Intro to NumPy

Dr Alan Ferguson

2024-10-11

Objectives

- Understand purpose and advantages of using NumPy.
- Perform basic array operations in NumPy:
  - creation
  - indexing
  - slicing
  - basic computations

What is NumPy?

- Open-source Python library for scientific computing
- Used for working with multi-dimensional *arrays* (which are more efficient than native Python lists)
- Essential for solving engineering tasks using Python, e.g.:
  - signal processing (*SciPy*)
  - machine learning (*scikit-learn*)
  - data science (*Pandas*)

NumPy

Why use NumPy?

- Efficient array operations (faster than Python lists due to better memory usage)
- Able to handle large datasets
- Supports mathematical operations directly on arrays

E.g. Compute the sine of the first 10,000 integers

Python Lists

```
%%timeit
import math

sines = []
for i in range(10000):
    angle = i / (2.0 * math.pi)
    sines.append(math.sin(angle))
```

```
1.16 ms ± 539 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
```

Numpy Arrays

```
%%timeit
import numpy as np

angles = np.arange(10000) / (2.0 * np.pi)
sines = np.sin(angles)
```

```
64.4 µs ± 7.2 µs per loop (mean ± std. dev. of 7 runs, 10,000 loops each)
```

Numpy Basics

How to install NumPy

```
pip install numpy
```

How to import NumPy

```
import numpy as np
```

NOTE: typically shortened to np

Creating arrays

1. Using lists to create arrays:

```
arr = np.array([1, 2, 3])
```

2. Use functions to generate arrays:

- Filled with zeros

```
arr = np.zeros(3)  # array([0., 0., 0. ])
```

- Filled with ones

```
arr = np.ones(3)   # array([1., 1., 1. ])
```

- Evenly spaced values in interval

```
arr = np.arange(3) # array([0, 1, 2])
arr = np.arange(1,3) # array([1, 2])
arr = np.arange(1, 6, 2) # array([1, 3, 5])
```

Equivalent to Python `range()` for ints

Array Dimensions and Shapes

- Number of axes of an array is called its dimension, e.g.:
  - 1-D array has 1 axis,
  - 2-D array has 2 axes, or
- The lengths of all the dimensions is called the shape of the array, e.g.:
  - has shape (3,), i.e. 3 elements
  - has shape (2, 2), i.e. 2 rows and 2 columns
  - has shape (2, 4), i.e. 2 rows and 2 columns

0-D (Scalars)

```
val = np.array(42)
print(val)
print(val.shape)
```

```
42
()
```

1-D (Vectors)

```
vec = np.array([1, 2, 3])
print(vec)
print(vec.shape)
```

```
[1 2 3]
(3,)
```

2-D (Matrices)

```
mat = np.array([[1, 2], [3, 4]])
print(f"{mat=}")
print(f"{mat.shape=}")
print()

mat = np.array([[0, 1, 2, 3], [4, 5, 6, 7]])
print(f"{mat=}")
print(f"{mat.shape=}")
```

```
mat=array([[1, 2],
       [3, 4]])
mat.shape=(2, 2)
```

```
mat=array([[0, 1, 2, 3],
        [4, 5, 6, 7]])
mat.shape=(2, 4)
```

Basic Operations

- Can perform elementwise addition, multiplication, etc.:

```python
a = np.array([1, 2, 3])
print(f"{a=}")
b = 2  * (a + 1)
print(f"{b=}")
```

```
a=array([1, 2, 3])
b=array([4, 6, 8])
```

- Also supports most mathematical functions, e.g.:

```python
a = np.array([1, 2, 3, 4, 5])
mean = np.mean(a)
std = np.std(a)
print(f"{mean=}")
print(f"{std=}")
```

```
mean=3.0
std=1.4142135623730951
```

- https://numpy.org/doc/stable/reference/

Indexing and slicing

- Can access elements via indices

```
a = np.arange(6)
a[0], a[2], a[-1]
```

```
(0, 2, 5)
```

- or as slices:

```
a[1:3], a[::-1]
```

```
(array([1, 2]), array([5, 4, 3, 2, 1, 0]))
```

- Slices are denoted by `start:end:step`, where:
    - `start` is the first index inside the slice
    - `end` is the first index AFTER the slice
    - `step` is increment of index between elements of slice
- Also have the empty slice `:` which extracts all of that dimension
- Also supports negative indices

- Using indexing and slices, we can modify parts of the array, e.g.:

```
arr = np.zeros((3, 3))
arr[(0, 0)] = 1 # make first element of first row 1
arr[(-1, -1)] = 100 # make last element of last row 100

```

Exercise

Objective

Simulate and analyse a basic time-series signal by generating data points, manipulating the data, and perform simple operations using NumPy

Task

Simulate a time-series signal, representing a noisy sine wave, and perform basic analysis on this:

- Step 1: Create timesteps
- Step 2: Generate sine wave
- Step 3: Add noise
- Step 4: Basic analysis (mean/std/min/max)
- Step 5: Extract segment from 2s to 4s and calculate mean
- Step 6: Smooth signal
- Step 7: Plot signal