ROUTING OPTIMIZATION FOR AERONAUTICAL NETWORKS.

MINI PROJECT

1) DIVYANSH RAWAL - 21BLC1123
2) SAI KRISHNA VS - 21BLC1211
3) HARSH SINHA - 21BLC1419

EXPLANATION OF THE CODE

```
IMPORT PANDAS AS PD
IMPORT MATPLOTLIB.PYPLOT AS PLT
IMPORT NUMPY AS NP

# READ THE DATASET
DATA = PD.READ_CSV('PLANE.CSV')

# MODIFY THE COLUMN NAMES TO MATCH THE DATASET
DATA.COLUMNS = ['FLIGHT NO.', 'TIMESTAMP', 'ALTITUDE', 'LATITUDE',
'LONGITUDE’]
```

1. *IMPORTING LIBRARIES:*
    1. *The code imports necessary libraries such as pandas, matplotlib.pyplot, numpy, and math for data manipulation, plotting, and mathematical calculations.*
2. *READING AND MODIFYING THE DATASET:*
    1. *The code reads a CSV file named 'plane.CSV' using the pandas library and assigns it to the 'data' variable.*
    2. *The column names of the DataFrame are modified to match the dataset.*

```
FROM MPL_TOOLKITS.MPLOT3D IMPORT AXES3D

# PLOTTING FLIGHT PATHS IN 3D
FIG = PLT.FIGURE(FIGSIZE=(10, 8))
AX = FIG.ADD_SUBPLOT(111, PROJECTION='3D')

FOR AIRPLANE IN AIRPLANES:
    FLIGHT_NO, _, ALTITUDE, LATITUDE, LONGITUDE = AIRPLANE
    AX.SCATTER(LONGITUDE, LATITUDE, ALTITUDE, MARKER='O', LABEL=FLIGHT_NO)
    AX.TEXT(LONGITUDE, LATITUDE, ALTITUDE, FLIGHT_NO, FONTSIZE=8, HA='LEFT', VA='BOTTOM')

FOR GS IN GSS:
    GS_NAME, GS_LATITUDE, GS_LONGITUDE, GS_ALTITUDE = GS
    AX.SCATTER(GS_LONGITUDE, GS_LATITUDE, GS_ALTITUDE, MARKER='S', COLOR='RED', LABEL=GS_NAME)
    AX.TEXT(GS_LONGITUDE, GS_LATITUDE, GS_ALTITUDE, GS_NAME, FONTSIZE=8, HA='LEFT', VA='BOTTOM')

AX.SET_XLABEL('LONGITUDE')
AX.SET_YLABEL('LATITUDE')
AX.SET_ZLABEL('ALTITUDE')
AX.SET_TITLE('FLIGHT PATHS AND GROUND STATIONS')
PLT.SHOW()
```

1. *Plotting flight paths in 3D:*
   1. *The code utilizes the matplotlib library to create a 3D plot of flight paths and ground stations.*
   2. *It initializes a figure and creates a 3D subplot.*
   3. *It iterates over the airplanes and ground stations, plotting their coordinates on the 3D plot using scatter plots.*
   4. *Text annotations are added to label the flight numbers and ground station names.*
   5. *Axis labels and a title are set, and the plot is displayed.*

2. *Converting latitude, longitude, and altitude to 3D Cartesian coordinates:*
   1. *The code defines a function named 'convert_to_cartesian' that takes latitude, longitude, and altitude as input and converts them to 3D Cartesian coordinates.*
   2. *The latitude and longitude are first converted to radians.*
   3. *The radius of the Earth (6371000 meters) is used to calculate the X, Y, and Z coordinates.*
   4. *The function returns the calculated X, Y, and Z values.*

```
IMPORT MATH

# CONVERT LATITUDE, LONGITUDE, AND ALTITUDE TO 3D CARTESIAN COORDINATES
DEF CONVERT_TO_CARTESIAN(LATITUDE, LONGITUDE, ALTITUDE):
    LAT_RAD = MATH.RADIANS(LATITUDE)
    LON_RAD = MATH.RADIANS(LONGITUDE)
    RADIUS = 6371000

    X = RADIUS * MATH.COS(LAT_RAD) * MATH.COS(LON_RAD)
    Y = RADIUS * MATH.COS(LAT_RAD) * MATH.SIN(LON_RAD)
    Z = ALTITUDE

    RETURN X, Y, Z

# CALCULATE THE DISTANCE BETWEEN TWO POINTS IN 3D CARTESIAN COORDINATES
DEF CALCULATE_DISTANCE(POINT1, POINT2):
    _, LAT1, LON1, ALT1 = POINT1[:4]
    _, LAT2, LON2, ALT2 = POINT2[:4]

    X1, Y1, Z1 = CONVERT_TO_CARTESIAN(LAT1, LON1, ALT1)
    X2, Y2, Z2 = CONVERT_TO_CARTESIAN(LAT2, LON2, ALT2)
    DISTANCE = MATH.SQRT((X2 - X1) ** 2 + (Y2 - Y1) ** 2 + (Z2 - Z1) ** 2)
    RETURN DISTANCE
```

CALCULATING THE DISTANCE BETWEEN TWO POINTS IN 3D CARTESIAN COORDINATES:

- *The code defines a function named 'calculate distance' that calculates the distance between two points in 3d cartesian coordinates.*

- *The function takes two points as input and extracts the latitude, longitude, and altitude values.*

- *The 'convert_to_cartesian' function is called to convert the latitude, longitude, and altitude to cartesian coordinates.*

- *the Euclidean distance between the two points is calculated using the cartesian coordinates.*
- *the distance value is returned.*

```
IMPORT MATPLOTLIB.PYPLOT AS PLT
IMPORT NUMPY AS NP
FROM SKLEARN.PREPROCESSING IMPORT STANDARDSCALER
FROM SCIPY IMPORT STATS

# FIT DISTRIBUTIONS TO THE SCALED COLUMNS
LATITUDE_DIST = STATS.NORM.FIT(SCALED_LATITUDE)
LONGITUDE_DIST = STATS.NORM.FIT(SCALED_LONGITUDE)
ALTITUDE_DIST = STATS.NORM.FIT(SCALED_ALTITUDE)

# PLOTTING DISTRIBUTIONS
PLT.FIGURE(FIGSIZE=(10, 6))

# LATITUDE DISTRIBUTION
PLT.SUBPLOT(1, 3, 1)
PLT.HIST(SCALED_LATITUDE, BINS='AUTO', DENSITY=TRUE, ALPHA=0.7, COLOR='B')
X = NP.LINSPACE(SCALED_LATITUDE.MIN(), SCALED_LATITUDE.MAX(), 100)
PLT.PLOT(X, STATS.NORM.PDF(X, *LATITUDE_DIST), 'R-', LW=2)
PLT.XLABEL('SCALED LATITUDE')
PLT.YLABEL('DENSITY')
PLT.TITLE('LATITUDE DISTRIBUTION')

# LONGITUDE DISTRIBUTION
PLT.SUBPLOT(1, 3, 2)
PLT.HIST(SCALED_LONGITUDE, BINS='AUTO', DENSITY=TRUE, ALPHA=0.7, COLOR='B')
X = NP.LINSPACE(SCALED_LONGITUDE.MIN(), SCALED_LONGITUDE.MAX(), 100)
PLT.PLOT(X, STATS.NORM.PDF(X, *LONGITUDE_DIST), 'R-', LW=2)
PLT.XLABEL('SCALED LONGITUDE')
PLT.YLABEL('DENSITY')
PLT.TITLE('LONGITUDE DISTRIBUTION')

# ALTITUDE DISTRIBUTION
PLT.SUBPLOT(1, 3, 3)
PLT.HIST(SCALED_ALTITUDE, BINS='AUTO', DENSITY=TRUE, ALPHA=0.7, COLOR='B')
X = NP.LINSPACE(SCALED_ALTITUDE.MIN(), SCALED_ALTITUDE.MAX(), 100)
PLT.PLOT(X, STATS.NORM.PDF(X, *ALTITUDE_DIST), 'R-', LW=2)
PLT.XLABEL('SCALED ALTITUDE')
PLT.YLABEL('DENSITY')
PLT.TITLE('ALTITUDE DISTRIBUTION')

PLT.TIGHT_LAYOUT()
PLT.SHOW()
```

EXPLANATION –

PLOTTING DISTRIBUTIONS:

- *The code uses the matplotlib library to plot distributions of latitude, longitude, and altitude.*
- *It fits normal distributions to the scaled latitude, longitude, and altitude values using the stats.norm.fit() function.*
- *Subplots are created for each distribution, and histograms of the scaled values are plotted along with the fitted normal distributions.*
- *Axis labels and titles are set, and the plot is displayed.*

- *The code assumes that there are already scaled versions of the variables (e.g., scaled_latitude, scaled_longitude, scaled_altitude) available.*
- *Distribution fitting is performed using the stats.norm.fit() function from the scipy.stats module.*
- *For each variable, the function estimates the parameters of a normal distribution (mean and standard deviation) that best fit the scaled values.*
- *The resulting parameters are stored in latitude_dist, longitude_dist, and altitude_dist, respectively.*

- *Creating the Figure and Subplots:*
- *The code initializes a figure using plt.figure(figsize=(10, 6)), specifying the size of the figure.*
- *Three subplots are created using plt.subplot(1, 3, 1), plt.subplot(1, 3, 2), and plt.subplot(1, 3, 3).*
- *The numbers 1, 3, 1 in plt.subplot(1, 3, 1) indicate that the figure should have 1 row, 3 columns, and the current plot is the first one.*

```
# CALCULATE THE DATA TRANSMISSION RATE FOR A GIVEN DISTANCE
DEF CALCULATE_DATA_TRANSMISSION_RATE(DISTANCE):
    IF DISTANCE < 300:
        RETURN 100
    ELIF DISTANCE < 400:
        RETURN 75
    ELIF DISTANCE < 500:
        RETURN 50
    ELSE:
        RETURN 25


# CALCULATE THE END-TO-END DATA TRANSMISSION RATE FOR A ROUTING PATH
DEF CALCULATE_END_TO_END_DATA_TRANSMISSION_RATE(PATH):
    DATA_TRANSMISSION_RATES =
[CALCULATE_DATA_TRANSMISSION_RATE(CALCULATE_DISTANCE(PATH[I], PATH[I+1]))
                               FOR I IN RANGE(LEN(PATH)-1)]
    RETURN MIN(DATA_TRANSMISSION_RATES)


# CALCULATE THE END-TO-END LATENCY FOR A ROUTING PATH
DEF CALCULATE_END_TO_END_LATENCY(PATH):
    LATENCIES = [CALCULATE_DISTANCE(PATH[I], PATH[I+1]) FOR I IN
RANGE(LEN(PATH)-1)]
    RETURN SUM(LATENCIES)
```

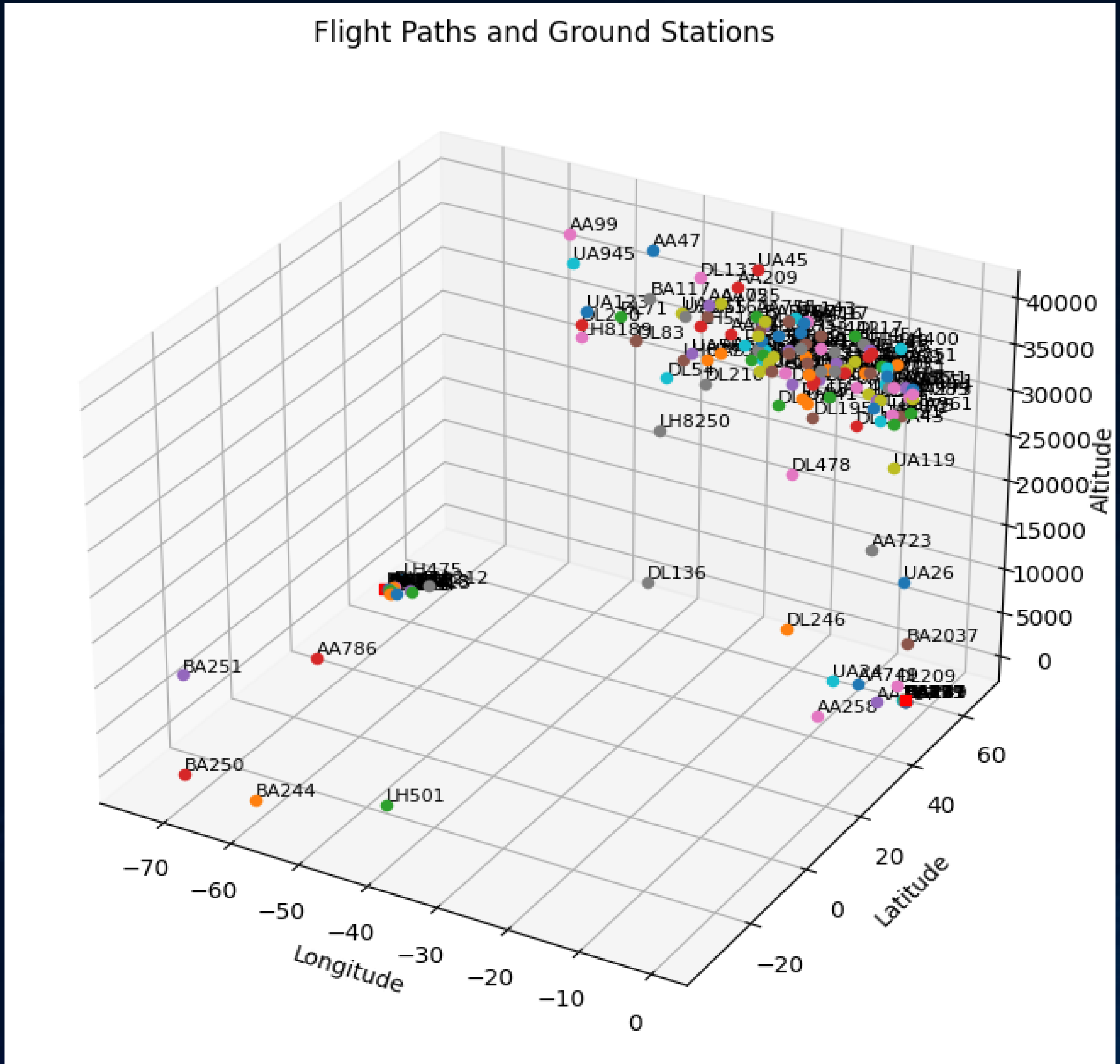
CALCULATING DATA TRANSMISSION RATES:

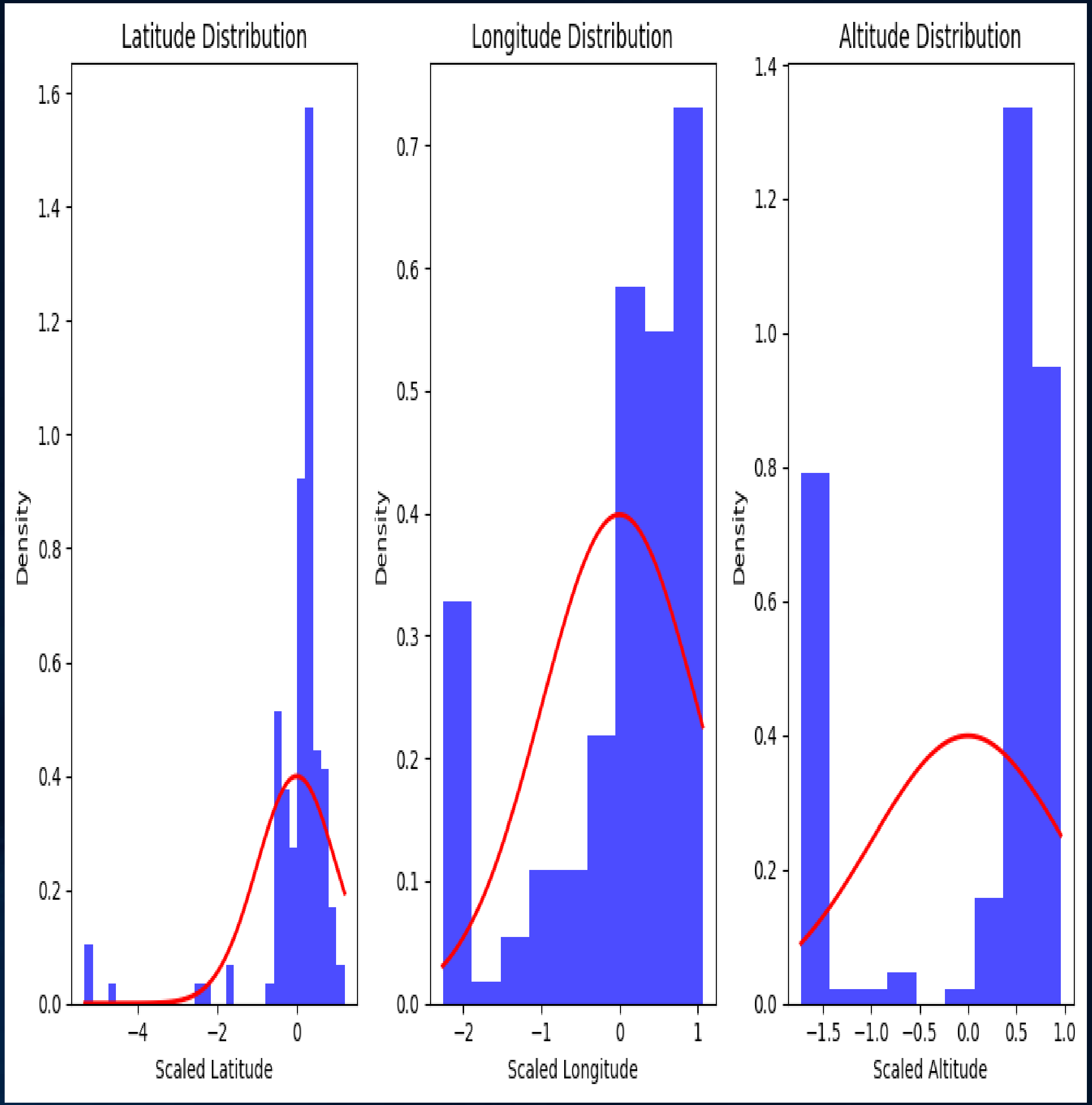
The code defines a function named 'calculate_data_transmission_rate' that calculates the data transmission rate based on the given distance.
The function takes the distance as input and applies different rate rules based on the distance ranges.
The calculated data transmission rate is returned.

CALCULATING END-TO-END DATA TRANSMISSION RATES AND LATENCIES:

The code defines a function named 'calculate_end_to_end_data_transmission_rate' that calculates the end-to-end data transmission rate for a given routing path.

The function takes a path (list of points) as input and calculates the data transmission rate for each segment of the path using the 'calculate_data_transmission_rate' function.
The minimum data transmission rate among all segments is returned as the end-to-end data transmission rate.
The code also defines a function named 'calculate_end_to_end_latency' that calculates the end-to-end latency for a given routing path.
The function takes a path as input and calculates the distance between each consecutive pair of points in the path.
The sum of distances is returned as the end-to-end latency.


Flight Paths and Ground Stations
Longitude
Latitude
Altitude
AA99
AA47
UA945
UA45
AA209
LH8250
DL478
UA119
AA723
DL136
UA26
DL246
BA2037
BA251
AA786
AA258
BA250
BA244
LH501


Latitude Distribution
Longitude Distribution
Altitude Distribution
Density
Scaled Latitude
Scaled Longitude
Scaled Altitude

```
FROM SCIPY.OPTIMIZE IMPORT MINIMIZE

# OPTIMIZATION OBJECTIVE FUNCTION FOR SINGLE-OBJECTIVE OPTIMIZATION
DEF OBJECTIVE_SINGLE(X, *ARGS):
    AIRPLANE_INDEX, GS_INDEX = ARGS
    AIRPLANE = AIRPLANES[AIRPLANE_INDEX]
    GS = GSS[GS_INDEX]

    PATH = [AIRPLANE, GS]
    DATA_TRANSMISSION_RATE =
CALCULATE_END_TO_END_DATA_TRANSMISSION_RATE(PATH)

    RETURN -DATA_TRANSMISSION_RATE
```

OPTIMIZATION OBJECTIVE FUNCTIONS:

- *The code defines two objective functions used for optimization: 'objective_single' and 'objective_multi'.*

- *The 'objective_single' function is used for single-objective optimization and maximizes the data transmission rate.*

- *The 'objective_multi' function is used for multiple-objective optimization and combines the data transmission rate and latency using a weighted sum approach.*

```
# DEFINE THE AIRPLANES AND GROUND STATIONS
AIRPLANES = DATA[['FLIGHT NO.', 'TIMESTAMP', 'ALTITUDE', 'LATITUDE', 'LONGITUDE']].VALUES
GSS = [('LHR', 51.4700, -0.4543, 81.73), ('EWR', 40.6895, -74.1745, 8.72)]


# DEFINE THE AIRPLANES AND GROUND STATIONS
AIRPLANES = DATA[['FLIGHT NO.', 'TIMESTAMP', 'ALTITUDE', 'LATITUDE', 'LONGITUDE']].VALUES
GSS = [('LHR', 51.4700, -0.4543, 81.73), ('EWR', 40.6895, -74.1745, 8.72)]


# DEFINE THE AIRPLANES AND GROUND STATIONS
AIRPLANES = DATA[['FLIGHT NO.', 'TIMESTAMP', 'ALTITUDE', 'LATITUDE', 'LONGITUDE']].VALUES
GSS = [('LHR', 51.4700, -0.4543, 81.73), ('EWR', 40.6895, -74.1745, 8.72)]


# DEFINE THE AIRPLANES AND GROUND STATIONS
AIRPLANES = DATA[['FLIGHT NO.', 'TIMESTAMP', 'ALTITUDE', 'LATITUDE', 'LONGITUDE']].VALUES
GSS = [('LHR', 51.4700, -0.4543, 81.73), ('EWR', 40.6895, -74.1745, 8.72)]


# DEFINE THE AIRPLANES AND GROUND STATIONS
AIRPLANES = DATA[['FLIGHT NO.', 'TIMESTAMP', 'ALTITUDE', 'LATITUDE', 'LONGITUDE']].VALUES
GSS = [('LHR', 51.4700, -0.4543, 81.73), ('EWR', 40.6895, -74.1745, 8.72)]


# SOLVE THE SINGLE-OBJECTIVE OPTIMIZATION PROBLEM
SINGLE_SOLUTIONS = []
FOR AIRPLANE_INDEX IN RANGE(LEN(AIRPLANES)):
    AIRPLANE_FLIGHT_NO, AIRPLANE_TIMESTAMP, AIRPLANE_ALTITUDE, AIRPLANE_LATITUDE,
AIRPLANE_LONGITUDE = AIRPLANES[AIRPLANE_INDEX]
    FOR GS_INDEX IN RANGE(LEN(GSS)):
        BOUNDS = [(0, 1)]
        ARGS = (AIRPLANE_INDEX, GS_INDEX)
        RESULT = MINIMIZE(OBJECTIVE_SINGLE, [0], ARGS=ARGS, BOUNDS=BOUNDS, METHOD='SLSQP')
        IF RESULT.SUCCESS:
            PATH = [(AIRPLANE_FLIGHT_NO, AIRPLANE_LATITUDE, AIRPLANE_LONGITUDE,
AIRPLANE_ALTITUDE), GSS[GS_INDEX]]
            DATA_TRANSMISSION_RATE = CALCULATE_END_TO_END_DATA_TRANSMISSION_RATE(PATH)
            SOLUTION = {'FLIGHT NO.': AIRPLANE_FLIGHT_NO,
                        'ROUTING PATH': PATH,
                        'END-TO-END DATA TRANSMISSION RATE': DATA_TRANSMISSION_RATE}
            SINGLE_SOLUTIONS.APPEND(SOLUTION)
```

```
# EXTRACTING DATA TRANSMISSION RATES FOR SINGLE-OBJECTIVE OPTIMIZATION
SINGLE_RATES = [SOLUTION['END-TO-END DATA TRANSMISSION RATE'] FOR SOLUTION IN
SINGLE_SOLUTIONS]

# BAR CHART FOR SINGLE-OBJECTIVE OPTIMIZATION
PLT.FIGURE(FIGSIZE=(10, 6))
PLT.BAR(RANGE(LEN(SINGLE_RATES)), SINGLE_RATES)
PLT.XLABEL('ROUTING PATH INDEX')
PLT.YLABEL('DATA TRANSMISSION RATE')
PLT.TITLE('END-TO-END DATA TRANSMISSION RATES (SINGLE OBJECTIVE)')
PLT.XTICKS(RANGE(LEN(SINGLE_RATES)))
PLT.GRID(TRUE)
PLT.SHOW()
```

SINGLE-OBJECTIVE OPTIMIZATION:

The code solves the single-objective optimization problem for each combination of airplanes and ground stations.

It iterates over airplanes and ground stations and uses the 'minimize' function from the scipy library to find the optimal solution.

The 'objective_single' function is passed as the objective function, and the bounds are set to [0, 1] for the optimization variable.

If the optimization is successful, the routing path, end-to-end data transmission rate, and flight number are stored in the 'single_solutions' list.

The data transmission rates from the solutions are extracted into the 'single_rates' list.

Finally, a bar chart is plotted to visualize the end-to-end data transmission rates for the single-objective optimization.

```
# OPTIMIZATION OBJECTIVE FUNCTION FOR MULTIPLE-OBJECTIVE OPTIMIZATION
DEF OBJECTIVE_MULTI(X, *ARGS):
    AIRPLANE_INDEX, GS_INDEX = ARGS
    AIRPLANE = AIRPLANES[AIRPLANE_INDEX]
    GS = GSS[GS_INDEX]

    PATH = [AIRPLANE, GS]
    DATA_TRANSMISSION_RATE = CALCULATE_END_TO_END_DATA_TRANSMISSION_RATE(PATH)
    LATENCY = CALCULATE_END_TO_END_LATENCY(PATH)

    # WEIGHTED SUM APPROACH TO COMBINE OBJECTIVES INTO A SINGLE SCALAR VALUE
    WEIGHT_DATA_TRANSMISSION_RATE = 1.0
    WEIGHT_LATENCY = 1.0
    SCALAR_VALUE = WEIGHT_DATA_TRANSMISSION_RATE * (-DATA_TRANSMISSION_RATE) +
WEIGHT_LATENCY * LATENCY
    RETURN SCALAR_VALUE

# SOLVE THE MULTIPLE-OBJECTIVE OPTIMIZATION PROBLEM
MULTI_SOLUTIONS = []
FOR AIRPLANE_INDEX IN RANGE(LEN(AIRPLANES)):
    AIRPLANE_FLIGHT_NO, AIRPLANE_TIMESTAMP, AIRPLANE_ALTITUDE, AIRPLANE_LATITUDE,
AIRPLANE_LONGITUDE = AIRPLANES[AIRPLANE_INDEX]
    FOR GS_INDEX IN RANGE(LEN(GSS)):
        BOUNDS = [(0, 1)]
        ARGS = (AIRPLANE_INDEX, GS_INDEX)
        RESULT = MINIMIZE(OBJECTIVE_MULTI, [0], ARGS=ARGS, BOUNDS=BOUNDS,
METHOD='SLSQP')
        IF RESULT.SUCCESS:
            PATH = [(AIRPLANE_FLIGHT_NO, AIRPLANE_LATITUDE, AIRPLANE_LONGITUDE,
AIRPLANE_ALTITUDE), GSS[GS_INDEX]]
            SCALAR_VALUE = RESULT.FUN
            DATA_TRANSMISSION_RATE =
CALCULATE_END_TO_END_DATA_TRANSMISSION_RATE(PATH)
            LATENCY = CALCULATE_END_TO_END_LATENCY(PATH)
            SOLUTION = {'FLIGHT NO.': AIRPLANE_FLIGHT_NO,
                        'ROUTING PATH': PATH,
                        'END-TO-END DATA TRANSMISSION RATE': DATA_TRANSMISSION_RATE,
                        'END-TO-END LATENCY': LATENCY}
            MULTI_SOLUTIONS.APPEND(SOLUTION)
```

```
IMPORT MATPLOTLIB.PYPLOT AS PLT

# EXTRACTING DATA TRANSMISSION RATES AND LATENCIES FOR
MULTIPLE-OBJECTIVE OPTIMIZATION
MULTI_RATES = [SOLUTION['END-TO-END DATA TRANSMISSION RATE']
FOR SOLUTION IN MULTI_SOLUTIONS]
MULTI_LATENCIES = [SOLUTION['END-TO-END LATENCY'] FOR
SOLUTION IN MULTI_SOLUTIONS]

# SCATTER PLOT FOR MULTIPLE-OBJECTIVE OPTIMIZATION
PLT.FIGURE(FIGSIZE=(10, 6))
PLT.SCATTER(MULTI_LATENCIES, MULTI_RATES, C='BLUE',
ALPHA=0.7)
PLT.XLABEL('END-TO-END LATENCY')
PLT.YLABEL('DATA TRANSMISSION RATE')
PLT.TITLE('TRADE-OFF BETWEEN DATA TRANSMISSION RATE AND
LATENCY (MULTIPLE OBJECTIVES)')
PLT.GRID(TRUE)
PLT.SHOW()
```

MULTIPLE-OBJECTIVE OPTIMIZATION:

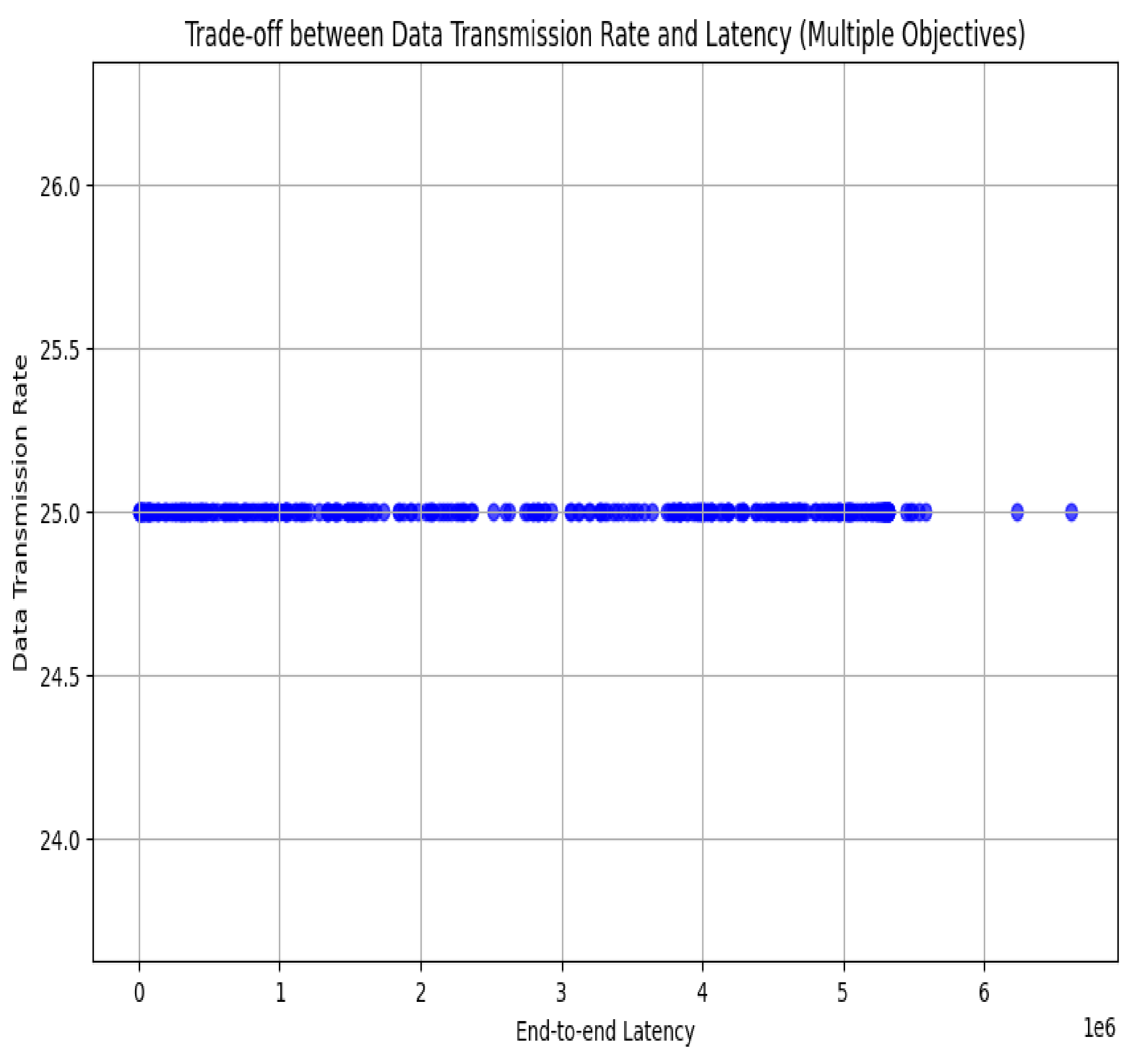
The code solves the multiple-objective optimization problem using a similar approach as in single-objective optimization.

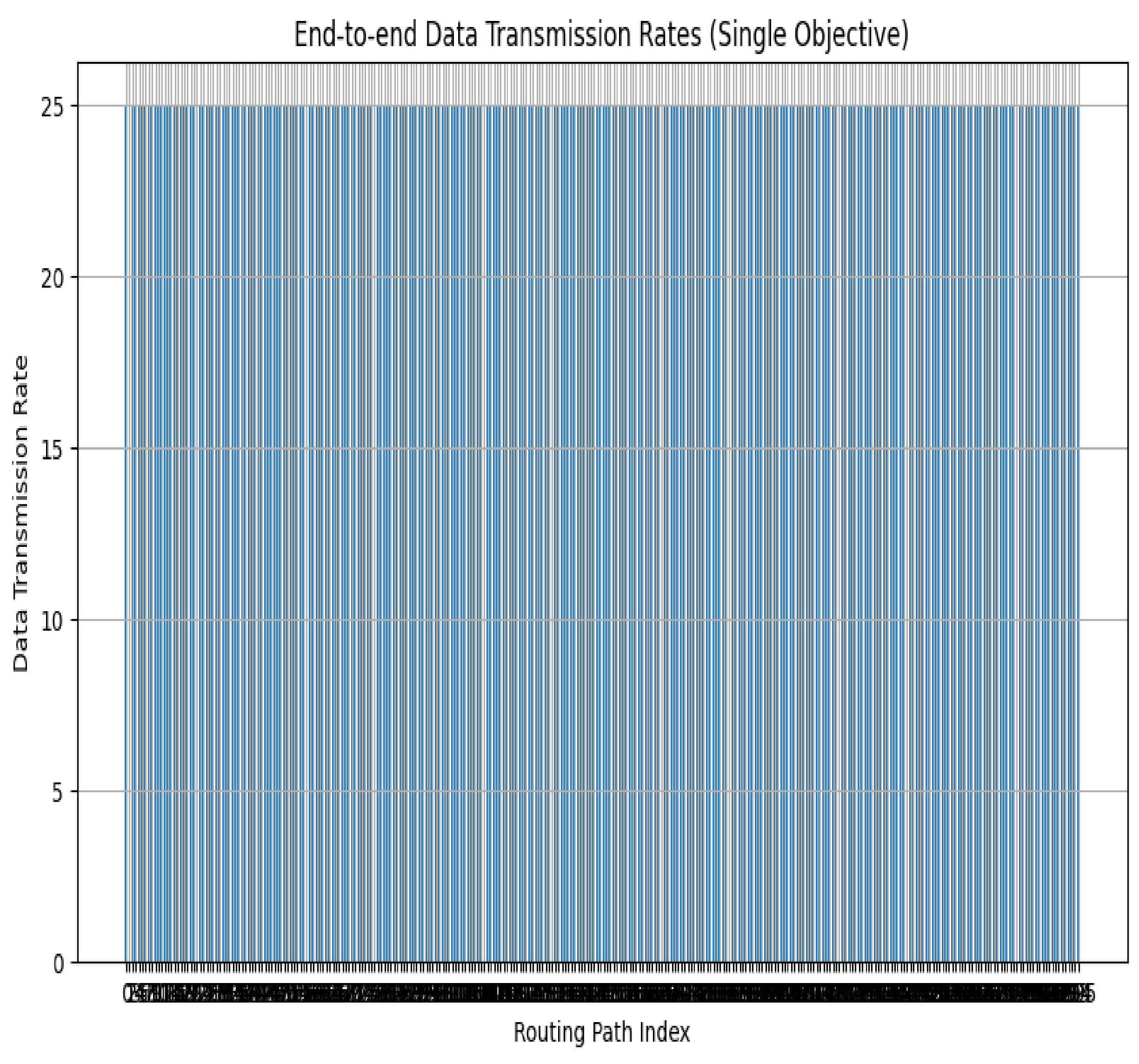
The 'objective_multi' function is used, which combines the data transmission rate and latency into a scalar value.

The solutions, including the routing path, data transmission rate, and latency, are stored in the 'multi_solutions' list.

The data transmission rates and latencies from the solutions are extracted into the 'multi_rates' and 'multi_latencies' lists.

A scatter plot is created to visualize the trade-off between data transmission rate and latency for the multiple-objective optimization.


Trade-off between Data Transmission Rate and Latency (Multiple Objectives)
Data Transmission Rate
End-to-end Latency
1e6
24.0
24.5
25.0
25.5
26.0
0
1
2
3
4
5
6


End-to-end Data Transmission Rates (Single Objective)
Data Transmission Rate
Routing Path Index
0
5
10
15
20
25

```
# PRINT THE SOLUTIONS FOR SINGLE-OBJECTIVE OPTIMIZATION
FOR SOLUTION IN SINGLE_SOLUTIONS:
    PRINT(SOLUTION)
```

PRINTING THE SOLUTIONS:

1. *The code iterates over the solutions obtained from single-objective optimization and prints the flight number, routing path, and end-to-end data transmission rate.*

- *This code demonstrates how to analyze flight data, visualize flight paths, calculate distances in 3D space, fit distributions, optimize routing paths for data transmission, and visualize trade-offs between multiple objectives.*