📘 Module-2: NumPy, SciPy, and Matplotlib

🔹 Introduction

This module introduces advanced capabilities of NumPy and SciPy libraries, which are widely used for scientific computing in Python. NumPy provides powerful subpackages for linear algebra, Fourier transforms, random number generation, and polynomial handling. SciPy builds on NumPy to offer additional functionalities like integration, interpolation, and optimization.

In addition, the module covers **Matplotlib**, a popular data visualisation library used to create 2D plots such as line graphs, bar charts, and pie charts. Learners will explore how to annotate plots, add legends, and save graphs to image files.

💻 Python Programmes

🔹 NumPy Subpackages

1. Linear Algebra with NumPy (linalg)

```
import numpy as np
a = np.array([[3, 1], [1, 2]])
b = np.linalg.inv(a)
print("Inverse of matrix:\n", b)
```

---

2. Fast Fourier Transform (fft)

```
import numpy as np
signal = np.array([0, 1, 0, -1])
fft_result = np.fft.fft(signal)
print("FFT of signal:", fft_result)
```

---

3. Random Numbers (random)

```
import numpy as np
random_arr = np.random.rand(3, 3)
print("Random array:\n", random_arr)
```

---

4. Polynomials with NumPy

```
import numpy as np
p = np.polynomial.Polynomial([1, 2, 3])  # 1 + 2x + 3x^2
print("Evaluate at x=2:", p(2))
print("Derivative:", p.deriv())
```

◆ SciPy Subpackages

5. Linear Algebra with SciPy (scipy.linalg)

```
import scipy.linalg as la
a = [[4, 3], [6, 3]]
lu, piv = la.lu(a)
print("LU Decomposition:\n", lu)
```

---

6. Fourier Transform using fftpack

```
from scipy.fftpack import fft
import numpy as np
x = np.array([0.0, 1.0, 0.0, -1.0])
y = fft(x)
print("FFT using fftpack:", y)
```

---

7. Integration with scipy.integrate

```
from scipy import integrate
import numpy as np
result, _ = integrate.quad(lambda x: x**2, 0, 1)
print("Integral of x^2 from 0 to 1:", result)
```

---

8. Interpolation with scipy.interpolate

```
from scipy import interpolate
import numpy as np
x = np.array([0, 1, 2, 3])
y = np.array([0, 1, 4, 9])
f = interpolate.interp1d(x, y)
print("Interpolated value at x=2.5:", f(2.5))
```

---

9. Optimization with scipy.optimize

```
from scipy.optimize import minimize
f = lambda x: x**2 + 5*np.sin(x)
result = minimize(f, x0=2)
print("Minimum value at x =", result.x)
```

◆ Matplotlib Basics

10. Simple Line Plot

```
import matplotlib.pyplot as plt
x = [1, 2, 3, 4]
y = [10, 20, 25, 30]
plt.plot(x, y)
plt.title("Line Plot")
```

```
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.show()
```

---

11. **Bar Chart**

```
import matplotlib.pyplot as plt
labels = ['A', 'B', 'C']
values = [5, 7, 3]
plt.bar(labels, values)
plt.title("Bar Chart")
plt.show()
```

---

12. **Pie Chart**

```
import matplotlib.pyplot as plt
sizes = [40, 35, 25]
labels = ['Python', 'Java', 'C++']
plt.pie(sizes, labels=labels, autopct='%1.1f%%')
plt.title("Language Usage")
plt.show()
```

---

13. **Add Legend and Annotation**

```
import matplotlib.pyplot as plt
x = [1, 2, 3]
y = [2, 4, 1]
plt.plot(x, y, label="Line 1")
plt.legend()
plt.annotate('Peak', xy=(2, 4), xytext=(2.2, 4.5),
arrowprops=dict(arrowstyle='->'))
plt.title("Annotated Plot")
plt.show()
```

---

14. **Saving Plot to File**

```
import matplotlib.pyplot as plt
x = [1, 2, 3]
y = [1, 4, 9]
plt.plot(x, y)
plt.title("Save to File")
plt.savefig("myplot.png")
```

---

15. **Multiple Line Plots**

```
import matplotlib.pyplot as plt
x = [0, 1, 2, 3]
y1 = [0, 1, 4, 9]
y2 = [0, 1, 2, 3]
plt.plot(x, y1, label="x^2")
plt.plot(x, y2, label="x")
plt.legend()
```

```
plt.title("Multiple Line Plots")
plt.show()
```