NumPy and Matplotlib

Cholwich Nattee and Nirattaya Khamsemanan May 20, 2023

1 NumPy Arrays

A NumPy array is a data structure that is similar to a Python list, but it has some important differences. First, arrays are homogeneous, which means that all of the elements in an array must be of the same data type. This makes arrays more efficient for working with large amounts of data, as the computer does not have to keep track of different data types.

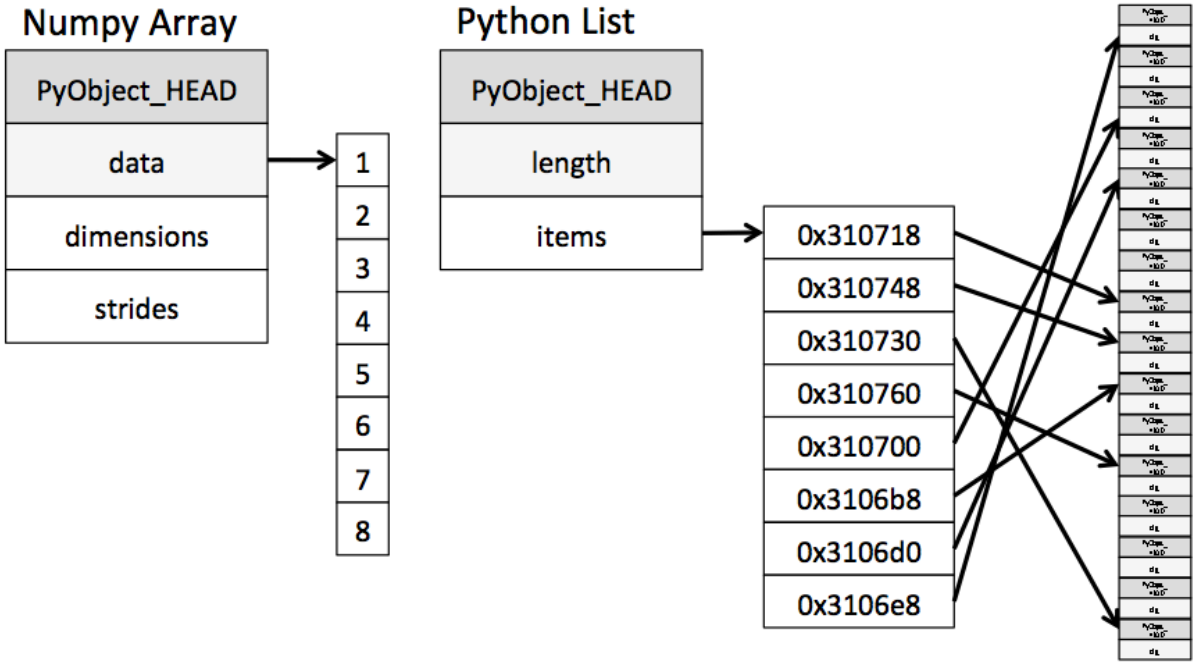
Second, arrays are more appropriate for working with the entire values at the same time. This is because arrays can be indexed and sliced, which allows you to access specific elements or subarrays. For example, you can use an array to represent a vector or a matrix, which are both important mathematical objects that are used in many different applications.

Here are some additional details about NumPy arrays:

- NumPy arrays are stored in contiguous memory, which makes them very efficient for mathematical operations.
- NumPy arrays support a wide range of mathematical operations, including addition, subtraction, multiplication, division, and more.
- NumPy arrays can be used to represent a variety of mathematical objects, including vectors, matrices, tensors, and more.
- NumPy arrays are a powerful tool for scientific computing and data analysis.

```
1 import numpy as np
2
3 a = np.array([1, 2, 3, 4])
4 b = np.array([[2], [3], [5]])
5 c = np.array([[1, 0, 0], [0, 1, 0], [0, 0, 1]])
6
7 print(a)
8 print(b)
9 print(c)
```

1.1 NumPy Array vs Python List



1.2 Dimensions and Shapes

We can check if an array is a vector or a matrix by using `np.ndim`. If the number of dimensions is 1, the array is a vector. If it is 2, the array is a matrix.

We can check the number of elements, or the number of rows and columns of an array using `np.shape`. This function returns a list of array dimensions.

```
1 print(a.ndim, a.shape)
2 print(b.ndim, b.shape)
3 print(c.ndim, c.shape)
```

1.3 Creating Arrays

The NumPy library provides a number of functions to create arrays. We have studied one function, i.e. `np.arange`. Here are functions that we can use to create arrays:

- `np.arange(start, stop, step)` creates an array with evenly spaced values starting from `start` and ending before `stop`. The parameter `step` specifies spacing between values.
- `np.zeros(shape)` creates an array filled with zeros. The parameter `shape` specifies the shape (the number of dimensions and the number of values in each dimension) of the array. It can be an integer or a list of integers.
- `np.ones(shape)` creates an array filled with ones.
- `np.identity(n)` creates an identity matrix (a 2-D array) when `n` is the number of rows and columns.

```
1 d = np.arange(1, 3, 0.5); print(d)
2 e = np.zeros([3, 3]);     print(e)
3 f = np.ones(4);           print(f)
4 g = np.identity(4);       print(g)
```

2 Array Operations

2.1 Array vs Scalar

An arithmetic operation between an array and a scalar is done by applying the operation with the scalar to all elements in the array.

```
a = np.arange(1, 11, 1)

print(2*a)
print(a-2)
print(a**2)
```

2.2 Array vs Array

An arithmetic operation between two arrays of the same shape is done by applying the operation between two elements at the same index.

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

$+$

1	0	0	0
0	1	0	0
0	0	1	0
0	0	0	1

$=$

2	2	3	4
5	7	7	8
9	10	12	12
13	14	15	17

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

$*$

1	0	0	0
0	2	0	0
0	0	3	0
0	0	0	4

$=$

1	0	0	0
0	12	0	0
0	0	33	0
0	0	0	64

※*This is not a matrix multiplication.*

```
a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]])
b = np.identity(4)
c = np.diag([1,2,3,4])

print(a+b)
print(a*c)
```

2.3 Dot Product and Matrix Multiplication

Since Python 3.5, The `@` operator can be used for a dot product operation and a matrix multiplication operation.

```
a = np.array([1, 0, 1, 0, 1])
b = np.array([1, 2, 3, 4, 5])
print(a @ b)

c = np.array([[1, 0], [2, 3]])
d = np.array([[1, 2, 3], [4, 5, 6]])
print(c @ d)
```

2.4 Broadcasting

NumPy arrays can only be operated on if they have the same shape. However, NumPy can replicate one array to have the same shape as the other array, allowing operations to be performed on them.
This is called broadcasting. Broadcasting allows NumPy to perform arithmetic operations on arrays of different shapes.

```
a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]])
b = np.array([0,1,2,3])
print(a+b)
```

$$\begin{array}{|cccc|} \hline 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \\ \hline \end{array} + \begin{array}{|cccc|} \hline 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ 0 & 1 & 2 & 3 \\ \hline \end{array} = \begin{array}{|cccc|} \hline 1 & 3 & 5 & 7 \\ 5 & 7 & 9 & 11 \\ 9 & 11 & 13 & 15 \\ 13 & 15 & 17 & 19 \\ \hline \end{array}$$

```
a = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12],[13,14,15,16]])
c = np.array([[0],[1],[2],[3]])
print(a*c)
```

$$\begin{array}{|cccc|} \hline 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \\ 13 & 14 & 15 & 16 \\ \hline \end{array} * \begin{array}{|cccc|} \hline 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 2 & 2 & 2 & 2 \\ 3 & 3 & 3 & 3 \\ \hline \end{array} = \begin{array}{|cccc|} \hline 0 & 0 & 0 & 0 \\ 5 & 6 & 7 & 8 \\ 18 & 20 & 22 & 24 \\ 39 & 42 & 45 & 48 \\ \hline \end{array}$$

3 Reshaping

The `np.reshape` function changes the shape of an array into a specified shape. However, the new shape must reflect the same number of elements.

```
a = np.arange(1, 11, 1)
b = np.reshape(a, (2, 5))
c = np.reshape(a, (5, 2))
d = np.reshape(a, (10, 1))

print(b)
print(c)
print(d)
```

4 Reductions

NumPy provides a number of functions to summarize or reduce the values in an array, e.g. `np.sum`, `np.max`, `np.min`, `np.median`, `np.mean`, `np.std`.

For 1-D arrays, these functions summarize all elements in the arrays.

For 2-D arrays, we can choose if we want to summarize all elements, by rows, or by columns. This can be chosen by specifying `axis` parameter.

```
1 import numpy as np
2
3 a = np.arange(1, 11, 1)
4 print(np.sum(a), end=" ")
5 print(np.max(a), end=" ")
6 print(np.min(a))
```

```
1 import numpy as np
2
3 b = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
4
5 print(np.sum(b))
6 print(np.sum(b, axis=1)) # sum every column in the same row
7 print(np.sum(b, axis=0)) # sum every row in the same column
```

The `np.sort` is a function returning the array ordered by the element values in the ascending order. The `np.argsort` function returns the indices when the array is sorted.

```
1 import numpy as np
2
3 a = np.array([10, 2, 3, 1, 20, 12, 5])
4 print(np.sort(a))
5 print(np.argsort(a))
```

Similarly, `np.argmax` returns the index of the maximum element in the array.

Example 1. Consider a vector v that represents a coordinate in three-dimensional space, and a matrix D with dimensions 4×3, where each row of the matrix represents a three-dimensional coordinate.

Write code to compute Euclidean distance between v and each row of D.

```
v = np.array([1.4, 5.0, -2.1])
D = np.array([[0.3, 4.2, 1.8],
              [6.7, 2.3, 7.8],
              [4.2, 0.7,-1.5],
              [1.2, 5.1,-1.9]])
```

Example 2. Write code to estimate the following integral using the (left) Riemann sum.

$$\int_0^b e^{-x^2} dx \approx \sum_{k=0}^{N-1} e^{-x_k^2} \Delta x$$

Note that `np.linspace(start, stop, num)` returns evenly spaced numbers over a specified interval.

5 Linear Algebra with NumPy

Function	Description
`.T`	Transpose
`np.linalg.inv`	Inverse
`np.linalg.det`	Determinant
`np.linalg.matrix_pow`	Matrix Powers

Example 3. A system of linear equations can be converted into matrices. We can then solve the system by calculating inverse matrix.
For example, we have a system of three linear equations with three variables.

$$\begin{aligned} 5x + 15y + 56z &= 35 \\ -4x - 11y - 41z &= -26 \\ -x - 3y - 11z &= -7 \end{aligned}$$

The system can be written in form of matrix multiplication as below.

$$\begin{bmatrix} 5 & 15 & 56 \\ -4 & -11 & -41 \\ -1 & -3 & -11 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 35 \\ -26 \\ -7 \end{bmatrix}$$

The system can be solved by multiplying the inverse matrix to both sides.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 & 15 & 56 \\ -4 & -11 & -41 \\ -1 & -3 & -11 \end{bmatrix}^{-1} \begin{bmatrix} 35 \\ -26 \\ -7 \end{bmatrix}$$

Write a code using NumPy library to solve the system of equations.

6 Matplotlib

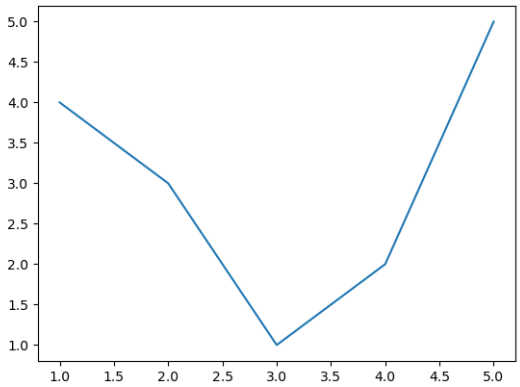
Matplotlib is a package for 2D plotting and data visualization. It provides a wide range of tools and functionalities for creating high-quality plots, charts, and graphs. It was designed to resemble MATLAB's plotting capabilities and is widely used in the scientific and data analysis communities.

```
1 import matplotlib.pyplot as plt
```

To create a plot,

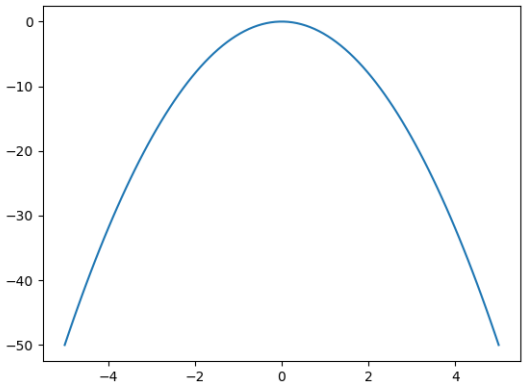
- Create an array of x values.
- Create an array of y values.
- Use `plt.plot(x, y, [fmt])` to create a plot where `fmt` specifies the styles.
- Use `plt.show()` to display the result.

```
1 x = np.array([1, 2, 3, 4, 5])
2 y = np.array([4, 3, 1, 2, 5])
3 plt.plot(x, y)
4 plt.show()
```



To plot a function, we can use `np.linspace` to create x and use NumPy functions to create y.

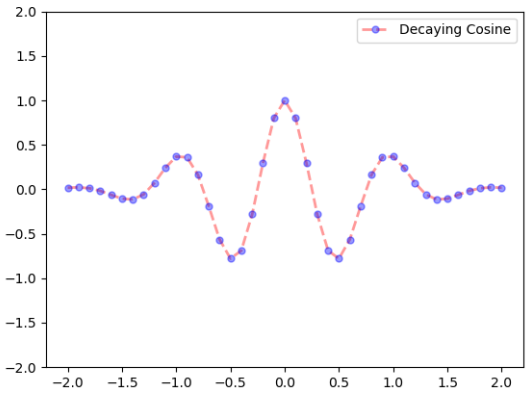
```
x = np.linspace(-5, 5, 100)
y = -2*x**2
plt.plot(x, y)
plt.show()
```



Here are some parameters:

`alpha`	transparency
`color (c)`	color
`label`	text appearing in legend
`linestyle (ls)`	`solid`, `dashed`, `dashdot`, `dotted`
`linewidth (lw)`	line width
`marker`	marker style `https://matplotlib.org/stable/api/markers_api.html`
`markeredgecolor (mec)`	marker edge color
`markerfacecolor (mfc)`	marker face color
`markersize (ms)`	marker size

```
x = np.linspace(-2,2,41)
y = np.exp(-x**2) * np.cos(2*np.pi*x)
plt.plot(x,y,alpha=0.4,label='Decaying Cosine',
         color='red',linestyle='dashed',linewidth=2,
         marker='o',markersize=5,markerfacecolor='blue',
         markeredgecolor='blue')
plt.ylim([-2,2])
plt.legend()
plt.show()
```



6.1 Scatter plot

Scatter plot is a simple technique to visualize datasets.

```
from sklearn.datasets import load_iris

iris = load_iris()

print(iris.feature_names)
print(iris.data)
```

```
x = iris.data[:, 0]
y = iris.data[:, 1]
z = iris.data[:, 2]
plt.scatter(x, y, alpha=0.4,
            s=100*z, c=iris.target)
plt.show()
```

6.2 Principal Component Analysis

Principal Component Analysis (PCA) is a statistical technique that reduces the dimensionality of data while preserving its important features by transforming it into a new set of uncorrelated variables called principal components.

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
iris_reduced = pca.fit_transform(iris.data)
print(iris_reduced.shape)
```

```
plt.scatter(iris_reduced[:, 0], iris_reduced[:, 1], c=iris.target)
plt.show()
```

6.3 Loading Data

The `np.loadtxt` is used to load a text file with values separated by specified characters, such as commas (`,`).

```
D = np.loadtxt('sample_data/mnist_train_small.csv', delimiter=',')
D.shape
```

```
d = D[0, 1:].reshape((28, 28))
plt.imshow(d, cmap='gray')
plt.show()
```

Example 4. Use PCA to visualize the MNIST dataset.

References

- https://patrickwalls.github.io/mathematicalpython