numpy-sugar Documentation

Release 1.0.47

Danilo Horta

Jun 30, 2017

Table of contents

1 Install **3**

2 Array **5**

3 Linear algebra **7**
- 3.1 Decomposition . . . 7
- 3.2 Determinant . . . 8
- 3.3 Dot and sum . . . 8
- 3.4 Properties . . . 9
- 3.5 Reducers . . . 9
- 3.6 Solvers . . . 10

4 Special functions **17**
- 4.1 Beta . . . 17
- 4.2 Chi-squared . . . 17
- 4.3 Gamma . . . 18
- 4.4 Log sum . . . 18
- 4.5 Normal . . . 19
- 4.6 Pearson . . . 19

5 Comments and bugs **21**

Python Module Index **23**

Missing NumPy functionalities.

CHAPTER 1

Install

The recommended way of installing it is via conda:

```
conda install -c conda-forge numpy-sugar
```

An alternative way would be via pip:

```
pip install numpy-sugar
```

CHAPTER 2

Array

numpy_sugar._array.**is_all_equal**(*arr*)
Check if the array values are all equal.

Parameters **arr** (*array_like*) – sequence of values.

Returns `True` if values are all equal.

Return type bool

numpy_sugar._array.**is_all_finite**(*arr*)
Check if the array values are all finite.

Parameters **arr** (*array_like*) – sequence of values.

Returns `True` if values are all finite.

Return type bool

numpy_sugar._array.**is_crescent**(*arr*)
Check if the array values are in non-decreasing order.

Parameters **arr** (*array_like*) – sequence of values.

Returns `True` for non-decreasing order.

Return type bool

numpy_sugar._array.**cartesian**(*shape*)
Cartesian indexing.

Returns a sequence of n-tuples indexing each element of a hypothetical matrix of the given shape.

Parameters **shape** (*tuple*) – tuple of dimensions.

Returns indices.

Return type array_like

Example

```
>>> from numpy_sugar import cartesian
>>> print(cartesian((2, 3)))
[[0 0]
 [0 1]
 [0 2]
 [1 0]
 [1 1]
 [1 2]]
```

Reference:

[1] http://stackoverflow.com/a/27286794

numpy_sugar._array.**is_all_equal**(*arr*)
Check if the array values are all equal.

Parameters **arr** (*array_like*) – sequence of values.

Returns `True` if values are all equal.

Return type bool

numpy_sugar._array.**is_all_finite**(*arr*)
Check if the array values are all finite.

Parameters **arr** (*array_like*) – sequence of values.

Returns `True` if values are all finite.

Return type bool

numpy_sugar._array.**is_crescent**(*arr*)
Check if the array values are in non-decreasing order.

Parameters **arr** (*array_like*) – sequence of values.

Returns `True` for non-decreasing order.

Return type bool

numpy_sugar._array.**unique**(*ar*)
Find the unique elements of an array.

It uses `dask.array.unique` if necessary.

Parameters **ar** (*array_like*) – Input array.

Returns the sorted unique elements.

Return type array_like

CHAPTER 3

Linear algebra

3.1 Decomposition

numpy_sugar.linalg.**economic_qs**(*K*, *epsilon=1.4901161193847656e-08*)

Economic eigen decomposition for symmetric matrices.

A symmetric matrix `K` can be decomposed in $Q_0 S_0 Q_0^\intercal + Q_1 S_1 Q_1^\intercal$, where S_1 is a zero matrix with size determined by `K`'s rank deficiency.

Parameters

- **K** (*array_like*) – Symmetric matrix.
- **epsilon** (*float*) – Eigen value threshold. Default is `sqrt(finfo(float).eps)`.

Returns `((Q0, Q1), S0)`.

Return type tuple

numpy_sugar.linalg.**economic_qs_linear**(*G*)

Economic eigen decomposition for symmetric matrices `dot(G, G.T)`.

It is theoretically equivalent to `economic_qs(dot(G, G.T))`. Refer to `numpy_sugar.economic_qs()` for further information.

Parameters **G** (*array_like*) – Matrix.

Returns `((Q0, Q1), S0)`.

Return type tuple

numpy_sugar.linalg.**economic_svd**(*G*, *epsilon=1.4901161193847656e-08*)

Economic Singular Value Decomposition.

Parameters

- **G** (*array_like*) – Matrix to be factorized.
- **epsilon** (*float*) – Threshold on the square root of the eigen values. Default is `sqrt(finfo(float).eps)`.

Returns Unitary matrix. numpy.ndarray: Singular values. numpy.ndarray: Unitary matrix.

Return type numpy.ndarray

See also:

numpy.linalg.svd() Cholesky decomposition.

3.2 Determinant

numpy_sugar.linalg.**plogdet**(*K*)

Log of the pseudo-determinant.

It assumes that K is a positive semi-definite matrix.

Parameters **K** (*array_like*) – matrix.

Returns log of the pseudo-determinant.

Return type float

3.3 Dot and sum

numpy_sugar.linalg.**sum2diag**(*A*, *D*, *out=None*)

Add values D to the diagonal of matrix A.

Parameters

- **A** (*array_like*) – Left-hand side.
- **D** (*array_like or float*) – Values to add.
- **out** (numpy.ndarray, optional) – copy result to.

Returns Resulting matrix.

Return type numpy.ndarray

numpy_sugar.linalg.**ddot**(*L*, *R*, *left=True*, *out=None*)

Dot product of a matrix and a diagonal one.

Parameters

- **L** (*array_like*) – Left matrix.
- **R** (*array_like*) – Right matrix.
- **left** (*bool*) – True if L is the diagonal matrix; False otherwise.
- **out** (numpy.ndarray, optional) – copy result to.

Returns Resulting matrix.

Return type numpy.ndarray

3.4 Properties

numpy_sugar.linalg.**check_definite_positiveness**(*A*)
Check if A is a definite positive matrix.

Parameters **A** (*array_like*) – Matrix.

Returns True if A is definite positive; False otherwise.

Return type bool

numpy_sugar.linalg.**check_symmetry**(*A*)
Check if A is a symmetric matrix.

Parameters **A** (*array_like*) – Matrix.

Returns True if A is symmetric; False otherwise.

Return type bool

3.5 Reducers

numpy_sugar.linalg.**lu_slogdet**(*LU*)
Natural logarithm of a LU decomposition.

Parameters **LU** (*tuple*) – LU decomposition.

Returns sign and log-determinant.

Return type tuple

numpy_sugar.linalg.**trace2**(*A*, *B*)
Trace of $AB^\intercal$.

Parameters

- **A** (*array_like*) – Left-hand side.
- **B** (*array_like*) – Right-hand side.

Returns Trace of $AB^\intercal$.

Return type float

numpy_sugar.linalg.**dotd**(*A*, *B*, *out=None*)
Diagonal of $AB^\intercal$.

If A is $n \times p$ and B is $p \times n$, it is done in $O(pn)$.

Parameters

- **A** (*array_like*) – Left matrix.
- **B** (*array_like*) – Right matrix.
- **out** (numpy.ndarray, optional) – copy result to.

Returns Resulting diagonal.

Return type numpy.ndarray

3.6 Solvers

numpy_sugar.linalg.**cho_solve**(*L*, *b*)
Solve for Cholesky decomposition.

Solve the linear equations $\mathrm{A}\mathbf{x} = \mathbf{b}$, given the Cholesky factorization of A.

Parameters

- **L** (*array_like*) – Lower triangular matrix.
- **b** (*array_like*) – Right-hand side.

Returns

The solution to the system $\mathrm{A}\mathbf{x} = \mathbf{b}$.

Return type numpy.ndarray

See also:

numpy.linalg.cholesky() Cholesky decomposition.

scipy.linalg.cho_solve() Solve linear equations given Cholesky factorization.

numpy_sugar.linalg.**lu_solve**(*LU*, *b*)
Solve for LU decomposition.

Solve the linear equations $\mathrm{A}\mathbf{x} = \mathbf{b}$, given the LU factorization of A.

Parameters

- **LU** (*array_like*) – LU decomposition.
- **b** (*array_like*) – Right-hand side.

Returns The solution to the system $\mathrm{A}\mathbf{x} = \mathbf{b}$.

Return type numpy.ndarray

See also:

scipy.linalg.lu_factor() LU decomposition.

scipy.linalg.lu_solve() Solve linear equations given LU factorization.

numpy_sugar.linalg.**solve**(*A*, *b*)
Solve for the linear equations $\mathrm{A}\mathbf{x} = \mathbf{b}$.

Parameters

- **A** (*array_like*) – Coefficient matrix.
- **b** (*array_like*) – Ordinate values.

Returns Solution x.

Return type numpy.ndarray

numpy_sugar.linalg.**rsolve**(*A*, *b*, *epsilon=1.4901161193847656e-08*)
Robust solve for the linear equations.

Parameters

- **A** (*array_like*) – Coefficient matrix.

- **b** (*array_like*) – Ordinate values.

Returns Solution x.

Return type numpy.ndarray

numpy_sugar.linalg.**stl**(*A*, *b*)
Shortcut to `solve_triangular(A, b, lower=True, check_finite=False)`.

Solve linear systems $\mathrm{A}\mathbf{x} = \mathbf{b}$ when A is a lower-triangular matrix.

Parameters

- **A** (*array_like*) – A lower-triangular matrix.
- **b** (*array_like*) – Ordinate values.

Returns Solution x.

Return type numpy.ndarray

See also:

scipy.linalg.solve_triangular() Solve triangular linear equations.

numpy_sugar.linalg.**lstsq**(*A*, *b*)
Return the least-squares solution to a linear matrix equation.

Parameters

- **A** (*array_like*) – Coefficient matrix.
- **b** (*array_like*) – Ordinate values.

Returns Least-squares solution.

Return type numpy.ndarray

numpy_sugar.linalg.**cho_solve**(*L*, *b*)
Solve for Cholesky decomposition.

Solve the linear equations $\mathrm{A}\mathbf{x} = \mathbf{b}$, given the Cholesky factorization of A.

Parameters

- **L** (*array_like*) – Lower triangular matrix.
- **b** (*array_like*) – Right-hand side.

Returns

The solution to the system $\mathrm{A}\mathbf{x} = \mathbf{b}$.

Return type numpy.ndarray

See also:

numpy.linalg.cholesky() Cholesky decomposition.

scipy.linalg.cho_solve() Solve linear equations given Cholesky factorization.

numpy_sugar.linalg.**sum2diag**(*A*, *D*, *out=None*)
Add values D to the diagonal of matrix A.

Parameters

- **A** (*array_like*) – Left-hand side.

- **D** (*array_like or float*) – Values to add.
- **out** (numpy.ndarray, optional) – copy result to.

Returns Resulting matrix.

Return type numpy.ndarray

numpy_sugar.linalg.**trace2**(*A*, *B*)
Trace of $\mathrm{AB}^\intercal$.

Parameters

- **A** (*array_like*) – Left-hand side.
- **B** (*array_like*) – Right-hand side.

Returns Trace of $\mathrm{AB}^\intercal$.

Return type float

numpy_sugar.linalg.**ddot**(*L*, *R*, *left=True*, *out=None*)
Dot product of a matrix and a diagonal one.

Parameters

- **L** (*array_like*) – Left matrix.
- **R** (*array_like*) – Right matrix.
- **left** (*bool*) – `True` if `L` is the diagonal matrix; `False` otherwise.
- **out** (numpy.ndarray, optional) – copy result to.

Returns Resulting matrix.

Return type numpy.ndarray

numpy_sugar.linalg.**dotd**(*A*, *B*, *out=None*)
Diagonal of $\mathrm{AB}^\intercal$.

If `A` is $n \times p$ and `B` is $p \times n$, it is done in $O(pn)$.

Parameters

- **A** (*array_like*) – Left matrix.
- **B** (*array_like*) – Right matrix.
- **out** (numpy.ndarray, optional) – copy result to.

Returns Resulting diagonal.

Return type numpy.ndarray

numpy_sugar.linalg.**lu_slogdet**(*LU*)
Natural logarithm of a LU decomposition.

Parameters **LU** (*tuple*) – LU decomposition.

Returns sign and log-determinant.

Return type tuple

numpy_sugar.linalg.**lu_solve**(*LU*, *b*)
Solve for LU decomposition.

Solve the linear equations $\mathrm{A}\mathbf{x} = \mathbf{b}$, given the LU factorization of A.

Parameters

- **LU** (*array_like*) – LU decomposition.
- **b** (*array_like*) – Right-hand side.

Returns The solution to the system $\mathrm{A}\mathbf{x} = \mathbf{b}$.

Return type numpy.ndarray

See also:

scipy.linalg.lu_factor() LU decomposition.

scipy.linalg.lu_solve() Solve linear equations given LU factorization.

numpy_sugar.linalg.**check_definite_positiveness**(*A*)
Check if A is a definite positive matrix.

Parameters **A** (*array_like*) – Matrix.

Returns True if A is definite positive; False otherwise.

Return type bool

numpy_sugar.linalg.**check_symmetry**(*A*)
Check if A is a symmetric matrix.

Parameters **A** (*array_like*) – Matrix.

Returns True if A is symmetric; False otherwise.

Return type bool

numpy_sugar.linalg.**economic_qs**(*K*, *epsilon=1.4901161193847656e-08*)
Economic eigen decomposition for symmetric matrices.

A symmetric matrix K can be decomposed in $\mathrm{Q}_0\mathrm{S}_0\mathrm{Q}_0^\intercal + \mathrm{Q}_1\,\mathrm{S}_1\mathrm{Q}_1^\intercal$, where S_1 is a zero matrix with size determined by K's rank deficiency.

Parameters

- **K** (*array_like*) – Symmetric matrix.
- **epsilon** (*float*) – Eigen value threshold. Default is sqrt(finfo(float).eps).

Returns ((Q0, Q1), S0).

Return type tuple

numpy_sugar.linalg.**economic_qs_linear**(*G*)
Economic eigen decomposition for symmetric matrices dot(G, G.T).

It is theoretically equivalent to economic_qs(dot(G, G.T)). Refer to numpy_sugar.economic_qs() for further information.

Parameters **G** (*array_like*) – Matrix.

Returns ((Q0, Q1), S0).

Return type tuple

numpy_sugar.linalg.**solve**(*A*, *b*)
Solve for the linear equations $\mathrm{A}\mathbf{x} = \mathbf{b}$.

Parameters

- **A** (*array_like*) – Coefficient matrix.
- **b** (*array_like*) – Ordinate values.

Returns Solution `x`.

Return type `numpy.ndarray`

`numpy_sugar.linalg.`**`rsolve`**(*A*, *b*, *epsilon=1.4901161193847656e-08*)
Robust solve for the linear equations.

Parameters

- **A** (*array_like*) – Coefficient matrix.
- **b** (*array_like*) – Ordinate values.

Returns Solution `x`.

Return type `numpy.ndarray`

`numpy_sugar.linalg.`**`economic_svd`**(*G*, *epsilon=1.4901161193847656e-08*)
Economic Singular Value Decomposition.

Parameters

- **G** (*array_like*) – Matrix to be factorized.
- **epsilon** (*float*) – Threshold on the square root of the eigen values. Default is `sqrt(finfo(float).eps)`.

Returns Unitary matrix. `numpy.ndarray`: Singular values. `numpy.ndarray`: Unitary matrix.

Return type `numpy.ndarray`

See also:

`numpy.linalg.svd()` Cholesky decomposition.

`numpy_sugar.linalg.`**`stl`**(*A*, *b*)
Shortcut to `solve_triangular(A, b, lower=True, check_finite=False)`.

Solve linear systems $\mathrm{A}\mathbf{x} = \mathbf{b}$ when A is a lower-triangular matrix.

Parameters

- **A** (*array_like*) – A lower-triangular matrix.
- **b** (*array_like*) – Ordinate values.

Returns Solution `x`.

Return type `numpy.ndarray`

See also:

`scipy.linalg.solve_triangular()` Solve triangular linear equations.

`numpy_sugar.linalg.`**`lstsq`**(*A*, *b*)
Return the least-squares solution to a linear matrix equation.

Parameters

- **A** (*array_like*) – Coefficient matrix.
- **b** (*array_like*) – Ordinate values.

Returns Least-squares solution.

Return type `numpy.ndarray`

numpy_sugar.linalg.**plogdet**(*K*)

Log of the pseudo-determinant.

It assumes that K is a positive semi-definite matrix.

Parameters **K** (*array_like*) – matrix.

Returns log of the pseudo-determinant.

Return type float

CHAPTER 4

Special functions

4.1 Beta

numpy_sugar.special.**beta_isf**(a, b, x)

Inverse of the Beta survival function.

Parameters

- **a** (*array_like*) – parameter a.
- **b** (*array_like*) – parameter b.
- **x** (*array_like*) – evaluation point.

4.2 Chi-squared

numpy_sugar.special.**chi2_sf**(k, x)

Chi-squared distribution [1] survival function.

Parameters

- **k** (*array_like*) – Degrees of fredom.
- **x** (*array_like*) – Evaluation point.

Returns

Survival function of the χ^2_k distribution.

Return type numpy.ndarray

References

[1] https://en.wikipedia.org/wiki/Chi-squared_distribution

4.3 Gamma

numpy_sugar.special.**lgamma**(*x*)
Natural logarithm of the Gamma function [1].

Parameters **x** (*array_like*) – evaluation point.

Returns Log-gamma of x.

Return type numpy.ndarray

References

[1] https://en.wikipedia.org/wiki/Gamma_function

4.4 Log sum

numpy_sugar.special.**logbinom**(*a*, *b*)
Natural logarithm of Binomial coefficient: $\log \binom{a}{b}$.

numpy_sugar.special.**logaddexp**(*x*, *y*)
Numerically-stable numpy.log(numpy.exp(x)+numpy.exp(y)).

Parameters

- **x** (*array_like*) – First array.
- **y** (*array_like*) – Second array.

Returns numpy.log(numpy.exp(x)+numpy.exp(y)).

numpy_sugar.special.**logaddexps**(*x*, *y*, *sx*, *sy*)
Numerically-stable numpy.log(sx*numpy.exp(x)+sy*numpy.exp(y)).

numpy_sugar.special.**logaddexpss**(*x*, *y*, *sx*, *sy*, *r*, *sign*)
Numerically-stable numpy.log(sx*numpy.exp(x)+sy*numpy.exp(y)).

Suppose you are interested in computing:

```
sx[i]*exp(x[i]) + sy[i]*exp(y[i])
```

where sx[i] and sy[i] are either -1 or +1. Often a direct calculation of the above is numerically innacurate. Instead, let:

```
sign[i]*exp(r[i]) = sx[i]*exp(x[i]) + sy[i]*exp(y[i])
```

This function accurately computes r[i] and sign[i], where sign[i] is either -1 or +1.

numpy_sugar.special.**logsumexp**(*x*)
Numerically-stable numpy.log(sum(numpy.exp(x))).

Parameters **x** (*array_like*) – We want the sum of their exponentiated values but in a numerically-stable way.

Returns numpy.log(sum(numpy.exp(x))).

4.5 Normal

numpy_sugar.special.**normal_pdf**(*x*)
P.d.f. of the Normal distribution.

Parameters **x** (*array_like*) – evaluation point.

numpy_sugar.special.**normal_cdf**(*x*)
C.d.f. of the Normal distribution.

Parameters **x** (*array_like*) – evaluation point.

numpy_sugar.special.**normal_icdf**(*x*)
Inverse of the c.d.f. of the Normal distribution.

Parameters **x** (*array_like*) – evaluation point.

numpy_sugar.special.**normal_sf**(*x*)
Survival function of the Normal distribution.

Parameters **x** (*array_like*) – evaluation point.

numpy_sugar.special.**normal_isf**(*x*)
Inverse of the survival function of the Normal distribution.

Parameters **x** (*array_like*) – evaluation point.

numpy_sugar.special.**normal_logpdf**(*x*)
Natural logarithm of the p.d.f. of the Normal distribution.

Parameters **x** (*array_like*) – evaluation point.

numpy_sugar.special.**normal_logcdf**(*x*)
Natural logarithm of the c.d.f. of the Normal distribution.

Parameters **x** (*array_like*) – evaluation point.

numpy_sugar.special.**normal_logsf**(*x*)
Natural logarithm of the survival function of the Normal distribution.

Parameters **x** (*array_like*) – evaluation point.

4.6 Pearson

numpy_sugar.special.**r_squared**(*x*, *y*)
Coefficient of determination between `x` and `y`.

It equals to `scipy.stats.pearsonr(x, y)[0]**2`.

Parameters

- **x** (*array_like*) – First array.
- **y** (*array_like*) – Second array.

Returns Squared Pearson correlation.

numpy_sugar.special.**beta_isf**(*a*, *b*, *x*)
Inverse of the Beta survival function.

Parameters

- **a** (*array_like*) – parameter *a*.

- **b** (*array_like*) – parameter *b*.
- **x** (*array_like*) – evaluation point.

numpy_sugar.special.**logaddexpss**(*x*, *y*, *sx*, *sy*, *r*, *sign*)
Numerically-stable `numpy.log(sx*numpy.exp(x)+sy*numpy.exp(y))`.

Suppose you are interested in computing:

```
sx[i]*exp(x[i]) + sy[i]*exp(y[i])
```

where `sx[i]` and `sy[i]` are either -1 or +1. Often a direct calculation of the above is numerically innacurate. Instead, let:

```
sign[i]*exp(r[i]) = sx[i]*exp(x[i]) + sy[i]*exp(y[i])
```

This function accurately computes `r[i]` and `sign[i]`, where `sign[i]` is either -1 or +1.

numpy_sugar.special.**logsumexp**(*x*)
Numerically-stable `numpy.log(sum(numpy.exp(x)))`.

Parameters **x** (*array_like*) – We want the sum of their exponentiated values but in a numerically-stable way.

Returns `numpy.log(sum(numpy.exp(x)))`.

numpy_sugar.special.**r_squared**(*x*, *y*)
Coefficient of determination between `x` and `y`.

It equals to `scipy.stats.pearsonr(x, y)[0]**2`.

Parameters

- **x** (*array_like*) – First array.
- **y** (*array_like*) – Second array.

Returns Squared Pearson correlation.

CHAPTER 5

Comments and bugs

You can get the source and open issues on Github.

Python Module Index

n

numpy_sugar._array, 3
numpy_sugar.linalg, 6
numpy_sugar.special, 15

Index

B

beta_isf() (in module numpy_sugar.special), 17, 19

C

cartesian() (in module numpy_sugar._array), 5
check_definite_positiveness() (in module numpy_sugar.linalg), 9, 13
check_symmetry() (in module numpy_sugar.linalg), 9, 13
chi2_sf() (in module numpy_sugar.special), 17
cho_solve() (in module numpy_sugar.linalg), 10, 11

D

ddot() (in module numpy_sugar.linalg), 8, 12
dotd() (in module numpy_sugar.linalg), 9, 12

E

economic_qs() (in module numpy_sugar.linalg), 7, 13
economic_qs_linear() (in module numpy_sugar.linalg), 7, 13
economic_svd() (in module numpy_sugar.linalg), 7, 14

I

is_all_equal() (in module numpy_sugar._array), 5, 6
is_all_finite() (in module numpy_sugar._array), 5, 6
is_crescent() (in module numpy_sugar._array), 5, 6

L

lgamma() (in module numpy_sugar.special), 18
logaddexp() (in module numpy_sugar.special), 18
logaddexps() (in module numpy_sugar.special), 18
logaddexpss() (in module numpy_sugar.special), 18, 20
logbinom() (in module numpy_sugar.special), 18
logsumexp() (in module numpy_sugar.special), 18, 20
lstsq() (in module numpy_sugar.linalg), 11, 14
lu_slogdet() (in module numpy_sugar.linalg), 9, 12
lu_solve() (in module numpy_sugar.linalg), 10, 12

N

normal_cdf() (in module numpy_sugar.special), 19
normal_icdf() (in module numpy_sugar.special), 19
normal_isf() (in module numpy_sugar.special), 19
normal_logcdf() (in module numpy_sugar.special), 19
normal_logpdf() (in module numpy_sugar.special), 19
normal_logsf() (in module numpy_sugar.special), 19
normal_pdf() (in module numpy_sugar.special), 19
normal_sf() (in module numpy_sugar.special), 19
numpy_sugar._array (module), 3
numpy_sugar.linalg (module), 6
numpy_sugar.special (module), 15

P

plogdet() (in module numpy_sugar.linalg), 8, 14

R

r_squared() (in module numpy_sugar.special), 19, 20
rsolve() (in module numpy_sugar.linalg), 10, 14

S

solve() (in module numpy_sugar.linalg), 10, 13
stl() (in module numpy_sugar.linalg), 11, 14
sum2diag() (in module numpy_sugar.linalg), 8, 11

T

trace2() (in module numpy_sugar.linalg), 9, 12

U

unique() (in module numpy_sugar._array), 6