Linear Algebra Functions in NumPy

Compiled by Sunil Kunwar

October 25, 2024

1 Basic Linear Algebra Functions

1.1 Matrix Operations

1.1.1 Matrix Multiplication

Description: Computes the dot product of two arrays or matrices.

```
import numpy as np

# Dot product
a = np.array([[1, 2], [3, 4]])
b = np.array([[5, 6], [7, 8]])
dot_product = np.dot(a, b) #can be done using @ operator
```

Description: Performs matrix multiplication, equivalent to the dot product for 2-D arrays.

```
# Matrix product
mat_product = np.matmul(a, b)
```

Description: Computes the dot product of two vectors.

```
# Dot product of two vectors
vdot_product = np.vdot(a[0], b[0])
```

1.1.2 Transpose

Description: Transposes the given array, flipping it over its diagonal.

```
# Transpose of an array
transposed_a = np.transpose(a)
```

Description: Swaps two axes of an array.

```
# Swap two axes of an array
swapped_axes = np.swapaxes(a, 0, 1)
```

1.1.3 Trace

Description: Computes the sum of the diagonal elements of a matrix.

```
# Trace of an array
trace_a = np.trace(a)
```

2 Solving Linear Systems

2.1 Solve $Ax = b$

Description: Solves the linear equation $Ax = b$ for x.

```
b = np.array([1, 2])
x = np.linalg.solve(a, b)  # Solving for x
```

2.2 Matrix Inversion

Description: Computes the inverse of a square matrix.

```
# Compute the inverse of a matrix
inv_a = np.linalg.inv(a)
```

2.3 Least-Squares Solution

Description: Computes the least-squares solution to a linear matrix equation.

```
# Least-squares solution
x_least_squares, residuals, rank, s = np.linalg.lstsq(a, b, rcond=
    None)
```

3 Decompositions

3.1 Eigenvalue Decomposition

Description: Computes the eigenvalues and right eigenvectors of a square array.

```
# Eigenvalue and eigenvector computation
eigenvalues, eigenvectors = np.linalg.eig(a)
```

3.2 Singular Value Decomposition (SVD)

Description: Factorizes a matrix into three matrices, representing its intrinsic properties.

```
# Singular Value Decomposition
u, s, vh = np.linalg.svd(a)
```

3.3 Cholesky Decomposition

Description: Decomposes a positive-definite matrix into a lower triangular matrix and its transpose.

```
# Cholesky decomposition
L = np.linalg.cholesky(a)
```

3.4 QR Decomposition

Description: Decomposes a matrix into an orthogonal matrix and an upper triangular matrix.

```
# QR decomposition
q, r = np.linalg.qr(a)
```

4 Norms and Determinants

4.1 Norms

Description: Computes the norm (length) of a vector or the Frobenius norm of a matrix.

```
# Matrix or vector norm
norm_a = np.linalg.norm(a, ord=None)
```

4.2 Determinant

Description: Computes the determinant of a square matrix.

```
# Compute the determinant of a matrix
det_a = np.linalg.det(a)
```

5 Other Useful Functions

5.1 Matrix Rank

Description: Returns the rank of a matrix, which is the dimension of the vector space generated by its rows or columns.

```
# Return the rank of a matrix
rank_a = np.linalg.matrix_rank(a)
```

5.2 Condition Number

Description: Computes the condition number of a matrix, indicating how sensitive the solution of a system of linear equations is to changes in the input.

```
# Compute the condition number of a matrix
cond_a = np.linalg.cond(a, p=None)
```

5.3 Pseudo-Inverse

Description: Computes the Moore-Penrose pseudo-inverse of a matrix.

```
# Compute the Moore-Penrose pseudo-inverse
pseudo_inv_a = np.linalg.pinv(a)
```

5.4 Cross Product

Description: Computes the cross product of two 3-dimensional vectors.

```
# Compute the cross product of two 3D vectors
a_vec = np.array([1, 2, 3])
b_vec = np.array([4, 5, 6])
cross_product = np.cross(a_vec, b_vec)
```

6 Utility Functions for Arrays

6.1 Identity Matrix

Description: Returns a 2D array with ones on the diagonal and zeros elsewhere.

```
# Return a 2D array with ones on the diagonal
identity_matrix = np.eye(3)
```

6.2 Diagonal

Description: Extracts or constructs a diagonal array.

```
# Extract or construct a diagonal array
diagonal_array = np.diag([1, 2, 3])
```

6.3 Flattening

Description: Returns a contiguous flattened array.

```
# Return a flattened array
flattened_a = np.ravel(a)
```

7 Advanced Linear Algebra Functions

7.1 Kronecker Product

Description: Computes the Kronecker product of two arrays.

```
# Compute the Kronecker product of two arrays
kron_product = np.kron(a, b)
```

7.2 Tensor Dot Product

Description: Computes the tensor dot product along specified axes.

```
# Compute the tensor dot product along specified axes
tensor_dot = np.tensordot(a, b, axes=1)
```

7.3 Element-wise Multiplication

Description: Performs element-wise multiplication of two arrays.

```python
# Element-wise multiplication
hadamard_product = np.multiply(a, b)
```

7.4 Matrix Power

Description: Raises a square matrix to an integer power n.

```python
# Raise a square matrix to the integer power n
matrix_power = np.linalg.matrix_power(a, 2)
```

8 Higher-Dimensional Tensors

8.1 Mode Product of a Tensor

Description: Computes the mode product of tensors.

```python
# Example of using einsum for tensor operations
tensor_product = np.einsum('ij,jk->ik', a, b)
```

9 Special Matrix Functions

9.1 Hermitian Matrix Check

Description: Checks if a matrix is Hermitian or symmetric.

```python
# Check if a matrix is Hermitian or symmetric
is_hermitian = np.iscomplexobj(a)
```