Linear algebra (`numpy.linalg`)

The NumPy linear algebra functions rely on BLAS and LAPACK to provide efficient low level implementations of standard linear algebra algorithms. Those libraries may be provided by NumPy itself using C versions of a subset of their reference implementations but, when possible, highly optimized libraries that take advantage of specialized processor functionality are preferred. Examples of such libraries are OpenBLAS, MKL (TM), and ATLAS. Because those libraries are multithreaded and processor dependent, environmental variables and external packages such as threadpoolctl may be needed to control the number of threads or specify th processor architecture.

The SciPy library also contains a `linalg` submodule, and there is overlap in the functionality provided by the SciPy and NumPy submodules. SciPy contains functions not found in `numpy.linalg`, such as functions related to LU decomposition and the Schur decomposition, multiple ways of calculating the pseudoinverse, and matrix transcendentals such as the matrix logarithm. Some functions that exist in both have augmented functionality in `scipy.linalg`. For example, `scipy.linalg.eig` can take a second matrix argument for solving generalized eigenvalue problems. Some functions in NumPy, however, have more flexible broadcasting options. For example, `numpy.linalg.solve` can handle “stacked” arrays, while `scipy.linalg.solve` accepts only a single square array as its first argument.

Note

The term *matrix* as it is used on this page indicates a 2d `numpy.array` object, and *not* a `numpy.matrix` object. The latter is no longer recommended, even for linear algebra. See the matrix object documentation for more information.

The `@` operator

Introduced in NumPy 1.10.0, the `@` operator is preferable to other methods when computing the matrix product between 2d arrays. The `numpy.matmul` function implements the `@` operato

Back to top

Matrix and vector products

`dot`(a, b[, out])	Dot product of two arrays.
`linalg.multi_dot`(arrays, *[, out])	Compute the dot product of two or more arrays in a single function call, while automatically selecting the fastest evaluation order.
`vdot`(a, b, /)	Return the dot product of two vectors.
`vecdot`(x1, x2, /[, out, casting, order, ...])	Vector dot product of two arrays.
`linalg.vecdot`(x1, x2, /, *[, axis])	Computes the vector dot product.
`inner`(a, b, /)	Inner product of two arrays.
`outer`(a, b[, out])	Compute the outer product of two vectors.
`matmul`(x1, x2, /[, out, casting, order, ...])	Matrix product of two arrays.
`linalg.matmul`(x1, x2, /)	Computes the matrix product.
`matvec`(x1, x2, /[, out, casting, order, ...])	Matrix-vector dot product of two arrays.
`vecmat`(x1, x2, /[, out, casting, order, ...])	Vector-matrix dot product of two arrays.
`tensordot`(a, b[, axes])	Compute tensor dot product along specified axes.
`linalg.tensordot`(x1, x2, /, *[, axes])	Compute tensor dot product along specified axes.
`einsum`(subscripts, *operands[, out, dtype, ...])	Evaluates the Einstein summation convention on the operands.
`einsum_path`(subscripts, *operands[, optimize])	Evaluates the lowest cost contraction order for an einsum expression by considering the creation of intermediate arrays.
`linalg.matrix_power`(a, n)	Raise a square matrix to the (integer) power *n*.
`kron`(a, b)	Kronecker product of two arrays.

`linalg.cross`(x1, x2, /, *[, axis])	Returns the cross product of 3-element vectors.

Decompositions

`linalg.cholesky`(a, /, *[, upper])	Cholesky decomposition.
`linalg.outer`(x1, x2, /)	Compute the outer product of two vectors.
`linalg.qr`(a[, mode])	Compute the qr factorization of a matrix.
`linalg.svd`(a[, full_matrices, compute_uv, ...])	Singular Value Decomposition.
`linalg.svdvals`(x, /)	Returns the singular values of a matrix (or a stack of matrices) `x`.

Matrix eigenvalues

`linalg.eig`(a)	Compute the eigenvalues and right eigenvectors of a square array.
`linalg.eigh`(a[, UPLO])	Return the eigenvalues and eigenvectors of a complex Hermitian (conjugate symmetric) or a real symmetric matrix.
`linalg.eigvals`(a)	Compute the eigenvalues of a general matrix.
`linalg.eigvalsh`(a[, UPLO])	Compute the eigenvalues of a complex Hermitian or real symmetric matrix.

Norms and other numbers

`linalg.norm`(x[, ord, axis, keepdims])	Matrix or vector norm.

`linalg.matrix_norm`(x, /, *[, keepdims, ord])	Computes the matrix norm of a matrix (or a stack of matrices) `x`.
`linalg.vector_norm`(x, /, *[, axis, ...])	Computes the vector norm of a vector (or batch of vectors) `x`.
`linalg.cond`(x[, p])	Compute the condition number of a matrix.
`linalg.det`(a)	Compute the determinant of an array.
`linalg.matrix_rank`(A[, tol, hermitian, rtol])	Return matrix rank of array using SVD method
`linalg.slogdet`(a)	Compute the sign and (natural) logarithm of the determinant of an array.
`trace`(a[, offset, axis1, axis2, dtype, out])	Return the sum along diagonals of the array.
`linalg.trace`(x, /, *[, offset, dtype])	Returns the sum along the specified diagonals of a matrix (or a stack of matrices) `x`.

Solving equations and inverting matrices

`linalg.solve`(a, b)	Solve a linear matrix equation, or system of linear scalar equations.
`linalg.tensorsolve`(a, b[, axes])	Solve the tensor equation `a x = b` for x.
`linalg.lstsq`(a, b[, rcond])	Return the least-squares solution to a linear matrix equation.
`linalg.inv`(a)	Compute the inverse of a matrix.
`linalg.pinv`(a[, rcond, hermitian, rtol])	Compute the (Moore-Penrose) pseudo-inverse of a matrix.
`linalg.tensorinv`(a[, ind])	Compute the 'inverse' of an N-dimensional array.

Other matrix operations

`diagonal`(a[, offset, axis1, axis2])	Return specified diagonals.
`linalg.diagonal`(x, /, *[, offset])	Returns specified diagonals of a matrix (or a stack of matrices) `x`.
`linalg.matrix_transpose`(x, /)	Transposes a matrix (or a stack of matrices) `x`.

Exceptions

`linalg.LinAlgError`	Generic Python-exception-derived object raised by linalg functions.

Linear algebra on several matrices at once

Several of the linear algebra routines listed above are able to compute results for several matrices at once, if they are stacked into the same array.

This is indicated in the documentation via input parameter specifications such as `a : (..., M, M) array_like`. This means that if for instance given an input array `a.shape == (N, M, M)`, it is interpreted as a “stack” of N matrices, each of size M-by-M. Similar specification applies to return values, for instance the determinant has `det : (...)` and will in this case return an array of shape `det(a).shape == (N,)`. This generalizes to linear algebra operations on higher-dimensional arrays: the last 1 or 2 dimensions of a multidimensional array are interpreted as