CSCI 1420 NumPy Guide

Basic Matrix Operations

$$\text{Let } A = \begin{pmatrix} a & b \\ c & d \end{pmatrix},\ B = \begin{pmatrix} e & f \\ g & h \end{pmatrix},\ C = \begin{pmatrix} i & j & k \\ l & m & n \end{pmatrix}.$$

Function	Example	Notes
`np.zeros(shape)`	`np.zeros((2, 2))` $= \begin{pmatrix} 0.0 & 0.0 \\ 0.0 & 0.0 \end{pmatrix}$	The default data type is float.
`np.ones(shape)`	`np.ones((2, 2))` $= \begin{pmatrix} 1.0 & 1.0 \\ 1.0 & 1.0 \end{pmatrix}$	The default data type is float.
`np.eye(n_rows)`	`np.eye(2)` $= \begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix}$	Identity matrix. The default data type is float.
`np.matmul(X, Y)`	`np.matmul(A, C)` $= \begin{pmatrix} ai+bl & aj+bm & ak+bn \\ ci+dl & cj+dm & ck+dn \end{pmatrix}$	Matrix multiplication. `X @ Y` is equivalent to `np.matmul(X, Y)`.
`np.add(X, Y)`	`np.add(A, B)` $= \begin{pmatrix} a+e & b+f \\ c+g & d+h \end{pmatrix}$	Element-wise addition. `X + Y` is equivalent to `np.add(X, Y)`.
`np.subtract(X, Y)`	`np.subtract(A, B)` $= \begin{pmatrix} a-e & b-f \\ c-g & d-h \end{pmatrix}$	Element-wise subtraction. `X - Y` is equivalent to `np.subtract(X, Y)`.
`np.multiply(X, Y)`	`np.multiply(A, B)` $= \begin{pmatrix} ae & bf \\ cg & dh \end{pmatrix}$	`np.multiply(X, Y)`, or `X * Y`, is element-wise multiplication. It is not the same as matrix multiplication (`np.matmul(X, Y)`).
`np.divide(X, Y)`	`np.divide(A, B)` $= \begin{pmatrix} a/e & b/f \\ c/g & d/h \end{pmatrix}$	Element-wise division. `X / Y` is equivalent to `np.divide(X, Y)`.
`np.transpose(X)`	`np.transpose(C)` $= \begin{pmatrix} i & l \\ j & m \\ k & n \end{pmatrix}$	`X.T` is equivalent to `np.transpose(X)`.
`np.mean(X)`	`np.mean(A)` $= \frac{a+b+c+d}{4}$	Computes the arithmetic mean of the (flattened) input array. You can also specify an `axis` along which to compute a mean.
`np.sum(X)`	`np.sum(A)` $= a+b+c+d$	Computes the sum all of the values in the input array. You can also specify an `axis` along which to compute a sum.
`np.reshape(X, shape)`	`np.reshape(A, (1, 4))` $= \begin{pmatrix} a & b & c & d \end{pmatrix}$	Reshapes array `X` into the specified `shape`, here transforming matrix `A` from a 2×2 to a 1×4 array. The total number of elements must remain constant.

Indexing

$$\text{Let } A = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix}.$$

Zero-Indexed Notation	NumPy Equivalent	Notes
$A_{2,2} = k$	`A[2, 2]` $= k$	Indexing one entry from a NumPy array returns a float.
$A_{1:3,1:3} = \begin{pmatrix} f & g \\ j & k \end{pmatrix}$	`A[1:3, 1:3]` $= \begin{pmatrix} f & g \\ j & k \end{pmatrix}$	NumPy indexing is very similar to Python list indexing.
$A_{1:4,0:2} = \begin{pmatrix} e & f \\ i & j \end{pmatrix}$	`A[1:, :2]` $= \begin{pmatrix} e & f \\ i & j \end{pmatrix}$	Indexing with `:i` returns all values up to but not including the value at index `i` in that dimension. Indexing with `i:` returns all values starting from and including the value at index `i` up to the last value in that dimension.
$A_{0:2,1} = \begin{pmatrix} b \\ f \end{pmatrix}$	`A[:2, 1]` $= \begin{pmatrix} b & f \end{pmatrix}$	Indexing a column from a NumPy array returns a 1-dimensional NumPy array. Reshape the output with, for example, `A[:2, 1].reshape(-1, 1)` if you specifically need a column vector.
$A_{1,0:2} = \begin{pmatrix} e & f \end{pmatrix}$	`A[1, :2]` $= \begin{pmatrix} e & f \end{pmatrix}$	Indexing a row from a NumPy array returns a 1-dimensional NumPy array.

Linear Algebra

$$\text{Let } A = \begin{pmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \end{pmatrix},\ B = \begin{pmatrix} m & n \\ o & p \end{pmatrix}$$

Function	Example	Notes
`np.linalg.norm(X)`	`np.linalg.norm(A[:,1])` $= \sqrt{b^2 + f^2 + j^2}$	By default, computes the L2 norm (Euclidean distance) of the input vector. If the input is a matrix, the default behaviour is to compute the Frobenius norm.
`np.linalg.inv(X)`	`np.linalg.inv(B)` $= B^{-1} = \frac{1}{mp-no} \begin{pmatrix} p & -n \\ -o & m \end{pmatrix}$	Computes the inverse of square matrix `X`. Raises an error if inversion fails.
`np.dot(X, Y)`	`np.dot(A[0], A[1])` $= \begin{pmatrix} a & b \end{pmatrix} \begin{pmatrix} c \\ d \end{pmatrix}$	Computes the inner (i.e dot) product of two arrays. For 2-dimenstional arrays, `np.dot(X, Y)` is equivalent to `X @ Y`. For computing general inner products, see `np.inner(X,Y)`.