Exercise 1 - Part 2: Introduction to Numpy

October 11, 2024

Gestione Energetica ed Automazione negli Edifici (GEAE) A.A. 2024/2025

1 Introduction to Numpy

1.1 Importing Numpy

In this section, we import the NumPy library, which is fundamental for numerical computing in Python. NumPy provides efficient data structures, such as arrays, that can be used to perform mathematical operations easily and efficiently. Importing NumPy is the first step to leverage its powerful features for energy analysis and automation tasks.

[1]:
```
import numpy as np
```

1.2 Creating NumPy Arrays

Creating arrays in NumPy is crucial because they are the primary data structure used for storing and manipulating numerical data. In this section, we create a 1D array (vector) that holds temperature values in Celsius. This example helps to introduce basic array creation, demonstrating the simplicity and efficiency of working with numerical data in NumPy compared to standard Python lists.

[2]:
```
temperatures_celsius = np.array([20, 25, 30, 35, 40])
print("Temperatures in Celsius:", temperatures_celsius)
```

```
Temperatures in Celsius: [20 25 30 35 40]
```

Creating a 2D array (matrix) representing a grid of pressure values. This introduces higher-dimensional arrays, which are essential for representing more complex datasets, such as spatial grids or matrices that often arise in energy systems modeling.

[3]:
```
pressures = np.array([
    [1.0, 1.2, 1.4],
    [1.1, 1.3, 1.5],
    [1.2, 1.4, 1.6]
```

```
])
print("Pressure grid (bar):\n", pressures)
```

```
Pressure grid (bar):
 [[1.  1.2 1.4]
 [1.1 1.3 1.5]
 [1.2 1.4 1.6]]
```

1.3 Array Operations

Converting temperatures from Celsius to Kelvin. Temperature values are converted from Celsius to Kelvin by applying a basic arithmetic operation directly to the entire NumPy array. This demonstrates the convenience of array-wide operations in NumPy, which is much faster and more readable than using loops to manipulate each element individually.

[4]:
```
temperatures_kelvin = temperatures_celsius + 273.15
print("Temperatures in Kelvin:", temperatures_kelvin)
```

```
Temperatures in Kelvin: [293.15 298.15 303.15 308.15 313.15]
```

Calculating the average temperature. The average temperature is calculated using NumPy's built-in functions. Statistical operations like average (mean) are important when analyzing datasets to identify general trends or patterns, such as average energy consumption or environmental temperatures.

[5]:
```
average_temperature = np.mean(temperatures_celsius)
print("Average temperature (°C):", average_temperature)
```

```
Average temperature (°C): 30.0
```

Performing element-wise multiplication.

[6]:
```
volumes = np.array([1.0, 1.5, 2.0, 2.5, 3.0])  # in cubic meters
# Calculating the product of pressures and volumes (P*V)
pv_product = pressures[0, 0] * volumes  # Using the first pressure value for␣
 ↪simplicity
print("P*V product:", pv_product)
```

```
P*V product: [1.  1.5 2.  2.5 3. ]
```

1.4 Multidimensional Arrays

In this section, 3D arrays are introduced, which can be used to represent more complex data structures. The 3D array helps in understanding how to work with higher-dimensional data, which is often necessary in real-world energy management applications. Creating a 3D array representing energy consumption over time and different units:

[7]:
```
energy_consumption = np.random.rand(2, 3, 4)  # Random values for demonstration
print("Energy consumption data:\n", energy_consumption)
```

```
Energy consumption data:
 [[[0.59238517 0.00975617 0.22662193 0.55337929]
  [0.82064181 0.8024685  0.20736114 0.853059  ]
  [0.68841606 0.20352671 0.1569099  0.69020491]]

 [[0.02156481 0.81779626 0.85579201 0.87172155]
  [0.91718953 0.47109075 0.91758818 0.18282847]
  [0.93890796 0.00312793 0.28386865 0.94308346]]]
```

1.5 Indexing and Slicing

Accessing elements in an array. Indexing is used to access specific elements in an array. This is fundamental for retrieving and analyzing data points, such as extracting a specific value from a dataset.

[8]:
```
first_temperature = temperatures_celsius[0]
print("First temperature (°C):", first_temperature)
```

```
First temperature (°C): 20
```

Slicing arrays. Slicing is a powerful way to select sub-portions of an array. It allows for efficient manipulation of data, such as selecting data for a specific period or subset of sensors. Understanding slicing is key to managing and analyzing time-series data in energy systems.

[9]:
```
subset_temperatures = temperatures_celsius[1:4]
print("Subset of temperatures (°C):", subset_temperatures)
```

```
Subset of temperatures (°C): [25 30 35]
```

Accessing elements in a 2D array. Accessing elements in a 2D array demonstrates how to work with tabular data. For example, this could represent temperature measurements from multiple locations, where accessing a specific row or column helps to analyze trends at a particular site.

[10]:
```
pressure_value = pressures[1, 2]
print("Pressure at position (1,2) (bar):", pressure_value)
```

```
Pressure at position (1,2) (bar): 1.5
```

1.6 Mathematical Functions

In this section, mathematical functions provided by NumPy are applied. These functions are essential tools for transforming data and performing necessary calculations.

[11]:
```
radii = np.array([0.05, 0.1, 0.15])  # in meters
areas = np.pi * radii ** 2  # Calculating areas of circles
print("Areas of circles (m^2):", areas)
```

```
Areas of circles (m^2): [0.00785398 0.03141593 0.07068583]
```

1.7 Statistical Functions

Generating a dataset of measured temperatures with random noise

[12]:
```
measured_temperatures = temperatures_celsius + np.random.normal(0, 0.5,␣
↪temperatures_celsius.shape)
print("Measured temperatures (°C):", measured_temperatures)
```

```
Measured temperatures (°C): [19.56116205 24.99801458 29.76369492 34.49982659
40.06415976]
```

Calculating standard deviation

[13]:
```
std_dev = np.std(measured_temperatures)
print("Standard deviation of measured temperatures (°C):", std_dev)
```

```
Standard deviation of measured temperatures (°C): 7.146089418486741
```