allow restrictions on the diagonal values.

#### References

**[1] Fang, H.-r. & O'Leary, D. P.,** Modified Cholesky algorithms: a catalog with new approaches, Mathematical Programming, 2008, 115, 319-349

**[2] Schnabel, R. & Eskow, E.,** A Revised Modified Cholesky Factorization Algorithm, SIAM Journal on Optimization, 1999, 9, 1135-1148

**[3] Schnabel, R. & Eskow, E.,** A New Modified Cholesky Factorization, SIAM Journal on Scientific and Statistical Computing, 1990, 11, 1136-1158

### 4.1.2 Nearest matrix with specific properties

This module provides functions to compute matrices with minimal difference to an input matrix and specific properties.

matrix.nearest.**correlation_matrix**(*A*, *max_iterations=1000*)

Computes a correlation matrix closest to *A* in the Frobenius norm. A correlation matrix is a positive semidefinite matrix with ones on the diagonal.

**Parameters**

- **A** (*numpy.ndarray*) – *A* must be Hermitian.
- **max_iterations** (*int*) – The maximal number of iterations used for the calculation.

**Returns** **B** – A correlation matrix closest to *A* in the Frobenius norm.

**Return type** numpy.ndarray

**Raises** TooManyIterationsError – If the computation needs more iterations than the maximal number of iterations.

### Notes

The method is presented in [1]. For the computation some code of Michael Croucher is used. See this source code for the license.

### References

**[1] Higham, N. J.** Computing the Nearest Correlation Matrix—A Problem from Finance IMA J. Numer. Anal., 2002, 22, 329-343

matrix.nearest.**positive_semidefinite_matrix**(*A*, *symmetric=False*)

Computes a symmetric positive semidefinite matrix with minimal difference to *A* in the Frobenius norm.

**Parameters**

- **A** (*numpy.ndarray*) – *A* must be Hermitian.
- **symmetric** (*bool*) – Whether *A* can be assumed to be symmetric or not. optional, default: False

**Returns** **B** – A symmetric positive semidefinite Hermitian matrix with minimal difference to *A* in the Frobenius norm.

**Return type** numpy.ndarray

### Notes

The method is presented in [1].

### References

**[1] Higham, N. J.** Computing a nearest symmetric positive semidefinite matrix Linear Algebra and its Applications, 1988, 103, 103-118

matrix.nearest.**skew_symmetric_matrix**(*A*)

Computes a skew-symmetric matrix with minimal difference to *A* for any unitarily invariant norm.

**Parameters** **A** (*numpy.ndarray*) – *A* must be a square matrix.

**Returns** **B** – A skew-symmetric matrix with minimal difference to *A* for any unitarily invariant norm.

**Return type** numpy.ndarray

matrix.nearest.**symmetric_matrix**(*A*)

Computes a symmetric matrix with minimal difference to *A* for any unitarily invariant norm.

**Parameters** **A** (*numpy.ndarray*) – *A* must be a square matrix.

**Returns** **B** – A symmetric matrix with minimal difference to *A* for any unitarily invariant norm.

**Return type** numpy.ndarray

### Notes

The optimality is proven in [1].

#### References

**[1] Fan, K., & Hoffman, A. (1955).** Some Metric Inequalities in the Space of Matrices. Proceedings of the American Mathematical Society, 6(1), 111-116. doi:10.2307/2032662

### 4.1.3 Decompose a matrix

matrix.**decompose**(*A*, *permutation=None*, *return_type=None*, *check_finite=True*, *overwrite_A=False*)
Computes a decomposition of a matrix.

**Parameters**

- **A** (*numpy.ndarray or scipy.sparse.spmatrix*) – Matrix to be decomposed. *A* must be Hermitian.
- **permutation** (*str or numpy.ndarray*) – The symmetric permutation method that is applied to the matrix before it is decomposed. It has to be a value in *matrix.UNIVERSAL_PERMUTATION_METHODS*. If *A* is sparse, it can also be a value in *matrix.SPARSE_ONLY_PERMUTATION_METHODS*. It is also possible to directly pass a permutation vector. optional, default: no permutation
- **return_type** (*str*) – The type of the decomposition that should be calculated. It has to be a value in *matrix.DECOMPOSITION_TYPES*. If return_type is None the type of the returned decomposition is chosen by the function itself. optional, default: the type of the decomposition is chosen by the function itself
- **check_finite** (*bool*) – Whether to check that *A* contains only finite numbers. Disabling may result in problems (crashes, non-termination) if the inputs do contain infinities or NaNs. Disabling gives a performance gain. optional, default: True
- **overwrite_A** (*bool*) – Whether it is allowed to overwrite *A*. Enabling may result in performance gain. optional, default: False

**Returns** A decomposition of *A* of type *return_type*.

**Return type** *matrix.decompositions.DecompositionBase*

**Raises**

- *matrix.errors.NoDecompositionPossibleError* – If the decomposition of *A* is not possible.
- *matrix.errors.MatrixNotSquareError* – If *A* is not a square matrix.
- *matrix.errors.MatrixNotFiniteError* – If *A* is not a finite matrix and *check_finite* is True.

matrix.**UNIVERSAL_PERMUTATION_METHODS = ('none', 'decreasing_diagonal_values', 'increasing_**
Supported permutation methods for decompose dense and sparse matrices.

matrix.**SPARSE_ONLY_PERMUTATION_METHODS = ()**
Supported permutation methods only for sparse matrices.

matrix.**DECOMPOSITION_TYPES = ('LDL', 'LDL_compressed', 'LL')**
Supported types of decompositions.

### 4.1.4 Examine a matrix

matrix.**is_positive_semidefinite**(*A*, *check_finite=True*)
Returns whether the passed matrix is positive semi-definite.

**Parameters**

- **A** (*numpy.ndarray or scipy.sparse.spmatrix*) – The matrix that should be checked. *A* must be Hermitian.
- **check_finite** (*bool*) – Whether to check that *A* contain only finite numbers. Disabling may result in problems (crashes, non-termination) if they contain infinities or NaNs. Disabling gives a performance gain. optional, default: True

**Returns** Whether *A* is positive semi-definite.

**Return type** bool

**Raises** *matrix.errors.MatrixNotFiniteError* – If *A* is not a finite matrix and *check_finite* is True.

matrix.**is_positive_definite**(*A*, *check_finite=True*)

Returns whether the passed matrix is positive definite.

**Parameters**

- **A** (*numpy.ndarray or scipy.sparse.spmatrix*) – The matrix that should be checked. *A* must be Hermitian.
- **check_finite** (*bool*) – Whether to check that *A* contain only finite numbers. Disabling may result in problems (crashes, non-termination) if they contain infinities or NaNs. Disabling gives a performance gain. optional, default: True

**Returns** Whether *A* is positive definite.

**Return type** bool

**Raises** *matrix.errors.MatrixNotFiniteError* – If *A* is not a finite matrix and *check_finite* is True.

matrix.**is_invertible**(*A*, *check_finite=True*)

Returns whether the passed matrix is an invertible matrix.

**Parameters**

- **A** (*numpy.ndarray or scipy.sparse.spmatrix*) – The matrix that should be checked. *A* must be Hermitian and positive semidefinite.
- **check_finite** (*bool*) – Whether to check that *A* contain only finite numbers. Disabling may result in problems (crashes, non-termination) if they contain infinities or NaNs. Disabling gives a performance gain. optional, default: True

**Returns** Whether *A* is invertible.

**Return type** bool

**Raises** *matrix.errors.MatrixNotFiniteError* – If *A* is not a finite matrix and *check_finite* is True.

### 4.1.5 Solve system of linear equations

matrix.**solve**(*A*, *b*, *overwrite_b=False*, *check_finite=True*)

Solves the equation $A\ x = b$ regarding $x$.

**Parameters**

- **A** (*numpy.ndarray or scipy.sparse.spmatrix*) – The matrix that should be checked. *A* must be Hermitian and positive definite.

- **b** (*numpy.ndarray*) – Right-hand side vector or matrix in equation $A\ x = b$. It must hold *b.shape[0] == A.shape[0]*.
- **overwrite_b** (*bool*) – Allow overwriting data in $b$. Enabling gives a performance gain. optional, default: False
- **check_finite** (*bool*) – Whether to check that $A$ and b‘ contain only finite numbers. Disabling may result in problems (crashes, non-termination) if they contain infinities or NaNs. Disabling gives a performance gain. optional, default: True

**Returns** An $x$ so that $A\ x = b$. The shape of $x$ matches the shape of $b$.

**Return type** numpy.ndarray

**Raises**

- *matrix.errors.MatrixNotSquareError* – If $A$ is not a square matrix.
- *matrix.errors.MatrixNotFiniteError* – If $A$ is not a finite matrix and *check_finite* is True.
- *matrix.errors.MatrixSingularError* – If $A$ is singular.

## 4.2 Matrix decompositions

Several matrix decompositions are supported. They are available in *matrix.decompositions*:

### 4.2.1 LL decomposition

**class** matrix.decompositions.**LL_Decomposition**(*L=None*, *p=None*)

Bases: *matrix.decompositions.DecompositionBase*

A matrix decomposition where $LL^H$ is the decomposed (permuted) matrix.

$L$ is a lower triangle matrix with ones on the diagonal. This decomposition is also called Cholesky decomposition.

**Parameters**

- **L** (*numpy.ndarray or scipy.sparse.spmatrix*) – The matrix $L$ of the decomposition. optional, If it is not set yet, it must be set later.
- **p** (*numpy.ndarray*) – The permutation vector used for the decomposition. This decomposition is of *A[p[:, np.newaxis], p[np.newaxis, :]]* where $A$ is a matrix. optional, default: no permutation

**L**

The matrix $L$ of the decomposition.

**Type** numpy.matrix or scipy.sparse.spmatrix

**P**

The permutation matrix. *P @ A @ P.T* is the matrix $A$ permuted by the permutation of the decomposition

**Type** scipy.sparse.dok_matrix

**append_block_decomposition**(*dec*)

Makes a new decomposition where this decompsition and the passed one are appended.

**Parameters** **dec** (DecompositionBase) – The decomposition that should be appended to this decomposition.

**Returns** **dec** – A new decomposition where this decompsition and the passed one are appended. This desomposition represents the first block and the passed one the second block.

**Return type** *DecompositionBase*

**as_LDL_Decomposition**()

**as_any_type**(**type_strs*, *copy=False*)
Converts this decomposition to any of the passed types.

**Parameters**

- ***type_strs** (*str*) – The decomposition types to any of them this this decomposition is converted.
- **copy** (*bool*) – Whether the data of this decomposition should always be copied or only if needed.

**Returns** If the type of this decomposition is not in *type_strs*, a decomposition of type *type_str[0]* is returned which represents the same decomposed matrix as this decomposition. Otherwise this decomposition or a copy of it is returned, depending on *copy*.

**Return type** *matrix.decompositions.DecompositionBase*

**as_same_type**(*dec*, *copy=False*)
Converts the passed decompositions to the same type as this decomposition.

**Parameters**

- **doc** (DecompositionBase) – The decomposition that should be converted.
- **copy** (*bool*) – Whether the data of the decomposition that sould be converted should always be copied or only if needed.

**Returns** If the type of the passed decomposition is not same type as this decomposition, a decomposition of this type is returned which represents the same decomposed matrix as the passed decomposition. Otherwise this decomposition or a copy of it is returned, depending on *copy*.

**Return type** *matrix.decompositions.DecompositionBase*

**as_type**(*type_str*, *copy=False*)
Converts this decomposition to passed type.

**Parameters**

- **type_str** (*str*) – The decomposition type to which this decomposition is converted.
- **copy** (*bool*) – Whether the data of this decomposition should always be copied or only if needed.

**Returns** If the type of this decomposition is not *type_str*, a decomposition of type *type_str* is returned which represents the same decomposed matrix as this decomposition. Otherwise this decomposition or a copy of it is returned, depending on *copy*.

**Return type** *matrix.decompositions.DecompositionBase*

**check_finite**(*check_finite=True*)
Check if this is a decomposition representing a finite matrix.

**Parameters** **check_finite** (*bool*) – Whether to perform this check. default: True

**Raises** *matrix.errors.DecompositionNotFiniteError* – If this is a decomposition representing a non-finite matrix.

**check_invertible**()
Check if this is a decomposition representing an invertible matrix.

**Raises** *matrix.errors.DecompositionSingularError* – If this is a decomposition representing a singular matrix.

**composed_matrix**
The composed matrix represented by this decomposition.

**Type** numpy.matrix or scipy.sparse.spmatrix

**copy**()
Copies this decomposition.

**Returns** A copy of this decomposition.

**Return type** *matrix.decompositions.DecompositionBase*

**inverse_matrix_both_sides_multiplication**(*x*, *y=None*, *dtype=None*)
Calculates the both sides (matrix-matrix or matrix-vector) product *y.H @ B @ x*, where *B* is the mattrix inverse of the composed matrix represented by this decomposition.

**Parameters**

- **x** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product *y.H @ B @ x*. It must hold *self.n == x.shape[0]*.
- **y** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product *y.H @ B @ x*. It must hold *self.n == y.shape[0]*. optional, default: If y is not passed, x is used as y.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** The result of *y.H @ A @ x*.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**Raises** *matrix.errors.DecompositionSingularError* – If this is a decomposition representing a singular matrix.

**inverse_matrix_right_side_multiplication**(*x*, *dtype=None*)
Calculates the right side (matrix-matrix or matrix-vector) product *B @ x*, where *B* is the matrix inverse of the composed matrix represented by this decomposition.

**Parameters**

- **x** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product in the matrix-matrix or matrix-vector *B @ x*. It must hold *self.n == x.shape[0]*.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** The result of *B @ x*.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**Raises** *matrix.errors.DecompositionSingularError* – If this is a decomposition representing a singular matrix.

**is_almost_equal**(*other*, *rtol=0.0001*, *atol=1e-06*)
Whether this decomposition is close to passed decomposition.

**Parameters**

- **other** (*str*) – The decomposition which to compare to this decomposition.

- **rtol** (*float*) – The relative tolerance parameter.
- **atol** (*float*) – The absolute tolerance parameter.

**Returns** Whether this decomposition is close to passed decomposition.

**Return type** bool

**is_equal**(*other*)
Whether this decomposition is equal to passed decomposition.

**Parameters** **other** (*str*) – The decomposition which to compare to this decomposition.

**Returns** Whether this decomposition is equal to passed decomposition.

**Return type** bool

**is_finite**()
Returns whether this is a decomposition representing a finite matrix.

**Returns** Whether this is a decomposition representing a finite matrix.

**Return type** bool

**is_invertible**()
Returns whether this is a decomposition representing an invertible matrix.

**Returns** Whether this is a decomposition representing an invertible matrix.

**Return type** bool

**is_permuted**
Whether this is a decompositon with permutation.

**Type** bool

**is_positive_definite**()
Returns whether this is a decomposition of a positive definite matrix.

**Returns** Whether this is a decomposition of a positive definite matrix.

**Return type** bool

**is_positive_semidefinite**()
Returns whether this is a decomposition of a positive semi-definite matrix.

**Returns** Whether this is a decomposition of a positive semi-definite matrix.

**Return type** bool

**is_singular**()
Returns whether this is a decomposition representing a singular matrix.

**Returns** Whether this is a decomposition representing a singular matrix.

**Return type** bool

**is_sparse**()
Returns whether this is a decomposition of a sparse matrix.

**Returns** Whether this is a decomposition of a sparse matrix.

**Return type** bool

**is_type**(*type_str*)
Whether this decomposition is of the passed type.

**Parameters** **type_str** (*str*) – The decomposition type according to which is checked.

**Returns** Whether this is a decomposition of the passed type.

**Return type** bool

**load**(*filename*)
Loads a decomposition of this type.

**Parameters** **filename** (*str*) – Where the decomposition is saved.

**Raises**

- FileNotFoundError – If the files are not found in the passed directory.
- DecompositionInvalidDecompositionTypeFile – If the files contains another decomposition type.

**matrix_both_sides_multiplication**(*x*, *y=None*, *dtype=None*)
Calculates the both sides (matrix-matrix or matrix-vector) product *y.H @ A @ x*, where *A* is the composed matrix represented by this decomposition.

**Parameters**

- **x** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product *y.H @ A @ x*. It must hold *self.n == x.shape[0]*.
- **y** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product *y.H @ A @ x*. It must hold *self.n == y.shape[0]*. optional, default: If y is not passed, x is used as y.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** The result of *y.H @ A @ x*.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**matrix_right_side_multiplication**(*x*, *dtype=None*)
Calculates the right side (matrix-matrix or matrix-vector) product *A @ x*, where *A* is the composed matrix represented by this decomposition.

**Parameters**

- **x** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product in the matrix-matrix or matrix-vector *A @ x*. It must hold *self.n == x.shape[0]*.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** The result of *A @ x*.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**n**
The dimension of the squared decomposed matrix.

**Type** int

**p**
The permutation vector. *A[p[:, np.newaxis], p[np.newaxis, :]]* is the matrix *A* permuted by the permutation of the decomposition

**Type** numpy.ndarray

**p_inverse**
The permutation vector that undoes the permutation.

**Type** numpy.ndarray

**permute_matrix**(*A*)
Permutes a matrix by the permutation of the decomposition.

**Parameters** **A** (*numpy.ndarray or scipy.sparse.spmatrix*) – The matrix that should be permuted.

**Returns** The matrix *A* permuted by the permutation of the decomposition.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**save**(*filename*)
Saves this decomposition.

**Parameters** **filename** (*str*) – Where this decomposition should be saved.

**solve**(*b*, *dtype=None*)
Solves the equation $A\ x = b$ regarding $x$, where $A$ is the composed matrix represented by this decomposition.

**Parameters**

- **b** (*numpy.ndarray or scipy.sparse.spmatrix*) – Right-hand side vector or matrix in equation $A\ x = b$. It must hold *self.n == b.shape[0]*.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** An $x$ so that $A\ x = b$. The shape of $x$ matches the shape of $b$.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**Raises** *matrix.errors.DecompositionSingularError* – If this is a decomposition representing a singular matrix.

**type_str = 'LL'**
The type of this decomposition represented as string.

**Type** str

**unpermute_matrix**(*A*)
Unpermutes a matrix permuted by the permutation of the decomposition.

**Parameters** **A** (*numpy.ndarray or scipy.sparse.spmatrix*) – The matrix that should be unpermuted.

**Returns** The matrix *A* unpermuted by the permutation of the decomposition.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

### 4.2.2 LDL decomposition

**class** matrix.decompositions.**LDL_Decomposition**(*L=None*, *d=None*, *p=None*)
Bases: *matrix.decompositions.DecompositionBase*

A matrix decomposition where $LDL^H$ is the decomposed (permuted) matrix.

$L$ is a lower triangle matrix with ones on the diagonal. $D$ is a diagonal matrix. Only the diagonal values of $D$ are stored.

**Parameters**

- **L** (*numpy.ndarray or scipy.sparse.spmatrix*) – The matrix $L$ of the decomposition. optional, If it is not set yet, it must be set later.

- **d** (*numpy.ndarray*) – The vector of the diagonal components of *D* of the decompositon. optional, If it is not set yet, it must be set later.
- **p** (*numpy.ndarray*) – The permutation vector used for the decomposition. This decomposition is of *A[p[:, np.newaxis], p[np.newaxis, :]]* where *A* is a matrix. optional, default: no permutation

**D**

The permutation matrix.

**Type** scipy.sparse.dia_matrix

**L**

The matrix *L* of the decomposition.

**Type** numpy.matrix or scipy.sparse.spmatrix

**LD**

A matrix whose diagonal values are the diagonal values of *D* and whose off-diagonal values are those of *L*.

**Type** numpy.matrix or scipy.sparse.spmatrix

**P**

The permutation matrix. *P @ A @ P.T* is the matrix *A* permuted by the permutation of the decomposition

**Type** scipy.sparse.dok_matrix

**append_block_decomposition**(*dec*)

Makes a new decomposition where this decompsition and the passed one are appended.

**Parameters** **dec** (DecompositionBase) – The decomposition that should be appended to this decomposition.

**Returns** **dec** – A new decomposition where this decompsition and the passed one are appended. This desomposition represents the first block and the passed one the second block.

**Return type** *DecompositionBase*

**as_LDL_DecompositionCompressed**()

**as_LL_Decomposition**()

**as_any_type**(**type_strs*, *copy=False*)

Converts this decomposition to any of the passed types.

**Parameters**

- ***type_strs** (*str*) – The decomposition types to any of them this this decomposition is converted.
- **copy** (*bool*) – Whether the data of this decomposition should always be copied or only if needed.

**Returns** If the type of this decomposition is not in *type_strs*, a decomposition of type *type_str[0]* is returned which represents the same decomposed matrix as this decomposition. Otherwise this decomposition or a copy of it is returned, depending on *copy*.

**Return type** *matrix.decompositions.DecompositionBase*

**as_same_type**(*dec*, *copy=False*)

Converts the passed decompositions to the same type as this decomposition.

**Parameters**

- **doc** (DecompositionBase) – The decomposition that should be converted.

- **copy** (*bool*) – Whether the data of the decomposition that sould be converted should always be copied or only if needed.

**Returns** If the type of the passed decomposition is not same type as this decomposition, a decomposition of this type is returned which represents the same decomposed matrix as the passed decomposition. Otherwise this decomposition or a copy of it is returned, depending on *copy*.

**Return type** *matrix.decompositions.DecompositionBase*

**as_type**(*type_str*, *copy=False*)
Converts this decomposition to passed type.

**Parameters**

- **type_str** (*str*) – The decomposition type to which this decomposition is converted.
- **copy** (*bool*) – Whether the data of this decomposition should always be copied or only if needed.

**Returns** If the type of this decomposition is not *type_str*, a decomposition of type *type_str* is returned which represents the same decomposed matrix as this decomposition. Otherwise this decomposition or a copy of it is returned, depending on *copy*.

**Return type** *matrix.decompositions.DecompositionBase*

**check_finite**(*check_finite=True*)
Check if this is a decomposition representing a finite matrix.

**Parameters** **check_finite** (*bool*) – Whether to perform this check. default: True

**Raises** *matrix.errors.DecompositionNotFiniteError* – If this is a decomposition representing a non-finite matrix.

**check_invertible**()
Check if this is a decomposition representing an invertible matrix.

**Raises** *matrix.errors.DecompositionSingularError* – If this is a decomposition representing a singular matrix.

**composed_matrix**
The composed matrix represented by this decomposition.

**Type** numpy.matrix or scipy.sparse.spmatrix

**copy**()
Copies this decomposition.

**Returns** A copy of this decomposition.

**Return type** *matrix.decompositions.DecompositionBase*

**d**
The diagonal vector of the matrix *D* of the decomposition.

**Type** numpy.ndarray

**inverse_matrix_both_sides_multiplication**(*x*, *y=None*, *dtype=None*)
Calculates the both sides (matrix-matrix or matrix-vector) product *y.H @ B @ x*, where *B* is the mattrix inverse of the composed matrix represented by this decomposition.

**Parameters**

- **x** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product *y.H @ B @ x*. It must hold *self.n == x.shape[0]*.

- **y** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product *y.H @ B @ x*. It must hold *self.n == y.shape[0]*. optional, default: If y is not passed, x is used as y.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** The result of *y.H @ A @ x*.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**Raises** *matrix.errors.DecompositionSingularError* – If this is a decomposition representing a singular matrix.

**inverse_matrix_right_side_multiplication**(*x*, *dtype=None*)

Calculates the right side (matrix-matrix or matrix-vector) product *B @ x*, where *B* is the matrix inverse of the composed matrix represented by this decomposition.

**Parameters**

- **x** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product in the matrix-matrix or matrix-vector *B @ x*. It must hold *self.n == x.shape[0]*.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** The result of *B @ x*.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**Raises** *matrix.errors.DecompositionSingularError* – If this is a decomposition representing a singular matrix.

**is_almost_equal**(*other*, *rtol=0.0001*, *atol=1e-06*)

Whether this decomposition is close to passed decomposition.

**Parameters**

- **other** (*str*) – The decomposition which to compare to this decomposition.
- **rtol** (*float*) – The relative tolerance parameter.
- **atol** (*float*) – The absolute tolerance parameter.

**Returns** Whether this decomposition is close to passed decomposition.

**Return type** bool

**is_equal**(*other*)

Whether this decomposition is equal to passed decomposition.

**Parameters** **other** (*str*) – The decomposition which to compare to this decomposition.

**Returns** Whether this decomposition is equal to passed decomposition.

**Return type** bool

**is_finite**()

Returns whether this is a decomposition representing a finite matrix.

**Returns** Whether this is a decomposition representing a finite matrix.

**Return type** bool

**is_invertible**()

Returns whether this is a decomposition representing an invertible matrix.

**Returns** Whether this is a decomposition representing an invertible matrix.

**Return type** bool

**is_permuted**
Whether this is a decompositon with permutation.

**Type** bool

**is_positive_definite**()
Returns whether this is a decomposition of a positive definite matrix.

**Returns** Whether this is a decomposition of a positive definite matrix.

**Return type** bool

**is_positive_semidefinite**()
Returns whether this is a decomposition of a positive semi-definite matrix.

**Returns** Whether this is a decomposition of a positive semi-definite matrix.

**Return type** bool

**is_singular**()
Returns whether this is a decomposition representing a singular matrix.

**Returns** Whether this is a decomposition representing a singular matrix.

**Return type** bool

**is_sparse**()
Returns whether this is a decomposition of a sparse matrix.

**Returns** Whether this is a decomposition of a sparse matrix.

**Return type** bool

**is_type**(*type_str*)
Whether this decomposition is of the passed type.

**Parameters** **type_str** (*str*) – The decomposition type according to which is checked.

**Returns** Whether this is a decomposition of the passed type.

**Return type** bool

**load**(*filename*)
Loads a decomposition of this type.

**Parameters** **filename** (*str*) – Where the decomposition is saved.

**Raises**

- FileNotFoundError – If the files are not found in the passed directory.
- DecompositionInvalidDecompositionTypeFile – If the files contains another decomposition type.

**matrix_both_sides_multiplication**(*x*, *y=None*, *dtype=None*)
Calculates the both sides (matrix-matrix or matrix-vector) product *y.H @ A @ x*, where *A* is the composed matrix represented by this decomposition.

**Parameters**

- **x** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product *y.H @ A @ x*. It must hold *self.n == x.shape[0]*.

- **y** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product *y.H @ A @ x*. It must hold *self.n == y.shape[0]*. optional, default: If y is not passed, x is used as y.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** The result of *y.H @ A @ x*.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**matrix_right_side_multiplication**(*x*, *dtype=None*)

Calculates the right side (matrix-matrix or matrix-vector) product *A @ x*, where *A* is the composed matrix represented by this decomposition.

**Parameters**

- **x** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product in the matrix-matrix or matrix-vector *A @ x*. It must hold *self.n == x.shape[0]*.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** The result of *A @ x*.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**n**

The dimension of the squared decomposed matrix.

**Type** int

**p**

The permutation vector. *A[p[:, np.newaxis], p[np.newaxis, :]]* is the matrix *A* permuted by the permutation of the decomposition

**Type** numpy.ndarray

**p_inverse**

The permutation vector that undoes the permutation.

**Type** numpy.ndarray

**permute_matrix**(*A*)

Permutes a matrix by the permutation of the decomposition.

**Parameters** **A** (*numpy.ndarray or scipy.sparse.spmatrix*) – The matrix that should be permuted.

**Returns** The matrix *A* permuted by the permutation of the decomposition.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**save**(*filename*)

Saves this decomposition.

**Parameters** **filename** (*str*) – Where this decomposition should be saved.

**solve**(*b*, *dtype=None*)

Solves the equation *A x = b* regarding *x*, where *A* is the composed matrix represented by this decomposition.

**Parameters**

- **b** (*numpy.ndarray or scipy.sparse.spmatrix*) – Right-hand side vector or matrix in equation *A x = b*. It must hold *self.n == b.shape[0]*.

- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** An $x$ so that $A\ x = b$. The shape of $x$ matches the shape of $b$.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**Raises** *matrix.errors.DecompositionSingularError* – If this is a decomposition representing a singular matrix.

**type_str = 'LDL'**
The type of this decomposition represented as string.

**Type** str

**unpermute_matrix**(*A*)
Unpermutes a matrix permuted by the permutation of the decomposition.

**Parameters** **A** (*numpy.ndarray or scipy.sparse.spmatrix*) – The matrix that should be unpermuted.

**Returns** The matrix $A$ unpermuted by the permutation of the decomposition.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

### 4.2.3 LDL decomposition compressed

**class** matrix.decompositions.**LDL_DecompositionCompressed**(*LD=None*, *p=None*)
Bases: *matrix.decompositions.DecompositionBase*

A matrix decomposition where $LDL^H$ is the decomposed (permuted) matrix.

$L$ is a lower triangle matrix with ones on the diagonal. $D$ is a diagonal matrix. $L$ and $D$ are stored in one matrix whose diagonal values are the diagonal values of $D$ and whose off-diagonal values are those of $L$.

**Parameters**

- **LD** (*numpy.ndarray or scipy.sparse.spmatrix*) – A matrix whose diagonal values are the diagonal values of $D$ and whose off-diagonal values are those of $L$. optional, If it is not set yet, it must be set later.
- **p** (*numpy.ndarray*) – The permutation vector used for the decomposition. This decomposition is of *A[p[:, np.newaxis], p[np.newaxis, :]]* where $A$ is a matrix. optional, default: no permutation

**D**
The permutation matrix.

**Type** scipy.sparse.dia_matrix

**L**
The matrix $L$ of the decomposition.

**Type** numpy.matrix or scipy.sparse.spmatrix

**LD**
A matrix whose diagonal values are the diagonal values of $D$ and whose off-diagonal values are those of $L$.

**Type** numpy.matrix or scipy.sparse.spmatrix

**P**
The permutation matrix. *P @ A @ P.T* is the matrix $A$ permuted by the permutation of the decomposition

**Type** scipy.sparse.dok_matrix

**append_block_decomposition**(*dec*)
Makes a new decomposition where this decompsition and the passed one are appended.

**Parameters** **dec** (DecompositionBase) – The decomposition that should be appended to this decomposition.

**Returns** **dec** – A new decomposition where this decompsition and the passed one are appended. This desomposition represents the first block and the passed one the second block.

**Return type** *DecompositionBase*

**as_LDL_Decomposition**()

**as_any_type**(**type_strs*, *copy=False*)
Converts this decomposition to any of the passed types.

**Parameters**

- ***type_strs** (*str*) – The decomposition types to any of them this this decomposition is converted.
- **copy** (*bool*) – Whether the data of this decomposition should always be copied or only if needed.

**Returns** If the type of this decomposition is not in *type_strs*, a decomposition of type *type_str[0]* is returned which represents the same decomposed matrix as this decomposition. Otherwise this decomposition or a copy of it is returned, depending on *copy*.

**Return type** *matrix.decompositions.DecompositionBase*

**as_same_type**(*dec*, *copy=False*)
Converts the passed decompositions to the same type as this decomposition.

**Parameters**

- **doc** (DecompositionBase) – The decomposition that should be converted.
- **copy** (*bool*) – Whether the data of the decomposition that sould be converted should always be copied or only if needed.

**Returns** If the type of the passed decomposition is not same type as this decomposition, a decomposition of this type is returned which represents the same decomposed matrix as the passed decomposition. Otherwise this decomposition or a copy of it is returned, depending on *copy*.

**Return type** *matrix.decompositions.DecompositionBase*

**as_type**(*type_str*, *copy=False*)
Converts this decomposition to passed type.

**Parameters**

- **type_str** (*str*) – The decomposition type to which this decomposition is converted.
- **copy** (*bool*) – Whether the data of this decomposition should always be copied or only if needed.

**Returns** If the type of this decomposition is not *type_str*, a decomposition of type *type_str* is returned which represents the same decomposed matrix as this decomposition. Otherwise this decomposition or a copy of it is returned, depending on *copy*.

**Return type** *matrix.decompositions.DecompositionBase*

**check_finite**(*check_finite=True*)
Check if this is a decomposition representing a finite matrix.

**Parameters** **check_finite** (*bool*) – Whether to perform this check. default: True

**Raises** *matrix.errors.DecompositionNotFiniteError* – If this is a decomposition representing a non-finite matrix.

**check_invertible**()
Check if this is a decomposition representing an invertible matrix.

**Raises** *matrix.errors.DecompositionSingularError* – If this is a decomposition representing a singular matrix.

**composed_matrix**
The composed matrix represented by this decomposition.

**Type** numpy.matrix or scipy.sparse.spmatrix

**copy**()
Copies this decomposition.

**Returns** A copy of this decomposition.

**Return type** *matrix.decompositions.DecompositionBase*

**d**
The diagonal vector of the matrix *D* of the decomposition.

**Type** numpy.ndarray

**inverse_matrix_both_sides_multiplication**(*x*, *y=None*, *dtype=None*)
Calculates the both sides (matrix-matrix or matrix-vector) product *y.H @ B @ x*, where *B* is the mattrix inverse of the composed matrix represented by this decomposition.

**Parameters**

- **x** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product *y.H @ B @ x*. It must hold *self.n == x.shape[0]*.
- **y** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product *y.H @ B @ x*. It must hold *self.n == y.shape[0]*. optional, default: If y is not passed, x is used as y.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** The result of *y.H @ A @ x*.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**Raises** *matrix.errors.DecompositionSingularError* – If this is a decomposition representing a singular matrix.

**inverse_matrix_right_side_multiplication**(*x*, *dtype=None*)
Calculates the right side (matrix-matrix or matrix-vector) product *B @ x*, where *B* is the matrix inverse of the composed matrix represented by this decomposition.

**Parameters**

- **x** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product in the matrix-matrix or matrix-vector *B @ x*. It must hold *self.n == x.shape[0]*.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** The result of $B @ x$.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**Raises** *matrix.errors.DecompositionSingularError* – If this is a decomposition representing a singular matrix.

**is_almost_equal**(*other*, *rtol=0.0001*, *atol=1e-06*)
Whether this decomposition is close to passed decomposition.

**Parameters**

- **other** (*str*) – The decomposition which to compare to this decomposition.
- **rtol** (*float*) – The relative tolerance parameter.
- **atol** (*float*) – The absolute tolerance parameter.

**Returns** Whether this decomposition is close to passed decomposition.

**Return type** bool

**is_equal**(*other*)
Whether this decomposition is equal to passed decomposition.

**Parameters** **other** (*str*) – The decomposition which to compare to this decomposition.

**Returns** Whether this decomposition is equal to passed decomposition.

**Return type** bool

**is_finite**()
Returns whether this is a decomposition representing a finite matrix.

**Returns** Whether this is a decomposition representing a finite matrix.

**Return type** bool

**is_invertible**()
Returns whether this is a decomposition representing an invertible matrix.

**Returns** Whether this is a decomposition representing an invertible matrix.

**Return type** bool

**is_permuted**
Whether this is a decompositon with permutation.

**Type** bool

**is_positive_definite**()
Returns whether this is a decomposition of a positive definite matrix.

**Returns** Whether this is a decomposition of a positive definite matrix.

**Return type** bool

**is_positive_semidefinite**()
Returns whether this is a decomposition of a positive semi-definite matrix.

**Returns** Whether this is a decomposition of a positive semi-definite matrix.

**Return type** bool

**is_singular**()
Returns whether this is a decomposition representing a singular matrix.

**Returns** Whether this is a decomposition representing a singular matrix.

**Return type** bool

**is_sparse**()
Returns whether this is a decomposition of a sparse matrix.

**Returns** Whether this is a decomposition of a sparse matrix.

**Return type** bool

**is_type**(*type_str*)
Whether this decomposition is of the passed type.

**Parameters** **type_str** (*str*) – The decomposition type according to which is checked.

**Returns** Whether this is a decomposition of the passed type.

**Return type** bool

**load**(*filename*)
Loads a decomposition of this type.

**Parameters** **filename** (*str*) – Where the decomposition is saved.

**Raises**

- FileNotFoundError – If the files are not found in the passed directory.
- DecompositionInvalidDecompositionTypeFile – If the files contains another decomposition type.

**matrix_both_sides_multiplication**(*x*, *y=None*, *dtype=None*)
Calculates the both sides (matrix-matrix or matrix-vector) product *y.H @ A @ x*, where *A* is the composed matrix represented by this decomposition.

**Parameters**

- **x** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product *y.H @ A @ x*. It must hold *self.n == x.shape[0]*.
- **y** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product *y.H @ A @ x*. It must hold *self.n == y.shape[0]*. optional, default: If y is not passed, x is used as y.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** The result of *y.H @ A @ x*.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**matrix_right_side_multiplication**(*x*, *dtype=None*)
Calculates the right side (matrix-matrix or matrix-vector) product *A @ x*, where *A* is the composed matrix represented by this decomposition.

**Parameters**

- **x** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product in the matrix-matrix or matrix-vector *A @ x*. It must hold *self.n == x.shape[0]*.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** The result of *A @ x*.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**n**
The dimension of the squared decomposed matrix.

**Type** int

**p**
The permutation vector. *A[p[:, np.newaxis], p[np.newaxis, :]]* is the matrix *A* permuted by the permutation of the decomposition

**Type** numpy.ndarray

**p_inverse**
The permutation vector that undoes the permutation.

**Type** numpy.ndarray

**permute_matrix**(*A*)
Permutes a matrix by the permutation of the decomposition.

**Parameters** **A** (*numpy.ndarray or scipy.sparse.spmatrix*) – The matrix that should be permuted.

**Returns** The matrix *A* permuted by the permutation of the decomposition.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**save**(*filename*)
Saves this decomposition.

**Parameters** **filename** (*str*) – Where this decomposition should be saved.

**solve**(*b*, *dtype=None*)
Solves the equation *A x = b* regarding *x*, where *A* is the composed matrix represented by this decomposition.

**Parameters**

- **b** (*numpy.ndarray or scipy.sparse.spmatrix*) – Right-hand side vector or matrix in equation *A x = b*. It must hold *self.n == b.shape[0]*.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** An *x* so that *A x = b*. The shape of *x* matches the shape of *b*.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**Raises** *matrix.errors.DecompositionSingularError* – If this is a decomposition representing a singular matrix.

**type_str = 'LDL_compressed'**
The type of this decomposition represented as string.

**Type** str

**unpermute_matrix**(*A*)
Unpermutes a matrix permuted by the permutation of the decomposition.

**Parameters** **A** (*numpy.ndarray or scipy.sparse.spmatrix*) – The matrix that should be unpermuted.

**Returns** The matrix *A* unpermuted by the permutation of the decomposition.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

### 4.2.4 Base decomposition

**class** matrix.decompositions.**DecompositionBase**(*p=None*)
Bases: object

A matrix decomposition.

This class is a base class for all other matrix decompositions.

**Parameters** **p** (*numpy.ndarray*) – The permutation vector used for the decomposition. This decomposition is of *A[p[:, np.newaxis], p[np.newaxis, :]]* where *A* is a matrix. optional, default: no permutation

**P**
The permutation matrix. *P @ A @ P.T* is the matrix *A* permuted by the permutation of the decomposition

**Type** scipy.sparse.dok_matrix

**append_block_decomposition**(*dec*)
Makes a new decomposition where this decompsition and the passed one are appended.

**Parameters** **dec** (DecompositionBase) – The decomposition that should be appended to this decomposition.

**Returns** **dec** – A new decomposition where this decompsition and the passed one are appended. This desomposition represents the first block and the passed one the second block.

**Return type** *DecompositionBase*

**as_any_type**(**type_strs*, *copy=False*)
Converts this decomposition to any of the passed types.

**Parameters**

- ***type_strs** (*str*) – The decomposition types to any of them this this decomposition is converted.
- **copy** (*bool*) – Whether the data of this decomposition should always be copied or only if needed.

**Returns** If the type of this decomposition is not in *type_strs*, a decomposition of type *type_str[0]* is returned which represents the same decomposed matrix as this decomposition. Otherwise this decomposition or a copy of it is returned, depending on *copy*.

**Return type** *matrix.decompositions.DecompositionBase*

**as_same_type**(*dec*, *copy=False*)
Converts the passed decompositions to the same type as this decomposition.

**Parameters**

- **doc** (DecompositionBase) – The decomposition that should be converted.
- **copy** (*bool*) – Whether the data of the decomposition that sould be converted should always be copied or only if needed.

**Returns** If the type of the passed decomposition is not same type as this decomposition, a decomposition of this type is returned which represents the same decomposed matrix as the passed decomposition. Otherwise this decomposition or a copy of it is returned, depending on *copy*.

**Return type** *matrix.decompositions.DecompositionBase*

**as_type**(*type_str*, *copy=False*)
Converts this decomposition to passed type.

**Parameters**

- **type_str** (*str*) – The decomposition type to which this decomposition is converted.
- **copy** (*bool*) – Whether the data of this decomposition should always be copied or only if needed.

**Returns** If the type of this decomposition is not *type_str*, a decomposition of type *type_str* is returned which represents the same decomposed matrix as this decomposition. Otherwise this decomposition or a copy of it is returned, depending on *copy*.

**Return type** *matrix.decompositions.DecompositionBase*

**check_finite**(*check_finite=True*)
Check if this is a decomposition representing a finite matrix.

**Parameters** **check_finite** (*bool*) – Whether to perform this check. default: True

**Raises** *matrix.errors.DecompositionNotFiniteError* – If this is a decomposition representing a non-finite matrix.

**check_invertible**()
Check if this is a decomposition representing an invertible matrix.

**Raises** *matrix.errors.DecompositionSingularError* – If this is a decomposition representing a singular matrix.

**composed_matrix**
The composed matrix represented by this decomposition.

**Type** numpy.matrix or scipy.sparse.spmatrix

**copy**()
Copies this decomposition.

**Returns** A copy of this decomposition.

**Return type** *matrix.decompositions.DecompositionBase*

**inverse_matrix_both_sides_multiplication**(*x*, *y=None*, *dtype=None*)
Calculates the both sides (matrix-matrix or matrix-vector) product *y.H @ B @ x*, where *B* is the mattrix inverse of the composed matrix represented by this decomposition.

**Parameters**

- **x** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product *y.H @ B @ x*. It must hold *self.n == x.shape[0]*.
- **y** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product *y.H @ B @ x*. It must hold *self.n == y.shape[0]*. optional, default: If y is not passed, x is used as y.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** The result of *y.H @ A @ x*.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**Raises** *matrix.errors.DecompositionSingularError* – If this is a decomposition representing a singular matrix.

**inverse_matrix_right_side_multiplication**(*x*, *dtype=None*)
Calculates the right side (matrix-matrix or matrix-vector) product *B @ x*, where *B* is the matrix inverse of the composed matrix represented by this decomposition.

**Parameters**

- **x** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product in the matrix-matrix or matrix-vector *B @ x*. It must hold *self.n == x.shape[0]*.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** The result of *B @ x*.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**Raises** *matrix.errors.DecompositionSingularError* – If this is a decomposition representing a singular matrix.

**is_almost_equal**(*other*, *rtol=0.0001*, *atol=1e-06*)

Whether this decomposition is close to passed decomposition.

**Parameters**

- **other** (*str*) – The decomposition which to compare to this decomposition.
- **rtol** (*float*) – The relative tolerance parameter.
- **atol** (*float*) – The absolute tolerance parameter.

**Returns** Whether this decomposition is close to passed decomposition.

**Return type** bool

**is_equal**(*other*)

Whether this decomposition is equal to passed decomposition.

**Parameters** **other** (*str*) – The decomposition which to compare to this decomposition.

**Returns** Whether this decomposition is equal to passed decomposition.

**Return type** bool

**is_finite**()

Returns whether this is a decomposition representing a finite matrix.

**Returns** Whether this is a decomposition representing a finite matrix.

**Return type** bool

**is_invertible**()

Returns whether this is a decomposition representing an invertible matrix.

**Returns** Whether this is a decomposition representing an invertible matrix.

**Return type** bool

**is_permuted**

Whether this is a decompositon with permutation.

**Type** bool

**is_positive_definite**()

Returns whether this is a decomposition of a positive definite matrix.

**Returns** Whether this is a decomposition of a positive definite matrix.

**Return type** bool

**is_positive_semidefinite**()

Returns whether this is a decomposition of a positive semi-definite matrix.

**Returns** Whether this is a decomposition of a positive semi-definite matrix.

**Return type** bool

**is_singular**()
Returns whether this is a decomposition representing a singular matrix.

**Returns** Whether this is a decomposition representing a singular matrix.

**Return type** bool

**is_sparse**()
Returns whether this is a decomposition of a sparse matrix.

**Returns** Whether this is a decomposition of a sparse matrix.

**Return type** bool

**is_type**(*type_str*)
Whether this decomposition is of the passed type.

**Parameters** **type_str** (*str*) – The decomposition type according to which is checked.

**Returns** Whether this is a decomposition of the passed type.

**Return type** bool

**load**(*filename*)
Loads a decomposition of this type.

**Parameters** **filename** (*str*) – Where the decomposition is saved.

**Raises**

- FileNotFoundError – If the files are not found in the passed directory.
- DecompositionInvalidDecompositionTypeFile – If the files contains another decomposition type.

**matrix_both_sides_multiplication**(*x*, *y=None*, *dtype=None*)
Calculates the both sides (matrix-matrix or matrix-vector) product *y.H @ A @ x*, where *A* is the composed matrix represented by this decomposition.

**Parameters**

- **x** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product *y.H @ A @ x*. It must hold *self.n == x.shape[0]*.
- **y** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product *y.H @ A @ x*. It must hold *self.n == y.shape[0]*. optional, default: If y is not passed, x is used as y.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** The result of *y.H @ A @ x*.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**matrix_right_side_multiplication**(*x*, *dtype=None*)
Calculates the right side (matrix-matrix or matrix-vector) product *A @ x*, where *A* is the composed matrix represented by this decomposition.

**Parameters**

- **x** (*numpy.ndarray or scipy.sparse.spmatrix*) – Vector or matrix in the product in the matrix-matrix or matrix-vector *A @ x*. It must hold *self.n == x.shape[0]*.

- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** The result of *A* @ *x*.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**n**
The dimension of the squared decomposed matrix.

**Type** int

**p**
The permutation vector. *A[p[:, np.newaxis], p[np.newaxis, :]]* is the matrix *A* permuted by the permutation of the decomposition

**Type** numpy.ndarray

**p_inverse**
The permutation vector that undoes the permutation.

**Type** numpy.ndarray

**permute_matrix**(*A*)
Permutes a matrix by the permutation of the decomposition.

**Parameters** **A** (*numpy.ndarray or scipy.sparse.spmatrix*) – The matrix that should be permuted.

**Returns** The matrix *A* permuted by the permutation of the decomposition.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**save**(*filename*)
Saves this decomposition.

**Parameters** **filename** (*str*) – Where this decomposition should be saved.

**solve**(*b*, *dtype=None*)
Solves the equation *A x = b* regarding *x*, where *A* is the composed matrix represented by this decomposition.

**Parameters**

- **b** (*numpy.ndarray or scipy.sparse.spmatrix*) – Right-hand side vector or matrix in equation *A x = b*. It must hold *self.n == b.shape[0]*.
- **dtype** (*numpy.dtype*) – Type to use in computation. optional, default: Determined by the method.

**Returns** An *x* so that *A x = b*. The shape of *x* matches the shape of *b*.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

**Raises** *matrix.errors.DecompositionSingularError* – If this is a decomposition representing a singular matrix.

**type_str = 'base'**
The type of this decomposition represented as string.

**Type** str

**unpermute_matrix**(*A*)
Unpermutes a matrix permuted by the permutation of the decomposition.

**Parameters** **A** (*numpy.ndarray or scipy.sparse.spmatrix*) – The matrix that should be unpermuted.

**Returns** The matrix *A* unpermuted by the permutation of the decomposition.

**Return type** numpy.ndarray or scipy.sparse.spmatrix

## 4.3 Errors

This is an overview about the exceptions that could arise in this library. They are available in *matrix.errors*:

The following exception is the base exception from which all other exceptions in this package are derived:

### 4.3.1 BaseError

**exception** matrix.errors.**BaseError**(*message*)

Bases: Exception

This is the base exception for all exceptions in this package.

If a matrix has an invalid properties, the following exceptions can occur:

### 4.3.2 MatrixError

**exception** matrix.errors.**MatrixError**(*matrix*, *message=None*)

Bases: *matrix.errors.BaseError*

An exception related to a matrix.

### 4.3.3 MatrixNotSquareError

**exception** matrix.errors.**MatrixNotSquareError**(*matrix*)

Bases: *matrix.errors.MatrixError*

A matrix is not a square matrix although a square matrix is required.

### 4.3.4 MatrixNotFiniteError

**exception** matrix.errors.**MatrixNotFiniteError**(*matrix*)

Bases: *matrix.errors.MatrixError*

A matrix has non-finite entries although a finite matrix is required.

### 4.3.5 MatrixSingularError

**exception** matrix.errors.**MatrixSingularError**(*matrix*)

Bases: *matrix.errors.MatrixError*

A matrix is singular although an invertible matrix is required.

### 4.3.6 MatrixNotHermitianError

**exception** matrix.errors.**MatrixNotHermitianError**(*matrix*, *i=None*, *j=None*)
Bases: *matrix.errors.MatrixError*

A matrix is not Hermitian although a Hermitian matrix is required.

### 4.3.7 MatrixComplexDiagonalValueError

**exception** matrix.errors.**MatrixComplexDiagonalValueError**(*matrix*, *i=None*)
Bases: *matrix.errors.MatrixNotHermitianError*

A matrix has complex diagonal values although real diagonal values are required.

If a desired decomposition is not computable, the following exceptions can be raised:

### 4.3.8 NoDecompositionPossibleError

**exception** matrix.errors.**NoDecompositionPossibleError**(*base*, *desired_type*)
Bases: *matrix.errors.BaseError*

It is not possible to calculate a desired matrix decomposition.

### 4.3.9 NoDecompositionPossibleWithProblematicSubdecompositionError

**exception** matrix.errors.**NoDecompositionPossibleWithProblematicSubdecompositionError**(*base*, *desired_type*, *problematic_lead*, *subdecomposition=Non*)
Bases: *matrix.errors.NoDecompositionPossibleError*

It is not possible to calculate a desired matrix decomposition. Only a subdecompostion could be calculated

### 4.3.10 NoDecompositionPossibleTooManyEntriesError

**exception** matrix.errors.**NoDecompositionPossibleTooManyEntriesError**(*matrix*, *desired_type*)
Bases: *matrix.errors.NoDecompositionPossibleError*

The decomposition is not possible for this matrix because it would have too many entries.

### 4.3.11 NoDecompositionConversionImplementedError

**exception** matrix.errors.**NoDecompositionConversionImplementedError**(*decomposition*, *desired_type*)

Bases: *matrix.errors.NoDecompositionPossibleError*

A decomposition conversion is not implemented for this type.

If the matrix, represented by a decomposition, has an invalid characteristic, the following exceptions can occur:

### 4.3.12 DecompositionError

**exception** matrix.errors.**DecompositionError**(*decomposition*, *message=None*)

Bases: *matrix.errors.BaseError*

An exception related to a decomposition.

### 4.3.13 DecompositionNotFiniteError

**exception** matrix.errors.**DecompositionNotFiniteError**(*decomposition*)

Bases: *matrix.errors.DecompositionError*

A decomposition of a matrix has non-finite entries although a finite matrix is required.

### 4.3.14 DecompositionSingularError

**exception** matrix.errors.**DecompositionSingularError**(*decomposition*)

Bases: *matrix.errors.DecompositionError*

A decomposition represents a singular matrix although a non-singular matrix is required.

If a decomposition could not be loaded from a file, the following exceptions can be raised:

### 4.3.15 DecompositionInvalidFile

**exception** matrix.errors.**DecompositionInvalidFile**(*filename*)

Bases: *matrix.errors.DecompositionError*, OSError

An attempt was made to load a decomposition from an invalid file.

### 4.3.16 DecompositionInvalidDecompositionTypeFile

**exception** matrix.errors.**DecompositionInvalidDecompositionTypeFile**(*filename*, *type_file*, *type_needed*)

Bases: *matrix.errors.DecompositionInvalidFile*

An attempt was made to load a decomposition from an file in which another decomposition type is stored.

If the computation needs more iterations than the maximal number of iterations, the following exception occurs:

### 4.3.17 TooManyIterationsError

**exception** matrix.errors.**TooManyIterationsError**(*message=None*, *iteration=None*, *result=None*)

Bases: *matrix.errors.BaseError*

Too many iterations are needed for the calculation.

## 4.4 Changelog

### 4.4.1 1.2

- Several functions to solve nearness problems (nearest symmetric matrix, nearest skew symmetric matrix, nearest positive semidefinite matrix, nearest correlation matrix) added.
- Significant speedup of approximation algorithms in *matrix.approximation.positive_semidefinite.Reimer*.
- Overflow handling improved in several functions.
- Decompositions in matrix.decompositions now have an *append_block_decomposition* and *as_same_type* method and their mutiplication methods have now a *dtype* argument.
- The approximation algorithms in *matrix.approximation.positive_semidefinite.Reimer* now have a *min_abs_value_L* argument.

### 4.4.2 1.1

- Positive semidefinite approximation algorithms of GMW and SE type have been added.
- Permutation method, with numerical stability as main focus, has been added to positive semidefinite approximation algorithm.
- Positive semidefinite approximation algorithm are moved into separate package. (*matrix.approximate* to *matrix.approximation.positive_semidefinite*)

### 4.4.3 1.0.1

- Approximation functions now also work if an overflows occurs.
- NumPys matrix is avoided because it is deprecated now.

### 4.4.4 1.0

- Approximation functions are slightly faster now.
- Better overflow handling is now used in approximation functions.
- Prebuild html documentation are now included.
- Function for approximating a matrix by a positive semidefinite matrix (*matrix.approximate.positive_semidefinite_matrix*) has been removed.

### 4.4.5 0.8

- Approximation functions have been replaced by more sophisticated approximation functions.
- Explicit function for approximating a matrix by a positive (semi)definite matrix has been added.
- Universal save and load functions have been added.
- Decompositions have obtained is_equal and is_almost_equal methods.
- Functions to multiply the matrix, represented by a decomposition, or its inverse with a matrix or a vector have been added.
- It is now possible to pass permutation vectors to approximate and decompose methods.

### 4.4.6 0.7

- Lineare systems associated to matrices or decompositions can now be solved.
- Invertibility of matrices and decompositions can now be examined.
- Decompositions can now be examined to see if they contain only finite values.

### 4.4.7 0.6

- Decompositions are now saveable and loadable.

### 4.4.8 0.5

- Matrices can now be approximated by decompositions.

### 4.4.9 0.4

- Positive definiteness and positive semi-definiteness of matrices and decompositions can now be examined.

### 4.4.10 0.3

- Dense and sparse matrices are now decomposable into several types (LL, LDL, LDL compressed).

### 4.4.11 0.2

- Decompositons are now convertable to other decompositon types.
- Decompositions are now comparable.

### 4.4.12 0.1

- Several decompositions types (LL, LDL, LDL compressed) have been added.
- Several permutation capabilities have been added.

## 4.5 Requirements

- Python >= 3.7
- NumPy >= 1.15
- SciPy >= 0.19
- scikit-sparse >= 0.4.2 (only for sparse matrix support)

CHAPTER 5

# Indices and tables

- genindex
- search

CHAPTER 6

# Copyright

Copyright (C) 2017-2019 Joscha Reimer jor@informatik.uni-kiel.de

This program is free software: you can redistribute it and/or modify it under the terms of the GNU Affero General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Affero General Public License for more details.

You should have received a copy of the GNU Affero General Public License along with this program. If not, see <http://www.gnu.org/licenses/>.

# Python Module Index

## m

# Index

## A



## B



## C

## D



## G



## I

## L

## M



## N



## P



## S

## T



## U