PyMatrixID Documentation

Release 0.1

Kenneth L. Ho

February 09, 2014

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

An interpolative decomposition (ID) of a matrix $A \in \mathbb{C}^{m \times n}$ of rank $k \leq \min\{m, n\}$ is a factorization

$$A\Pi = \begin{bmatrix} A\Pi_1 & A\Pi_2 \end{bmatrix} = A\Pi_1 \begin{bmatrix} I & T \end{bmatrix},$$

where $\Pi = [\Pi_1, \Pi_2]$ is a permutation matrix with $\Pi_1 \in \{0, 1\}^{n \times k}$, i.e., $A\Pi_2 = A\Pi_1 T$. This can equivalently be written as $A = BP$, where $B = A\Pi_1$ and $P = [I, T]\Pi^{\mathsf{T}}$ are the *skeleton* and *interpolation matrices*, respectively.

If A does not have exact rank k, then there exists an approximation in the form of an ID such that $A = BP + E$, where $\|E\| \sim \sigma_{k+1}$ is on the order of the $(k+1)$-th largest singular value of A. Note that σ_{k+1} is the best possible error for a rank-k approximation and, in fact, is achieved by the singular value decomposition (SVD) $A \approx USV^*$, where $U \in \mathbb{C}^{m \times k}$ and $V \in \mathbb{C}^{n \times k}$ have orthonormal columns and $S = \mathrm{diag}(\sigma_i) \in \mathbb{C}^{k \times k}$ is diagonal with nonnegative entries. The principal advantages of using an ID over an SVD are that:

- it is cheaper to construct;
- it preserves the structure of A; and
- it is more efficient to compute with in light of the identity submatrix of P.

Note: PyMatrixID has been merged into SciPy 0.13 (in a slightly modified form) as `scipy.linalg.interpolative` by Andreas Klöckner and Pauli Virtanen. It is highly recommended to henceforth use that package instead.

1.1 Overview

The ID software package [1] by Martinsson, Rokhlin, Shkolnisky, and Tygert is a Fortran library to compute IDs using various algorithms, including the deterministic pivoted QR approach of [2] and the more recent randomized methods described in [3], [4], and [5]. PyMatrixID is a Python wrapper for this package that exposes its functionality in a more convenient manner. Note that PyMatrixID does not add any functionality beyond that of organizing a simpler and more consistent interface.

[1] P.G. Martinsson, V. Rokhlin, Y. Shkolnisky, M. Tygert. ID: a software package for low-rank approximation of matrices via interpolative decompositions, version 0.3. http://cims.nyu.edu/~tygert/id_doc.pdf.

[2] H. Cheng, Z. Gimbutas, P.G. Martinsson, V. Rokhlin. On the compression of low rank matrices. *SIAM J. Sci. Comput.* 26 (4): 1389–1404, 2005. doi:10.1137/030602678.

[3] E. Liberty, F. Woolfe, P.G. Martinsson, V. Rokhlin, M. Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proc. Natl. Acad. Sci. U.S.A.* 104 (51): 20167–20172, 2007. doi:10.1073/pnas.0709640104.

[4] P.G. Martinsson, V. Rokhlin, M. Tygert. A randomized algorithm for the decomposition of matrices. *Appl. Comput. Harmon. Anal.* 30 (1): 47–68, 2011. doi:10.1016/j.acha.2010.02.003.

[5] F. Woolfe, E. Liberty, V. Rokhlin, M. Tygert. A fast randomized algorithm for the approximation of matrices. *Appl. Comput. Harmon. Anal.* 25 (3): 335–366, 2008. doi:10.1016/j.acha.2007.12.002.

We advise the user to consult also the documentation for the ID package, which is included in full as part of PyMatrixID.

1.2 Licensing and availability

PyMatrixID is freely available under the BSD license and can be downloaded at https://github.com/klho/PyMatrixID. To request alternate licenses, please contact the author.

PyMatrixID also distributes the ID software package, which is likewise released under the BSD license.

1.3 References

CHAPTER

TWO

INSTALLING

This section describes how to compile and install PyMatrixID on Unix-like systems. Primary prerequisites include Git, GNU Make, a Fortran compiler such as GFortran, F2PY, Python, and NumPy. Secondary prerequisites include Sphinx and LaTeX for the documentation.

PyMatrixID has only been tested using GFortran; the use of all other compilers should be considered "at your own risk" (though they should really be fine).

2.1 Code repository

All source files for PyMatrixID (including those for this documentation) are available at https://github.com/klho/PyMatrixID. To download PyMatrixID using Git, type the following command at the shell prompt:

```
$ git clone https://github.com/klho/PyMatrixID /path/to/local/repository/
```

2.2 Compiling

There are several targets available to compile, namely:

- the Python wrapper;
- the ID package; and
- this documentation.

To see all available targets, switch the working directory to the root of the local repository and type:

```
$ make help
```

Hopefully the instructions are self-explanatory; for more explicit directions, please see below. Before beginning, view and edit the file `Makefile` to ensure that all options are properly set for your system. In particular, if you will not be using GFortran, be sure to set an alternate compiler as appropriate.

To compile the Python wrapper, type:

```
$ make
```

or:

```
$ make all
```

or:

```
$ make python
```

This creates the F2PY-ed library `bin/id_dist.so`.

To compile the ID package, type:

```
$ make id_dist
```

It is not necessary to compile the ID package in order to use the Python wrapper; the required binaries are created automatically by F2PY. However, compiling the ID package itself may be useful, for example, to test the library independently. The ID package is located in the directory `external/id_dist`.

To compile the documentation files, type:

```
$ make doc
```

Output HTML and PDF files are placed in the directory `doc`.

2.3 Driver program

PyMatrixID also contains a driver program to demonstrate its use. To run the driver, type:

```
$ make driver
```

The driver program is discussed in more detail in *Tutorial*.

CHAPTER

THREE

TUTORIAL

We now present a tutorial on using PyMatrixID. From here on, we will therefore assume that PyMatrixID has been properly installed; if this is not the case, please go back to *Installing*.

3.1 Overview

The Python interface is located in the directory `python`, which contains the directory `pymatrixid`, organizing the main Python package, and `id_dist.so`, the F2PY-ed Fortran library. The file `id_dist.so` contains all wrapped routines and is imported by `pymatrixid.backend`, which in turn is imported by `pymatrixid` to create a more convenient interface around it. For details on the Python modules, please see the *Python API*.

We will now step through the process of using PyMatrixID, following the driver program as a guide.

3.2 Initializing

The first step is to import `pymatrixid` by issuing the command:

```
>>> import pymatrixid
```

at the Python prompt. This should work if you are in the `python` directory; otherwise, you may have to first type something like:

```
>>> import sys
>>> sys.path.append(/path/to/pymatrixid/python/)
```

in order to tell Python where to look.

Now let's build a matrix. For this, we consider a Hilbert matrix, which is well known to have low rank:

```
>>> from scipy.linalg import hilbert
>>> n = 1000
>>> A = hilbert(n)
```

We can also do this explicitly via:

```
>>> import numpy as np
>>> n = 1000
>>> A = np.empty((n, n), order='F')
>>> for j in range(n):
>>>     for i in range(m):
>>>         A[i,j] = 1. / (i + j + 1)
```

Note the use of the flag `order='F'` in `numpy.empty()`. This instantiates the matrix in Fortran-contiguous order and is important for avoiding data copying when passing to the backend.

We then define multiplication routines for the matrix by regarding it as a `scipy.sparse.linalg.LinearOperator`:

```
>>> from scipy.sparse.linalg import aslinearoperator
>>> L = aslinearoperator(A)
```

This automatically sets up methods describing the action of the matrix and its adjoint on a vector.

3.3 Computing an ID

We have several choices of algorithm to compute an ID. These fall largely according to two dichotomies:

1. how the matrix is represented, i.e., via its entries or via its action on a vector; and
2. whether to approximate it to a fixed relative precision or to a fixed rank.

We step through each choice in turn below.

In all cases, the ID is represented by three parameters:

1. a rank `k`;
2. an index array `idx`; and
3. interpolation coefficients `proj`.

The ID is specified by the relation `np.dot(A[:,idx[:k]], proj) = A[:,idx[k:]]`.

3.3.1 From matrix entries

We first consider a matrix given in terms of its entries.

To compute an ID to a fixed precision, type:

```
>>> k, idx, proj = pymatrixid.interp_decomp(A, eps)
```

where `eps < 1` is the desired precision.

To compute an ID to a fixed rank, use:

```
>>> idx, proj = pymatrixid.interp_decomp(A, k)
```

where `k >= 1` is the desired rank.

Both algorithms use random sampling and are usually faster than the corresponding older, deterministic algorithms, which can be accessed via the commands:

```
>>> k, idx, proj = pymatrixid.interp_decomp(A, eps, rand=False)
```

and:

```
>>> idx, proj = pymatrixid.interp_decomp(A, k, rand=False)
```

respectively.

3.3.2 From matrix action

Now consider a matrix given in terms of its action on a vector as a `scipy.sparse.linalg.LinearOperator`.

To compute an ID to a fixed precision, type:

```
>>> k, idx, proj = pymatrixid.interp_decomp(L, eps)
```

To compute an ID to a fixed rank, use:

```
>>> idx, proj = pymatrixid.interp_decomp(L, k)
```

These algorithms are randomized.

3.4 Reconstructing an ID

The ID routines above do not output the skeleton and interpolation matrices explicitly but instead return the relevant information in a more compact (and sometimes more useful) form. To build these matrices, write:

```
>>> B = pymatrixid.reconstruct_skel_matrix(A, k, idx)
```

for the skeleton matrix and:

```
>>> P = pymatrixid.reconstruct_interp_matrix(idx, proj)
```

for the interpolation matrix. The ID approximation can then be computed as:

```
>>> C = np.dot(B, P)
```

This can also be constructed using:

```
>>> C = pymatrixid.reconstruct_matrix_from_id(B, idx, proj)
```

without having to first compute `P`.

Alternatively, this can be done explicitly as well using:

```
B = A[:,idx[:k]]
P = np.hstack([np.eye(k), proj])[:,np.argsort(idx)]
C = np.dot(B, P)
```

3.5 Computing an SVD

An ID can be converted to an SVD via the command:

```
>>> U, S, V = pymatrixid.id_to_svd(B, idx, proj)
```

The SVD approximation is then:

```
>>> C = np.dot(U, np.dot(np.diag(S), np.dot(V.conj().T)))
```

The SVD can also be computed "fresh" by combining both the ID and conversion steps into one command. Following the various ID algorithms above, there are correspondingly various SVD algorithms that one can employ.

3.5.1 From matrix entries

We consider first SVD algorithms for a matrix given in terms of its entries.

To compute an SVD to a fixed precision, type:

```
>>> U, S, V = pymatrixid.svd(A, eps)
```

To compute an SVD to a fixed rank, use:

```
>>> U, S, V = pymatrixid.svd(A, k)
```

Both algorithms use random sampling; for the determinstic versions, issue the keyword `rand=False` as above.

3.5.2 From matrix action

Now consider a matrix given in terms of its action on a vector.

To compute an SVD to a fixed precision, type:

```
>>> U, S, V = pymatrixid.svd(L, eps)
```

To compute an SVD to a fixed rank, use:

```
>>> U, S, V = pymatrixid.svd(L, k)
```

3.6 Utility routines

Several utility routines are also available.

To estimate the spectral norm of a matrix, use:

```
>>> snorm = pymatrixid.estimate_spectral_norm(A)
```

This algorithm is based on the randomized power method and thus requires only matrix-vector products. The number of iterations to take can be set using the keyword `its` (default: `its=20`). The matrix is interpreted as a `scipy.sparse.linalg.LinearOperator`, but it is also valid to supply it as a `numpy.ndarray`, in which case it is trivially converted using `scipy.sparse.linalg.aslinearoperator()`.

The same algorithm can also estimate the spectral norm of the difference of two matrices `A1` and `A2` as follows:

```
>>> diff = pymatrixid.estimate_spectral_norm_diff(A1, A2)
```

This is often useful for checking the accuracy of a matrix approximation.

Some routines in `pymatrixid` require estimating the rank of a matrix as well. This can be done with either:

```
>>> k = pymatrixid.estimate_rank(A, eps)
```

or:

```
>>> k = pymatrixid.estimate_rank(L, eps)
```

depending on the representation. The parameter `eps` controls the definition of the numerical rank.

Finally, the random number generation required for all randomized routines can be controlled via `pymatrixid.rand()`. To reset the seed values to their original values, use:

```
>>> pymatrixid.rand()
```

To specify the seed values, use:

```
>>> pymatrixid.rand(s)
```

where `s` must be an array of 55 floats. To simply generate some random numbers, type:

```
>>> pymatrixid.rand(n)
```

where `n` is the number of random numbers to generate.

3.7 Remarks

The above functions all automatically detect the appropriate interface and work with both real and complex data types, passing input arguments to the proper backend routine.

All backend functions can be accessed via the `pymatrixid.backend` module, which wraps the Fortran functions directly, perhaps with some minor simplification.

CHAPTER

FOUR

PYTHON API

This section provides the auto-generated API for the Python modules `pymatrixid` and `pymatrixid.backend`. The functions in `pymatrixid.backend` are wrappers for the routines of the same name in the Fortran ID software package; for details, please consult the Fortran source code (in `external/id_dist/src`).

4.1 pymatrixid

Python module for interfacing with *id_dist*.

`pymatrixid.`**`estimate_rank`**(*A*, *eps*)

Estimate matrix rank to a specified relative precision using randomized methods.

The matrix *A* can be given as either a `numpy.ndarray` or a `scipy.sparse.linalg.LinearOperator`, with different algorithms used for each case. If *A* is of type `numpy.ndarray`, then the output rank is typically about 8 higher than the actual numerical rank.

This function automatically detects the form of the input parameters and passes them to the appropriate backend. For details, see `backend.idd_estrank()`, `backend.idd_findrank()`, `backend.idz_estrank()`, and `backend.idz_findrank()`.

Parameters

- **A** (`numpy.ndarray` or `scipy.sparse.linalg.LinearOperator`) – Matrix whose rank is to be estimated, given as either a `numpy.ndarray` or a `scipy.sparse.linalg.LinearOperator` with the *rmatvec* method (to apply the matrix adjoint).
- **eps** (*float*) – Relative error for numerical rank definition.

Returns Estimated matrix rank.

Return type int

`pymatrixid.`**`estimate_spectral_norm`**(*A*, *its=20*)

Estimate spectral norm of a matrix by the randomized power method.

This function automatically detects the matrix data type and calls the appropriate backend. For details, see `backend.idd_snorm()` and `backend.idz_snorm()`.

Parameters

- **A** (`scipy.sparse.linalg.LinearOperator`) – Matrix given as a `scipy.sparse.linalg.LinearOperator` with the *matvec* and *rmatvec* methods (to apply the matrix and its adjoint).
- **its** (*int*) – Number of power method iterations.

Returns Spectral norm estimate.

Return type float

pymatrixid.**estimate_spectral_norm_diff**(*A*, *B*, *its=20*)

Estimate spectral norm of the difference of two matrices by the randomized power method.

This function automatically detects the matrix data type and calls the appropriate backend. For details, see `backend.idd_diffsnorm()` and `backend.idz_diffsnorm()`.

Parameters

- **A** (`scipy.sparse.linalg.LinearOperator`) – First matrix given as a `scipy.sparse.linalg.LinearOperator` with the *matvec* and *rmatvec* methods (to apply the matrix and its adjoint).
- **B** (`scipy.sparse.linalg.LinearOperator`) – Second matrix given as a `scipy.sparse.linalg.LinearOperator` with the *matvec* and *rmatvec* methods (to apply the matrix and its adjoint).
- **its** (*int*) – Number of power method iterations.

Returns Spectral norm estimate of matrix difference.

Return type float

pymatrixid.**id_to_svd**(*B*, *idx*, *proj*)

Convert ID to SVD.

The SVD reconstruction of a matrix with skeleton matrix *B* and ID indices and coefficients *idx* and *proj*, respectively, is:

```
U, S, V = id_to_svd(B, idx, proj)
A = numpy.dot(U, numpy.dot(numpy.diag(S), V.conj().T))
```

See also `svd()`.

This function automatically detects the matrix data type and calls the appropriate backend. For details, see `backend.idd_id2svd()` and `backend.idz_id2svd()`.

Parameters

- **B** (`numpy.ndarray`) – Skeleton matrix.
- **idx** (`numpy.ndarray`) – Column index array.
- **proj** (`numpy.ndarray`) – Interpolation coefficients.

Returns Left singular vectors.

Return type `numpy.ndarray`

Returns Singular values.

Return type `numpy.ndarray`

Returns Right singular vectors.

Return type `numpy.ndarray`

pymatrixid.**interp_decomp**(*A*, *eps_or_k*, *rand=True*)

Compute ID of a matrix.

An ID of a matrix *A* is a factorization defined by a rank *k*, a column index array *idx*, and interpolation coefficients *proj* such that:

```
numpy.dot(A[:,idx[:k]], proj) = A[:,idx[k:]]
```

The original matrix can then be reconstructed as:

```
numpy.hstack([A[:,idx[:k]],
              numpy.dot(A[:,idx[:k]], proj)]
             )[:,numpy.argsort(idx)]
```

or via the routine `reconstruct_matrix_from_id()`. This can equivalently be written as:

```
numpy.dot(A[:,idx[:k]],
          numpy.hstack([numpy.eye(k), proj])
         )[:,np.argsort(idx)]
```

in terms of the skeleton and interpolation matrices:

```
B = A[:,idx[:k]]
```

and:

```
P = numpy.hstack([numpy.eye(k), proj])[:,np.argsort(idx)]
```

respectively. See also `reconstruct_interp_matrix()` and `reconstruct_skel_matrix()`.

The ID can be computed to any relative precision or rank (depending on the value of *eps_or_k*). If a precision is specified (*eps_or_k* < *1*), then this function has the output signature:

```
k, idx, proj = interp_decomp(A, eps_or_k)
```

Otherwise, if a rank is specified (*eps_or_k* >= *1*), then the output signature is:

```
idx, proj = interp_decomp(A, eps_or_k)
```

This function automatically detects the form of the input parameters and passes them to the appropriate backend. For details, see `backend.iddp_id()`, `backend.iddp_aid()`, `backend.iddp_rid()`, `backend.iddr_id()`, `backend.iddr_aid()`, `backend.iddr_rid()`, `backend.idzp_id()`, `backend.idzp_aid()`, `backend.idzp_rid()`, `backend.idzr_id()`, `backend.idzr_aid()`, and `backend.idzr_rid()`.

Parameters

- **A** (`numpy.ndarray` or `scipy.sparse.linalg.LinearOperator`) – Matrix to be factored, given as either a `numpy.ndarray` or a `scipy.sparse.linalg.LinearOperator` with the *rmatvec* method (to apply the matrix adjoint).
- **eps_or_k** (*float or int*) – Relative error (if *eps_or_k* < *1*) or rank (if *eps_or_k* >= *1*) of approximation.
- **rand** (*bool*) – Whether to use random sampling if *A* is of type `numpy.ndarray` (randomized algorithms are always used if *A* is of type `scipy.sparse.linalg.LinearOperator`).

Returns Rank required to achieve specified relative precision if *eps_or_k* < *1*.

Return type int

Returns Column index array.

Return type `numpy.ndarray`

Returns Interpolation coefficients.

Return type numpy.ndarray

pymatrixid.**rand**(**args*)

Generate standard uniform pseudorandom numbers via a very efficient lagged Fibonacci method.

This routine is used for all random number generation in this package and can affect ID and SVD results.

Several call signatures are available:

- If no arguments are given, then the seed values are reset to their original values.
- If an integer *n* is given as input, then an array of *n* pseudorandom numbers are returned.
- If an array *s* of 55 values is given as input, then the seed values are set to *s*.

For details, see backend.id_srand(), backend.id_srandi(), and backend.id_srando().

pymatrixid.**reconstruct_interp_matrix**(*idx*, *proj*)

Reconstruct interpolation matrix from ID.

The interpolation matrix can be reconstructed from the ID indices and coefficients *idx* and *proj*, respectively, as:

```
P = numpy.hstack([numpy.eye(proj.shape[0]), proj])[:,numpy.argsort(idx)]
```

The original matrix can then be reconstructed from its skeleton matrix *B* via:

```
numpy.dot(B, P)
```

See also reconstruct_matrix_from_id() and reconstruct_skel_matrix().

This function automatically detects the matrix data type and calls the appropriate backend. For details, see backend.idd_reconint() and backend.idz_reconint().

Parameters

- **idx** (numpy.ndarray) – Column index array.
- **proj** (numpy.ndarray) – Interpolation coefficients.

Returns Interpolation matrix.

Return type numpy.ndarray

pymatrixid.**reconstruct_matrix_from_id**(*B*, *idx*, *proj*)

Reconstruct matrix from its ID.

A matrix *A* with skeleton matrix *B* and ID indices and coefficients *idx* and *proj*, respectively, can be reconstructed as:

```
numpy.hstack([B, numpy.dot(B, proj)])[:,numpy.argsort(idx)]
```

See also reconstruct_interp_matrix() and reconstruct_skel_matrix().

This function automatically detects the matrix data type and calls the appropriate backend. For details, see backend.idd_reconid() and backend.idz_reconid().

Parameters

- **B** (numpy.ndarray) – Skeleton matrix.
- **idx** (numpy.ndarray) – Column index array.
- **proj** (numpy.ndarray) – Interpolation coefficients.

Returns Reconstructed matrix.

Return type numpy.ndarray

pymatrixid.**reconstruct_skel_matrix**(*A*, *k*, *idx*)
Reconstruct skeleton matrix from ID.

The skeleton matrix can be reconstructed from the original matrix *A* and its ID rank and indices *k* and *idx*, respectively, as:

```
B = A[:,idx[:k]]
```

The original matrix can then be reconstructed via:

```
numpy.hstack([B, numpy.dot(B, proj)])[:,numpy.argsort(idx)]
```

See also `reconstruct_matrix_from_id()` and `reconstruct_interp_matrix()`.

This function automatically detects the matrix data type and calls the appropriate backend. For details, see `backend.idd_copycols()` and `backend.idz_copycols()`.

Parameters

- **A** (`numpy.ndarray`) – Original matrix.
- **k** (*int*) – Rank of ID.
- **idx** (`numpy.ndarray`) – Column index array.

Returns Skeleton matrix.

Return type `numpy.ndarray`

pymatrixid.**svd**(*A*, *eps_or_k*, *rand=True*)
Compute SVD of a matrix via an ID.

An SVD of a matrix *A* is a factorization:

```
A = numpy.dot(U, numpy.dot(numpy.diag(S), V.conj().T))
```

where *U* and *V* have orthonormal columns and *S* is nonnegative.

The SVD can be computed to any relative precision or rank (depending on the value of *eps_or_k*).

See also `interp_decomp()` and `id_to_svd()`.

This function automatically detects the form of the input parameters and passes them to the appropriate backend. For details, see `backend.iddp_svd()`, `backend.iddp_asvd()`, `backend.iddp_rsvd()`, `backend.iddr_svd()`, `backend.iddr_asvd()`, `backend.iddr_rsvd()`, `backend.idzp_svd()`, `backend.idzp_asvd()`, `backend.idzp_rsvd()`, `backend.idzr_svd()`, `backend.idzr_asvd()`, and `backend.idzr_rsvd()`.

Parameters

- **A** (`numpy.ndarray` or `scipy.sparse.linalg.LinearOperator`) – Matrix to be factored, given as either a `numpy.ndarray` or a `scipy.sparse.linalg.LinearOperator` with the *matvec* and *rmatvec* methods (to apply the matrix and its adjoint).
- **eps_or_k** (*float or int*) – Relative error (if *eps_or_k < 1*) or rank (if *eps_or_k >= 1*) of approximation.
- **rand** (*bool*) – Whether to use random sampling if *A* is of type `numpy.ndarray` (randomized algorithms are always used if *A* is of type `scipy.sparse.linalg.LinearOperator`).

Returns Left singular vectors.

Return type `numpy.ndarray`

Returns Singular values.

Return type numpy.ndarray

Returns Right singular vectors.

Return type numpy.ndarray

4.2 pymatrixid.backend

Direct wrappers for Fortran *id_dist* backend.

pymatrixid.backend.**id_srand**(*n*)
Generate standard uniform pseudorandom numbers via a very efficient lagged Fibonacci method.

Parameters **n** (*int*) – Number of pseudorandom numbers to generate.

Returns Pseudorandom numbers.

Return type numpy.ndarray

pymatrixid.backend.**id_srandi**(*t*)
Initialize seed values for id_srand() (any appropriately random numbers will do).

Parameters **t** (numpy.ndarray) – Array of 55 seed values.

pymatrixid.backend.**id_srando**()
Reset seed values to their original values.

pymatrixid.backend.**idd_copycols**(*A*, *k*, *idx*)
Reconstruct skeleton matrix from real ID.

Parameters

- **A** (numpy.ndarray) – Original matrix.
- **k** (*int*) – Rank of ID.
- **idx** (numpy.ndarray) – Column index array.

Returns Skeleton matrix.

Return type numpy.ndarray

pymatrixid.backend.**idd_diffsnorm**(*m*, *n*, *matvect*, *matvect2*, *matvec*, *matvec2*, *its=20*)
Estimate spectral norm of the difference of two real matrices by the randomized power method.

Parameters

- **m** (*int*) – Matrix row dimension.
- **n** (*int*) – Matrix column dimension.
- **matvect** (*function*) – Function to apply the transpose of the first matrix to a vector, with call signature *y = matvect(x)*, where *x* and *y* are the input and output vectors, respectively.
- **matvect2** (*function*) – Function to apply the transpose of the second matrix to a vector, with call signature *y = matvect2(x)*, where *x* and *y* are the input and output vectors, respectively.
- **matvec** (*function*) – Function to apply the first matrix to a vector, with call signature *y = matvec(x)*, where *x* and *y* are the input and output vectors, respectively.
- **matvec2** (*function*) – Function to apply the second matrix to a vector, with call signature *y = matvec2(x)*, where *x* and *y* are the input and output vectors, respectively.

- **its** (*int*) – Number of power method iterations.

Returns Spectral norm estimate of matrix difference.

Return type float

pymatrixid.backend.**idd_estrank**(*eps*, *A*)
Estimate rank of a real matrix to a specified relative precision using random sampling.

The output rank is typically about 8 higher than the actual rank.

Parameters

- **eps** (*float*) – Relative precision.
- **A** (numpy.ndarray) – Matrix.

Returns Rank estimate.

Return type int

pymatrixid.backend.**idd_findrank**(*eps*, *m*, *n*, *matvect*)
Estimate rank of a real matrix to a specified relative precision using random matrix-vector multiplication.

Parameters

- **eps** (*float*) – Relative precision.
- **m** (*int*) – Matrix row dimension.
- **n** (*int*) – Matrix column dimension.
- **matvect** (*function*) – Function to apply the matrix transpose to a vector, with call signature *y = matvect(x)*, where *x* and *y* are the input and output vectors, respectively.

Returns Rank estimate.

Return type int

pymatrixid.backend.**idd_frm**(*n*, *w*, *x*)
Transform real vector via a composition of Rokhlin's random transform, random subselection, and an FFT.

In contrast to idd_sfrm(), this routine works best when the length of the transformed vector is the power-of-two integer output by idd_frmi(), or when the length is not specified but instead determined a posteriori from the output. The returned transformed vector is randomly permuted.

Parameters

- **n** (*int*) – Greatest power-of-two integer satisfying *n* <= *x.size* as obtained from idd_frmi(); *n* is also the length of the output vector.
- **w** (numpy.ndarray) – Initialization array constructed by idd_frmi().
- **x** (numpy.ndarray) – Vector to be transformed.

Returns Transformed vector.

Return type numpy.ndarray

pymatrixid.backend.**idd_frmi**(*m*)
Initialize data for idd_frm().

Parameters **m** (*int*) – Length of vector to be transformed.

Returns Greatest power-of-two integer *n* satisfying *n* <= *m*.

Return type int

Returns Initialization array to be used by idd_frm().

Return type numpy.ndarray

pymatrixid.backend.**idd_id2svd**(*B*, *idx*, *proj*)
Convert real ID to SVD.

Parameters

- **B** (numpy.ndarray) – Skeleton matrix.
- **idx** (numpy.ndarray) – Column index array.
- **proj** (numpy.ndarray) – Interpolation coefficients.

Returns Left singular vectors.

Return type numpy.ndarray

Returns Right singular vectors.

Return type numpy.ndarray

Returns Singular values.

Return type numpy.ndarray

pymatrixid.backend.**idd_reconid**(*B*, *idx*, *proj*)
Reconstruct matrix from real ID.

Parameters

- **B** (numpy.ndarray) – Skeleton matrix.
- **idx** (numpy.ndarray) – Column index array.
- **proj** (numpy.ndarray) – Interpolation coefficients.

Returns Reconstructed matrix.

Return type numpy.ndarray

pymatrixid.backend.**idd_reconint**(*idx*, *proj*)
Reconstruct interpolation matrix from real ID.

Parameters

- **idx** (numpy.ndarray) – Column index array.
- **proj** (numpy.ndarray) – Interpolation coefficients.

Returns Interpolation matrix.

Return type numpy.ndarray

pymatrixid.backend.**idd_sfrm**(*l*, *n*, *w*, *x*)
Transform real vector via a composition of Rokhlin's random transform, random subselection, and an FFT.

In contrast to idd_frm(), this routine works best when the length of the transformed vector is known a priori.

Parameters

- **l** (*int*) – Length of transformed vector, satisfying *l* <= *n*.
- **n** (*int*) – Greatest power-of-two integer satisfying *n* <= *x.size* as obtained from idd_sfrmi().
- **w** (numpy.ndarray) – Initialization array constructed by idd_sfrmi().
- **x** (numpy.ndarray) – Vector to be transformed.

Returns Transformed vector.

Return type numpy.ndarray

pymatrixid.backend.**idd_sfrmi**(*l*, *m*)
Initialize data for idd_sfrm().

Parameters

- **l** (*int*) – Length of output transformed vector.
- **m** (*int*) – Length of the vector to be transformed.

Returns Greatest power-of-two integer *n* satisfying *n* <= *m*.

Return type int

Returns Initialization array to be used by idd_sfrm().

Return type numpy.ndarray

pymatrixid.backend.**idd_snorm**(*m*, *n*, *matvect*, *matvec*, *its=20*)
Estimate spectral norm of a real matrix by the randomized power method.

Parameters

- **m** (*int*) – Matrix row dimension.
- **n** (*int*) – Matrix column dimension.
- **matvect** (*function*) – Function to apply the matrix transpose to a vector, with call signature *y = matvect(x)*, where *x* and *y* are the input and output vectors, respectively.
- **matvec** (*function*) – Function to apply the matrix to a vector, with call signature *y = matvec(x)*, where *x* and *y* are the input and output vectors, respectively.
- **its** (*int*) – Number of power method iterations.

Returns Spectral norm estimate.

Return type float

pymatrixid.backend.**iddp_aid**(*eps*, *A*)
Compute ID of a real matrix to a specified relative precision using random sampling.

Parameters

- **eps** (*float*) – Relative precision.
- **A** (numpy.ndarray) – Matrix.

Returns Rank of ID.

Return type int

Returns Column index array.

Return type numpy.ndarray

Returns Interpolation coefficients.

Return type numpy.ndarray

pymatrixid.backend.**iddp_asvd**(*eps*, *A*)
Compute SVD of a real matrix to a specified relative precision using random sampling.

Parameters

- **eps** (*float*) – Relative precision.
- **A** (numpy.ndarray) – Matrix.

Returns Left singular vectors.

Return type numpy.ndarray

Returns Right singular vectors.

Return type numpy.ndarray

Returns Singular values.

Return type numpy.ndarray

pymatrixid.backend.**iddp_id**(*eps*, *A*)
Compute ID of a real matrix to a specified relative precision.

Parameters

- **eps** (*float*) – Relative precision.
- **A** (numpy.ndarray) – Matrix.

Returns Rank of ID.

Return type int

Returns Column index array.

Return type numpy.ndarray

Returns Interpolation coefficients.

Return type numpy.ndarray

pymatrixid.backend.**iddp_rid**(*eps*, *m*, *n*, *matvect*)
Compute ID of a real matrix to a specified relative precision using random matrix-vector multiplication.

Parameters

- **eps** (*float*) – Relative precision.
- **m** (*int*) – Matrix row dimension.
- **n** (*int*) – Matrix column dimension.
- **matvect** (*function*) – Function to apply the matrix transpose to a vector, with call signature *y = matvect(x)*, where *x* and *y* are the input and output vectors, respectively.

Returns Rank of ID.

Return type int

Returns Column index array.

Return type numpy.ndarray

Returns Interpolation coefficients.

Return type numpy.ndarray

pymatrixid.backend.**iddp_rsvd**(*eps*, *m*, *n*, *matvect*, *matvec*)
Compute SVD of a real matrix to a specified relative precision using random matrix-vector multiplication.

Parameters

- **eps** (*float*) – Relative precision.
- **m** (*int*) – Matrix row dimension.
- **n** (*int*) – Matrix column dimension.

- **matvect** (*function*) – Function to apply the matrix transpose to a vector, with call signature *y = matvect(x)*, where *x* and *y* are the input and output vectors, respectively.
- **matvec** (*function*) – Function to apply the matrix to a vector, with call signature *y = matvec(x)*, where *x* and *y* are the input and output vectors, respectively.

Returns Left singular vectors.

Return type numpy.ndarray

Returns Right singular vectors.

Return type numpy.ndarray

Returns Singular values.

Return type numpy.ndarray

pymatrixid.backend.**iddp_svd**(*eps*, *A*)

Compute SVD of a real matrix to a specified relative precision.

Parameters

- **eps** (*float*) – Relative precision.
- **A** (numpy.ndarray) – Matrix.

Returns Left singular vectors.

Return type numpy.ndarray

Returns Right singular vectors.

Return type numpy.ndarray

Returns Singular values.

Return type numpy.ndarray

pymatrixid.backend.**iddr_aid**(*A*, *k*)

Compute ID of a real matrix to a specified rank using random sampling.

Parameters

- **A** (numpy.ndarray) – Matrix.
- **k** (*int*) – Rank of ID.

Returns Column index array.

Return type numpy.ndarray

Returns Interpolation coefficients.

Return type numpy.ndarray

pymatrixid.backend.**iddr_aidi**(*m*, *n*, *k*)

Initialize array for iddr_aid().

Parameters

- **m** (*int*) – Matrix row dimension.
- **n** (*int*) – Matrix column dimension.
- **k** (*int*) – Rank of ID.

Returns Initialization array to be used by iddr_aid().

Return type numpy.ndarray

pymatrixid.backend.**iddr_asvd**(*A*, *k*)
Compute SVD of a real matrix to a specified rank using random sampling.

Parameters

- **A** (numpy.ndarray) – Matrix.
- **k** (*int*) – Rank of SVD.

Returns Left singular vectors.

Return type numpy.ndarray

Returns Right singular vectors.

Return type numpy.ndarray

Returns Singular values.

Return type numpy.ndarray

pymatrixid.backend.**iddr_id**(*A*, *k*)
Compute ID of a real matrix to a specified rank.

Parameters

- **A** (numpy.ndarray) – Matrix.
- **k** (*int*) – Rank of ID.

Returns Column index array.

Return type numpy.ndarray

Returns Interpolation coefficients.

Return type numpy.ndarray

pymatrixid.backend.**iddr_rid**(*m*, *n*, *matvect*, *k*)
Compute ID of a real matrix to a specified rank using random matrix-vector multiplication.

Parameters

- **m** (*int*) – Matrix row dimension.
- **n** (*int*) – Matrix column dimension.
- **matvect** (*function*) – Function to apply the matrix transpose to a vector, with call signature *y = matvect(x)*, where *x* and *y* are the input and output vectors, respectively.
- **k** (*int*) – Rank of ID.

Returns Column index array.

Return type numpy.ndarray

Returns Interpolation coefficients.

Return type numpy.ndarray

pymatrixid.backend.**iddr_rsvd**(*m*, *n*, *matvect*, *matvec*, *k*)
Compute SVD of a real matrix to a specified rank using random matrix-vector multiplication.

Parameters

- **m** (*int*) – Matrix row dimension.
- **n** (*int*) – Matrix column dimension.

- **matvect** (*function*) – Function to apply the matrix transpose to a vector, with call signature *y = matvect(x)*, where *x* and *y* are the input and output vectors, respectively.
- **matvec** (*function*) – Function to apply the matrix to a vector, with call signature *y = matvec(x)*, where *x* and *y* are the input and output vectors, respectively.
- **k** (*int*) – Rank of SVD.

Returns Left singular vectors.

Return type numpy.ndarray

Returns Right singular vectors.

Return type numpy.ndarray

Returns Singular values.

Return type numpy.ndarray

pymatrixid.backend.**iddr_svd**(*A*, *k*)

Compute SVD of a real matrix to a specified rank.

Parameters

- **A** (numpy.ndarray) – Matrix.
- **k** (*int*) – Rank of SVD.

Returns Left singular vectors.

Return type numpy.ndarray

Returns Right singular vectors.

Return type numpy.ndarray

Returns Singular values.

Return type numpy.ndarray

pymatrixid.backend.**idz_copycols**(*A*, *k*, *idx*)

Reconstruct skeleton matrix from complex ID.

Parameters

- **A** (numpy.ndarray) – Original matrix.
- **k** (*int*) – Rank of ID.
- **idx** (numpy.ndarray) – Column index array.

Returns Skeleton matrix.

Return type numpy.ndarray

pymatrixid.backend.**idz_diffsnorm**(*m*, *n*, *matveca*, *matveca2*, *matvec*, *matvec2*, *its=20*)

Estimate spectral norm of the difference of two complex matrices by the randomized power method.

Parameters

- **m** (*int*) – Matrix row dimension.
- **n** (*int*) – Matrix column dimension.
- **matveca** (*function*) – Function to apply the adjoint of the first matrix to a vector, with call signature *y = matveca(x)*, where *x* and *y* are the input and output vectors, respectively.

- **matveca2** (*function*) – Function to apply the adjoint of the second matrix to a vector, with call signature *y = matveca2(x)*, where *x* and *y* are the input and output vectors, respectively.
- **matvec** (*function*) – Function to apply the first matrix to a vector, with call signature *y = matvec(x)*, where *x* and *y* are the input and output vectors, respectively.
- **matvec2** (*function*) – Function to apply the second matrix to a vector, with call signature *y = matvec2(x)*, where *x* and *y* are the input and output vectors, respectively.
- **its** (*int*) – Number of power method iterations.

Returns Spectral norm estimate of matrix difference.

Return type float

pymatrixid.backend.**idz_estrank**(*eps*, *A*)

Estimate rank of a complex matrix to a specified relative precision using random sampling.

The output rank is typically about 8 higher than the actual rank.

Parameters

- **eps** (*float*) – Relative precision.
- **A** (numpy.ndarray) – Matrix.

Returns Rank estimate.

Return type int

pymatrixid.backend.**idz_findrank**(*eps*, *m*, *n*, *matveca*)

Estimate rank of a complex matrix to a specified relative precision using random matrix-vector multiplication.

Parameters

- **eps** (*float*) – Relative precision.
- **m** (*int*) – Matrix row dimension.
- **n** (*int*) – Matrix column dimension.
- **matveca** (*function*) – Function to apply the matrix adjoint to a vector, with call signature *y = matveca(x)*, where *x* and *y* are the input and output vectors, respectively.

Returns Rank estimate.

Return type int

pymatrixid.backend.**idz_frm**(*n*, *w*, *x*)

Transform complex vector via a composition of Rokhlin's random transform, random subselection, and an FFT.

In contrast to idz_sfrm(), this routine works best when the length of the transformed vector is the power-of-two integer output by idz_frmi(), or when the length is not specified but instead determined a posteriori from the output. The returned transformed vector is randomly permuted.

Parameters

- **n** (*int*) – Greatest power-of-two integer satisfying *n <= x.size* as obtained from idz_frmi(); *n* is also the length of the output vector.
- **w** (numpy.ndarray) – Initialization array constructed by idz_frmi().
- **x** (numpy.ndarray) – Vector to be transformed.

Returns Transformed vector.

Return type numpy.ndarray

pymatrixid.backend.**idz_frmi**(*m*)
Initialize data for idz_frm().

Parameters **m** (*int*) – Length of vector to be transformed.

Returns Greatest power-of-two integer *n* satisfying *n* <= *m*.

Return type int

Returns Initialization array to be used by idz_frm().

Return type numpy.ndarray

pymatrixid.backend.**idz_id2svd**(*B*, *idx*, *proj*)
Convert complex ID to SVD.

Parameters

- **B** (numpy.ndarray) – Skeleton matrix.
- **idx** (numpy.ndarray) – Column index array.
- **proj** (numpy.ndarray) – Interpolation coefficients.

Returns Left singular vectors.

Return type numpy.ndarray

Returns Right singular vectors.

Return type numpy.ndarray

Returns Singular values.

Return type numpy.ndarray

pymatrixid.backend.**idz_reconid**(*B*, *idx*, *proj*)
Reconstruct matrix from complex ID.

Parameters

- **B** (numpy.ndarray) – Skeleton matrix.
- **idx** (numpy.ndarray) – Column index array.
- **proj** (numpy.ndarray) – Interpolation coefficients.

Returns Reconstructed matrix.

Return type numpy.ndarray

pymatrixid.backend.**idz_reconint**(*idx*, *proj*)
Reconstruct interpolation matrix from complex ID.

Parameters

- **idx** (numpy.ndarray) – Column index array.
- **proj** (numpy.ndarray) – Interpolation coefficients.

Returns Interpolation matrix.

Return type numpy.ndarray

pymatrixid.backend.**idz_sfrm**(*l*, *n*, *w*, *x*)
Transform complex vector via a composition of Rokhlin's random transform, random subselection, and an FFT.

In contrast to idz_frm(), this routine works best when the length of the transformed vector is known a priori.

Parameters

- **l** (*int*) – Length of transformed vector, satisfying *l* <= *n*.
- **n** (*int*) – Greatest power-of-two integer satisfying *n* <= *x.size* as obtained from `idz_sfrmi()`.
- **w** (`numpy.ndarray`) – Initialization array constructed by `idd_sfrmi()`.
- **x** (`numpy.ndarray`) – Vector to be transformed.

Returns Transformed vector.

Return type `numpy.ndarray`

`pymatrixid.backend.`**`idz_sfrmi`**(*l*, *m*)

Initialize data for `idz_sfrm()`.

Parameters

- **l** (*int*) – Length of output transformed vector.
- **m** (*int*) – Length of the vector to be transformed.

Returns Greatest power-of-two integer *n* satisfying *n* <= *m*.

Return type int

Returns Initialization array to be used by `idz_sfrm()`.

Return type `numpy.ndarray`

`pymatrixid.backend.`**`idz_snorm`**(*m*, *n*, *matveca*, *matvec*, *its=20*)

Estimate spectral norm of a complex matrix by the randomized power method.

Parameters

- **m** (*int*) – Matrix row dimension.
- **n** (*int*) – Matrix column dimension.
- **matveca** (*function*) – Function to apply the matrix adjoint to a vector, with call signature *y* = *matveca(x)*, where *x* and *y* are the input and output vectors, respectively.
- **matvec** (*function*) – Function to apply the matrix to a vector, with call signature *y* = *matvec(x)*, where *x* and *y* are the input and output vectors, respectively.
- **its** (*int*) – Number of power method iterations.

Returns Spectral norm estimate.

Return type float

`pymatrixid.backend.`**`idzp_aid`**(*eps*, *A*)

Compute ID of a complex matrix to a specified relative precision using random sampling.

Parameters

- **eps** (*float*) – Relative precision.
- **A** (`numpy.ndarray`) – Matrix.

Returns Rank of ID.

Return type int

Returns Column index array.

Return type `numpy.ndarray`

Returns Interpolation coefficients.

Return type numpy.ndarray

pymatrixid.backend.**idzp_asvd**(*eps*, *A*)
Compute SVD of a complex matrix to a specified relative precision using random sampling.

Parameters

- **eps** (*float*) – Relative precision.
- **A** (numpy.ndarray) – Matrix.

Returns Left singular vectors.

Return type numpy.ndarray

Returns Right singular vectors.

Return type numpy.ndarray

Returns Singular values.

Return type numpy.ndarray

pymatrixid.backend.**idzp_id**(*eps*, *A*)
Compute ID of a complex matrix to a specified relative precision.

Parameters

- **eps** (*float*) – Relative precision.
- **A** (numpy.ndarray) – Matrix.

Returns Rank of ID.

Return type int

Returns Column index array.

Return type numpy.ndarray

Returns Interpolation coefficients.

Return type numpy.ndarray

pymatrixid.backend.**idzp_rid**(*eps*, *m*, *n*, *matveca*)
Compute ID of a complex matrix to a specified relative precision using random matrix-vector multiplication.

Parameters

- **eps** (*float*) – Relative precision.
- **m** (*int*) – Matrix row dimension.
- **n** (*int*) – Matrix column dimension.
- **matveca** (*function*) – Function to apply the matrix adjoint to a vector, with call signature *y = matveca(x)*, where *x* and *y* are the input and output vectors, respectively.

Returns Rank of ID.

Return type int

Returns Column index array.

Return type numpy.ndarray

Returns Interpolation coefficients.

Return type numpy.ndarray

pymatrixid.backend.**idzp_rsvd**(*eps*, *m*, *n*, *matveca*, *matvec*)

Compute SVD of a complex matrix to a specified relative precision using random matrix-vector multiplication.

Parameters

- **eps** (*float*) – Relative precision.
- **m** (*int*) – Matrix row dimension.
- **n** (*int*) – Matrix column dimension.
- **matveca** (*function*) – Function to apply the matrix adjoint to a vector, with call signature *y = matveca(x)*, where *x* and *y* are the input and output vectors, respectively.
- **matvec** (*function*) – Function to apply the matrix to a vector, with call signature *y = matvec(x)*, where *x* and *y* are the input and output vectors, respectively.

Returns Left singular vectors.

Return type numpy.ndarray

Returns Right singular vectors.

Return type numpy.ndarray

Returns Singular values.

Return type numpy.ndarray

pymatrixid.backend.**idzp_svd**(*eps*, *A*)

Compute SVD of a complex matrix to a specified relative precision.

Parameters

- **eps** (*float*) – Relative precision.
- **A** (numpy.ndarray) – Matrix.

Returns Left singular vectors.

Return type numpy.ndarray

Returns Right singular vectors.

Return type numpy.ndarray

Returns Singular values.

Return type numpy.ndarray

pymatrixid.backend.**idzr_aid**(*A*, *k*)

Compute ID of a complex matrix to a specified rank using random sampling.

Parameters

- **A** (numpy.ndarray) – Matrix.
- **k** (*int*) – Rank of ID.

Returns Column index array.

Return type numpy.ndarray

Returns Interpolation coefficients.

Return type numpy.ndarray

pymatrixid.backend.**idzr_aidi**(*m*, *n*, *k*)

Initialize array for idzr_aid().

Parameters

- **m** (*int*) – Matrix row dimension.
- **n** (*int*) – Matrix column dimension.
- **k** (*int*) – Rank of ID.

Returns Initialization array to be used by `idzr_aid()`.

Return type `numpy.ndarray`

`pymatrixid.backend.`**`idzr_asvd`**(*A*, *k*)

Compute SVD of a complex matrix to a specified rank using random sampling.

Parameters

- **A** (`numpy.ndarray`) – Matrix.
- **k** (*int*) – Rank of SVD.

Returns Left singular vectors.

Return type `numpy.ndarray`

Returns Right singular vectors.

Return type `numpy.ndarray`

Returns Singular values.

Return type `numpy.ndarray`

`pymatrixid.backend.`**`idzr_id`**(*A*, *k*)

Compute ID of a complex matrix to a specified rank.

Parameters

- **A** (`numpy.ndarray`) – Matrix.
- **k** (*int*) – Rank of ID.

Returns Column index array.

Return type `numpy.ndarray`

Returns Interpolation coefficients.

Return type `numpy.ndarray`

`pymatrixid.backend.`**`idzr_rid`**(*m*, *n*, *matveca*, *k*)

Compute ID of a complex matrix to a specified rank using random matrix-vector multiplication.

Parameters

- **m** (*int*) – Matrix row dimension.
- **n** (*int*) – Matrix column dimension.
- **matveca** (*function*) – Function to apply the matrix adjoint to a vector, with call signature *y = matveca(x)*, where *x* and *y* are the input and output vectors, respectively.
- **k** (*int*) – Rank of ID.

Returns Column index array.

Return type `numpy.ndarray`

Returns Interpolation coefficients.

Return type numpy.ndarray

pymatrixid.backend.**idzr_rsvd**(*m*, *n*, *matveca*, *matvec*, *k*)

Compute SVD of a complex matrix to a specified rank using random matrix-vector multiplication.

Parameters

- **m** (*int*) – Matrix row dimension.
- **n** (*int*) – Matrix column dimension.
- **matveca** (*function*) – Function to apply the matrix adjoint to a vector, with call signature *y = matveca(x)*, where *x* and *y* are the input and output vectors, respectively.
- **matvec** (*function*) – Function to apply the matrix to a vector, with call signature *y = matvec(x)*, where *x* and *y* are the input and output vectors, respectively.
- **k** (*int*) – Rank of SVD.

Returns Left singular vectors.

Return type numpy.ndarray

Returns Right singular vectors.

Return type numpy.ndarray

Returns Singular values.

Return type numpy.ndarray

pymatrixid.backend.**idzr_svd**(*A*, *k*)

Compute SVD of a complex matrix to a specified rank.

Parameters

- **A** (numpy.ndarray) – Matrix.
- **k** (*int*) – Rank of SVD.

Returns Left singular vectors.

Return type numpy.ndarray

Returns Right singular vectors.

Return type numpy.ndarray

Returns Singular values.

Return type numpy.ndarray

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

p

INDEX