Python Programming

Lab:- 30(Scipy Transform,Interpolation & IO)

Student Id:- AF0417098

Student Name:- Ankush

Introduction to SciPy

SciPy is a powerful Python library used for scientific and technical computing. It builds on top of NumPy and provides a wide range of functions and tools for mathematical computations, optimization, integration, interpolation, eigenvalue problems, and signal processing, among many other tasks.

SciPy is essential for data scientists, engineers, and researchers who require advanced computational methods. It is widely used for solving problems in scientific computing, mathematics, engineering, and machine learning.

1. SciPy vs. NumPy

SciPy builds upon NumPy, which is the fundamental package for array operations in Python. NumPy handles basic array operations like matrix and vector multiplication, while SciPy provides higher-level functions to perform operations such as optimization, signal processing, and solving differential equations.

- NumPy is mainly for numerical operations with arrays and matrices.
- SciPy extends NumPy and provides a collection of mathematical algorithms and convenience functions that make scientific computing easier.

2. SciPy Ecosystem

SciPy is part of the broader SciPy ecosystem which includes libraries like:

- NumPy: For numerical operations and array manipulation.
- Matplotlib: For data visualization.
- Pandas: For data manipulation and analysis.
- SymPy: For symbolic computation.
- scikit-learn: For machine learning.

3. Installation of SciPy

To install SciPy, you can use Python's package manager, pip:

```
pip install scipy
```

Once installed, you can import the library into your Python script:

```
import scipy
```

4. SciPy Sub-packages

SciPy is organized into sub-packages, each focusing on a specific scientific computing domain. Here are some of the key sub-packages:

- **scipy.integrate:** For integration (numerical integration, solving ordinary differential equations).
- **scipy.optimize:** For optimization and root finding (finding minimums and maximums).
- **scipy.linalg:** For linear algebra operations (matrix decompositions, solving systems of linear equations).
- **scipy.fftpack:** For Fourier transforms (used in signal processing and other areas).
- **scipy.signal:** For signal processing (filtering, convolution, etc.).

- scipy.spatial: For working with spatial data structures and algorithms (KD-trees, distance computations).
- scipy.stats: For statistical functions (probability distributions, hypothesis testing)

5.Real-World Applications of SciPy

- SciPy is versatile and widely used in several fields:
- Engineering: Used for solving differential equations, system modeling, signal processing, etc.
- Physics: For simulations, modeling dynamic systems, and solving physical equations.
- Machine Learning: SciPy is often used in feature extraction and data processing stages of machine learning pipelines.
- Finance: Optimization algorithms help solve financial modeling problems, portfolio optimization, and risk management.

6. Why Use SciPy?

- Ease of Use: SciPy provides high-level functions for complex mathematical operations, making it accessible even for non-experts in numerical computing.
- Performance: SciPy is optimized for performance, leveraging compiled libraries such as BLAS, LAPACK, and Fortran.
- Comprehensive: It has an extensive range of functions that cover most aspects of scientific computing.

Assignment Questions:-

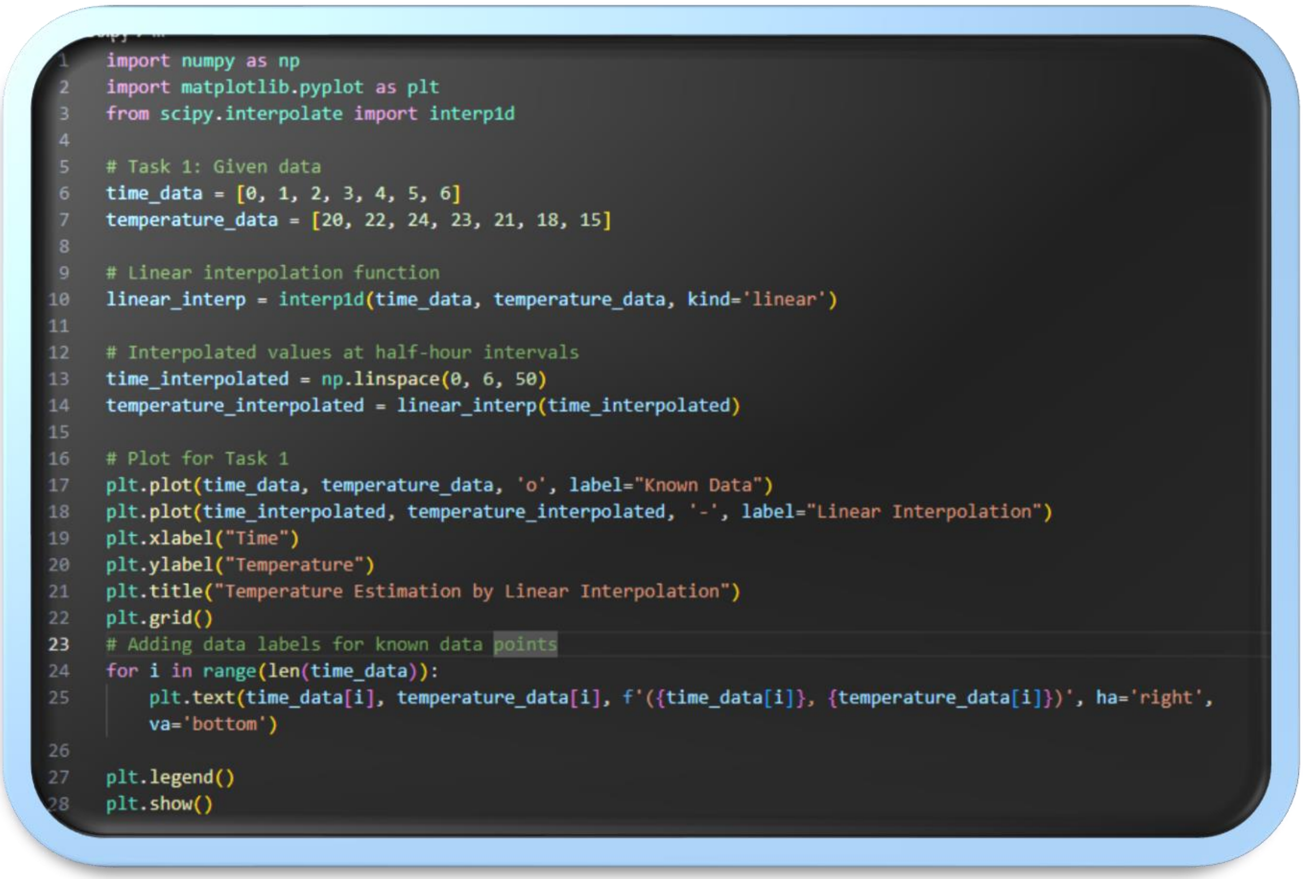
Task 1: To Find estimate temperature given known data points by using interpolation:

Input -

time_data =[0, 1, 2, 3, 4, 5, 6])

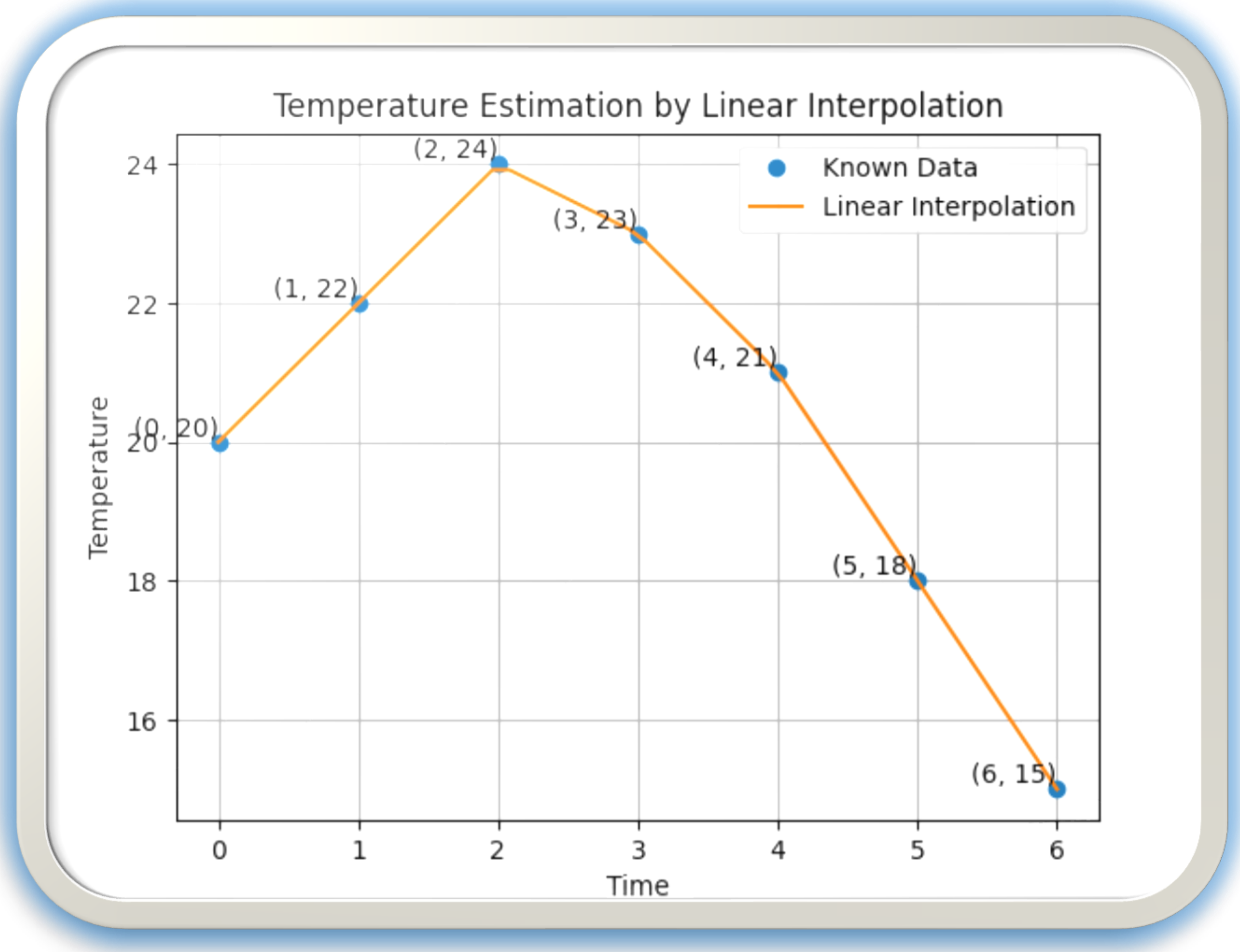
temperature_data=[20, 22, 24, 23, 21, 18, 15]

Program:-



```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.interpolate import interp1d

# Task 1: Given data
time_data = [0, 1, 2, 3, 4, 5, 6]
temperature_data = [20, 22, 24, 23, 21, 18, 15]

# Linear interpolation function
linear_interp = interp1d(time_data, temperature_data, kind='linear')

# Interpolated values at half-hour intervals
time_interpolated = np.linspace(0, 6, 50)
temperature_interpolated = linear_interp(time_interpolated)

# Plot for Task 1
plt.plot(time_data, temperature_data, 'o', label="Known Data")
plt.plot(time_interpolated, temperature_interpolated, '-', label="Linear Interpolation")
plt.xlabel("Time")
plt.ylabel("Temperature")
plt.title("Temperature Estimation by Linear Interpolation")
plt.grid()
# Adding data labels for known data points
for i in range(len(time_data)):
    plt.text(time_data[i], temperature_data[i], f'({time_data[i]}, {temperature_data[i]})', ha='right',
    va='bottom')

plt.legend()
plt.show()
```

Output:-



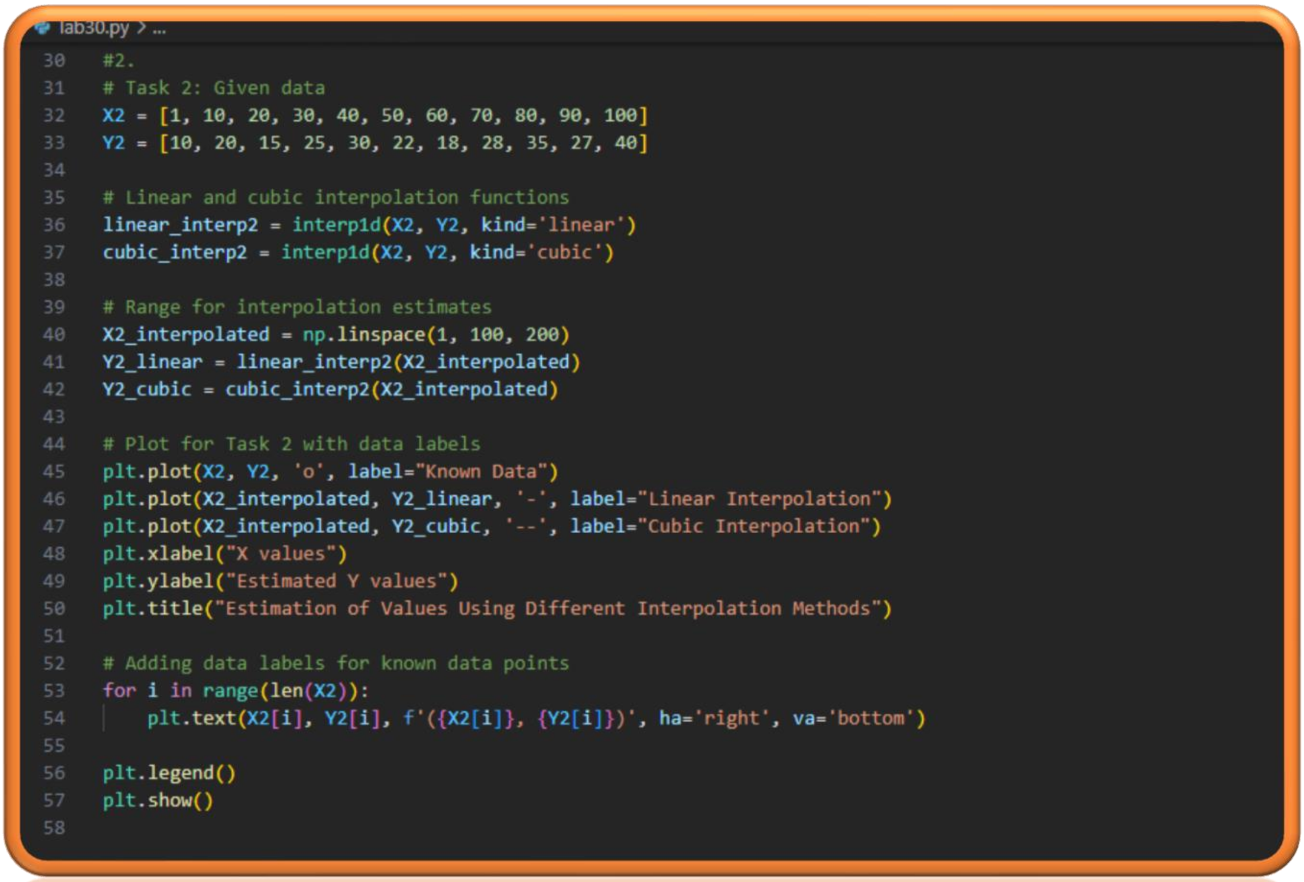
Task 2: To Find estimate values range 1 to 100 with known data values

X2= known data

Y2= Estimate data

with different types of interpolation chart.

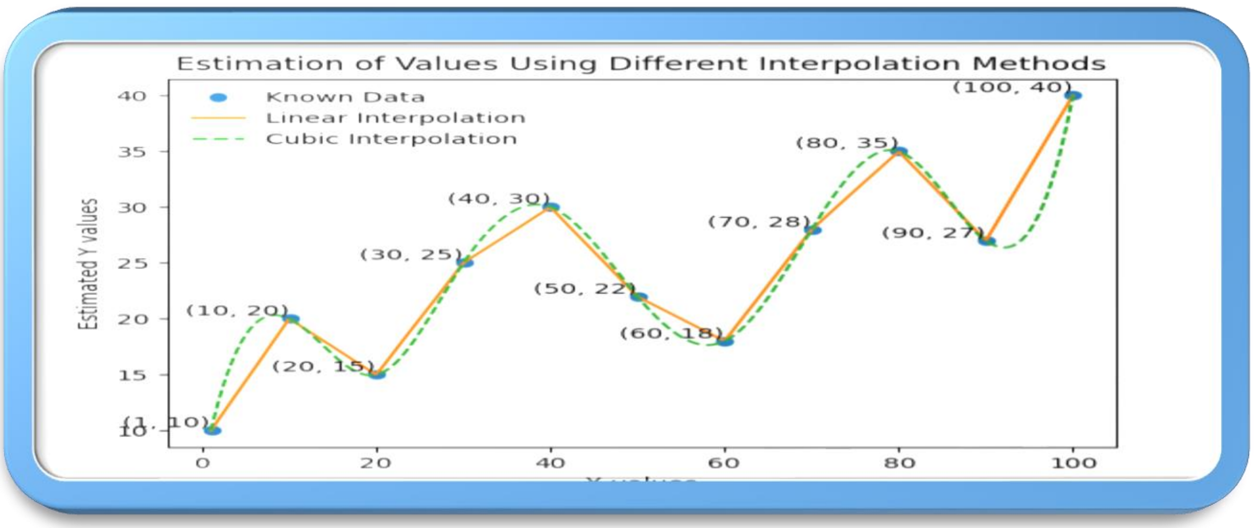
Program:-



lab30.py > ...

```python
#2.
# Task 2: Given data
X2 = [1, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
Y2 = [10, 20, 15, 25, 30, 22, 18, 28, 35, 27, 40]

# Linear and cubic interpolation functions
linear_interp2 = interp1d(X2, Y2, kind='linear')
cubic_interp2 = interp1d(X2, Y2, kind='cubic')

# Range for interpolation estimates
X2_interpolated = np.linspace(1, 100, 200)
Y2_linear = linear_interp2(X2_interpolated)
Y2_cubic = cubic_interp2(X2_interpolated)

# Plot for Task 2 with data labels
plt.plot(X2, Y2, 'o', label="Known Data")
plt.plot(X2_interpolated, Y2_linear, '-', label="Linear Interpolation")
plt.plot(X2_interpolated, Y2_cubic, '--', label="Cubic Interpolation")
plt.xlabel("X values")
plt.ylabel("Estimated Y values")
plt.title("Estimation of Values Using Different Interpolation Methods")

# Adding data labels for known data points
for i in range(len(X2)):
    plt.text(X2[i], Y2[i], f'({X2[i]}, {Y2[i]})', ha='right', va='bottom')

plt.legend()
plt.show()

```

Output:-


Estimation of Values Using Different Interpolation Methods
Known Data
Linear Interpolation
Cubic Interpolation
Estimated Y values
(1, 10)
(10, 20)
(20, 15)
(30, 25)
(40, 30)
(50, 22)
(60, 18)
(70, 28)
(80, 35)
(90, 27)
(100, 40)